Cray Management System for XC Systems with SMW 8.0/CLE 6.0

Harold Longley, Cray Inc.

CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- Configuration
- Booting
- Reconfiguration
- Software update
- Summary
- Q & A

Introduction – Why?

- **Why change?**
  - Refresh system software architecture
    - Previous generation tools were developed over 10 years ago for XT systems and then grew with add-on features for XE/XK and XC systems
  - Move away from Cray unique solutions
    - sharedroot with specialized /etc for default, class, node
    - persistent /var for service nodes
    - Release switching with xtrelswitch
  - Make system administration:
    - easier
    - more common
    - require less downtime
  - Provide common solutions for multiple Cray products

Introduction – Why?

- **Why is this hard?**
  - Cray software installation and configuration management must scale for nodes which utilize non-persistent root filesystems
  - CMS tools must preserve the system reliability and scalability upon which Cray customers depend

Introduction – How?

- **Leverage standard Linux and common open source tools**
  - rpm with zypper for SUSE or yum for CentOS
  - Manage storage using LVM with BTRFS and XFS filesystems
  - BTRFS snapshots on SMW
  - Configuration data in YAML and JSON file formats
  - Configuration management tools (Ansible)
- **Common installation process for SMW and CLE**
  - Support different Linux versions
  - Staged upgrades

Introduction – How?

- **Separation of software and configuration**
  - Prescriptive image creation
    - Create images based on “recipes”
    - Utilizes rpm dependencies
  - Centralization of configuration
    - Provide structure for configuration data
    - Provide a tool to manage the configuration data
    - Provide a framework for configuration
  - Ability to “share” config data – eLogin
  - Ability to “clone” a configuration and test
  - Boot images are system agnostic
    - ability to create and test boot images on one system and push to other systems
  - Node customization at boot time or after adjusting configuration

Introduction – When?

- **Next major releases for CLE and SMW**
  - CLE 6.0 (Rhine)
  - SMW 8.0 (Redwood)
- **SMW and CLE based on SLES 12 for x86_64**
- **HSS controllers based on OpenSUSE 13.2 for 32bit**
- **New software installers**
- **New system management**
  - IMPS (Image Management and Provisioning System)
  - CMF (Configuration Management Framework)
  - NIMS (Node Image Mapping Service)
  - Ansible configuration management
- **Much software is mostly unchanged (HSS, ALPS, NHC, RUR, etc.)**

Separation of Software and Configuration

- **Image Management and Provisioning System (IMPS)**
  - IMPS repository management
    - Create and distribute repository content (rpms)
  - IMPS Prescriptive Image Creation
    - Create and update standard or custom images
- **Configuration Management Framework (CMF)**
  - Centralized Configuration on SMW
  - Create, change, or add new configuration information, including site specific configuration

NIMS

- **Node Image Mapping Service**
  - Which images get booted on specific nodes
  - Additional kernel parameters to pass to the nodes on boot
  - Which load file to use within a boot image
- **NIMS daemon – nimsd**
  - Holds that information
  - Responds to requests from the boot manager
- **NIMS group**
  - assign a node to one group: service, login, DAL, compute, mygroup
- **Administrator commands to NIMS daemon**
  - Actions can be performed on a set of nodes specified with filters

Fresh Install

- **Install SMW with SLES12**
  - SLES12 with root filesystem on software RAID1 for R815 SMW and hardware RAID5 for R630 SMW
- **Configure disk space on SMW and Boot RAID**
  - Everything uses LVM and btrfs, xfs, or ext4
- **Create and populate repositories**
  - Shared across multiple Cray products
- **Install/Configure Hardware Supervisory System (HSS)**
  - SLES12/OpenSUSE, but otherwise largely unmodified
  - Cabinet controllers (CC) and blade controllers (BC)

Fresh Install

- **Configure CLE**
  - The “configurator” command cfgset
- **Build CLE images**
  - Build boot images using prescriptive image recipes
- **Boot system**
  - Nodes boot unconfigured boot image
  - Configuration applied by Ansible plays in two phases
    - Early /init
    - Multi-user boot
  - Nodes with storage configure it during initial boot

Why Ansible?

- Modern open source configuration management solution
- Easiest to use from current configuration management solutions
- Easy to pass “variables” via files or scripts that output JSON or YAML
- Least client dependencies of modern solutions
- Can work in “client-less” mode only requiring SSH and Python
- Can work in pull or push mode
- Written in Python
- Good library of “modules”
  - User can provide their own modules

System upgrades

- **While running current system in production…**
  - Create snapshots and clone config sets
  - New repositories and image recipes loaded
  - New software applied to snapshots
  - New CLE images built
  - New configuration applied to cloned config sets
- **Shut down CLE system and reboot SMW**
  - Reboot cabinet controllers (CC) and blade controllers (BC) with new controller image
  - Update firmware on various controllers
  - Refresh snapshots and config sets where necessary
- **Boot new CLE system**
- **Fallback if necessary**

CLE Software Updates (patches, security fixes)

- **Install and configure in a BTRFS snapshot on SMW**
  - Instead of contaminating your existing, working environment
- **Apply rpms to update repositories on SMW**
- **Rebuild images and stage for booting**
- **Live update feature**
  - CLE service and compute nodes have zypper or yum repositories which use scalable services (smw, tier1, tier2) as http servers for software repos
    - tier1 servers (boot node and SDB node) reference SMW
    - tier2 servers reference tier1 servers
    - All other nodes reference tier2 servers
  - On each node use zypper or yum commands to update rpms from those repos
- **Rolling the software update out:**
  - Compute Nodes
    - Live Update compute nodes if possible
    - Reboot between jobs
  - Service Nodes
    - Live Update service nodes if possible
    - Reboot service nodes if necessary

CMS Agenda

- Introduction
- **Overview of new concepts**
- Software installation
- Configuration
- Booting
- Reconfiguration
- Software update
- Summary
- Q & A

Overview of New Concepts

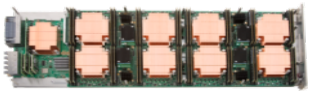
- **Review of Cray XC System**
- **Separate software and configuration**
- **Management of software**
  - Repositories
  - Image recipes
  - Package collections
  - Image root
  - Boot image
- **Configuration Management Framework**
  - How it gets created and managed from centralized location
- **Boot process**
  - Customizing software with configuration during boot process
- **Software installation**
- **eLogin**

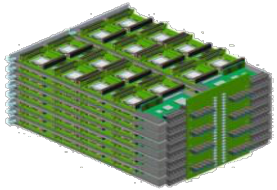
Review – Cray XC System

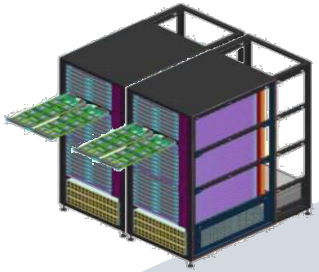
Blower cabinet Compute cabinets Blower cabinet Compute cabinets

EMI/RFI filter
on both inlet (left side)
and outlet (right side)

Review – Cray XC System Building Blocks







Compute Blade

4 Nodes

Service (I/O) Blade

2 Nodes

Chassis

Rank 1 Network

16 Compute Blades

No Cables

64 Compute Nodes

Group

Rank 2 Network

Passive Electrical Network

2 Cabinets

6 Chassis

384 Compute Nodes

System

Rank 3 Network

Active Optical Network

Hundreds of Cabinets

Up to 10s of thousands of nodes

Review – HSS Cabinet View

- **1 Cabinet Controller (CC) per cabinet**
  - Connects to the Power Supply rack via an Environmental board
- **1 Chassis Host per chassis (3 per cabinet)**
  - Connect to the Cabinet Controller
- **1 Blade Controller (BC) per blade**
- **Operators Panel**
  - Sets/displays X:Y coordinates
  - Sets/displays hardware soft limits
  - Cabinet type and elevation
  - Located in rear of the cabinet, upper left corner of chassis 2
- **Various control boards**
  - Blower control boards (1 per blower cabinet)
  - Coil control boards (1 for each cooling coil)
  - Air velocity and temperature sensor strips

Review - Identifying Components

- System components are labeled according to physical ID (HSS Identification), node ID, IP address, or class

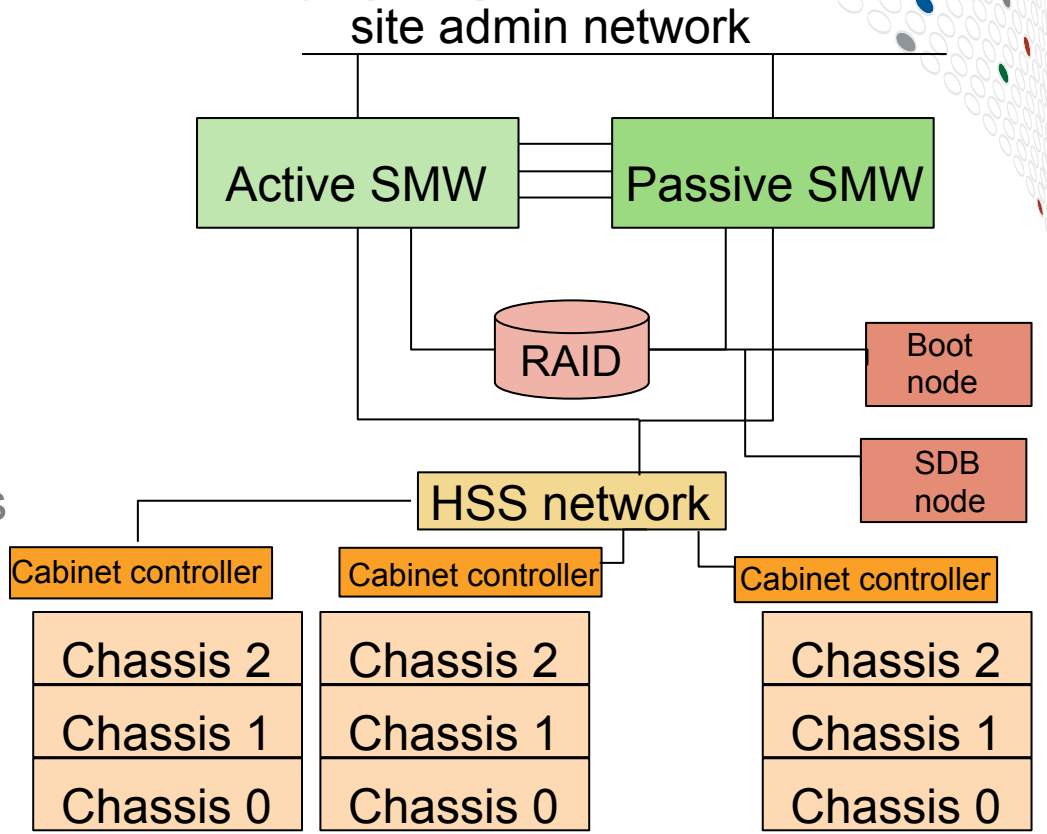
Component	Format	Description
System	s0, p0 all	All components attached to the SMW.
Cabinet	cX-Y	Cabinet number and row; this is the cabinet controller (CC) host name.
Chassis	cX-Yc#	Physical chassis in cabinet: 0, 1, 2. Chassis are numbered from bottom to top.
Blade or slot	cX-Yc#s#	Physical blade slot in chassis:0 – 15, numbered from lower left to upper right; this is the Blade controllers (BC) hosts name.
Node	cX-Yc#s#n#	Node on a blade: 0 - 3 for compute blades 1 and 2 for I/O blades
Aries ASIC	cX-Yc#s#a#	Cray Aries ASIC on a blade: always 0
Link	cX-Yc#s#a#l#	Link port of a Aries ASIC: 00 – 57 (octal)

Review – System Management Workstation (SMW)

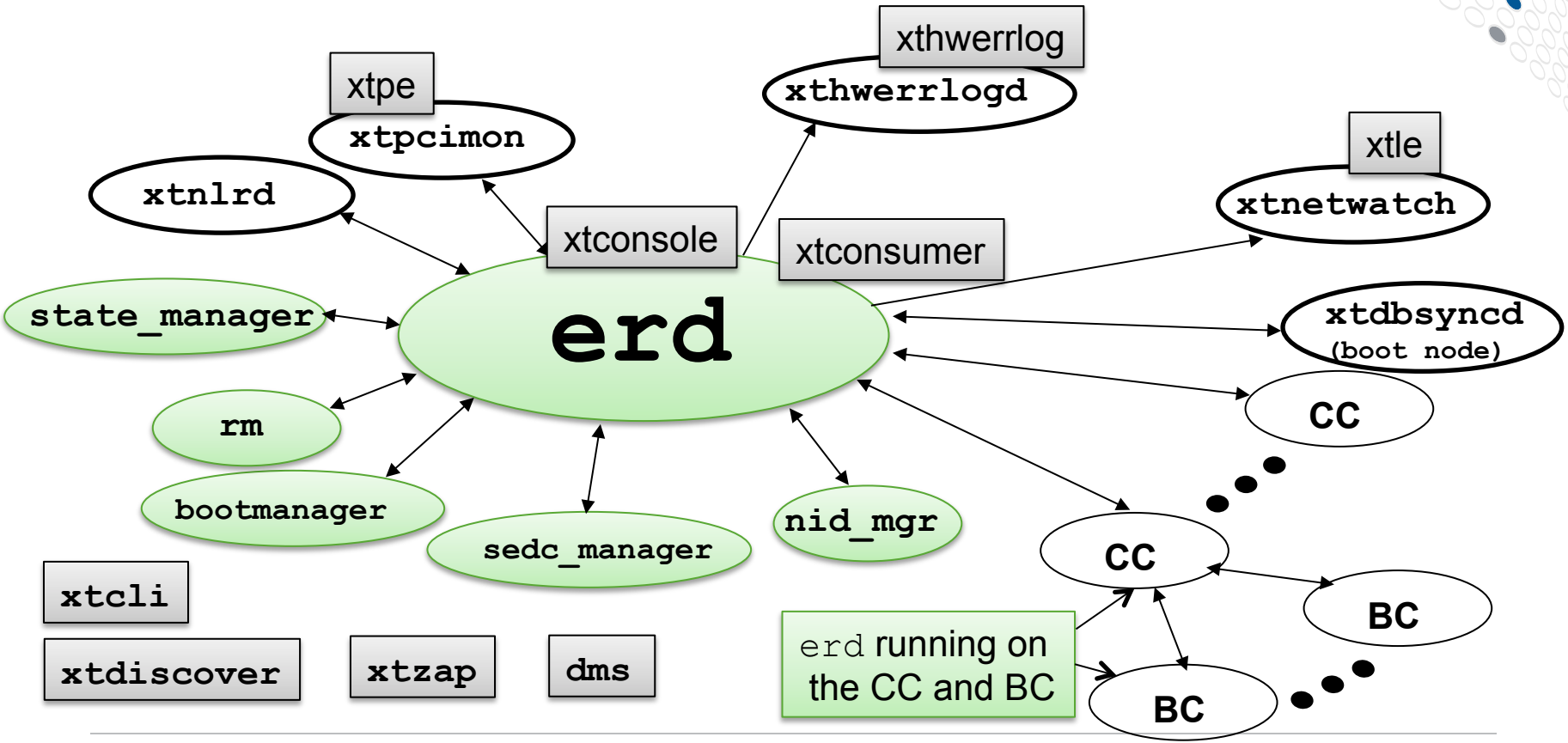
- **SMW is the single point of control for the HSS and system administration**
  - Used by operators, administrators and service personnel
    - Management of daily operations (booting, halting, and dumping), installing software, system configuration, administration, and diagnosing system faults
  - Ethernet connections:
    - Customer network
    - HSS (Hardware Supervisory System) network
    - Admin network (boot and SDB nodes)
    - With optional SMW High Availability software (SMW HA)
      - 2 heartbeat networks
      - DRBD network to replicate disk for Power Management database
  - Includes a Fibre Channel or SAS HBA connected to the boot RAID
  - Includes iDRAC (integrated Dell Remote Access Controller) on R815 and R630 SMWs
    - Remote power control
    - Remote console

Review – Optional High Availability (HA) SMW

- **Two SMWs using cluster management software (SuSE High Availability Extension) in an active/ passive mode**
  - This allows for the passive SMW to take over the duties of the active SMW in the event of a software or hardware fault on the active SMW



Review – HSS software (in SMW 7.2 and 8.0)



Review – Lightweight Log Manager (LLM)

- Automatic collection of cabinet and blade controller logs
- Automatic collection of logs from service nodes
- Inherent log rotation to avoid logs to becoming too large
- Provides system to compress and remove old logs
- Allows for external log forwarding
- Provides centralized control of log names, file formats, file sizes, file locations
- Standard log file data elements like time and host name
- Standard log file format simplifies log file parsing
- Messages logged with LLM utilize a message severity to help identify important messages
- Since log file names are date/time stamped, this allows for easier identification of which log files contain messages from which time-frames
- Log files from boot sessions are collected in separate directories for each boot session

Review – xtdiscover

- **`xtdiscover` – creates and updates the hardware database on the SMW**
  - Note: This command powers the XC system on if necessary
  - NOT used for system diagnosis
  - `xtdiscover` discovers new or changed hardware and updates the database
  - `xtdiscover --bootstrap` is used during the initial configuration of the system
    - At initial system configuration discovers the Cray system hardware and correctly identifies missing or non-responsive cabinets, empty or non-functioning slots, the blade type in each slot, CPU type, and other node attributes
  - `xtdiscover --forcenewcfg` deletes the current configuration and rediscovers the system

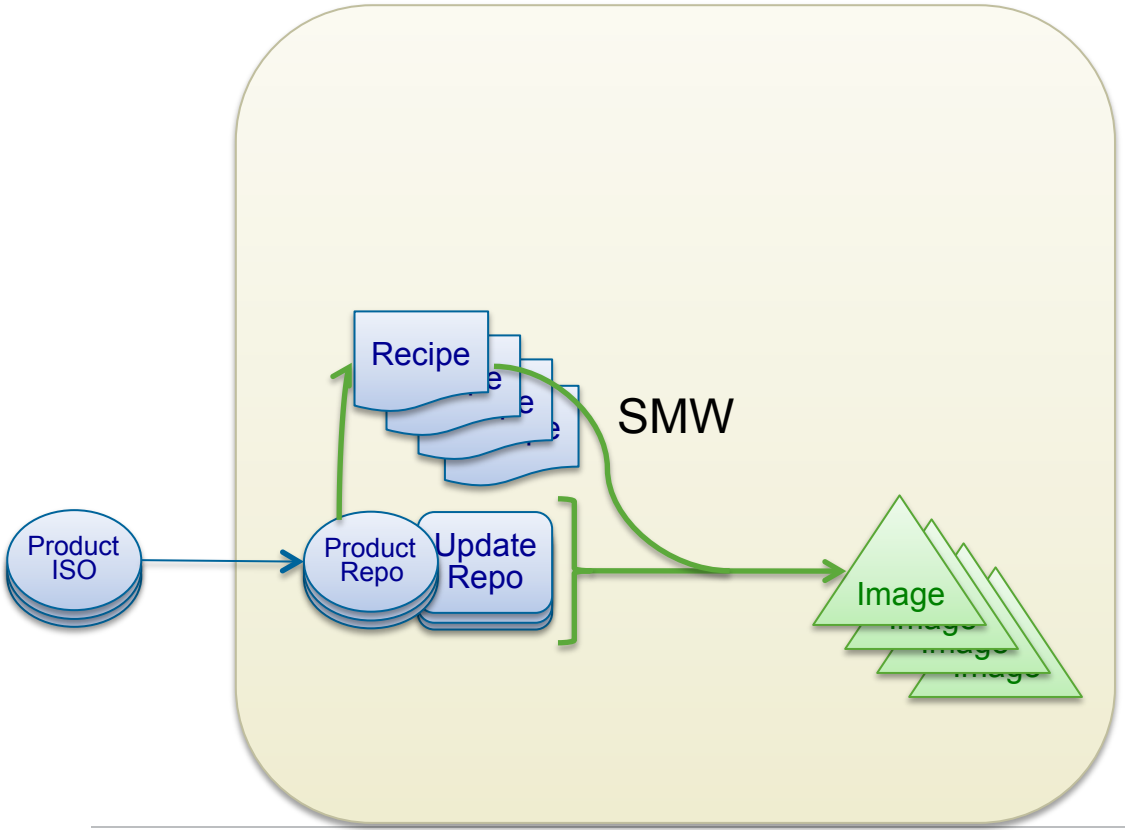
Review – Data Virtualization Service (DVS)

- **Cray DVS is a distributed network service that provides transparent access to filesystems residing on the service I/O nodes and/or remote servers in the data center**
  - Projects local filesystems resident on service nodes or remote file servers to compute and service nodes within the Cray system
    - *Projecting* makes a filesystem available on nodes where it does not physically reside
    - Uses the Linux-supplied VFS interface to process filesystem access operations
    - Can project any POSIX-compliant filesystem
      - Cray has extensively tested DVS with NFS™ and General Parallel File System (GPFS™)
  - Represents a software layer that provides scalable transport for filesystem services
    - Provides I/O performance and scalability to a large number of nodes, far beyond the typical number of clients supported by a single NFS server
    - Operating system noise and impact on compute node memory resources are both minimized in the Cray DVS configuration

Overview – Separate Software and Configuration

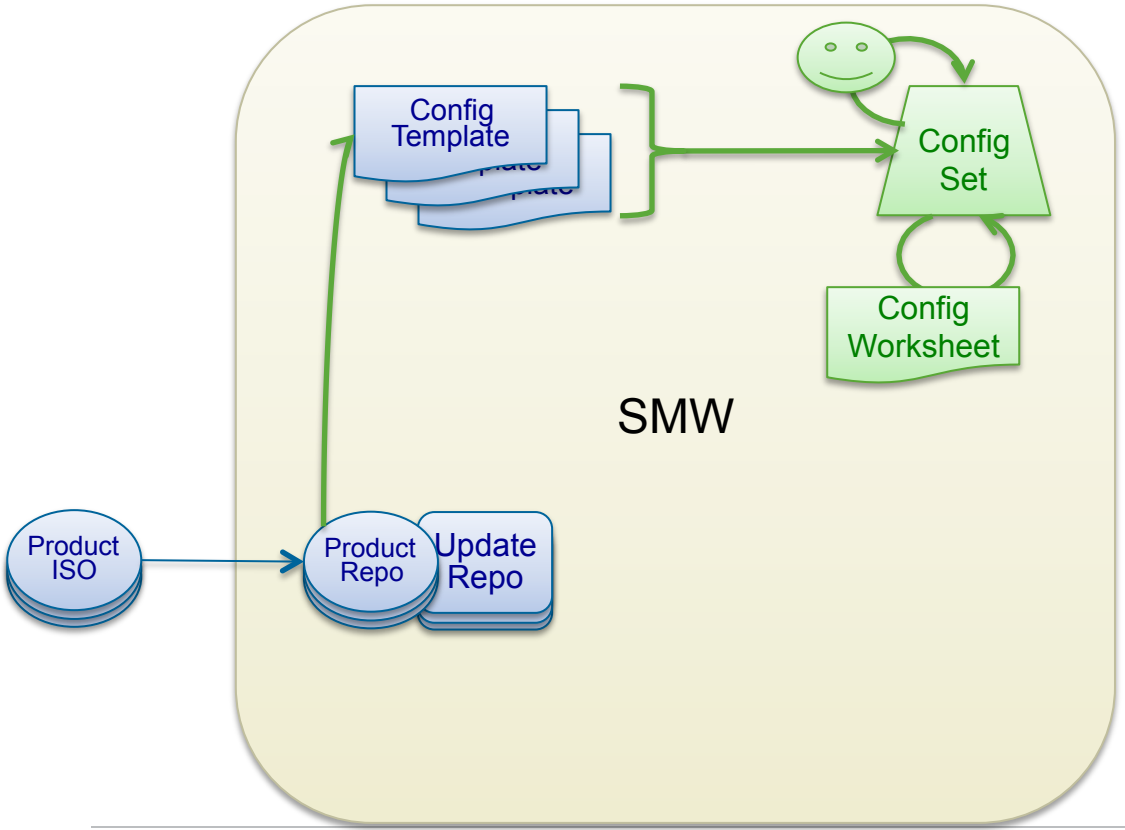
- **Node Images contain [unconfigured] code**
  - Different images for compute, service, login, DAL, …
- **Config sets contain centralized configuration**
  - Global config set used by SMW and CLE
  - CLE config set used by CLE
- **Putting software and configuration together at boot**
- **Some configuration changes can be applied after boot**

Overview – Node Images



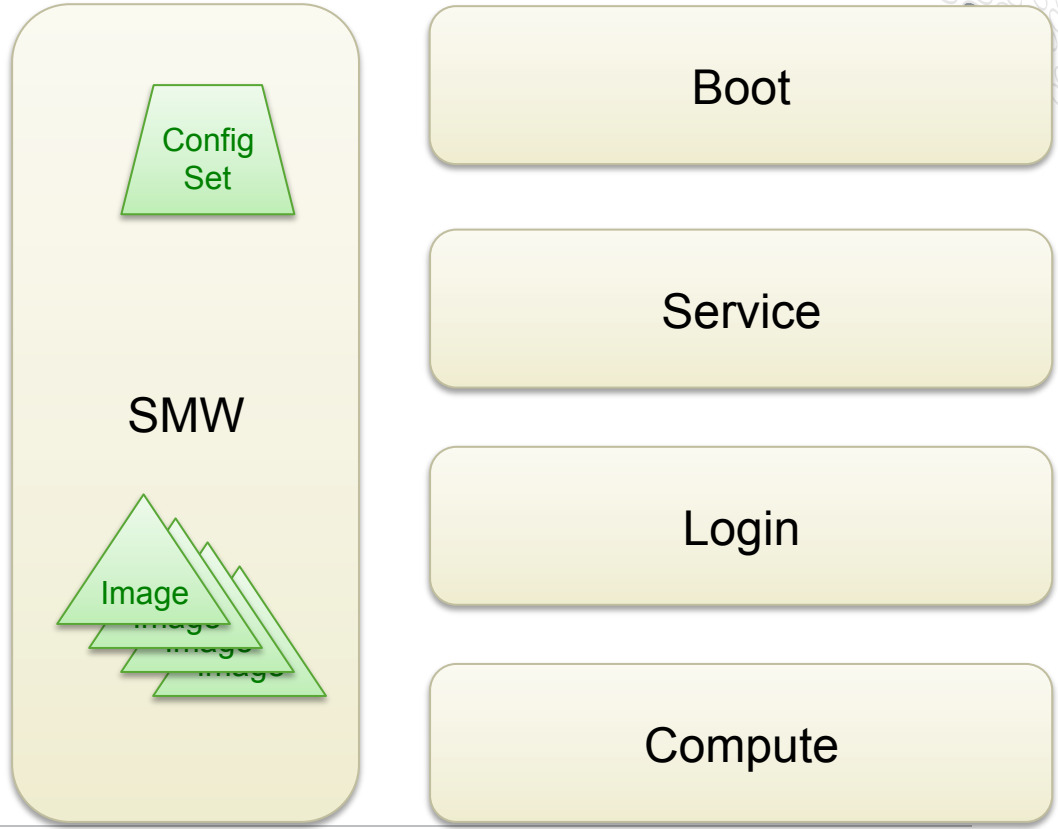
- **Install software**
  - Repositories created from product DVDs
    - Read-only
    - Empty writeable update repositories created for future use
  - Recipes installed for default image types.
- **Create images**
  - Create image roots from recipes and rpm repositories
- **The resulting images are mostly unconfigured**

Overview – Centralized Configuration



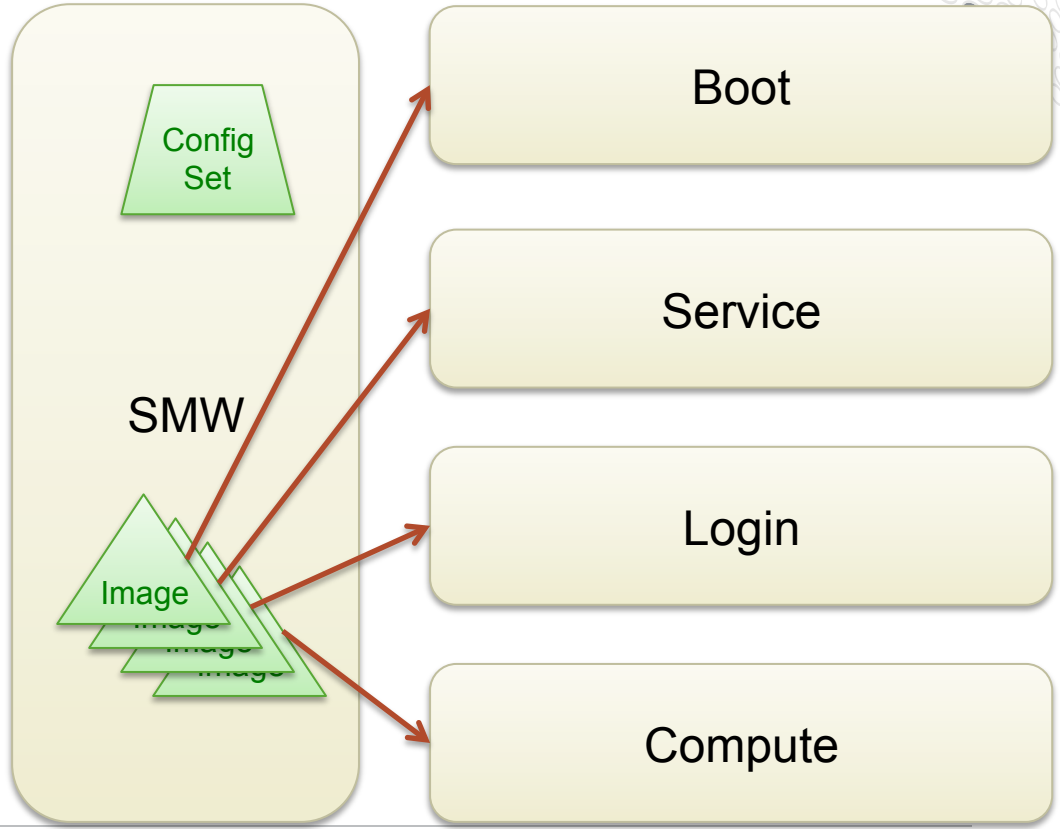
- **Config Set container**
  - Centralized configuration
  - Contains multiple configuration files, for different areas.
  - Well defined schema facilitates tools
  - YAML format allows direct editing
  - Other (non-YAML) content can be added as necessary.
- **Configurator tool**
  - Understands the schema
  - Validates values, prompts for missing content
  - Allows quick updating of specific values
  - Creates and consumes configuration worksheets
- **Configuration Worksheets**
  - For fresh installs, big changes
  - Alternative display and input method
  - Editable
  - Import into Configurator

Overview – Boot Software With Configuration



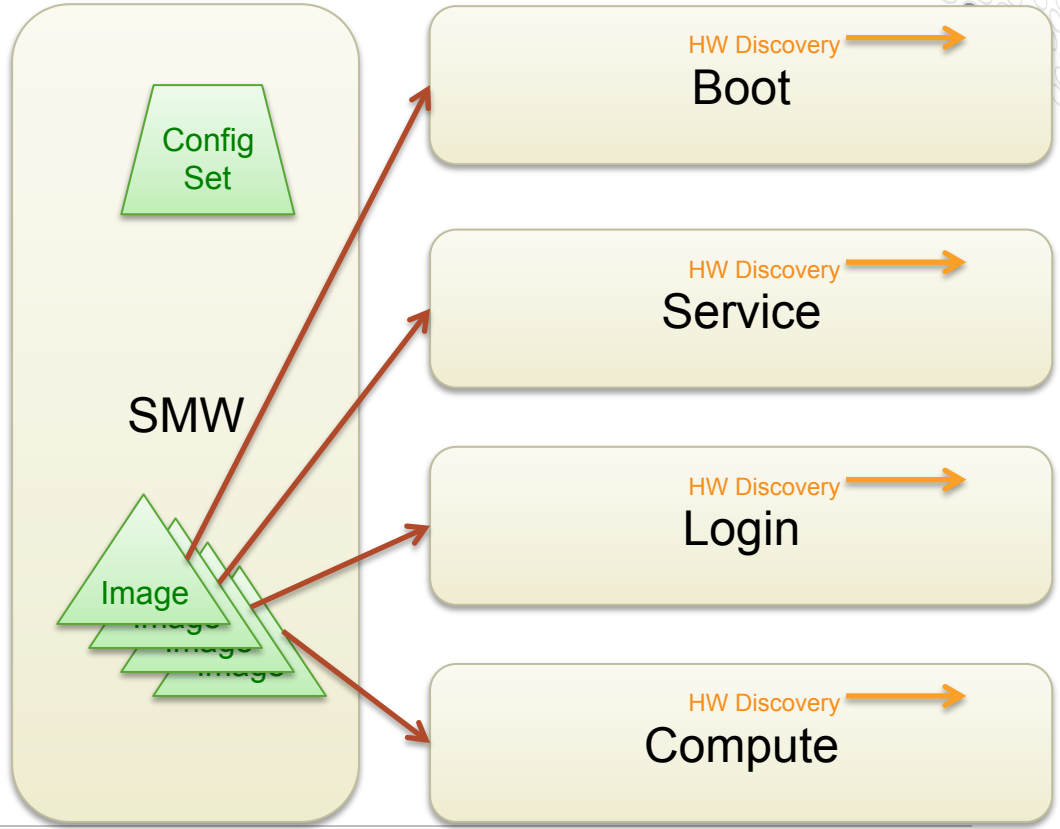
Overview – Boot Software With Configuration

- Nodes boot unconfigured image



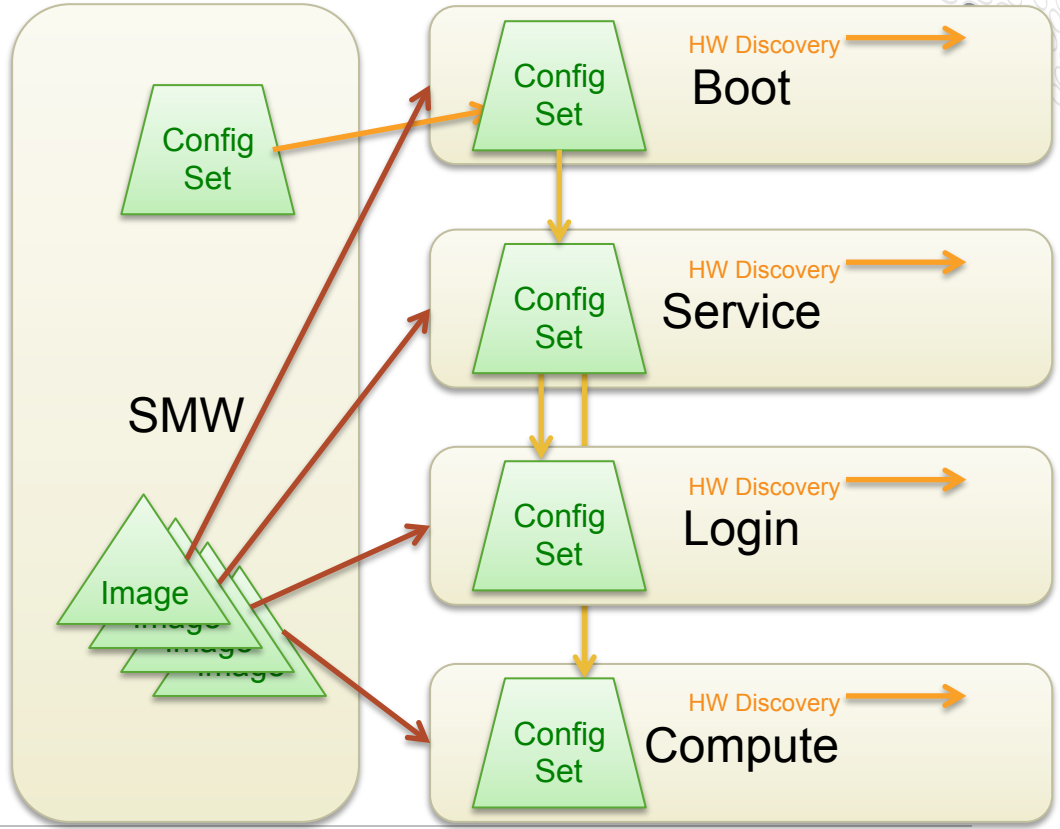
Overview – Boot Software With Configuration

- Nodes boot unconfigured image
- **Early init does:**
  - Basic discovery of node ID, kernel parameters, etc.



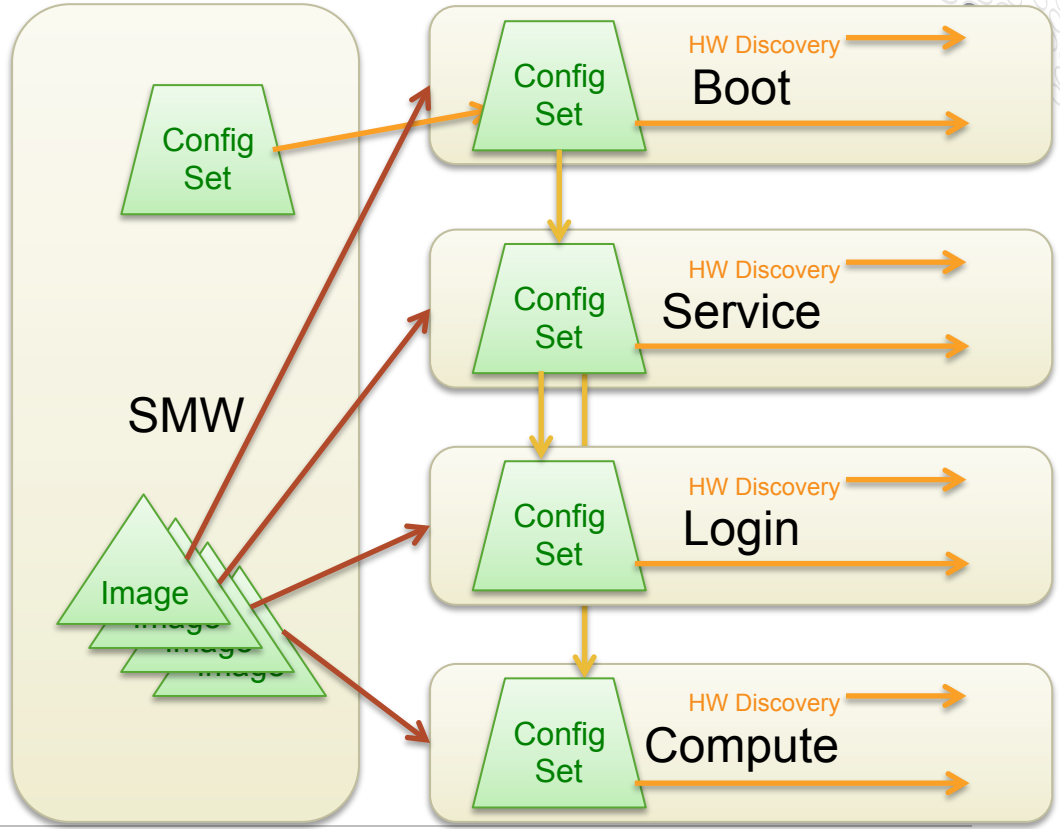
Overview – Boot Software With Configuration

- Nodes boot unconfigured image
- **Early init does:**
  - Basic discovery of node ID, kernel parameters, etc.
  - Imports read-only config set



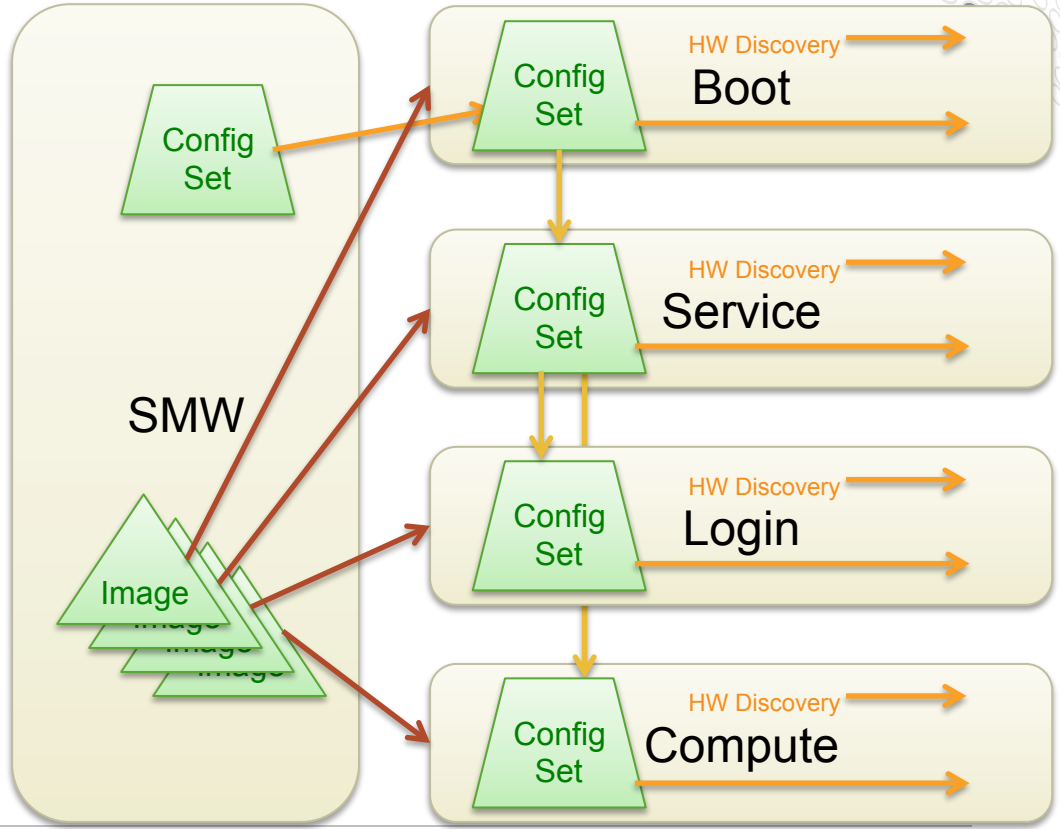
Overview – Boot Software With Configuration

- Nodes boot unconfigured image
- **Early init does:**
  - Basic discovery of node ID, kernel parameters, etc.
  - Imports read-only config set
  - Runs Cray Ansible plays
    - Ansible plays contained in the image
    - Consumes system facts from the discovery
    - Consumes config set data
    - Updates /etc configuration
    - Updates running system



Overview – Boot Software With Configuration

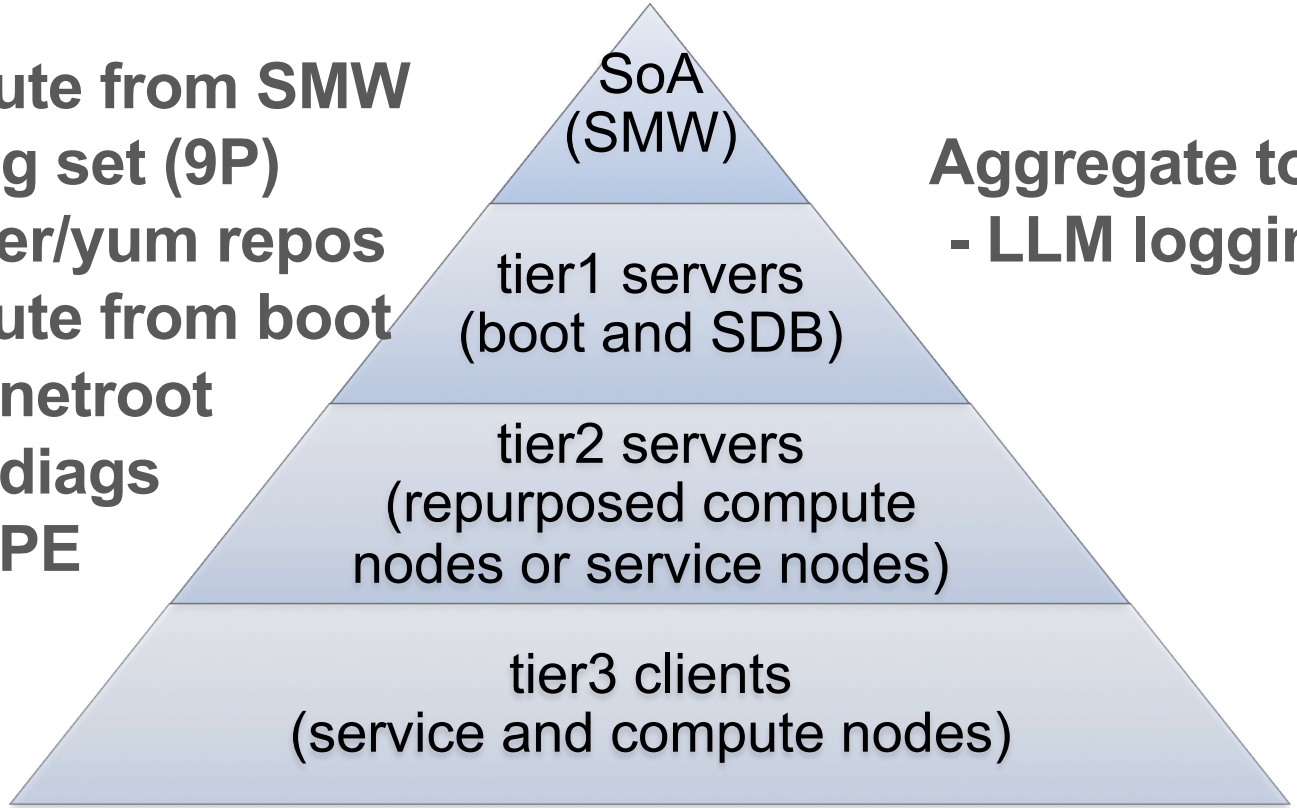
- Nodes boot unconfigured image
- Early init does:
  - Basic discovery of node ID, kernel parameters, etc.
  - Imports read-only config set
  - Runs Cray Ansible plays in init
    - Ansible plays contained in the image
    - Consumes system facts from the discovery
    - Consumes config set data
    - Updates /etc configuration
    - Updates running system
- **Node is booted and configured**
  - Normal init process starts, systemd takes over
  - Some Cray software started via systemd
  - Run Cray Ansible plays in multi-user



Overview – Cray Scalable Services

Distribute from SMW
- config set (9P)
- zypper/yum repos

Distribute from boot
- DVS netroot
- DVS diags
- DVS PE



Aggregate to SMW
- LLM logging

Overview – Cray Scalable Services – tier2

- **tier2 nodes**
  - Recommended ratio of clients to tier2 servers is 400 to 1
  - Repurposed compute nodes (RCN) are best choice
  - Distribute nodes throughout the system for resiliency in event of hardware failure
- **Fine print**
  - At least one server must be provided
  - Minimum of two nodes on different blades for resiliency
  - Never use these nodes as tier2
    - KNL (KNights Landing) compute nodes as RCN
    - Direct Attached Lustre (DAL) servers
    - RSIP (Realm Specific IP) servers
    - login nodes

Overview – Cray Scalable Services – tier2

- **Fine Print (continued)**
  - Small Test Deployment Systems (TDS) may use tier2 nodes which have additional roles
    - Dedicated repurposed compute nodes (RCN) are optimal
    - Dedicated service nodes
    - Nodes with UNIFORM light to moderate load
    - Should not be nodes with slower individual CPU cores, such as KNL nodes
    - Should have relatively homogeneous single core speeds to reduce resource contention disparity during periods of partial availability
  - More tier2 nodes does not always yield additional performance and are subject to diminishing returns

Overview - management of software

- **Management of software with IMPS**
  - File formats
  - Repositories
  - Image recipes
  - Package collections
  - Image root
  - Boot image

File formats

- **YAML (YAML Ain't Markup Language)**
  - Common data types easily mapped to most high-level languages
    - list, associative array, and scalar
  - Suited for humans to view or edit data structures
  - IMPS commands for changing, searching, displaying, validating
    - Ensure files stay in correct format
- **JSON (JavaScript Object Notation)**
  - Open standard format
  - Uses human-readable text
  - Data objects consist of attribute–value pairs
  - Not intended to be human-editable
- **Both formats are "importable" into Python and Ansible**

Repositories

- **All repositories are housed on SMW**
  - /var/opt/cray/repos
- **Some repositories may be shared by SMW and CLE**
  - SLE Server
  - SLE Software Developer's Kit
  - SLE Workstation Extension
  - SLE Module Legacy
  - SLE Module Public Cloud
- **Other repositories unique to SMW or to CLE**
  - SMW software to be installed on SMW
  - CLE software to be installed on SMW
  - CLE software to be installed on CLE SLES nodes
  - CLE software to be installed on CLE DAL nodes
  - CentOS for CLE DAL nodes
- **Empty "update" repositories created for future use**
  - Patches
  - Security updates

Image Recipes

- **Each default image type has an image recipe installed on SMW**
  - Compute, service, login, DAL (Direct Attached Lustre)
  - All Cray image recipes are named to avoid naming conflicts
- **Each image recipe is in a JSON file**
  - Has name and description
  - Includes package collections, packages (rpms), repositories, and other recipes
- **JSON file may contain more than one image recipe**
  - Versioned JSON file(s) for each Cray software release
- **Recipes can include other recipes, package collections, packages, repositories**
- **Everything has a rationale**
  - Description explaining why each package collection, package, repository, or recipe is listed
- **Custom image recipes can be created to serve specific purposes**
- **SMW location:**
  - /etc/opt/cray/imps/image_recipes.d/

Package Collections

- Represent logical groupings of packages (rpms)
- Contain versioned and unversioned package names
- CLE Installed package collections are read only
- Package collections can include packages and other package collections
- SMW location:
  - /etc/opt/cray/imps/package_collections.d/

Image Recipe Example 1

```
{
    "compute_cle_6.0up01_sles_12_x86-64_ari": {
        "description": “Compute image for SLES 12",
        "package_collections": {
            "cle-compute_6.0up01_sles_12_kernel_ari": {
                "rationale": "Provides the needed kernel and kernel drivers."
            },
            "cle_6.0up01_sles_12_compute": {
                "rationale": "This image recipe is a SLES 12 compute node; add all package
collections befitting a Cray SLES 12 compute image."
            }
        },
        "packages": {},
        "recipes": {
            "seed_common_6.0up01_sles_12_x86-64": {
                "rationale": "Start all SLES recipes with common base UID/GIDs"
            }
        },
        "repositories": {
…
        }
    },
}
```

Image Recipe Example 2

```
"repositories": {
            "cle_6.0up01_sles_12_x86-64_ari": {
                "rationale": "A base set of Cray provided packages for SLES
12."
            },
            "cle_6.0up01_sles_12_x86-64_ari_updates": {
                "rationale": "A repository for Cray provided updates to
packages for SLES 12."
            },
            "sles_12_x86-64": {
                "rationale": "The base OS used to build SLES 12 based nodes."
            },
            "sles_12_x86-64_updates": {
                “rationale": "Needed for updating an image recipe for new SLES
12 package updates."
            }
        }
```

Package Collection Example 1

```
{
    "cle_6.0up01_sles_12_base": {
        "description": "Collection of packages for base SLES node capabilities.",
        "package_collections": {},
        "packages": {
            "ansible": {
                "rationale": "Configuration management package needed to configure
nodes."
            },
...
            "zypper": {
                "rationale": "This utility allows install/update of packages
dynamically from within a SLES node."
            }
        }
…
    },
}
```

Package Collection Example 2

```
"cle_6.0up01_sles_12_compute": {
        "description": "Collection of packages for base SLES compute node
capabilities.",
        "package_collections": {
            "cle_6.0up01_sles_12_base": {
                "rationale": "compute nodes need base software"
            },
            "cle_6.0up01_sles_12_compute_cray": {
                "rationale": "Cray packages installed on a compute node"
            },
        },
        "packages": {
            "ksh": {
                "rationale": "Needed for Test group."
            },
            "tcsh": {
                "rationale": "Required by some applications and some customer sites."
            }
```

Extended Image Recipe Support

- **Adding non-rpm content to an image root**
  - Modify JSON image recipe file to
    - Copy content from location on SMW
    - Execute post-build commands and/or scripts
  - Post-build scripts can use several environmental variables
    - IMPS_IMAGE_NAME
    - IMPS_VERSION
    - IMPS_IMAGE_RECIPE_NAME
    - IMPS_POSTBUILD_FILES
  - Post-build commands and scripts always run chrooted
  - Automatic cleanup of files which were copied into the image root

Extended Image Recipe Example

```
"image_recipe_name": {
        ...
        "package_collections": { ... },
        "packages": { ... },
        ”recipes": { ... },
        "postbuild_copy": [
            "/file/on/smw/sample.py",
            ...
            "/dir/on/smw"
        ],
        "postbuild_chroot": [
            "chroot_command1",
            ...
            "chroot_commandN"
        ],
        "repositories": { ... }
    },
```

Extended Image Recipe – seed recipe

```
 "seed_common_6.0up01_sles_12_x86-64": {
        "description": "Common seed image for SLES 12 images",
        "package_collections": {},
        "packages": {
            "rpm": {
                "rationale": "allow IMPS to build RPM database"
            },
            "shadow": {
                "rationale": "minimal package to add users"
            }
        },
        "postbuild_chroot": [
            "/usr/sbin/groupadd --gid 499 mysql",
            "/usr/sbin/groupadd --gid 492 ntp",
            "/usr/sbin/useradd --uid 60 --gid 499 --no-user-group --home-dir /var/lib/mysql --
shell /bin/false --comment added_during_image_create mysql",
            "/usr/sbin/useradd --uid 74 --gid 492 --no-user-group --home-dir /var/lib/ntp --
shell /bin/false --comment added_during_image_create ntp",
        "repositories": { ... }
    },
```

Extended Image Recipe – WLM recipe

```
"wlm_service": {
       "description": "WLM service node image",
       "recipes": {
           "service_cle_6.0up01_sles_12_x86-64_ari": {
             "rationale": "Start from standard service node recipe"
         },
     "packages": {},
     "package_collections": {},
     "postbuild_chroot": [
         "${IMPS_POSTBUILD_FILES}/pbs/pbs_imps_installer.py",
         "${IMPS_POSTBUILD_FILES}/moab_torque/moab_torque_imps_installer.py",
         "rpm -ivh ${IMPS_POSTBUILD_FILES}/moab_torque/*.rpm"
     ],
     "postbuild_copy": [
         "/home/crayadm/wlm_install/pbs_service/pbs",
         "/home/crayadm/wlm_install/p1/moab_torque"
     ],
     "repositories": {}
 },
```

Image Roots and Boot Images

- **Image root**
  - Root filesystem tree on the SMW
  - Created from image recipe
  - All rpm dependencies are resolved from repositories
  - Each image root is related to a single image recipe
  - /var/opt/cray/imps/image_roots
- **Boot image**
  - Created from image root
  - Packaged into a format suitable for booting
  - Each boot image related to a single image root
  - /var/opt/cray/imps/boot_images
- **The resulting images are essentially unconfigured!**

Boot Images

- **Multiple images used to boot CLE**
  - Service node boot image – used by most service nodes
  - Login node boot image – used by login nodes
  - Compute node boot image – used by compute nodes
  - DAL node boot image – used by DAL nodes
  - Custom boot images created by the site
- **NIMS associates a boot image with each node**
- **Image used to boot external login nodes**
  - eLogin node boot image

Overview – Configuration Management Framework

- Config sets
- IMPS Distribution Service (IDS)
- Configuration data
- Configurator
- Boot process configuration
  - cray-ansible and Ansible

Config sets

- All configuration information needed to operate the logical system will be stored in a central repository called a “configuration set” or “config set”
- More than one config set can exist to support partitioned systems or alternate configurations.
- The config sets reside on the SMW and are made available to all nodes in the system read-only
- All config sets are shared throughout the system, but only one is active on a given node at a time.
- Two config sets
  - global config set which covers both the management domain (“SMW” and “CMC”) as well as truly global data
  - CLE config set (for p0, or on a partitioned system for p1, p2, p3, etc.)

Config sets – directory structure on node

- **From the end node's perspective, it's just a directory of config files for the current CLE and global config sets**
  `/etc/opt/cray/config/current`
  `/etc/opt/cray/config/global`
- **/etc/opt/cray/config/current subdirectories**
  ansible, changelog, config, dist, files, worksheets
- **/etc/opt/cray/config/current/config YAML files**
  cray_alps_config.yaml, cray_logging_config.yaml, cray_net_config.yaml, cray_scalable_services_config.yaml, etc.
- **/etc/opt/cray/config/current/worksheets YAML files**
  cray_alps_worksheet.yaml, cray_logging_worksheet.yaml, cray_net_worksheet.yaml, cray_scalable_services_worksheet.yaml, etc.

Config sets – directory structure on SMW

- The config set that is mounted on the nodes lives on the SMW

```
smw:/var/opt/cray/imps/config/sets/p0
```

- Other config sets on SMW

```
smw:/var/opt/cray/imps/config/sets/p0
smw:/var/opt/cray/imps/config/sets/p0-preupgrade-20150324
smw:/var/opt/cray/imps/config/sets/p0-postupgrade-20150324
smw:/var/opt/cray/imps/config/sets/p1
smw:/var/opt/cray/imps/config/sets/p2
smw:/var/opt/cray/imps/config/sets/global
```

- The global config set is also available on the SMW as a link to the /var/opt/cray/imps/config/sets/global

```
smw:/etc/opt/cray/config/global
```

Config set distribution - IDS

- **In order for the config set to be available on all nodes it is distributed by the IMPS Distribution Service (IDS)**
- **IDS leverages the 9P network filesystem and the Linux automounter facility to share the files via Cray scalable services from the SMW, tier1, tier2 to the entire XC system**
  - diod (user-space export daemon) can re-share a 9P mount
  - Read-only allows us to leverage caching
  - 9P support built into modern kernels
  - autofs allows for resiliency and failover
- **Config set caching**
  - cray-cfgset-cache daemon on SMW
    - Responds to kernel inotify events for changes in config sets
    - After change noticed, 4 seconds later a new squashfs file and checksum are generated
    - Cached version of the config set is copied to the nodes and checksum verified
- **The content and use of the config set is independent of the distribution mechanism**

Config set data

- Stored in YAML
- Configuration files include both user data and management metadata
- Configurator will merge and manage configuration data within the config set
- Schema standardized to support configuration tool and provide common look and feel

Configurator

- **The configurator**
  - Completely data driven by files called templates
  - Merge existing configuration data with new templates
- **Configuration templates**
  - Provide useful documentation for the value
  - Provide useful defaults
  - Provide value and syntax checking to be used by configurator
- **Run configurator to collect new data**
  - Will automatically prompt/merge new data elements
  - System administrator's "answers" to questions become new default

Configurator

- **Iterate on the configurator as necessary**
  - Admin can configure a specific service
    - cfgset update -s cray_alps -S unset -l basic p0
    - cfgset update -s cray_alps -S all -l advanced p0
- **Can run configurator in 3 different modes**
  - All modes
    - Merge in new templates
    - Run pre/post-configuration scripts (unless suppressed with --no-scripts)
    - Create configuration worksheets based on current settings
  - auto – Asks questions based on filters (level and state) from command line
  - interactive – View or change any setting in any service
  - prepare – Does not ask any questions

Config Templates - schema

- **Design of template schema drives how information is gathered**
  - YAML format
  - Cray-provided templates start with “cray_”
  - <service_name>_config.yaml
    - cray_logging_config.yaml
- **Template sections**
  - Service
    - Describes the service
    - Initial question about whether the service should be configured further
  - Settings
    - Contains questions to be answered to configure service

Config Templates Example – schema

```
---
cray_service_name:
    ...
    [service meta]
    …
    settings:
        ...
        [service settings]
        ...
...
```

Config Templates - service

- **Fields in template for service**
  - Title - explanation of the service
  - Guidance – description to aid in enable/disabling service
  - Enabled – boolean decision to configure service (or not)
  - Configured – whether this has been configured already
  - Level – required, basic, or advanced
  - Config_after – this should be configured after these other services
  - Template_type – CLE or global

Config Templates Example – service

```
cray_scalable_services:
   enabled: true
   configurator:
      allow_none: true
      comments: []
      configured: true
      config_after: []
      default_value: true
      guidance: "Cray scalable services defines which servers (nodes) are used in
         the scaling of the system. Scalable services must be configured to ensure
         a properly functioning system. \nOnce defined, these servers will be used
         in various services, such as DVS, to provide horizontal scaling, or scaling
         out, of those services. Horizontal scaling allows the system to utilize
         user-defined nodes to work together as a single unit to increase workflow
         output. \nScalable services defines a tree of servers starting with the
         Server of Authority (SoA), followed by tier1 and tier2 servers as represented
         below.\n                    tier2\n          tier1    --\nSoA
         \ --              tier2\n          tier1    --\n"
      level: required
      template_type: cle
      title: Cray Scalable Services
```

Config Templates - settings

- **Fields in template for settings**
  - Title - explanation of the class
  - Guidance – description to aid in setting the value(s)
  - Members – values of the class
  - Regex – regular expression to validate input
  - Configured - whether this has been configured already
  - Level – required, basic, or advanced
  - Argspec – one or more values to be configured
  - Data - one or more values which have been configured

Config Templates Example – settings

```
cray_scalable_services:
  settings:
    scalable_service:
      data:
        tier1:
        - c0-0c0s0n1
      configurator:
        argspec:
          tier1:
            allow_none: false
            configured: true
            default_value: []
            guidance: 'A list of tier1 server cnames. \nThe tier1
              servers must have a direct network connection to
              the Server of Authority (SoA). The SoA is typically
              the SMW. Any node that is directly connected to
              the SoA can function as a tier1 server.\On Cray
              CLE systems, the boot node must be specified as
              a tier1 server.  The SDB should also be specified
              assuming it is properly configured to reach the
              SOA.\nAdding additional tier1
               servers provide enhanced resiliency.\nThere
               must be at least one tier1 server listed.'
            level: required
            multival_key: false
            purge: false
            regex: ^c(\d+)-(\d+)c([0-2])s(\d[0-5]?)n([0-3])$
            title: Tier1 servers
            type: list
      scope_type: class
      comments: []
      guidance: null
      purge: false
```

Config Templates – settings multival

- **Users prompted for each key**
  - then data which applies to it
- **Example**
  - boot_node_ethernet
    - Key1
      - Values
    - Key2
      - Values

Config Templates Example – settings multival

```
[ ... ]
settings:
   some_node_ethernet:
   [ ... ]
      data:
      - key: eth0
        netmask: 255.255.255.0
        ipaddress: 123.45.67.89
      - key: eth1
        netmask: 255.255.240.0
        ipaddress: 192.168.0.1
      configurator:
         scope_type: multival
         argspec:
            interface:
               multival_key: true
               type: string
               level: basic
               default_value: "eth0"
               title: Ethernet Interface
               guidance: Enter the ethernet interface name like "eth0".
               [ ... ]
            netmask:
               type: string
               level: basic
               default_value: "255.255.255.0"
               title: Netmask
               guidance: Enter the netmask.
               [ ... ]
            ipaddress:
               type: string
               level: basic
               default_value: "192.168.0.1"
               title: IP Address
               guidance: Enter the ethernet IP address.
               [ ... ]
```

Config Templates – cray_node_groups

- **Node groups define logical, non-exclusive groupings of CLE nodes**
  - Enables CLE nodes to be logically grouped and referenced by group name within the configuration data of CLE services
    - Cray Simple Sync (UP01)
    - Other Cray config templates (After UP01)
  - Provides a single location in the CLE configuration data where nodes can be managed at the system administrator's discretion
  - Group name can be used by Ansible plays
    - Site created Ansible plays
    - Cray Simple Sync Ansible play (UP01)
    - Other Cray Ansible plays (After UP01)

Config Templates – cray_node_groups

```
cray_node_groups:
    configurator:
        template_type: [ 'cle' ]
        level: required
        default_value: true
        ...
    settings:
        groups:
            data:
            - key: example_group_1
              members:
              - c0-0c0s8n0
              - c0-0c0s8n1
            - key: example_group_2
              members:
              - c0-0c0s8n1
              - c0-0c0s8n2
              - c0-0c0s8n3
    configurator:
    scope: multival
    argspec:
        group_name:
            multival_key: true
            type: string
            ...
        members:
            title: Group Members
            type: list
            default_value: []
            ...
```

Configurator - cfgset

- **Updating a config set – actions by the configurator**
  - Clone the config set as a backup
  - Run pre-configuration scripts
  - Validate templates and configuration data
    - YAML syntax validation check
    - Schema validation check
    - Merge templates
  - Prompt for all information to be configured
  - Write changlog entry
    /var/opt/cray/imps/config/sets/p0/changelog/changelog_2015-09-29T12:00:37.yaml
  - Run post-configuration scripts
  - Remove backup config set

Overview – Boot process configuration

- **All Ansible plays run ON the system at boot time**
- **Ansible "pull" mode**
  - Configuration happens locally on the node instead of being initiated from some central management node.
- **"self configuring model”**
  - **cray-ansible** finds all Ansible plays installed and executes them
  - Ansible plays are packaged with their application software
    - In other words, ALPS plays get packaged with the ALPS software

Overview – Augmenting CMF

- **Cray's CMF allows additional configuration tasks**
  - Add site-specific config temples with site data settings
  - Add site-specific tasks in concert with Cray-provided Ansible boot-time execution acting on config set with Cray data and site data
- **Extending cray-ansible with site Ansible plays**
  - Start/stop services
  - Change crontab entries
  - Modify files
  - Copy files

Ansible – terms

- **Playbook**
  - One or more plays
- **Play**
  - Maps groups of hosts to tasks
- **Task**
  - Sequence of actions performed against group of hosts that match a pattern in the play
- **Modules**
  - Large Ansible library of common code
    - Manage basic system resources
    - Send notifications
- **Roles**
  - Abstraction for naming a group of things that perform same function
- **Separate code from data**
  - Jinja2 templates (code)
  - Variables (data)
- **Jinja2**
  - Python-based template engine
  - Templates have placeholders for parameter values which can be replaced with variables
- **Data**
  - Facts
    - Automatically available
    - Discovered at run time
  - Variables
    - User-defined

Ansible

- **Framework for developers to write Ansible plays**
- **Ansible plays**
  - Will configure the software
  - Can be either in the image or in the config set
    - **cray-ansible** will find plays in both locations and include them automatically
    - Config set is the proper place for site playbooks to retain separation between image and configuration
- **Integrating site Ansible plays into config set**
  - smw:/var/opt/cray/imps/config/sets/<config_set>/ansible/myplay.yaml
  - smw:/var/opt/cray/imps/config/sets/p0/dist/<other files>
- **Play runs automatically with all Cray provided plays**
  - Simple mechanism to influence play ordering
  - For example, to amend what ALPS configuration is done
    - ensure site play runs after the ALPS play

Ansible – local system facts

- Boot process provides access to a group of Ansible facts that describe the running node
- Ansible provides a large amount of system information by default, but additional facts are useful when deciding when and where to perform configuration tasks
  - http://docs.ansible.com/ansible/playbooks_variables.html#local-facts-facts-d
- Cray system facts
  - /etc/ansible/facts.d/cray_system.fact

Ansible – Best Practices for Playbooks/Roles

- **Ansible expects that all tasks are idempotent**
  - (action performed only once, even if play is run more than once)
  - Care should be taken to ensure that tasks prescribe the desired state of the running system, making changes only when necessary
  - See http://docs.ansible.com/ansible/glossary.html#resource-model
- **When modifying files on a running system**
  - Keep in mind that other services may access the file
  - Take the appropriate measures to ensure the modifications do not interfere with other operations.
  - Leave a breadcrumb that the file is updated by an automated process
    - The “insertbefore” or “insertafter” options in the Ansible “lineinfile” module are well-suited to help with this.
- **Look at the directory of Ansible modules, if you find that you are trying to do something that is difficult to achieve in a few simple steps**
  - It is likely that Ansible already has a module that provides the functionality
  - See http://docs.ansible.com/ansible/modules_by_category.html.
- **The Ansible “ini_file” module was specially created for modifying INI-style configuration files**
  - Use this instead of “lineinfile” when applicable.

Ansible – example.yaml

boot# **grep example /etc/ansible/site.yaml**
- include: /etc/opt/cray/config/current/ansible/example.yaml
boot# **cat /etc/opt/cray/config/current/ansible/example.yaml**

```
- hosts: localhost

  vars:
    run_after:
    - common

  roles:
    - example
```

Ansible – ansible-playbook 1

boot# **ansible-playbook –v /etc/opt/cray/config/current/ansible/example.yaml**
PLAY [localhost]
****

GATHERING FACTS
****

ok: [localhost]
TASK: [example | task template, set variables]
****

ok: [localhost] => {"ansible_facts": {"myservice_bar": "9999",
"myservice_baz": "turnip", "myservice_foo": "true"}}

TASK: [example | task template, create myservice.conf config]
****

ok: [localhost] => {"changed": false, "gid": 0, "group": "root", "mode":
"0644", "owner": "root", "path": "/etc/myservice.conf", "size": 199,
"state": "file", "uid": 0}

Ansible – ansible-playbook 2

```
TASK: [example | task copy, check for file]
***********************************
ok: [localhost] => {"changed": false, "stat": {"atime": 1429911953.80648,
"ctime": 1429911256.7013516, "dev": 3, "exists": true, "gid": 0, "inode":
110300, "isblk": false, "ischr": false, "isdir": false, "isfifo": false,
"isgid": false, "islnk": false, "isreg": true, "issock": false, "isuid":
false, "md5": "9d4ec22f000e91f8cc39dcfd6864d46c", "mode": "0644", "mtime":
1429911256.7013516, "nlink": 1, "pw_name": "root", "rgrp": true, "roth":
true, "rusr": true, "size": 198, "uid": 0, "wgrp": false, "woth": false,
"wusr": true, "xgrp": false, "xoth": false, "xusr": false}}

TASK: [example | task copy, make copy of myservice.conf]
**********************
skipping: [localhost]
```

Ansible – ansible-playbook 3

```
 TASK: [example | task lineinfile, customize existing config]
******************
ok: [localhost] => {"backup": "", "changed": false, "msg": ""}

TASK: [example | task service, turn on rsyncd]
*********************************
ok: [localhost] => {"changed": false, "name": "rsyncd", "state": "started"}

TASK: [example | task shell, do something]
***********************************
skipping: [localhost]

PLAY RECAP
********************************************************************
localhost                  : ok=6    changed=0    unreachable=0
failed=0
```

Ansible – example tasks

boot# **cd /etc/opt/cray/config/current/ansible/roles/example/tasks**
boot# **ls**
copy.yaml lineinfile.yaml main.yaml service.yaml shell.yaml template.yaml
boot# **cat main.yaml**

```
---
- name: task main, template example
  include: template.yaml
- name: task main, make copy of config file
  include: copy.yaml
- name: task main, customize for second instance
  include: lineinfile.yaml
- name: task main, turn on rsyncd
  include: service.yaml
- name: task main, run a shell script, but only once
  include: shell.yaml
```

Ansible – example tasks

boot# **cat template.yaml**

```
---

- name: task template, set variables
  set_fact:
    myservice_foo=true
    myservice_bar=9999
    myservice_baz=turnip

- name: task template, create myservice.conf
config
  template:
    src=myservice.conf.j2
    dest=/etc/myservice.conf
```

boot# **cat copy.yaml**

```
---

- name: task copy, check for file
  stat:
    path=/etc/myservice2.conf
  register: result

- name: task copy, make copy of myservice.conf
  synchronize:
    src=/etc/myservice.conf
    dest=/etc/myservice2.conf
  when: not result.stat.exists
```

Ansible – example tasks

boot# **cat lineinfile.yaml**

```
---
- name: task lineinfile, customize existing config
  lineinfile:
    dest=/etc/myservice2.conf
    regexp="^baz="
    line="baz=onion"
    backup=yes
```

boot# **cat service.yaml**

```
---
- name: task service, turn on rsyncd
  service:
    name=rsyncd
    state=started
  when: not ansible_local.cray_system.in_init
```

boot# **cat shell.yaml**

```
---
- name: task shell, do something
  shell:  "echo hello > /tmp/foo && touch /var/run/
something"
  args:
    creates: /var/run/something
```

Ansible – Cray system facts

- **Cray system facts are in /etc/ansible/facts.d/cray_system.fact**
- **in_init**
  - True if current Ansible run is prior to the Linux systemd/sysvinit startup phase, false if after
- **roles**
  - A list of node roles assigned to the node. Possible values are: “boot”, “sdb”, and “smw”
- **platform**
  - One of “service” or “compute” (undefined for SMW)
- **is_cray_blade**
  - True if the node is a Cray-proprietary blade, otherwise false (for example: SMW, CMC, and eLogin nodes)
- **uses_systemd**
  - True if the base distribution uses systemd or not

Ansible – Cray system facts

- **cname**
  - Component name of the node (c0-0c0s0n1)
- **nid**
  - Node ID of the current node (example: for nid00045, nid = 45)
- **sessionid**
  - XC boot session identifier
- **hostid**
  - Hostname for non-Cray proprietary blades (for example, SMW), cname for Cray blades
- **nims_group**
  - From the kernel parameter in /proc/cmdline which was assigned for this node on SMW with cnode command
- **node_groups**
  - A list of node_groups which have this node as a member

Ansible – Using Cray system facts

```
- hosts: localhost
  vars:          # Cray-provided node “facts” + config set data
    nid:         ansible_local.cray_system.nid
    is_nid7:     ansible_local.cray_system.nid == "7"
    is_login:    ansible_local.cray_system.hostid in
                 cray_login.settings.login_nodes.data.members
    is_sdb:      “sdb” in ansible_local.cray_system.roles
    in_init:     ansible_local.cray_system.in_init
    is_svc:      ansible_local.cray_system.platform == "service“
    is_nims_blue: “blue” in ansible_local.cray_system.nims_group
    is_node_group_red: “red” in ansible_local.cray_system.node_groups

    run_after:          # Call out a runtime dependency
    - simple_sync
```

Ansible – Using Cray system facts

```
tasks:
# Option:Use Ansible modules to do individual steps (example: start a service)
- name: start awesomed service on nid0007, sdb, login nodes,
        blue nims group, red node_group
  service: name=awesomed state=started args="-f /path/to/awesome_config.conf"
  when:
    (is_nid7 or is_login or is_sdb or is_nims_blue or is_node_group_red) and
    not in_init

# Option: Let me just do everything in my script
- name: run my script on all service nodes
  shell: /etc/opt/cray/config/current/dist/site_script.sh >> somelog.txt
  when:
    is_svc and not in_init
```

Ansible – Using Cray system facts

```
- name: task nfshomedir, make mount point
  file:
    path=/home/users
    state=directory
    mode=755
    when: ansible_local.cray_system.hostid in cray_login.settings.login_nodes.data.members

- name: task nfshomedir, add mount to fstab
  lineinfile:
    dest=/etc/fstab
    regexp="^172.30.79.66:/home/users"
    line="172.30.79.66:/home/users /home/users nfs nfsvers=3,noacl 0 0"
    backup=yes
    when: ansible_local.cray_system.hostid in cray_login.settings.login_nodes.data.members
```

Ansible – Using Cray system facts

```
node# cat someplay.yaml
---
# Stop a service on some nodes
- name: don't run cron except on login nodes
  hosts: localhost

  tasks:
  - name: control cron
    service:
      name: cron
      state: stopped
    when:
      ansible_local.cray_system.hostid not in cray_login.settings.login_nodes.data.members
      and not ansible_local.cray_system.in_init
```

Ansible – Using Cray system facts

```
- hosts: localhost
  vars:
    nid: [ansible_local.cray_system.nid]
    run_before:
    - ssh
  tasks:
  - name: find nid match in external hosts file, capture IP address
    shell: “grep {{nid}} /etc/mysitelocal/hosts-external | head -1 | awk '{ print $4 }’”
    register: external_ipaddr
  - name: add ListenAddress/external options to file
    lineinfile:
      dest: /etc/sshd/sshd_config
      regexp="^SSHD_OPTS="
      line="SSHD_OPTS=‘-u0 -f /etc/ssh/sshd_config.external -o ListenAddress={{external_ipaddr}}’"
      backup: yes
    when:
      external_ipaddr is defined
  - debug: “Did not find external interface to start SSHD on...”
    when: external_ipaddr is not defined
```

Ansible – play ordering

```
boot# cat /etc/ansible/site.yaml
---
#This file was autogenerated at 2016-04-21T16:30:38+06:00
- include: /etc/ansible/set_hostname.yaml
- include: /etc/ansible/simple_sync.yaml
- include: /etc/ansible/early.yaml
- include: /etc/ansible/dws-dvs.yaml
- include: /etc/ansible/local_users.yaml
- include: /etc/ansible/firewall_init.yaml
- include: /etc/ansible/networking.yaml
- include: /etc/ansible/ssh.yaml
- include: /etc/ansible/lnet.yaml
- include: /etc/ansible/common.yaml
- include: /etc/ansible/persistent_data.yaml
- include: /etc/ansible/ipforward_routes.yaml
- include: /etc/ansible/llm.yaml
- include: /etc/ansible/sm_inv.yaml
- include: /etc/ansible/rsip.yaml
- include: /etc/ansible/compute_node.yaml
- include: /etc/ansible/liveupdates.yaml
- include: /etc/ansible/db.yaml
- include: /etc/ansible/alps.yaml
- include: /etc/ansible/munge.yaml
- include: /etc/ansible/drc.yaml
- include: /etc/ansible/capmc.yaml
- include: /etc/ansible/wlm_detect.yaml
- include: /etc/ansible/service_node.yaml
- include: /etc/ansible/login_node.yaml
- include: /etc/ansible/dvs.yaml
- include: /etc/ansible/cle_lustre_client.yaml
- include: /etc/ansible/dws.yaml
# Play's play types (netroot_setup) are excluded
#- include: /etc/ansible/netroot_setup.yaml
- include: /etc/ansible/netroot_cop.yaml
- include: /etc/ansible/multipath.yaml
- include: /etc/ansible/rca.yaml
- include: /etc/ansible/node_health.yaml
- include: /etc/ansible/rur.yaml
- include: /etc/ansible/ccm.yaml
- include: /etc/ansible/baseopts.yaml
- include: /etc/ansible/cray_image_binding.yaml
- include: /etc/ansible/sysconfig.yaml
- include: /etc/ansible/sysenv.yaml
- include: /etc/ansible/wlm_trans.yaml
- include: /etc/ansible/xtremoted.yaml
- include: /etc/ansible/cle_node.yaml
- include: /etc/ansible/freemem.yaml
- include: /etc/ansible/cle_motd.yaml
- include: /etc/ansible/allow_users.yaml
```

Ansible – play ordering log

boot# **more /var/opt/cray/log/ansible/sitelog-booted**

```
2016-04-21 16:30:35,494 Starting Ansible configuration start-cle phase
2016-04-21 16:30:36,135 Ignoring '/etc/opt/cray/config/current/config/cray_ipforward_config.yaml': Global inherit requested
2016-04-21 16:30:36,484 Ignoring '/etc/opt/cray/config/current/config/cray_logging_config.yaml': Global inherit requested
2016-04-21 16:30:36,760 Ignoring '/etc/opt/cray/config/current/config/cray_multipath_config.yaml': Global inherit requested
2016-04-21 16:30:37,816 Ignoring '/etc/opt/cray/config/current/config/cray_time_config.yaml': Global inherit requested
2016-04-21 16:30:38,006 Ignoring lower precedence file: /etc/opt/cray/config/global/config/cray_firewall_config.yaml
2016-04-21 16:30:38,530 Writing  updated gathering to /etc/ansible/ansible.cfg
2016-04-21 16:30:38,531 Writing  updated library to /etc/ansible/ansible.cfg
2016-04-21 16:30:38,532 Writing  updated log_path to /etc/ansible/ansible.cfg
2016-04-21 16:30:46,739
2016-04-21 16:30:46,739 PLAY [local set_hostname playbook] ********************************************
2016-04-21 16:30:46,739
2016-04-21 16:30:46,739 GATHERING FACTS ***************************************************************
2016-04-21 16:30:46,739
2016-04-21 16:30:46,739 ok: [localhost]
2016-04-21 16:30:46,739
2016-04-21 16:30:46,739 TASK: [set_hostname | task main, define nid format hostname for Cray blades else leave null] ***
2016-04-21 16:30:46,739
2016-04-21 16:30:46,739 ok: [localhost] => {"ansible_facts": {"host": "nid00001"}}
2016-04-21 16:30:46,739
```

Ansible set_hostname play

boot# **cat /etc/ansible/set_hostname.yaml**

```
---

# Cray top level configuration management play set_hostname
# Copyright 2016 Cray Inc. All Rights Reserved.

- name: local set_hostname playbook
  hosts: localhost

  vars:
    run_early: True

    cray_play_type:
    - cle
    - netroot_setup

  roles:
    - role: set_hostname
      when: cray_net.enabled and cray_net.settings.service.data.cray_managed
```

Ansible set_hostname role

boot# **cat /etc/ansible/roles/set_hostname/tasks/main.yaml**

```
# Cray role set_hostname
# Copyright 2014-2016 Cray Inc. All Rights Reserved.
- name: task main, define nid format hostname for Cray blades else leave null
  set_fact:
    host={{ 'nid%05d' |format(ansible_local.cray_system.nid|int)
    if ansible_local.cray_system.is_cray_blade else '' }}

- name: task main, redefine hostname if found in networking config
  set_fact:
    host={{item.hostname}}
  with_items:
    cray_global_net.settings.hosts.data|
union(cray_net.settings.hosts.data)
  when: ansible_local.cray_system.hostid == item.hostid
    and item.hostname != ""

# If we've determined a hostname, write it out
- name: task main, update hostname file and trigger hostname command
  template:
    src=hostname.j2
    dest=/etc/hostname
    backup=yes
  when: host != ""
  notify: sethostname
```

Ansible ansible_cfg_search

- Search Ansible plays in config set and image root to see which plays use which configuration variables

  ansible_cfg_search [-h] [-p PLAYBOOK] [-s CONFIG_SETTING]
  [-e LOOKUP_EXPRESSION] [-q]
  config_set image

- **List the Ansible playbooks in config set and image root**

  smw# **ansible_cfg_search -q p0 custom_compute_cle**
  /var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/allow_users.yaml
  /var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/alps.yaml
  …
  /var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/set_hostname.yaml
  ...
  /var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/sysenv.yaml
  /var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/wlm_detect.yaml

Ansible ansible_cfg_search

- **Search one Ansible playbook for plays and variables in config set and image root**

```
smw# ansible_cfg_search -p set_hostname.yaml p0 custom_compute_cle
/var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/set_hostname.yaml:
  - /var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/roles/set_hostname/tasks/main.yaml:
    - /var/opt/cray/imps/config/sets/p0/config/cray_netroot_preload_config.yaml:
      - cray_net.settings.hosts.data
    - /var/opt/cray/imps/config/sets/global/config/cray_network_boot_packages_config.yaml:
      - cray_net.settings.hosts.data
…
  - /var/opt/cray/imps/image_roots/custom_compute_cle/etc/ansible/set_hostname.yaml:
    - /var/opt/cray/imps/config/sets/global/config/cray_network_boot_packages_config.yaml:
      - cray_net.enabled
      - cray_net.settings.service.data.cray_managed
...
```

Ansible references

- **Ansible web site:**
  - http://www.ansible.com/configuration-management
- **Wikipedia:**
  - http://en.wikipedia.org/wiki/Ansible_%28software%29
- **Source:**
  - https://github.com/ansible/ansible
- **Documentation:**
  - http://docs.ansible.com/
- **Books (many more are available)**
  - Ansible: Up & Running
  - Ansible for DevOps
  - Ansible Playbook Essentials
  - Mastering Ansible

Boot process - boot sequence

- **Ansible plays happen in two phases during boot**
  - Execution of Ansible in initrd /init
  - Normal Linux multi-user startup with systemd
  - Another execution of Ansible at the end of multiuser
- **Ansible**
  - If you ask it to perform an action, it will generally not perform any action the second time if the first succeeds.
  - The exception is for actions that MUST ONLY be performed at a certain time
    - For example, if your play starts a process you only want to have that happen at multiuser mode

Boot process - execution in initrd

- **The first execution of Ansible in_init**
  - Create a config file for a service before the service is started in multi-user
  - Prepare the storage prior to the boot of the system
    - Create LVM volume groups, volumes and filesystems
    - When the system starts, these filesystems will be mounted and ready for use.
- **An Ansible play running in_init should not execute processes**
  - These should only be launched in multiuser
  - When enabling systemd processes in_init
    - Manage the default links instead of using a service enable/disable because systemd isn't running

Boot process – Linux startup

- **Linux startup**
  - Because we configured many things in_init, when the standard Linux startup occurs utilizing systemd, system services should start up properly configured
  - Filesystems will be mounted at this time

Boot process – second Ansible run

- **Ansible in multi-user**
  - Many of the configuration files were modified during in_init those actions will be no-ops
  - Ansible plays can specify dependencies on other plays to ensure they are performed first
    - For example, the ALPS play can depend on the database play such that we know the database is up by the time it gets to ALPS
- **Your play should do whatever it takes to get your service into operation**
  - For example, some plays like the database play have to first ensure the database is up, but then also load the schemas if needed, and load data into the database, all in the correct order

How A Config Set Becomes Applied Config

1. **Configurator**
   - Creates the config set, gathers data
2. **NIMS**
   - Maps image and config set to nodes
3. **IDS: Distributes config set to node during early boot**
   - Mounts the config set on the node for consumption
4. **cray-ansible takes over**

How A Config Set Becomes Applied Config

- **cray-ansible (wraps Ansible runs)**
  - Assembles/orders Ansible plays
    - From the image
    - Provided by the site in the config set
  - Gathers “facts” (system data: OS type, runlevel, hostname)
  - Reads in config set data
  - Is executed in two phases to apply configuration data via plays

Plays are responsible for determining the configuration to apply during each phase

Configuring MySQL Example

- **cray-ansible runs Ansible twice (sandwiches Linux startup)**
  - *Phase 1* - during Linux init
    - "pre-configures" Linux
    - Shut off Linux default services we don't want
    - Setup HSN network interfaces
  - …Linux/systemd takes over
  - *Phase 2* – after Linux is booted multi-user
    - configures software, starts services

Node Image at Boot time

kernel loaded	
/init bootstrapping	
cray-ansible in /init	Prepare mysql volume
	Prep mysql filesystem
	Configure mysql
Linux boot (systemd)	Mount disk
	Start mysql
cray-ansible in booted multi-user	Ensure mysql running
	Load mysql schema
	Load mysql data
	Start ALPS

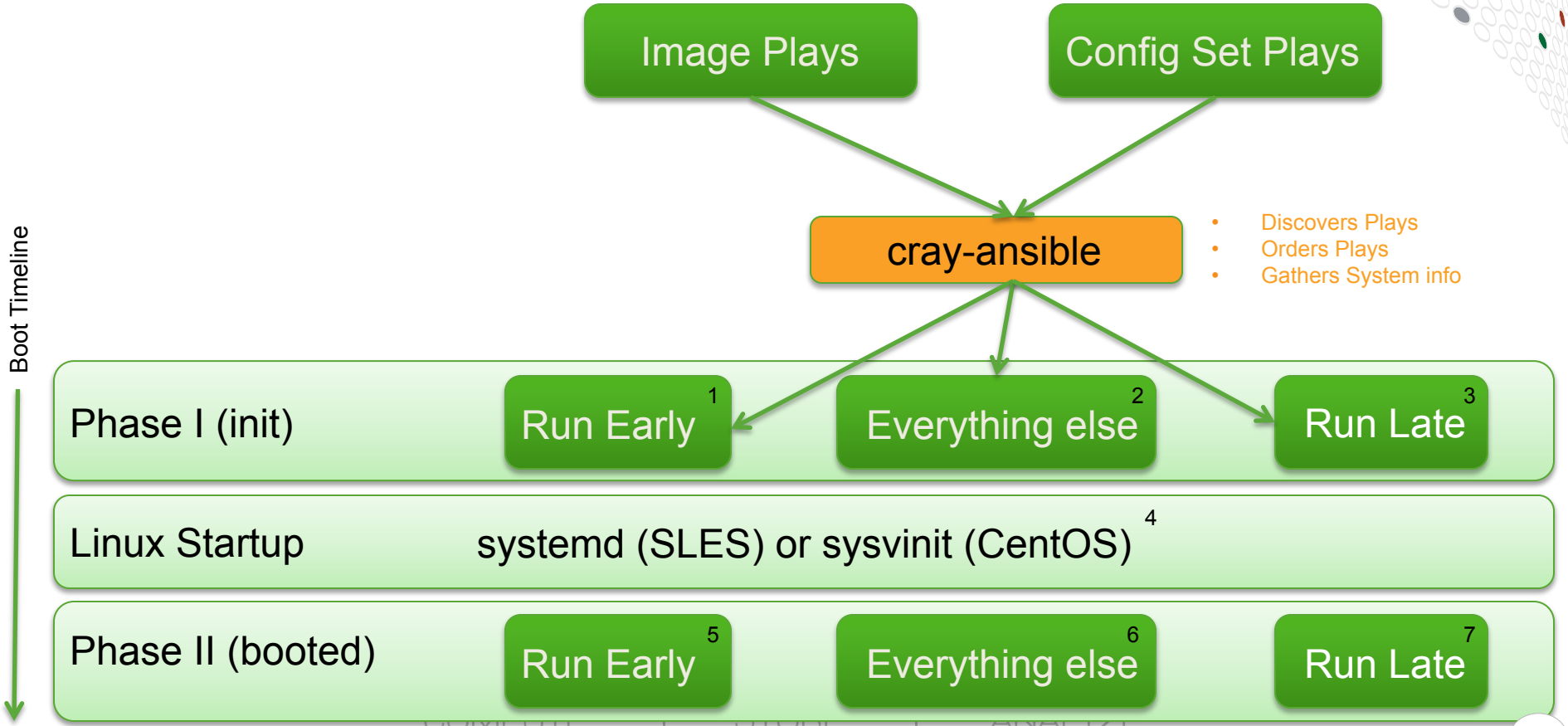
Play Ordering During Ansible Runs

- **cray-ansible determines order via play directives**

1. run_early – a container of plays, first group to be run
2. run_late – a container of plays, last group to be run
3. run_after – specifies dependencies
4. run_before – specifies dependencies (reserved for customers)

- Everything not specified is run between “early” and “late” containers
- *Dependencies take precedence over run_early/run_late*
- cray-ansible sorts all plays (in image and those in config set)
  - Dependency-ordered list of plays stored in /etc/ansible/site.yaml

Play Ordering During Ansible Runs



tmpfs versus netroot

- **tmpfs**
  - Entire root filesystem resides in memory resident (tmpfs) filesystem
  - Root filesystem access very efficient
  - Filesystem content is restricted to reduce memory footprint
    - Used in cases where minimal command set and libraries are required
- **netroot**
  - Root filesystem mounted from network source
  - Reduces memory footprint
  - Provides robust set of commands and libraries
  - Increases root filesystem access latency
  - Increases node boot time
- **Service nodes always use tmpfs (except login nodes)**

tmpfs comparison with netroot

	Memory consumption	Number of rpms
Service node tmpfs	1800MB	600-620
Login node tmpfs	3200MB	950-1000
Login node netroot	115MB	2450-2500
Compute tmpfs	1300MB	650-700
Compute netroot	115MB	2350-2400

Cray image recipes for tmpfs and netroot can be cloned and extended with rpms needed or reduced to remove rpms not needed

Ansible on node using netroot

- **cray-ansible runs three times**
  - *Special netroot setup – during Linux init*
    - cray-ansible runs only play_type netroot_setup
      - Mounts netroot image
  - *Phase 1* - during Linux init
    - Uses full set of Ansible plays from the netroot image instead of tmpfs
    - cray-ansible runs only play_type cle
  - Linux/systemd takes over
  - *Phase 2* – after Linux is booted multi-user
    - cray-ansible runs only play_type cle
    - configures software, starts services

Node Image at Boot time

kernel loaded	
/init bootstrapping	
cray-ansible only netroot_setup plays	simple sync v2
	LNet play
	DVS play
	netroot mount play
cray-ansible in /init	set_hostname
	simple sync v2
	Other plays
Linux boot (systemd)	
cray-ansible in booted multi-user	set_hostname
	simple sync v2
	Other plays

Overview of software installation

- **Software installation**
  - SLES12
  - Storage layout
  - Fresh install
  - Staged upgrades reduce downtime
  - Reverting to a previous software version
  - PE software

SLES 12 - Differences

- **Default filesystem type changed from ext3 to**
  - btrfs for the operating system
  - xfs for data filesystems
- **Bootloader has changed from grub1 to grub2**
  - This affects the SMW, but not CLE nodes
- **sysvinit (/etc/init.d) replaced by systemd**
- **wicked network configuration**
  - A modern, dynamic network configuration infrastructure
- **MariaDB open source database replaces the MySQL database system, but is still called mysql**
- **https://www.suse.com/releasenotes/x86_64/SUSE-SLES/12**

SLES 12 btrfs (B-trees filesystem)

- Copy-On-Write (COW) logging-style filesystem
  - Writes block changes to new location
  - Links in the change
  - Until last write, the new changes are not committed
- Writable snapshots that allow you to easily roll back your system
- Can define subvolumes which will not be part of snapshot
- Data and metadata checksums improve the reliability of the filesystem
- Integrated with LVM (Logical Volume Manager) storage objects
- Multiple device support allows one to grow or shrink the filesystem
- https://www.suse.com/documentation/sles11/stor_admin/data/sec_filesystems_major.html

SLES 12 - systemd

- **Suite of basic building blocks for a Linux system**
  - Provides a system and service manager that runs as PID 1 and starts the rest of the system.
- **systemd**
  - Provides aggressive parallelization capabilities
  - Uses socket and D-Bus activation for starting services
  - Offers on-demand starting of daemons
  - Keeps track of processes using Linux control groups
  - Supports snapshotting and restoring of the system state
  - Maintains mount and automount points
  - Implements an elaborate transactional dependency-based service control logic
  - Can watch a process and restart if it fails

SLES 12 - systemd

- **systemd supports SysV and LSB init scripts and works as a replacement for sysvinit**
  - Best to replace SysV init scripts with systemd unit files
  - Get status on a service

    smw# **systemctl status rsms.service**
    - Shows output from process to verify it started or help debug why it didn't
  - Restart a service

    smw# **systemctl restart rsms.service**
- **Other parts include:**
  - a logging daemon
  - utilities to control basic system configuration like the hostname, date, locale
  - maintain a list of logged-in users, system accounts, runtime directories, and settings
  - daemons to manage simple network configuration, network time synchronization, log forwarding, and name resolution

Storage Layout – concepts

- **Storage set**
  - Defines the filesystems, volumes, and volume groups used by a node
  - SMW has storage set for its filesystems on the boot RAID
  - CLE has storage set which groups filesystems used by boot and SDB nodes
  - System with two partitions needs two CLE storage sets
  - System could have CLE test storage set and CLE production storage set
- **cray_bootraid_config YAML**
  - storage_sets– cledefault and smwdefault
    - key, list of LVM volume_groups
  - LVM volume groups (VGs) – SMW, boot, and sdb
    - key, owner of VG, List of Physical Volumes (PVs), list of LVM volume groups
  - LVM volumes
    - key, description, fs_mount-point, fs_size, fs_type, lvm_volume, lvm_volume_group

Storage layout – SMW internal disks

- R815 SMW boot disk is a RAID1 pair of drives (mirrored) for swap, /boot, and /
- R630 SMW boot disk is a RAID5 group of drives for swap, /boot, and /

Filesystem	Type	Description
/boot	ext3	Booting area
swap	swap	Swap
/	btrfs	Root (/) filesystem with btrfs subvolumes
/var/lib/pgsql	ext4	HSS postgresql database

Storage layout – SMW df

- **SMW (R815 example, R630 would show /dev/sdb* instead of /dev/md*)**

Filesystem	1K-blocks	Used	Available	Use%	Mounted on
/dev/md126	118528896	103600408	11934184	90%	/
devtmpfs	8132124	0	8132124	0%	/dev
tmpfs	8173804	80	8173724	1%	/dev/shm
tmpfs	8173804	18948	8154856	1%	/run
tmpfs	8173804	0	8173804	0%	/sys/fs/cgroup
/dev/md126	118528896	103600408	11934184	90%	/media/root-sv
/dev/md126	118528896	103600408	11934184	90%	/var/tmp
/dev/md126	118528896	103600408	11934184	90%	/tmp
/dev/md126	118528896	103600408	11934184	90%	/var/spool
/dev/md126	118528896	103600408	11934184	90%	/var/log
/dev/md126	118528896	103600408	11934184	90%	/etc/grub.d
/dev/md126	118528896	103600408	11934184	90%	/var/crash
/dev/md126	118528896	103600408	11934184	90%	/var/adm/cray
/dev/sde	118528896	10360040	119341840	9%	/var/lib/pgsql
/dev/md126	118528896	103600408	11934184	90%	/var/lib/named
/dev/md127	4003248	89420	3703812	3%	/boot

Storage layout – SMW /etc/fstab

- **SMW /etc/fstab**
  - Devices specified with /dev/disk/by-uuid type of identifiers
  - Notice the subvolumes on the same device with the / (root) filesystem

```
UUID=1b132132-ad28-4822-a4cd-35e635372930 swap             swap       defaults               0 0
UUID=e39ae3b7-56c1-42a1-b386-a6e6c1ec8e13 /               btrfs      defaults               0 0
UUID=e39ae3b7-56c1-42a1-b386-a6e6c1ec8e13 /etc/grub.d     btrfs      subvol=@/etc/grub.d    0 0
UUID=e39ae3b7-56c1-42a1-b386-a6e6c1ec8e13 /tmp            btrfs      subvol=@/tmp           0 0
UUID=e39ae3b7-56c1-42a1-b386-a6e6c1ec8e13 /var/adm/cray   btrfs      subvol=@/var/adm/cray  0 0
UUID=e39ae3b7-56c1-42a1-b386-a6e6c1ec8e13 /var/crash      btrfs      subvol=@/var/crash     0 0
UUID=e39ae3b7-56c1-42a1-b386-a6e6c1ec8e13 /var/lib/named  btrfs      subvol=@/var/lib/named 0 0
UUID=03e629bd-6856-403d-af71-ba5e68d4b0fa  /var/lib/pgsql btrfs      subvol=@/var/lib/pgsql 0 0
UUID=e39ae3b7-56c1-42a1-b386-a6e6c1ec8e13 /var/log        btrfs      subvol=@/var/log       0 0
UUID=e39ae3b7-56c1-42a1-b386-a6e6c1ec8e13 /var/spool      btrfs      subvol=@/var/spool     0 0
UUID=e39ae3b7-56c1-42a1-b386-a6e6c1ec8e13 /var/tmp        btrfs      subvol=@/var/tmp       0 0
UUID=2e31e655-0983-4676-8200-b76d6aafc403 /boot           ext3       acl,user_xattr         1 2
```

Storage layout – SMW on Boot RAID

- SMW filesystems in LVM Volume Group

Owning node	Filesystem	Type	Description
SMW	/home	xfs	Home directories on SMW
SMW	/var/lib/mysql	btrfs	HSS MySQL database
SMW	/var/opt/cray/disk/1	xfs	Logs, dumps, debug
SMW	/var/opt/cray/repos	btrfs	Software repositories
SMW	/var/opt/cray/imps	btrfs	image roots, boot images, and configuration data

Storage layout – boot node on Boot RAID

- CLE boot node filesystems in LVM Volume Group

Owning node	Filesystem	Type	Description
boot	/cray_home	xfs	Home directory for crayadm
boot	/var/opt/cray/imps	btrfs	image roots for PE, netroot, and diags
boot	/non_volatile	xfs	Persistent data for service nodes (not just /var)

Storage layout – SDB node on Boot RAID

- CLE SDB node filesystems in LVM Volume Group

Owning node	Filesystem	Type	Description
SDB	/var/lib/mysql	xfs	SDB database
SDB	/alps_shared	xfs	ALPS data

Fresh install - preparation

- **Prepare - gather basic configuration information**
  - Nodes with special roles
    - boot, sdb, login, tier1, tier2, DVS, RSIP, LNet, DAL, etc.
  - Network information
    - DNS servers, search domains
    - Networks (other than admin, login, HSN, HSS, SMW failover)
      - address, netmask, broadcast, gateway
    - Host information
      - cname, Ethernet or InfiniBand interfaces, hostname, hostname aliases, IP address, MTU, static or DHCP, etc.
  - LNet router and Lustre client information if using external Lustre server
- **Plan utilization of storage on boot RAID**
  - Create storage sets configuration

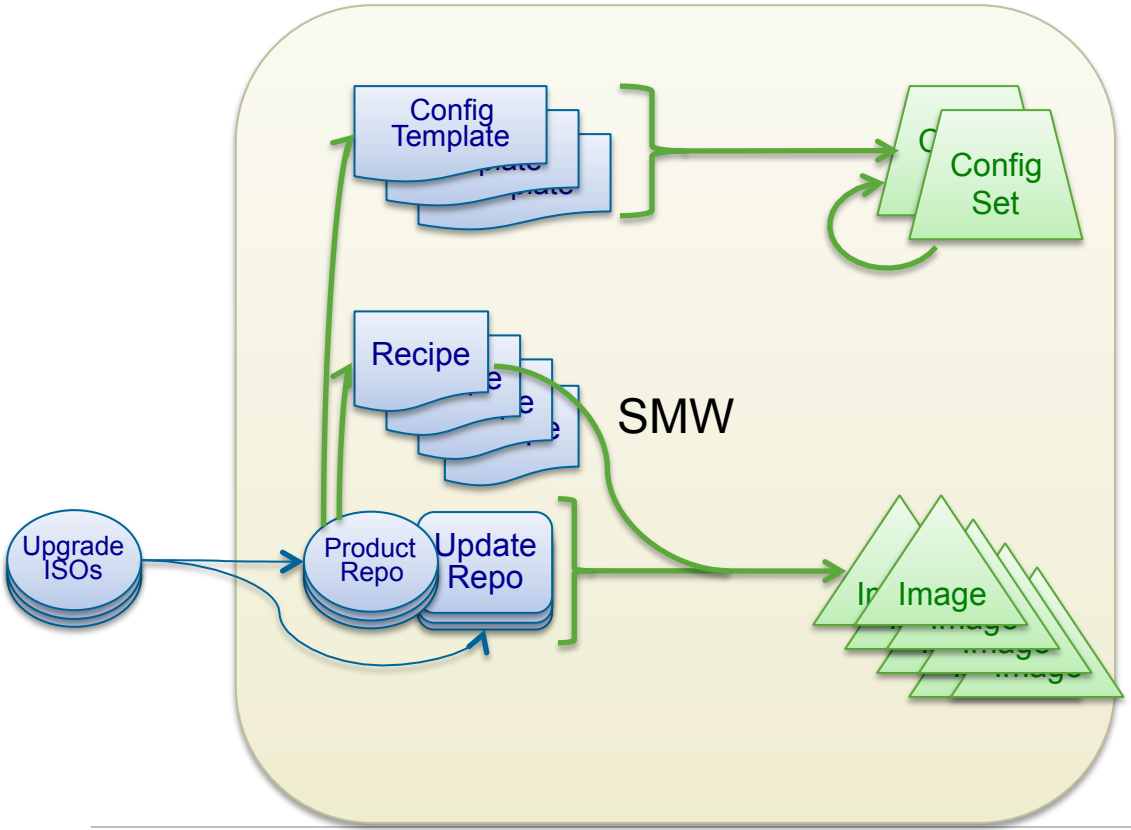
Fresh install

- **Run installer to**
  - Create filesystems from SMW storage set
  - Install software and configuration templates into a snapshot on SMW
- **Use snaputil to choose new SMW snapshot then**
  - Reboot SMW
  - Reboot cabinet and blade controllers with new HSS images
  - Update cabinet and blade controller firmware and node BIOS (if needed)
- **Run imgbuilder to create CLE boot images from image recipes**
- **Use NIMS to map boot images and kernel parameters to nodes**
- **Create config set using configurator**
- **Boot CLE**
  - Creates filesystems from CLE storage set via Ansible plays

Installer

- **Install SMW and CLE software together**
- **Installer is modular and can run tasks from its own media and extra media provided to it**
  - SMW media plus CLE media plus SLE (security) update media
  - Linux vendor media (no tasks)
- **Software repositories created on SMW**
  - /var/opt/cray/repos
  - rpms are synchronized from software media to repos

Staged Upgrades



- **Focus:**
  - Minimal downtime
  - Safe: rollback possible
- **Snapshots and Versions**
  - SMW root gets btrfs snapshot
  - CLE objects are all explicitly versioned
- **While running current system in production…**
  - Update repositories
  - Apply software to snapshots
  - Build new CLE images
  - Apply new configuration to config sets
- **Switch to new release**
  - Shut down system
  - Update firmware
  - Refresh snapshots and config sets where necessary
  - Boot new system
- **Revert if necessary**

Staged upgrades reduce downtime

- **Create a btrfs snapshot using the snaputil command or installer**
  - Installation of new software happens to that snapshot
- **Use snaputil to chroot into the snapshot to**
  - Run imgbuilder to create CLE boot images from image recipes and optionally update NIMS map
  - Update config sets using IMPS configurator (cfgset)
    - Update config set for CLE without pre/post-scripts
    - Update global config set without pre/post-scripts
  - Use NIMS to map boot images and kernel parameters to nodes
- **When ready to use the new software, use snaputil to choose snapshot**
  - Shutdown CLE
  - Reboot SMW to the new snapshot
  - Reboot cabinet and blade controllers with new HSS images
  - Update cabinet and blade controller firmware and node BIOS (if needed)
  - Update global and CLE config set with pre/post-scripts
  - Boot CLE

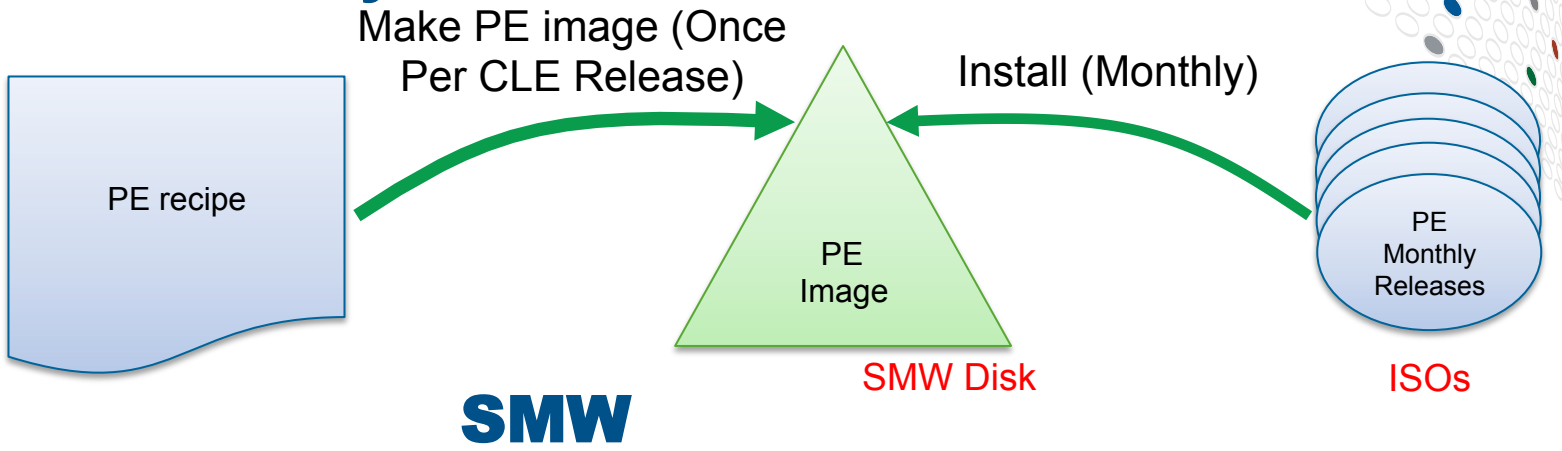
Reverting to a previous software version

- **Reverting to an older snapshot for fallback is easy**
  - Shutdown CLE
  - **snaputil list** will show available snapshots
  - **snaputil default oldname** will set the next SMW reboot to use the “oldname” snapshot
  - Reboot SMW
  - Reboot cabinet and blade controllers with controller images from this snapshot
  - Update cabinet and blade controller firmware (if needed)
  - Boot CLE

Programming Environment

- **Same PE software content can be used for:**
  - Compute
  - Login
  - eLogin
- **Installed and managed on the SMW**
  - Uses the craype-installer
  - Deployed to boot node for internal XC nodes
  - Deployed to Cray Management Controller (CMC) for eLogin
- **PE available via a network filesystem for diskless XC nodes**

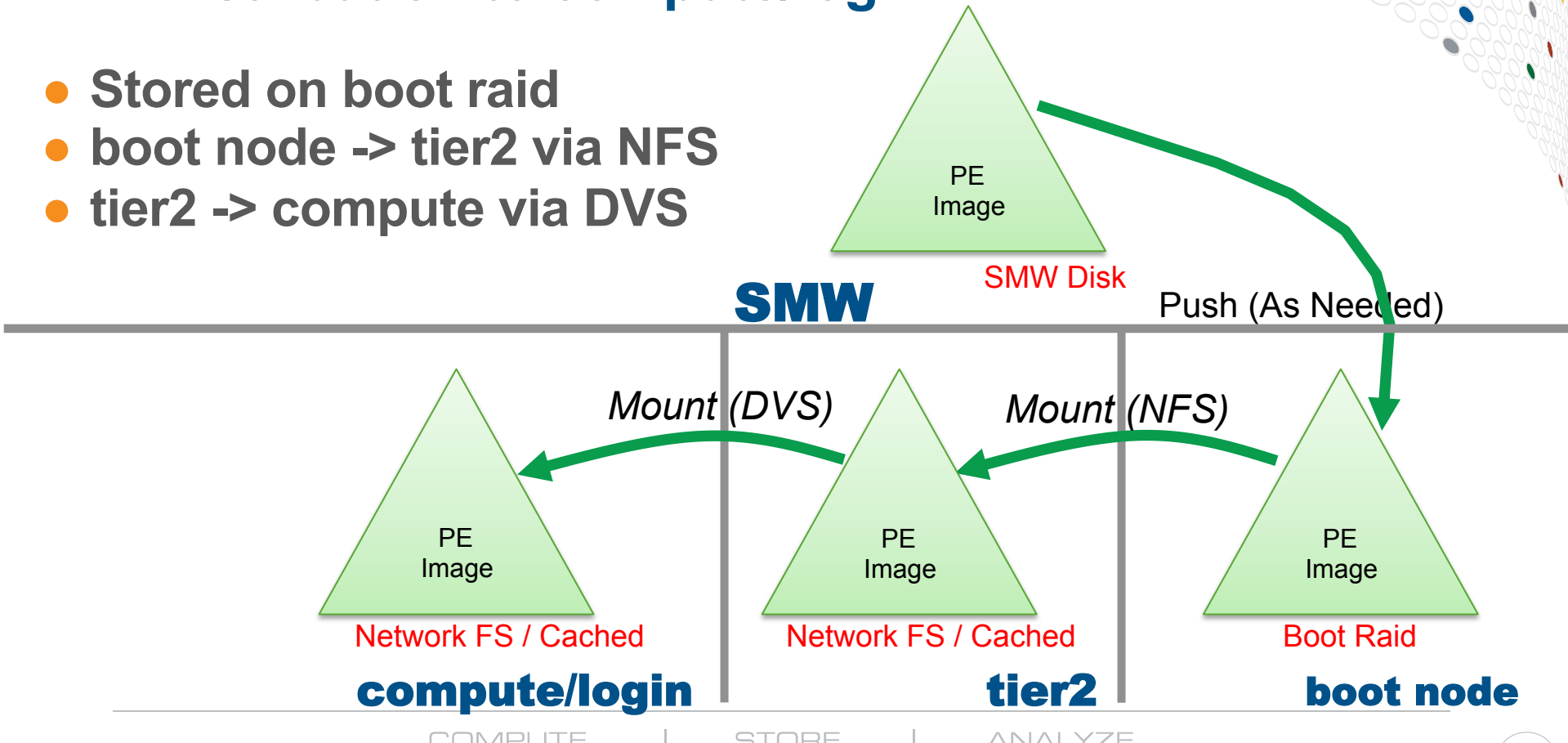
PE Install and Lifecycle



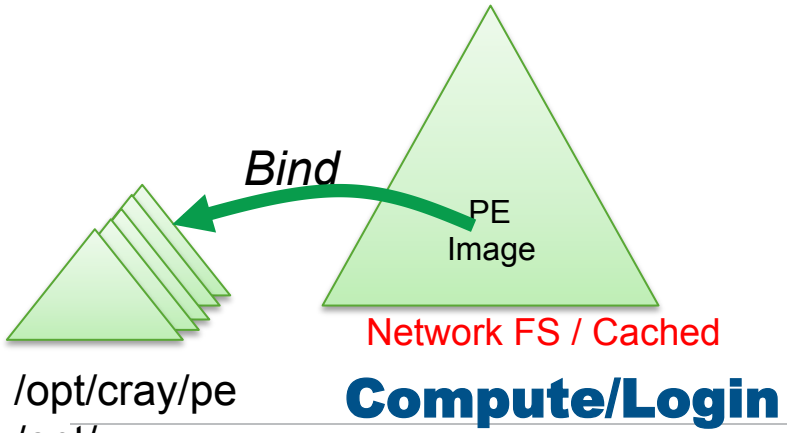
- **PE Installer targets image root**
- **Lifecycle of PE:**
  - Build new PE image with each CLE release
    - Do this as part of fresh install or staged CLE upgrade
    - Add PE releases that are needed
    - If older PE releases are no longer needed, leave out
    - Validates dependencies are still valid for older PE releases
  - Add monthly PE releases to existing PE image within a CLE release

PE Distribution to compute/login

- Stored on boot raid
- boot node -> tier2 via NFS
- tier2 -> compute via DVS

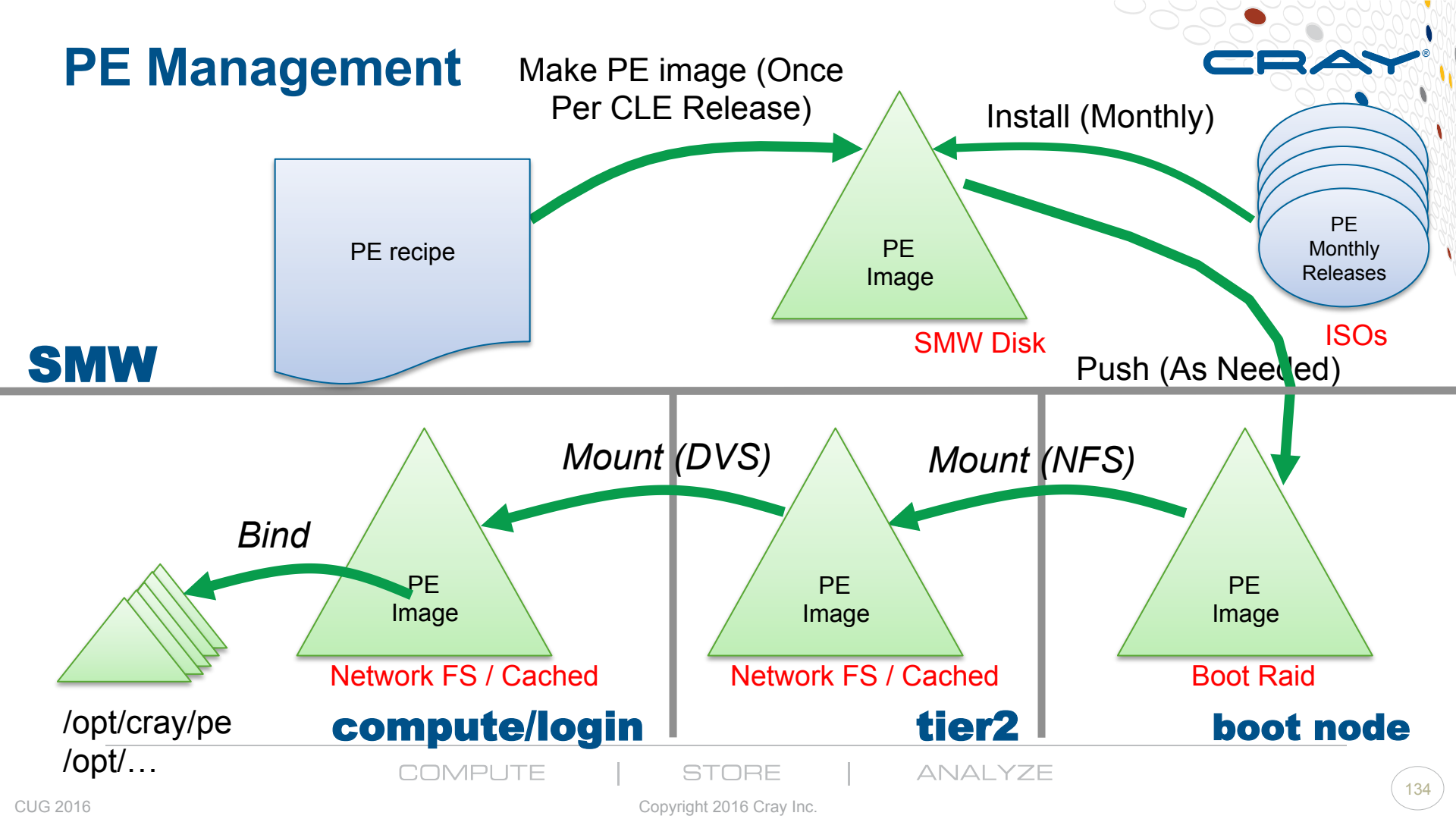


PE Node Setup and Configuration



- Driven by PE setup script
- Does bind mounts from image:
  - /opt/cray/pe
  - /opt/totalview
  - …
- Sets up modules
- Tweaks default shell profile
- Run at boot time
- Can be run again if PE image changes

PE Management



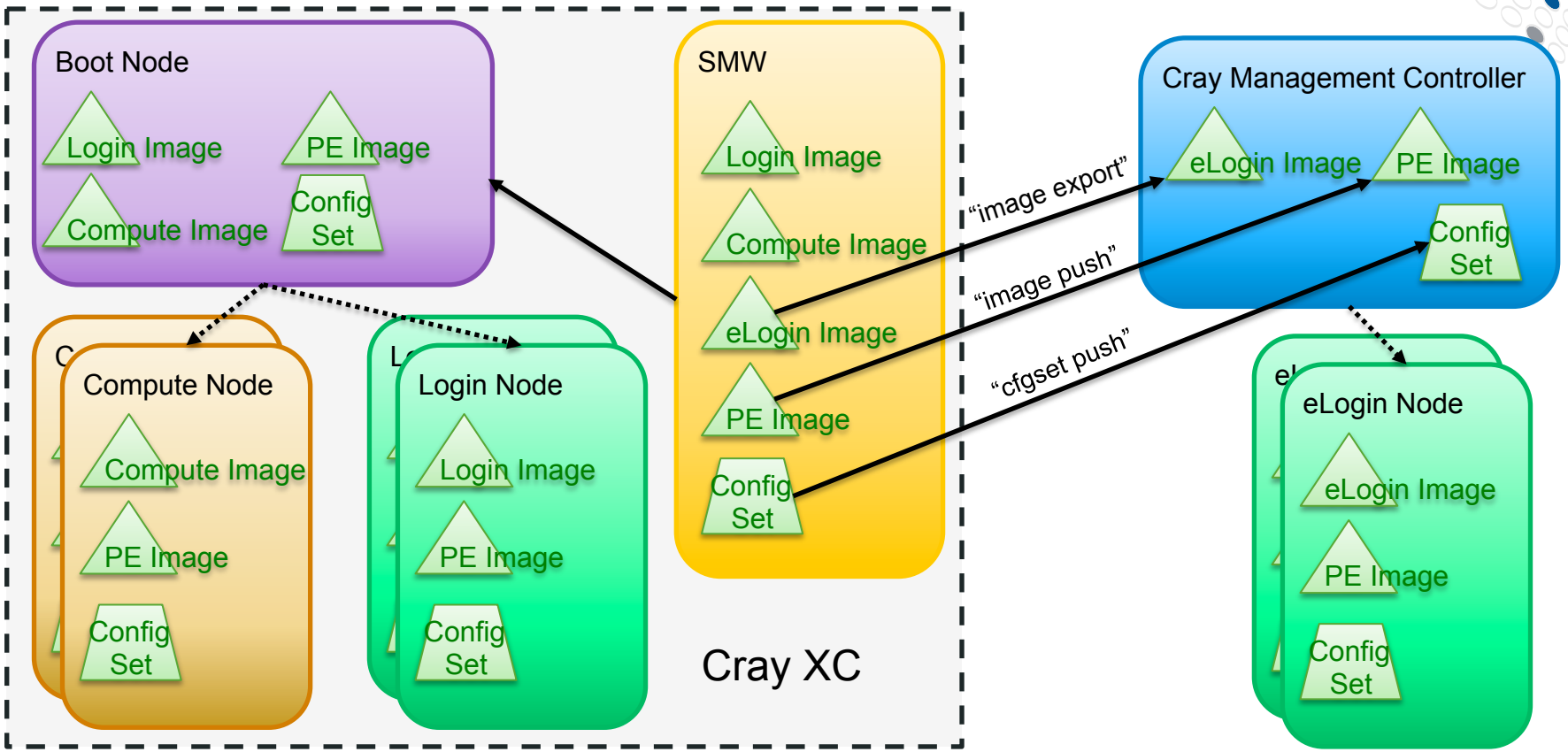
eLogin

- **eLogin**
  - Provides a login, job submission, and development environment for the Cray XC
  - Available to users independent of Cray XC availability
  - Replaces previous CDL (Cray Development and Login) and esLogin
  - Integrated with site workload manager (WLM), Lustre filesystem, etc.
- **eLogin recipe**
  - Another recipe, built with the same IMPS Image Tool
  - Shares base SLES 12 and CLE packages with login image
  - Includes content specific to eLogin nodes
  - Excludes content specific to XC nodes (Cray kernel, drivers for the high speed network, etc.)
- **eLogin uses the same config set as the login nodes**
  - Includes eLogin-specific configuration
  - Some content in config set can be excluded from eLogin's view of config set
- **eLogin uses the same Programming Environment software**

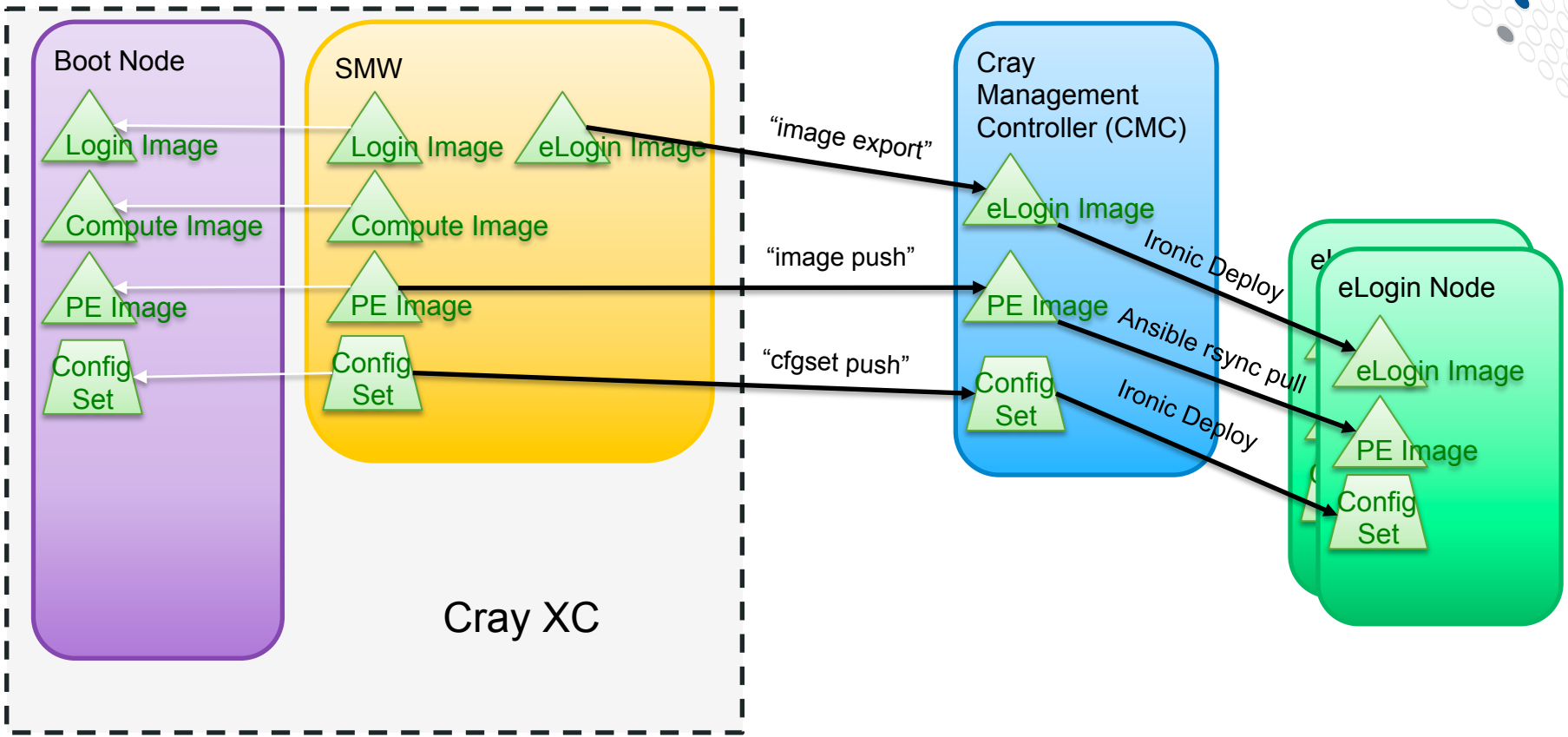
eLogin CMC

- **Cray Management Controller (CMC) for eLogin**
  - Software
    - Linux via Cray bootable CentOS 7.1 DVD
    - Cray System Management Software (CSMS) ISO
    - eLogin software ISO
  - Interacts with SMW to get shared image and configuration services
    - Must be network connected to the SMW
  - Cray System Management Software (CSMS) uses OpenStack framework with eLogin specific customizations
    - Cinder – block storage service
    - Glance - Image service
    - Heat – orchestration service (booting nodes, deploying nodes)
    - Ironic with Fuel – bare metal provisioning service
      - Cray customized with conman for remote console and console logging
    - Keystone - authentication (LDAP) and authorization (mysql) between services
    - Neutron - network management
    - Nova - node lifecycle management
    - Swift – object storage service
  - eLogin image sent from SMW to OpenStack Glance

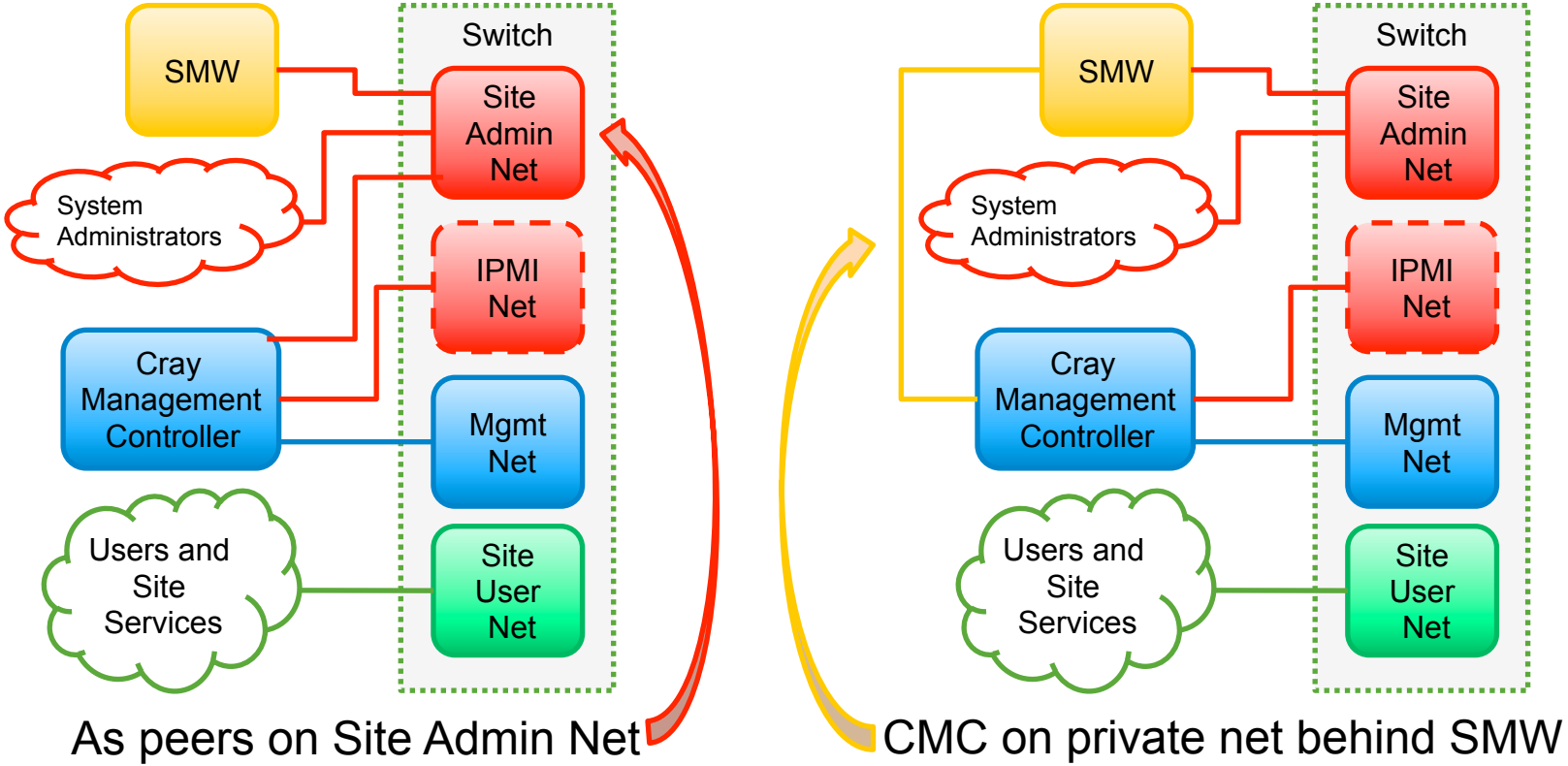
eLogin Image/Config Flow



eLogin: Detailed Data Flow



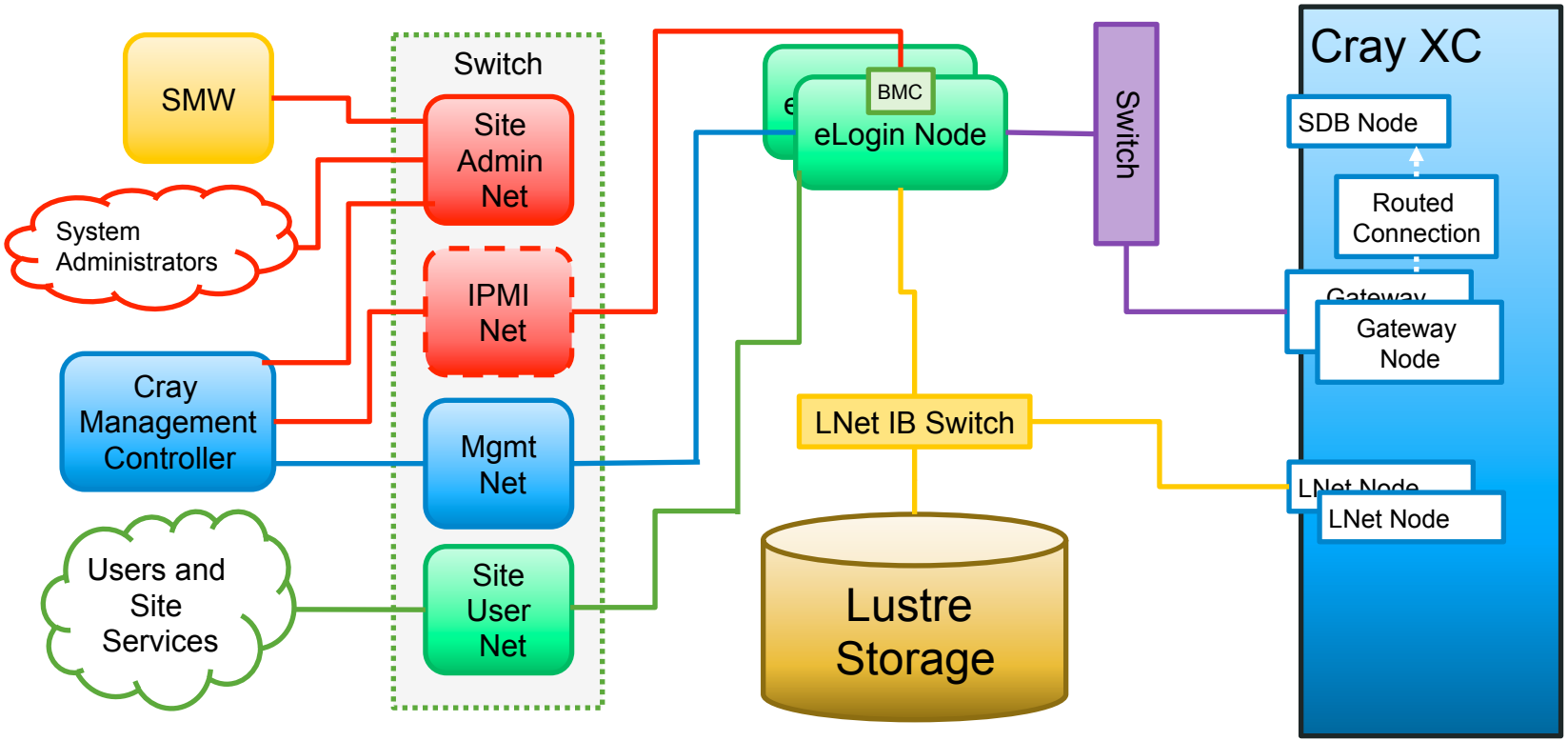
eLogin: Networking
SMW to Cray Management Controller options



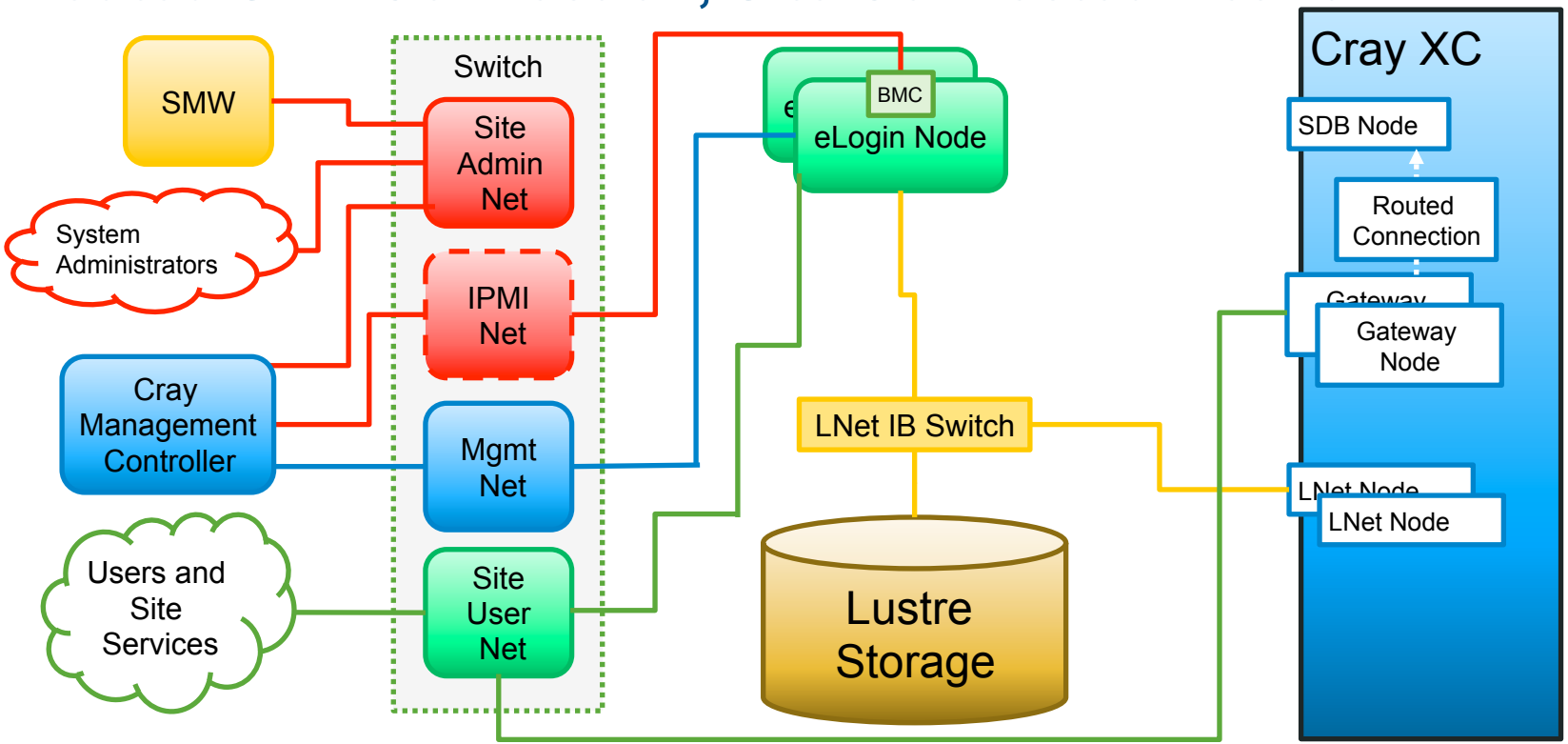
eLogin Network Architecture

- **eLogin/CMC networks**
  - Site admin network
    - Used by sysadmins to login to CMC
  - IPMI network
    - Connects CMC to eLogin nodes, switches
    - Used to manage an provision the eLogin systems
  - Management network
    - Connects CMC to eLogin IPMI devices
  - Site user network
    - User access and authentication services (LDAP)
  - IB-network
    - InfiniBand network connects eLogin and XC LNet nodes to Lustre servers
- **There are four distinct configurations for attaching eLogin to XC**
  - Common to all of them
    - DRAC port of each eLogin node must be connected to the IPMI network
    - Ethernet 1 of each eLogin node must be connected to the management network
    - Ethernet 2 must be connected to the site user network to allow users to login

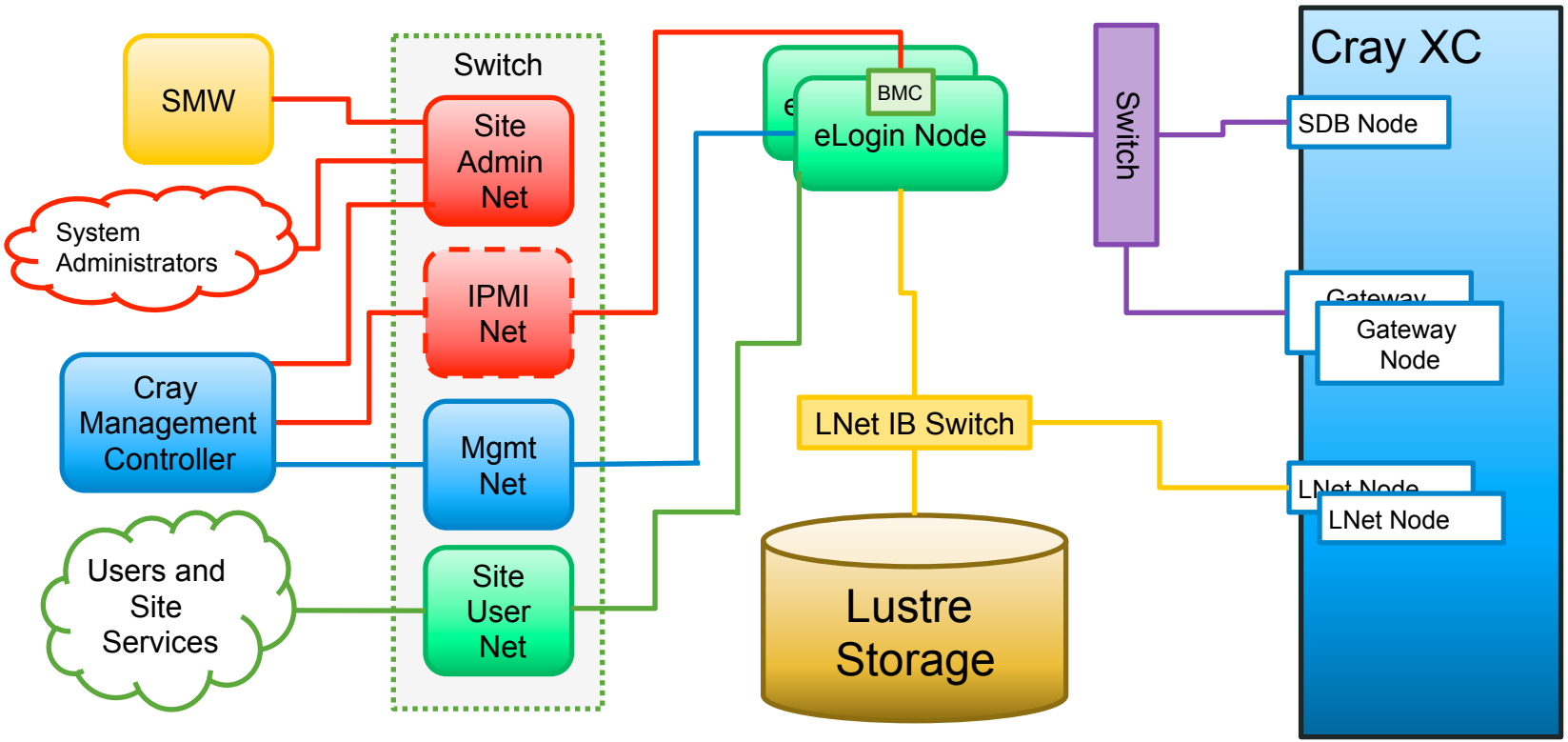
eLogin: Networking Example
Routed SDB Connection



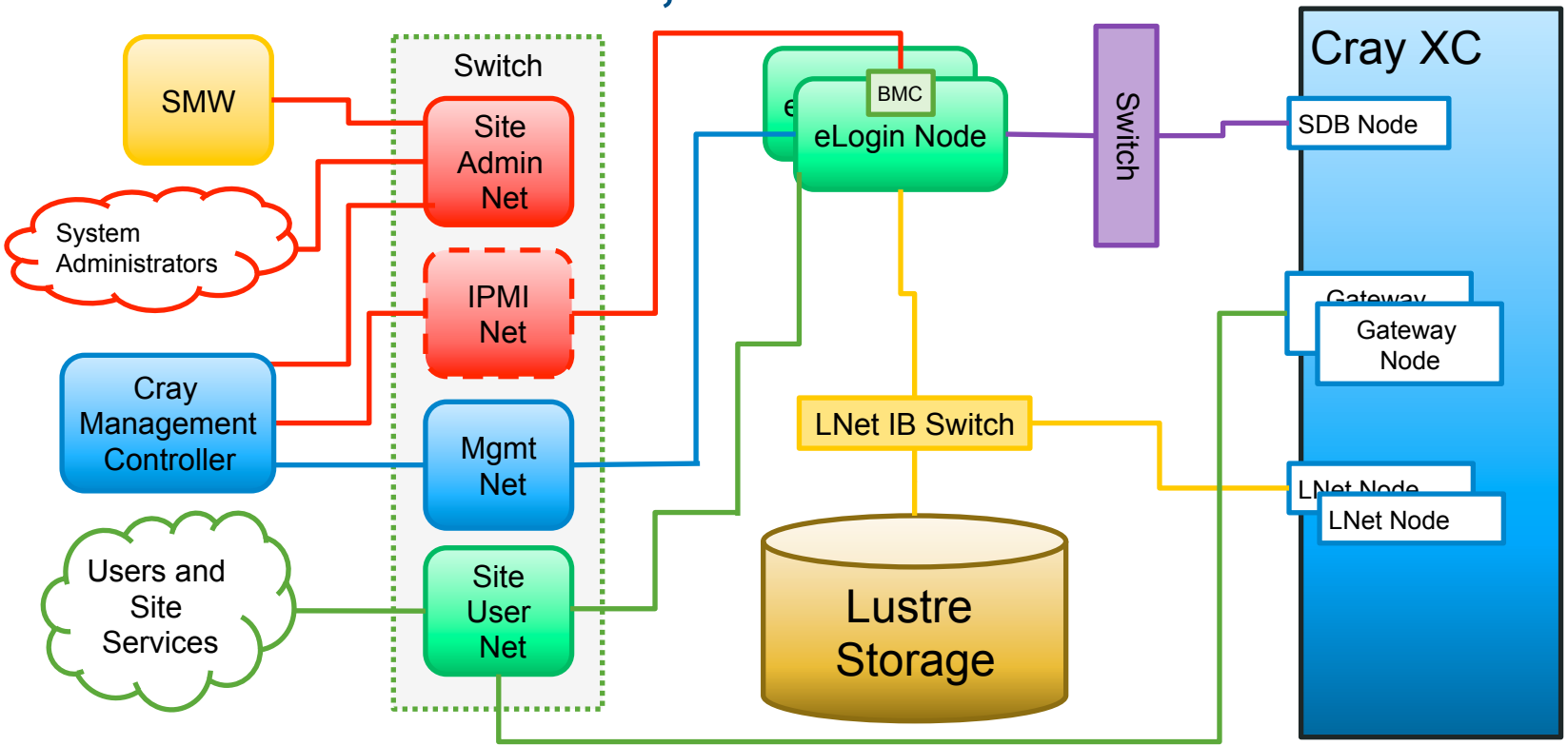
eLogin: Networking Example
Routed SDB Connection, Site Connected Network



eLogin: Networking Example
Direct SDB Connection



eLogin: Networking Example
Direct SDB Connection, Site Connected Network



CMS Agenda

- Introduction
- Overview of new concepts
- **Software installation**
- Configuration
- Booting
- Reconfiguration
- Software update
- Summary
- Q & A

Software Installation

- **Fresh install only**
  - Install SLES 12 on SMW
  - Gather software ISOs
  - Create storage set configuration
  - Provision storage used by SMW
- **Run installer**
- **snaputil – manage snapshots**
- **Configure SMW for XC system hardware**
- **imgbuilder – prepare CLE boot images**
- **NIMS – manage boot images and kernel parameters**
- **Create/assign images**
- **Security Updates**
- **Live Updates**

Software Installation – Fresh Install Only

- **Install SLES 12 on SMW from bootable DVD**
  - R815 SMW: configure software RAID1 on 2 SMW internal disks
  - R630 SMW: configure RAID5 on internal RAID controller with 4 disks
  - Create /, swap, and /boot filesystems
  - Install software after confirming installation choices
- **Gather software ISOs**
  - All SUSE and CentOS ISOs should be in /root/isos
- **Mount SMW media**

  smw# **mkdir -p /media/SMW**

  smw# **mount -o loop,ro /root/isos/smw-8.0.20-201511131259.iso /media/SMW**

Software Installation – Installer

SMWinstall [--target=NAME] [--media=PATH] [--plus-media=PATH, --plus-media=PATH, ...] [options]
SMWinstall --mode bootstrap [--reconfigure] [options]
SMWinstall --mode provision-storage [--force] [options]
Options:

--target=NAME	Install software into btrfs snapshot NAME
--media=PATH	Path to installation media
--plus-media=PATH	Additional media to process after --media is processed
--forceupdate	Force installation of packages, even if versions match or we're asking to downgrade packages, which zypper won't do by default
--storage-config=FILE	Use specified storage configuration template
--storage-set=NAME	Name of storage set in storage configuration template to use for the management node (default: smwdefault)
--iso-dir=DIR	Location where Linux distribution ISOs can be found (default: /root/isos)

Software Installation – Storage Sets

- **Install in bootstrap mode**
  - smw# **/media/SMW/SMWinstall --mode bootstrap**
  - Installs rpms and config templates to bootstrap creating storage configuration
  - Runs the configurator interactively for the cray_bootraid config template
  - Customize with disk devices from boot RAID for the SMW, boot, and SDB LVM volume groups
  - Check sizes for filesystems, adjust as needed
    - Sizes shown in this presentation might have different recommendations in final release
    - Consult with Cray before changing any of the filesystem types
  - Nodes will “self configure” volume groups, volumes, and filesystems from this file
    - SMW – while running installer to provision storage during fresh install
    - boot and SDB nodes – during first boot after fresh install
- **Install in bootstrap mode with existing storage configuration template**
  - smw# **/media/SMW/SMWinstall --mode bootstrap --storage-config /path/to/ cray_bootraid_config.yaml**

Software Installation – Storage Sets 1

```
cray_bootraid:
  enabled: true
  settings:
    storage_sets:
          data:
          -   key: cledefault
              volume_groups:
              -   key: boot_node_vg
                  owner: c0-0c0s0n1
                  devices:
                  - /dev/disk/by-id/wwn-0x60080e500036ae3e0000042554818230
                  volumes:
```

Software Installation – Storage Sets 2

```
volumes:
  - key: home
    description: LVM volume for user home
                 directories on the boot node.
    type: lvm
    fs_type: xfs
    fs_size: 100
    fs_mount_point: /cray_home
    snapshot: false
    mount_options: ''
  - key: imps
    description: LVM volume for IMPS on the
                 boot node.
    type: lvm
    fs_type: btrfs
    fs_size: 250
    fs_mount_point: /var/opt/cray/imps
    snapshot: false
    mount_options: noacl
  - key: nvolatile
    description: LVM Volume for persistent
                 (non-volatile) storage.
    type: lvm
    fs_type: xfs
    fs_size: 100
    fs_mount_point: /non_volatile
    snapshot: false
    mount_options: ''
```

Software Installation – Storage Sets 3

```
- key: sdb_node_vg
  owner: c0-0c0s1n1
  devices:
  - /dev/disk/by-id/
wwn-0x60080e500036ae3e0000044e5523b8c6
  volumes:
  - key: db
    description: LVM volume for the CLE system
                 database (SDB).
    type: lvm
    fs_type: xfs
    fs_size: 20
    fs_mount_point: /var/lib/mysql
    snapshot: false
    mount_options: ''
```

```
- key: alps
  description: LVM Volume for ALPS shared
               data.
  type: lvm
  fs_type: xfs
  fs_size: 20
  fs_mount_point: /alps_shared
  snapshot: false
  mount_options: ''
```

Software Installation – Storage Sets 4

```
- key: smwdefault
        volume_groups:
        - key: smw_node_vg
          owner: smw
          devices:
          - /dev/disk/by-id/
wwn-0x60080e500036ae3e0000042354818169
          - /dev/disk/by-id/
wwn-0x60080e500036ae3e0000044b54f568d1
          volumes:
          - key: home
            description: LVM volume for user
                  home directories on the SMW.
            type: lvm
            fs_type: xfs
            fs_size: 200
            fs_mount_point: /home
            snapshot: false
            mount_options: ''
- key: db
  description: LVM volume for HSS
                database.
  type: lvm
  fs_type: btrfs
  fs_size: 10
  fs_mount_point: /var/lib/mysql
  snapshot: true
  mount_options: ”
```

Software Installation – Storage Sets 5

```
- key: log
  description: LVM Volume to store log, debug,
               and dump data.
  type: lvm
  fs_type: xfs
  fs_size: 1000
  fs_mount_point: /var/opt/cray/disk/1
  snapshot: false
  mount_options: ”
- key: imps
  description: LVM Volume for storage of IMPS
               configuration and images.
  type: lvm
  fs_type: btrfs
  fs_size: 1000
  fs_mount_point: /var/opt/cray/imps
  snapshot: false
  mount_options: "
- key: repos
  description: LVM Volume to store RPM
               repository data.
  type: lvm
  fs_type: btrfs
  fs_size: 200
  fs_mount_point: /var/opt/cray/repos
  snapshot: true
  mount_options: "
```

Software Installation – Installer

- **Provision SMW storage set on boot RAID**
  smw# **/media/SMW/SMWinstall --mode=provision-storage**
  - Prepare SMW storage on boot RAID
  - SMW LVM Volume Group created, SMW filesystems created
- **Install SMW, CLE and Security Updates together**
  smw# **/media/SMW/SMWinstall --plus-media /root/isos/sleupdate-image-201510141357.iso --plus-media=/root/isos/ cle-6.0.20-201511131259.iso --target=${SNAPSHOT}**
  - If no target on command line, a snapshot will be created
- **All tasks on SMW media done first, then tasks from the other media**
  - Changes made in snapshot
- **Logs created in /var/adm/cray/logs/install/install.YYYYMMDD.log**
  - Very verbose log file with all zypper/rpm messages

Software Installation – snaputil

- **snaputil – manage SMW root volume btrfs subvolume snapshots**
- **Full log output can be found in /var/adm/cray/logs/snaputil/ snaputil.YYYYMMDD.log**

snaputil list [<name>] [options] [--sort=(name|size|created) [--desc]] [--quiet]
snaputil default <name> [options]
snaputil create <name> [options] [--readonly] [--from=snapshot]
snaputil delete <name> [<name>...] [options]
snaputil show <name> [options]
snaputil bootmenu-enable <name> [options]
snaputil bootmenu-disable <name> [options]
snaputil diff <snap1> <snap2> [<filename>] [options]
snaputil rename <name> <new_name> [options]
snaputil chroot <name> [options]

Software Installation – snaputil list

- **List all snapshots**

smw# **snaputil list**

Status	Name	Size (MB unshared)	Created
	@	20355.4	2015-11-07 11:10:12
	SLES12	8.36	2015-11-07 11:58:38
	SMW-8.0UP01_CLE-6.0UP01.20160304	12.53	2016-03-03 07:15:57
cur,def	SMW-8.0UP01_CLE-6.0UP01.20160323	757.5	2016-03-23 08:54:40

Software Installation – snaputil create

- **Create a new snapshot**

  smw# **snaputil create demo**

  Created subvolume demo in /media/root-sv/snapshots/demo

  smw# **snaputil list**

Status	Name	Size (MB unshared)	Created
	@	20355.4	2015-11-07 11:10:12
	SLES12	8.36	2015-11-07 11:58:38
	SMW-8.0UP01_CLE-6.0UP01.20160304	12.53	2016-03-03 07:15:57
cur,def	SMW-8.0UP01_CLE-6.0UP01.20160323	757.5	2016-03-23 08:54:40
	demo	0.2	2016-03-26 13:32:57

Software Installation – snaputil show

- **Show a snapshot**

```
smw# snaputil show demo
boot menu      : False
booted         : False
btrfs_object_id : 1177
cle_version    : 6.0.68
created        : 2016-03-26 13:32:40
default        : False
initrd         : initrd-3.12.28-4-default
kernel         : vmlinuz-3.12.28-4-default
name           : demo
parent         : smw-8.0.66_cle-6.0.66.20160323
path           : /media/root-sv/snapshots/demo
read-only      : False
smw_version    : 8.0.66
smwha_version  : 12.1.17
storage_set    : smwdefault
subvolumes     :
  /var/lib/mysql:demo
  /var/opt/cray/repos:demo
total size     : 1729.60 MB
unshared size  : 0.02 MB
updated        : 2016-03-26 13:32:57.969610
```

Software Installation – snaputil default

- **Set snapshot to be used for next SMW boot**

  smw# **snaputil default demo**

  subvolume demo is now default.

  smw# **snaputil list**

Status	Name	Size (MB unshared)	Created
	@	20355.4	2015-11-07 11:10:12
	SLES12	8.36	2015-11-07 11:58:38
	SMW-8.0UP01_CLE-6.0UP01.20160304	12.53	2016-03-03 07:15:57
cur	SMW-8.0UP01_CLE-6.0UP01.20160323	757.5	2016-03-23 08:54:40
def	demo	0.2	2016-03-26 13:32:57

Software Installation – snaputil diff

- **What files are different between snapshots?**

  smw# **snaputil diff SMW-8.0UP01_CLE-6.0UP01.20160323 demo**
  etc/motd
  root/.bash_history
  root/.viminfo

- **Compare files which are different between snapshots**

  smw# **snaputil diff demo SMW-8.0UP01_CLE-6.0UP01.20160323 etc/motd**
  --- /media/root-sv/snapshots/demo/etc/motd 2014-10-14 03:52:43.000000000 -0500
  +++ /media/root-sv/snapshots/SMW-8.0UP01_CLE-6.0UP01.20160323/etc/motd 2016-03-26 13:46:35.738501158 -0500
  @@ -0,0 +1 @@
  +test of change to /etc/motd

Software Installation – snaputil rename/delete

- **Rename a snapshot**
  smw# **snaputil rename demo mydemo**
  subvolume was renamed to mydemo
- **Delete a snapshot**
  smw# **snaputil delete mydemo**
  mydemo was deleted
  smw# **snaputil list**

Status	Name	Size (MB unshared)	Created
	@	20355.4	2015-11-07 11:10:12
	SLES12	8.36	2015-11-07 11:58:38
	SMW-8.0UP01_CLE-6.0UP01.20160304	12.53	2016-03-03 07:15:57
cur,def	SMW-8.0UP01_CLE-6.0UP01.20160323	757.5	2016-03-23 08:54:40

Software Installation – Configure SMW

- **Fresh install only - very similar to previous releases**
  - After software installed, reboot SMW to new snapshot
  - Initialize Power Management database
  - Discover XC hardware with xtdiscover
  - Discover routing configuration of HSN with rtr
  - Update firmware and BIOS with xtzap
  - Create new boot images with imgbuilder
  - Prepare global config set worksheets
  - Update global config set
  - Prepare CLE config set worksheets
  - Create config set for CLE

Software Installation – Configure SMW

- **Update only - many actions done in new snapshot before SMW reboot**
  - After software installed, chroot to new snapshot
    - Create new boot images with imgbuilder
    - Update config set for CLE without pre/post-scripts
    - Update global config set without pre/post-scripts
  - When new configuration is ready, reboot SMW
  - Update global and CLE config set with pre/post-scripts
  - Discover XC hardware with xtdiscover
  - Discover routing configuration of HSN with rtr
  - Update firmware and BIOS with xtzap

Software Installation – imgbuilder

- **Build and package a set of IMPS images and update NIMS node mappings**

```
imgbuilder [--map [--partition=PART,...]] [options] [--verbose] [-- <key>=<val>...]
imgbuilder --bootstrap-nims [--nims-map=MAP] [--partition=PART] [--verbose]
    Options:
      --bootstrap-nims          Update nims table with groups derived from "type" where group is empty.
                                This helps create a meaningful mapping on new systems without node
                                groups defined.
     --nims-map=MAP             Maps the images to a specific NIMS map. By default, imgbuilder maps
                                images to the active NIMS map.  This option is valid only with
                                the --bootstrap-nims option.
      -c --config=FILE          Use the specified configuration YAML
      -g --image-group=GROUP    Use the specified image group
      --map                     Add newly-built images to the NIMS
      --partition=PART          When mapping images, update a partition's active map
      --force                   Tell IMPS to build image roots even if they already exist.
```

Software Installation – imgbuilder

- **Logs created**
  - /var/adm/cray/logs/imgbuilder/imgbuilder.YYYYMMDD.log
- **imgbuilder configuration file lists the set of images to build**

  /etc/opt/cray/config/global/config/cray_image_groups.yaml
- **Image names defined in the above configuration file have runtime values available to them. This includes:**

  {date} includes the current system date (20140314)
  {time} includes the current system time. (134514)
  {host} includes the current system hostname
  {cle_release}
  {cle_build}

Software Installation – imgbuilder

- **Additional values can be added by passing in options after '--' to imgbuilder**
- **To add a runtime-specified prefix to some other tags for compute node images**
  - In the configuration file:

    - recipe: "compute_cle_6.0up01_sles_12_x86-64_ari"
      dest: "{compute_prefix}_{cle_release}-build{cle_build}_my{date}.cpio”
- **Then when invoking imgbuilder, we can specify the value to use for {compute_prefix}:**

  smw# **imgbuilder -- compute_prefix=my_compute**
- **This will yield a compute image with a name such as:**

  my_compute_cle_6.0.UP01-build6.0.68-my20160317.cpio

Software Installation – IMPS commands

- **IMPS commands used by installer and imgbuilder**
  - repo
    - Manage creation and distribution of repository content (rpms)
  - image
    - Create image roots
    - Provision and package for booting
    - Deploy
      - boot image for XC systems
      - push an image to remote system for projection
      - register image with Openstack Glance server on CMC for eLogin
- **Other IMPS commands**
  - pkgcoll
    - Manage package collections (logical groupings of rpms)
  - recipe
    - Manage image recipes to associate rpms (individual or in package collections) with one or more repositories
    - Can extend build and configuration actions via post build operations

Software Installation – NIMS

- **Node Image Mapping Service (NIMS) maps a node to "boot attributes" when a node is booted**
  - boot image
  - loadfile
  - config set
  - kernel parameters
- **NIMS daemon (nimsd)**
  - Responds to HSS boot manager via HSS events to provide boot attributes when booting and rebooting nodes
  - Interacts with administrative commands nimscli (UP00 only) or cmap and cnode (UP01)

Software Installation – NIMS map and node

- **NIMS Map**
  - Collection of nodes, either entire machine (p0) or a system partition (p2)
  - Multiple maps possible, but only one map active for a given partition
  - Maps are associated with one partition and cannot span partitions
  - The system administrator can control which map is the active map for a partition
  - Contains NIMS node groups
    - A node can be assigned to an arbitrary group for ease of changing boot attributes
- **NIMS Node**
  - Represents the physical, bootable node on the XC system

Software Installation – NIMS cmap

- **cmap - take administrative actions on NIMS map for the specified partition and config set**
  - create – Create new empty maps or clone existing maps
  - list – Display maps and associated metadata
  - merge – Copy settings from one map and apply them to another
  - update – Modify the attributes or contents of a map
  - setactive – Make a map active

Software Installation – NIMS cmap

- **List all NIMS maps**

```
smw# cmap list
NAME                                PARTITION  ACTIVE_MAP
cle6.0.79.created201604010730       p0         True
p0                                  p0         False
```

- **Show info for a single map**

```
smw# cmap list p0
NAME                                PARTITION  ACTIVE_MAP
p0                                  p0         False
```

- **Lists all maps with the specified fields in columns**

```
smw# cmap list --fields name,active_map,default_config_set,version,path
NAME                                ACTIVE_MAP  DEFAULT_CONFIG_SET  VERSION  PATH
cle6.0.79.created201604010730       False       p0                  2        /var/opt/
cray/imps/config/sets/global/nims/maps/p1/cle6.0.79.created201604010730
```

Software Installation – NIMS cmap

- **Create new map**
  smw# cmap create demo
- **Create new map as a clone**
  smw# cmap create --clone cle6.0.79.created201604010730 demo
- **Set map to be active**
  smw# cmap setactive demo
- **Find currently active map**
  smw# cmap list --fields name,active_map|grep "True"
  demo True
- **Update config set for a map**
  smw# cmap update –config-set p0-test demo
- **Merge contents from one map into another**
  smw# cmap merge test_changes demo

Software Installation – NIMS cnode

- **cnode – set and display boot attributes for NIMS nodes**
  - list – Display nodes and associated metadata
  - update – Modify a node by setting or unsetting attributes
    - add/remove NIMS group
    - set/unset boot image
    - set/unset loadfile
    - add/remove/set kernel parameter key or key=value pairs
    - set/unset configset
    - partition (for the NIMS map)
    - map (if not the active NIMS map)

Software Installation – NIMS cnode

- **cnode --filter FILTER**
  - Works for both **cnode list** and **cnode update**
  - Comma-separated string of key=value pairs used to create a subset of the node list
  - Valid keys are name (the node), image, type, parameters, loadfile, config_set, and group
  - Specifying multiple key-value pairs will perform a logical AND of all provided criteria
    - --filter group=my_group,image=/path/to/image.cpio
      - Only nodes in my_group with that boot image will be listed or updated

Software Installation – NIMS cnode list

- **To display all nodes in the current active map:**

smw# **cnode list**

```
 Node          Type      Group                                          Image
Loadfile    Config Set   Parameters
c0-0c0s0n0   service  service  /var/opt/cray/imps/boot_images/service_cle_6.0.UP01-
build6.0.68_sles_12-created20160212.cpio          -          NIMS_GROUP=service
ids=10.128.0.130,10.128.0.138 config_set=p0

c0-0c0s10n3  compute  compute  /var/opt/cray/imps/boot_images/initrd-compute-
large_cle_6.0.UP01-build6.0.68_sles_12-created20160212.cpio  -
NIMS_GROUP=compute netroot=compute-large_cle_6.0.UP01-build6.0.68_sles_12-
created20160210 ids=10.128.0.130,10.128.0.138 config_set=p0

c0-0c1s1n1   service  login    /var/opt/cray/imps/boot_images/login_cle_6.0.UP01-
build6.0.68_sles_12-created20160212.cpio          -          NIMS_GROUP=login
ids=10.128.0.130,10.128.0.138 config_set=p0
```

Software Installation – NIMS cnode list

- To display subset of nodes (based on glob pattern) in 'my_map' on partition 'p1:**

  smw# cnode list -p p1 -m my_map c0-0c0s[1-2]***
- To display the name and parameters fields of all nodes in the current active map:**

  smw# cnode list --fields name,parameters**
- To write comma separated node names and groups to the /tmp/node_list.csv file, one node per line, and omit the header row:**

  smw# cnode list --fields name,groups --format csv -o /tmp/node_list.csv -H**
- To display all nodes in the current active map which are in a group called 'my_group' and have kernel parameter LOG_LEVEL=1 set:**

  smw# cnode list –filter group=my_group,parameters=LOG_LEVEL=1**

Software Installation – NIMS cnode update

- To set an image to boot on all service nodes: (--set-image)
  smw# cnode update -i /path/to/image.cpio -g service
- To set a group on some nodes, and then set the image for that group (--remove-group,--add-group)
  smw# cnode update -G service c0-0c0s1n1 c0-0c0s3n2
  smw# cnode update -g dal c0-0c0s1n1 c0-0c0s3n2
  smw# cnode update -i /path/to/dalimage.cpio -g dal
- To unset an image on some nodes: (--unset-image)
  smw# cnode update -I c1-2 c1-3 c1-4
- To set a group for all n1 compute nodes: (--add-group)
  smw# cnode update -g n1group --filter type=compute *n1

Software Installation – NIMS cnode update

- To set kernel parameters for huge page sizes on compute nodes: (--set-parameter)
  smw# cnode update -s hugepagesz=1M -g compute
- To add kernel parameters for huge page sizes on compute nodes: (--add-parameter)
  smw# cnode update -k hugepagesz=512M -g compute
  smw# cnode update -k hugepagesz=128M -g compute
- To remove kernel parameter for a particular hugepagesz key on one node: (--remove-parameter)
  smw# cnode update -K hugepagesz=512M c1-2c0s4n3
- To test a config set on one node (--set-config-set)
  smw# cnode update -c demo c1-2c0s4n3
- To return node back to default config set for map (--unset-config-set)
  smw# cnode update -C c1-2c0s4n3

Software Installation – Create/assign images

- **Fresh install**
  - smw# imgbuilder --bootstrap-nims
  - smw# cnode update -G service c0-0c0s0n2
  - smw# cnode update -g login c0-0c0s0n2
  - smw# cnode update -G service c0-0c0s1n1,c0-0c0s3n2
  - smw# cnode update -g dal c0-0c0s1n1,c0-0c0s3n2
  - smw# imgbuilder --map
- **Software update**
  - smw# imgbuilder –map
- **Check which NIMS map is active**
  - smw# cmap list
- **Check NIMS information per node**
  - smw# cnode list

CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- **Configuration**
- Booting
- Reconfiguration
- Software update
- Summary
- Q & A

Configuration

- **Configuration worksheets**
- **cfgset subcommands**
- **Manipulate config sets**
  - Create or update CLE config set
  - Update global config set
  - Display or search config set data
  - Validate config sets
  - List config sets
  - Compare two config sets
  - Push config set
  - Remove config set
- **Configurator user interface (UI)**

Configuration – Configuration Worksheets

- **Configuration worksheet**
  - YAML file with stub of configuration values and guidance for a service
  - Used as input to the configurator
  - Convenient when large amounts of data needed (network interfaces) which would be awkward and time consuming to gather via interactive user interface
- **Fresh install**
  - Worksheets can be prepared so that someone can start looking at them to fill in answers while other activities such as discovering hardware, flashing firmware, and other hardware or HSS software activities are being done
- **Reconfiguration**
  - Worksheet can be prepared from config set, changed, and then imported back into the config set

Configuration – cfgset subcommands

- **cfgset create**
  - Create new empty config sets or clone existing config sets or prepare configuration worksheets
- **cfgset diff**
  - Show changes between files in config sets
- **cfgset push**
  - Copies contents of a config set to remote host
- **cfgset remove**
  - Remove config set
- **cfgset search**
  - Searches configuration data entries in config sets
- **cfgset update**
  - Modify the attributes or contents of a config set or rename the config set
- **cfgset validate**
  - Checks a config set for syntax, structure, and configuration status of required-level items

Configuration – cfgset options

- **General options for update, create, search**
  - --state or -s {set,unset,all}
  - --level or -l {required,basic,advanced}
  - --service or -S SERVICE
- **Options only for update and create**
  - --mode or -m {auto, interactive, prepare}
  - --worksheet-path or -w WORKSHEET_PATH
- **Options only for search**
  - --term or -t TERM

Configuration – cfgset update

cfgset update <config_set_name>

cfgset update --**state unset** <config_set_name>

cfgset update -**S set** <config_set_name>

cfgset update --**state all** <config_set_name>

cfgset update -S all **--service <service_name>** <config_set_name>

cfgset update -**s <service_name> --level advanced** <config_set_name>

cfgset update **-l required** <config_set_name>

cfgset update **-m prepare** –s <service_name> <config_set_name>

cfgset update **-w <worksheet_path>** -s <service_name> <config_set_name>

Configuration – Manipulate global config set install

- **Prepare global configuration worksheets into work area**
  smw# **cfgset create -m prepare -t global global_example**
  smw# **cd /var/opt/cray/imps/config/sets**
  smw# **cp -p global/worksheets global_example/worksheets**
- **Update global config set with worksheets from work area**
  - Edit some or all global configuration worksheets in work area

  smw# **vi /var/opt/cray/imps/config/sets/global_example/worksheets/***
  smw# **cfgset create -t global --worksheet-path '/var/opt/cray/imps/config/sets/global_example/worksheets/*_worksheet.yaml' global**
- **Update global config set without worksheets**
  smw# **cfgset update -m interactive global**
- **Clone global config set**
  smw# **cfgset create --clone global global-preupgrade-YYYYMMDD**
- **Run Ansible plays on SMW**
  smw# **/etc/init.d/cray-ansible start**

Configuration – Manipulate CLE config set install

- **Create CLE configuration worksheets into work area**

  smw# **cfgset create -m prepare -t cle --no-scripts p0_example**

- **Edit all CLE configuration worksheets in work area**

  smw# **vi /var/opt/cray/imps/config/sets/p0_example/worksheets/***

- **Create CLE config set for partition p0 from work area**

  smw# **cfgset create -t cle --worksheet-path '/var/opt/cray/imps/config/sets/p0_example/worksheets/*_worksheet.yaml' p0**

- **Update CLE config set for partition p0**

  smw# **cfgset update p0**

Configuration – Manipulate config sets update

- **Clone and update global config set**

  smw# **cfgset create --clone global global-preupgrade-YYYYMMDD**

  smw# **cfgset update global**

- **Clone and update CLE config set for partition p0**
  - Software update or reconfiguration

  smw# **cfgset create --clone p0 p0-preupgrade-YYYYMMDD**

  smw# **cfgset update p0**

- **Update config set using a worksheet for a single service**

  smw# **cfgset update p0 -w /my/new/cray_net_worksheet.yaml**

Configuration – Search config set

- **Display or search config set data**

```
smw# cfgset search -l advanced --format full -s cray_net p0
# 606 matches for '.' from cray_net_config.yaml
#-------------------------------------------------------------------------------
cray_net.settings.networks.data.login.ipv4_netmask:
    value:   255.255.240.0
    default: # (empty) (string)
    level:   required
    status:  Configured
cray_net.settings.networks.data.login.ipv4_broadcast:
    value:   # (empty)
    default: # (empty) (string)
    level:   advanced
    status:  Configured
cray_net.settings.networks.data.login.dns_servers:
    value:   172.30.84.40, 172.31.84.40, 172.28.84.40
    default: # (empty) (list)
    level:   basic
    status:  Configured
```

Configuration – Search config set

- **Display or search config set data**

```
smw# cfgset search -l advanced global
# 41 matches for '.' from cray_logging_config.yaml
#-------------------------------------------------------------------------------
cray_logging.settings.global_options.data.global_log_level: 4
cray_logging.settings.global_options.data.raid: 10.1.0.

smw# cfgset search p0 --term c0-0c0s0n1
# 1 match for 'c0-0c0s0n1' from cray_net_config.yaml
#-------------------------------------------------------------------------------
cray_net.settings.hosts.data.bootnode.hostid: c0-0c0s0n1

# 1 match for 'c0-0c0s0n1' from cray_scalable_services_config.yaml
#-------------------------------------------------------------------------------
cray_scalable_services.settings.scalable_service.data.tier1: c0-0c0s0n1, c0-0c1s0n1
```

Configuration – Search config set

- **Display or search config set data**

```
smw# cfgset search p0 -t ipv4_address
# 27 matches for 'ipv4_address' from cray_net_config.yaml
#-------------------------------------------------------------------------------
cray_net.settings.hosts.data.sdbnode.interfaces.hsn_boot_alias.ipv4_address: 10.131.255.253
cray_net.settings.hosts.data.sdbnode.interfaces.primary_ethernet.ipv4_address: 10.3.1.253
cray_net.settings.hosts.data.lnet5.interfaces.ib2.ipv4_address: 10.156.103.3
cray_net.settings.hosts.data.lnet5.interfaces.ib0.ipv4_address: 10.156.101.3
cray_net.settings.hosts.data.rsip2.interfaces.eth0.ipv4_address: 172.30.49.23
cray_net.settings.hosts.data.login1.interfaces.login_ethernet.ipv4_address: 172.30.49.21
cray_net.settings.hosts.data.bootnode.interfaces.hsn_boot_alias.ipv4_address: 10.131.255.254
cray_net.settings.hosts.data.bootnode.interfaces.primary_ethernet.ipv4_address: 10.3.1.254
```

Configuration – Validate config set

- **Validate config sets**

```
smw# cfgset validate p0
INFO - Checking config set directory access
INFO - Checking config set templates
INFO - Validating configuration templates for YAML syntax.
INFO - Validating configuration templates for schema compliance.
INFO - Merging configuration templates and validating schema.
INFO - Validating config data
INFO - cray_persistent_data service validates
INFO - cray_sysenv service validates
INFO - Data validation skipped for service 'cray_lmt': not enabled
INFO - Data validation skipped for service 'cray_logging': inherits global config set values.
…
INFO - ConfigSet 'p0' is valid.

smw# cfgset validate global
```

Configuration – Clone and Diff config set

- **Clone config sets to archive them post upgrade**

  smw# **cfgset create --clone p0 p0-postupgrade-YYYYMMDD**

  smw# **cfgset create --clone global global-postupgrade-YYYYMMDD**

- **Compare pre-upgrade and post-upgrade**

  smw# **cfgset diff p0-preupgrade-20151217 p0-postupgrade-20151217**

  INFO - Calculating path entry differences..

  INFO - Unique entries to IMPS ConfigSet 'p0-postupgrade-20151217':

  - changelog/changelog_2015-12-16T17:00:39.yaml
  - changelog/changelog_2015-12-16T17:04:43.yaml
  - changelog/changelog_2015-12-17T09:26:40.yaml
  - changelog/changelog_2015-12-17T13:28:35.yaml

Configuration – List config set

smw# **cfgset list**

smw# **cfgset list --fields name,created,health**

NAME	CREATED	HEALTH
global	2015-12-05T14:09:40	True
global-postupgrade-YYYYMMDD	2016-01-17T12:00:11	True
p0	2015-12-05T15:11:30	True
p0-postupgrade-YYYYMMDD	2016-01-17T12:03:40	True

smw# **cfgset list --fields name,created,path,health --format csv**

Configuration – More config set manipulations

- **Copy config set to remote host**

  smw# **cfgset push --dest other-smw p0-test**

- **Remove unneeded config set**

  smw# **cfgset remove p0-test**

- **Rename config set**

  smw# **cfgset update --rename newname oldname**

Configuration – Configurator UI

- **Updating or creating a config set with cfgset**
  - Modes
    - automatic
      - Unconfigured questions matching level and state filters from command line asked in predetermined order
    - interactive
      - Can navigate to any question
  - For each question
    - Guidance is displayed for the question
    - Current setting (which may be default) is shown
    - System administrator “answers” the “question” with appropriate data
    - Can “skip” questions or entire services on the first pass
    - Or disable service on the first pass and then selectively configure a previously unconfigured service on another pass
- **User interface for cfgset is hard to describe**
  - Simple example on next few slides

Configuration – Configurator UI – service

```
**********Service Configuration**********
Service: Cray Scalable Services
Guidance: Cray scalable services defines which servers (nodes) are used in the scaling of the system.
          Scalable services must be configured to ensure a properly functioning system.
        * Once defined, these servers will be used in various services, such as DVS, to provide
          horizontal scaling, or scaling out, of those services. Horizontal scaling allows the system
          to utilize user-defined nodes to work together as a single unit to increase workflow output.
        * Scalable services defines a tree of servers starting with the Server of Authority (SoA), followed
          by tier1 and tier2 servers as represented below.
                                tier2
                        tier1   --
                SoA  --         tier2
                        tier1   --
                                tier2
Enable this service? [y/n/?=help] (default: y): y
```

Configuration – Configurator UI – setting

```
******** cray_scalable_services.settings.scalable_service.data.server_of_authority *********
     server_of_authority -- Server of Authority (SoA)
       The Server of Authority (SoA) is the holder of the authoritative configuration information
       for the entire Cray system.  The specific name used here must be reachable (pingable)
       by the tier1 servers such as the boot node.
       On Cray CLE systems, the SoA is typically "smw".
       Default:         Current:
          smw              not configured yet
    Value: string, blank values not allowed, regex=^[A-Za-z0-9][A-Za-z0-9.-]{0,252}$
          level=advanced, state=unset
    Inputs: <string>  -- OR --  menu commands (? for help)

cray_scalable_services.settings.scalable_service.data.server_of_authority
[<cr>=set 'smw', <new value>, ?=help, @=less] $
```

Configuration – Configurator UI context-sensitive help

```
[<cr>=set 'smw', <new value>, ?=help, @=less] $ ?

|--- Command Help
| *   <cr>  - accept the default value(s)
| *   #     - set the value to its default
| *   <     - go back to the previous setting
| *   >     - skip and go to the next setting
| *   ^     - Go to the 'cray_scalable_services' service menu (interactive mode)
| *   ^^    - Go to the service list menu (interactive mode)
| *   Q     - write out changes and exit the configurator
| *   x     - revert all changes and exit the configurator
| *   r     - refresh the screen
| *   @     - toggle more/less info
| *   ?     - show this help
|
| Press Enter/return to take default action (current: <cr> - accept), OR enter a new value at prompt
```

Configuration – Configurator UI – setting

```
******** cray_scalable_services.settings.scalable_service.data.tier1 *********
    tier1 -- Tier1 servers
      The tier1 servers must have a direct network connection to the Server of Authority (SoA).
      On Cray CLE systems, the boot node must be specified as a tier1 server.  The SDB should
      also be specified assuming it is properly configured to reach the SMW.  Adding additional
      tier1 servers provides enhanced resiliency.  Log  traffic from the CLE portion of the system
      to the SMW will be spread across the tier1 nodes providing additional load capacity.
      Default:          Current:
        (none)            1) c0-0c0s0n1
                          2) c0-0c1s0n1
  Value: list, blank values not allowed, regex=^c(\d+)-(\d+)c([0-2])s(\d[0-5]?)n([0-3])$
       level=required, state=unset
  Inputs: menu commands (? for help)
cray_scalable_services.settings.scalable_service.data.tier1
[<cr>=set 2 entries, +=add an entry, ?=help, @=less] $
```

Configuration – global config set services

cray_bootraid_config.yaml
cray_firewall_config.yaml
cray_global_net_config.yaml
cray_image_groups.yaml
cray_ipforward_config.yaml
cray_logging_config.yaml
cray_network_boot_packages_config.yaml
cray_time_config.yaml

Configuration – CLE config set services

cray_alps_config.yaml
cray_auth_config.yaml
cray_ccm_config.yaml
cray_cnat_config.yaml
cray_dvs_config.yaml
cray_dws_config.yaml
cray_firewall_config.yaml
cray_image_binding_config.yaml
cray_ipforward_config.yaml
cray_liveupdates_config.yaml
cray_lmt_config.yaml
cray_lnet_config.yaml
cray_local_users_config.yaml
cray_logging_config.yaml
cray_login_config.yaml
cray_lustre_client_config.yaml
cray_multipath_config.yaml
cray_munge_config.yaml
cray_net_config.yaml
cray_network_boot_packages_config.yaml
cray_node_health_config.yaml
cray_persistent_data_config.yaml
cray_rsip_config.yaml
cray_rur_config.yaml
cray_scalable_services_config.yaml
cray_sdb_config.yaml
cray_service_node_config.yaml
cray_simple_shares_config.yaml
cray_simple_sync_config.yaml
cray_ssh_config.yaml
cray_storage_config.yaml
cray_sysconfig_config.yaml
cray_sysenv_config.yaml
cray_time_config.yaml
cray_user_settings_config.yaml
cray_wlm_detect_config.yaml
cray_wlm_trans_config.yaml

Configuration – CLE/eLogin config set services

- **Only for eLogin**
  - cray_elogin_lnet_config.yaml
  - cray_elogin_networking_config.yaml
  - cray_eswrap_config.yaml
- **Shared between CLE and eLogin**
  - cray_auth_config.yaml
  - cray_local_users_config.yaml
  - cray_lustre_client_config.yaml
  - cray_net_config.yaml
  - cray_simple_sync_config.yaml
  - cray_ssh_config.yaml
  - cray_time_config.yaml
  - cray_user_settings_config.yaml

CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- Configuration
- **Booting**
- Reconfiguration
- Software update
- Summary
- Q & A

Booting

- What is new with booting?
- Run simple jobs
- Troubleshoot a boot
- Dumping Cray XC system

Booting – What is New?

- **Boot the system**

  crayadm@smw> **xtbootsys -a auto.pluto**

- **What is new with booting?**
  - boot manager interacts with nimsd for boot images
  - xtbootsys extracts debugging information from all boot images
  - xtbootsys sets up mappings for the config set being used
  - Boot automation files should avoid strict boot ordering of service nodes

Booting – Run simple jobs

- **Check status on nodes**

  crayadm@login> **xtprocadmin**

  crayadm@login> **xtnodestat**

  crayadm@login> **apstat -v**

- **Test basic aprun functionality**

  crayadm@login> **NUMNODES=$(($(apstat -v | grep XT | awk "{print \$3}")));**
  echo NUMNODES is $NUMNODES

  crayadm@login> **cd /tmp; aprun -b -n $NUMNODES -N 1 /bin/cat /proc/sys/kernel/hostname**

  crayadm@login> **aprun -n $NUMNODES -N2 python -c "print 'hello world.'"**

Troubleshoot A Boot

- **Dealing with systemd (SLES 12)**
  - `systemd` forgoes traditional logging mechanisms and stores the following messages in a custom database
    - `syslogd` messages
    - Kernel log messages
    - Initial ram and early boot messages
    - Messages written to stdout/stderr for all services
  - `journalctl` is used to access this information
    - '`journalctl -a`' displays all kernel messages and other available information
    - '`journalctl -f`' behaves like '`tail -f`', displaying updates as they happen

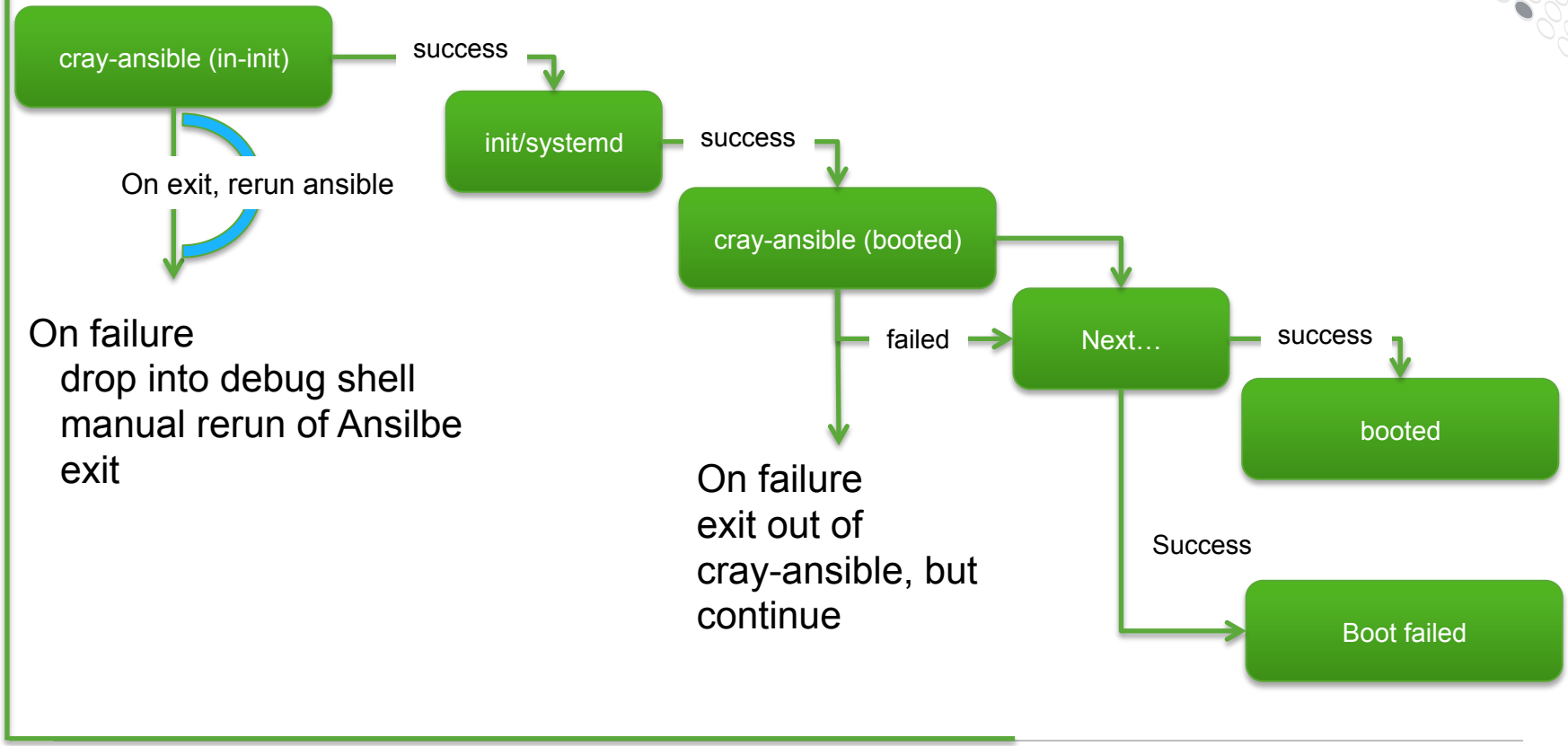
Troubleshoot A Boot

- **Boot automation file starts nodes in a certain order**
  - boot, sdb, service, compute
  - Additional actions can run commands on certain nodes
- **Logs on the SMW in /var/opt/cray/log**
  - HSS daemons and rsyslogd daemon running on the SMW will log to files in this directory
    - nimsd, xtpmd, xtremoted, xtpowerd, xtsnmpd, xtdiagd, erfsd, state_manager, bootmanager, sedc_manager, nid_mgr, erdh, erd
- **The output from booting CLE will be /var/opt/cray/log/p0-current**
  - Several files with more detailed and specific information
    - bootinfo.* file is the output from running xtbootsys
      - New: timing information for how long sections of the boot process take
    - console-YYYYMMDD is the combined console output from every node
- **Commands on SMW logged in /var/opt/cray/log/commands**

Troubleshoot A Boot

- **Detect Ansible failure in /init for a node**
  - Check the console log for the node
    cray-ansible: /etc/ansible/site.yaml completed in init – FAILED.
- **Debugging Ansible failures in /init for a node**
  - Ansible failures in init always cause node to drop into debug shell
  - Debug shell can be accessed via xtcon from SMW
    smw# **xtcon c0-0c0s8n3**
    nid00035#
  - Inspect ansible log, change config set, or other corrective action
  - Exiting from debug shell will cause cray-ansible to run again
    nid00035# **exit**
  - Boot will not proceed for this node until cray-ansible in init succeeds

Troubleshoot A Boot – boot flow and exits



Troubleshoot A Boot

- **Full Ansible logs too verbose to send to SMW for each node so inspect on the node**
- **Does ssh succeed to login to the node?**
  - Look at Ansible logs
  - Change config set data and rerun cray-ansible
- **Does ssh fail to login to the node?**
  - Try connecting with xtcon

    smw# **xtcon c0-0c0s8n3**
  - Look at Ansible logs
  - Change config set data and rerun cray-ansible
- **Does xtcon fail to login to the node?**
  - Try rebooting the node (with a warm boot)
    - Set `DEBUG=true` in kernel parameters in NIMS for the node
    - Reboot node, connect to console with xtcon, step through /init
      - Look at config set, network interface, etc.
      - Check Ansible logs
      - Change config set data and rerun cray-ansible

Troubleshoot A Boot

- **Once on the node, logs are in /var/opt/cray/log/ansible**
  - First (init) phase
    - sitelog-init has Ansible play output
    - file-changelog-init shows each file changed by an Ansible play
    - file-changelog-init.yaml shows each file changed by an Ansible play in YAML
  - Second (booted) phase
    - sitelog-booted has Ansible play output
    - file-changelog-booted shows each file changed by an Ansible play
    - file-changelog-booted.yaml shows each file changed by an Ansible play in YAML
- **Ansible writes changelogs for most files changed by the Ansible modules affecting files**
  - acl, assemble, blockinfile, copy, fetch, file, find, ini_file, lineinfile, patch, replace, stat, synchronize, template, unarchive, xattr
  - See http://docs.ansible.com/ansible/list_of_files_modules.html

Troubleshoot A Boot

- **sitelog files show output from each task in executed plays**

```
2016-01-17 12:15:27,671 TASK: [cle_motd | task motd, release]
*************************************
2016-01-17 12:15:27,671 changed: [localhost] => {"changed": true, "cmd": "grep RELEASE /etc/opt/
cray/release/cle-release | awk -F\\='{print $2}'", "delta": "0:00:00.002536", "end": "2016-01-17
12:15:27.471384", "rc": 0, "start": "2016-01-17 12:15:27.468848", "stderr": "", "stdout":
"6.0.UP01", "warnings": []}
```

- **Location of failing task can be found in plays**

```
boot# grep -Rn "task motd, release" /etc/ansible /etc/opt/cray/config/current/ansible
/etc/ansible/roles/cle_motd/tasks/motd.yaml:15:- name: task motd, release
```

- **file-changelog files show Ansible phase then each file and which play changed it**
  - file-changelog-init

```
Apr 05 2016 21:07:47 (init) template: file '/etc/nologin' changed by Ansible task file '/etc/ansible/roles/
early/tasks/nologin.yaml' with owner=root, group=root, mode=0775
```

  - file-changelog-booted

```
Apr 05 2016 16:09:43 (booted) lineinfile: file '/etc/sysconfig/nfs' changed by Ansible task file '/etc/ansible/
roles/fs_share/tasks/nfs_shares.yaml' with owner=None, group=None, mode=None
```

Dumping Cray XC System

- **xtdumpsys collects and analyzes information from a Cray XC system that is failing or has failed, has crashed, or is hung**
  - event log data, active heartbeat probing, voltages, temperatures, health faults, in-memory console buffers, and high-speed interconnection network errors
  - config sets from SMW
  - Ansible logs from nodes can be collected
  - Ansible changed files log from nodes can be collected
  - NIMS logs from SMW can be collected
- **cdump for a panicked or hung node**
  - Dumps node memory to a file
  - Analyzed with crash

CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- Configuration
- Booting
- **Reconfiguration**
- Software update
- Summary
- Q & A

Reconfiguration

- Configure for external Lustre server
- Update /etc/motd
- Install file with Simple Sync
- Configure for external NFS server
- Configure netroot images
- Configure Direct Attached Lustre (DAL)
- PE (Programming Environment) Install and Update
- Extend an Image Recipe
- Stage Changes in Snapshot

Reconfiguration – external Lustre server

- **Clone config set**

  smw# cfgset create --clone p0 p0beforeLustre

- **Configure LNet node**

  smw# cfgset update -s cray_lnet -l advanced -m interactive p0

- **Configure Lustre client**

  smw# cfgset update -s cray_lustre_client -m interactive p0

- **Update network configuration**

  smw# cfgset update -s cray_net -l advanced -m interactive p0

Reconfiguration – external Lustre server

- **Validate config set**

  smw# cfgset validate p0

- **Shutdown CLE**

  crayadm@smw> xtbootsys -s last -a auto.xtshutdown

- **Boot CLE**

  crayadm@smw> xtbootsys -a auto.pluto

- **Test simple job**

  crayadm@login> cd /lustre/crayadm; aprun -b –n 5 -N 1 /bin/ls

Reconfiguration - /etc/motd

- **Update /etc/motd to add custom message**

  smw# **cd /var/opt/cray/imps/config/sets/p0/files/roles**

  smw# **vi common/etc/motd**

  CUG tutorial

- **Content delivered during a boot, but you can also deliver it immediately to the nodes**

  boot# **cat /etc/motd**

  *** Welcome to IMPS service node c0-0c0s0n1 (nid 1) ***
  Running 1.6GB Suse 12 image service_cle_6.0up01_sles_12_x86-64_ari
  CLE release 6.0up01, build 6.0.68
  16 vcores, boot_freemem: 28868mb

  boot# **/etc/init.d/cray-ansible start**

  boot# **cat /etc/motd**

  *** Welcome to IMPS service node c0-0c0s0n1 (nid 1) ***
  Running 1.4GB Suse 12 image service_cle_6.0up01_sles_12_x86-64_ari
  CLE release 6.0up01, build build 6.0.79(201604010620)
  16 vcores, boot_freemem: 29045mb
  CUG tutorial

  login# **/etc/init.d/cray-ansible start**

Reconfiguration – Simple Sync v2

- **Simple Sync service provides a simple and easy to use mechanism for administrators to copy files onto their system without resorting to writing an Ansible play**
- **Files that are placed in the following directory structure will be copied onto nodes with matching criteria:**

  smw:/var/opt/cray/imps/config/sets/p0/files/simple_sync/

  ./common/files/ # matches all nodes
  ./platform/[compute, service]/files/ # matches compute or service nodes
  ./nodegroups/<node_group_name>/files/ # matches members of <node_group_name>
  ./hardwareid/<hardwareid>/files/ # matches nodes with matching hardware id
  ./hostname/<hostname>/files/ # matches nodes with hostname
- **Files, including directory structure below ./files/ will be replicated on the target node starting at /**

  smw:/var/opt/cray/imps/config/sets/p0/files/simple_sync/

  ./common/files/etc/myapplication.conf
  - will be placed on all nodes as /etc/myapplication.conf

Reconfiguration – Simple Sync v2

- **Change to simple sync directory for CLE config set**
  smw# **cd /var/opt/cray/imps/config/sets/p0/files/simple_sync**
- **Make a file for all nodes**
  smw# **touch common/files/pluto.common**
- **Make a file for all service nodes**
  smw# **touch platform/service/files/pluto.service**
- **Make a file for all compute nodes**
  smw# **touch platform/compute/files/pluto.compute**
- **Make a file for nodegroup called testgroup**
  smw# **touch nodegroup/testgroup/files/pluto.testgroup**
- **Make a file for node c0-0c0s3n2**
  smw# **touch hardwareid/c0-0c0s3n2/files/pluto.c0-0c0s3n2**
- **Content will be delivered during a boot, but you can deliver it immediately to the nodes**
  boot# **/etc/init.d/cray-ansible start**
  sdb# **/etc/init.d/cray-ansible start**
  sdb# **pcmd -r -n ALL_NODES_NOT_ME "/etc/init.d/cray-ansible start"**
  - ALL_NODES, ALL_COMPUTE, ALL_SERVICE, ALL_SERVICE_NOT_ME

Reconfiguration – Simple Sync v1 (eLogin UP01)

- **Files in these config set locations are copied to the target eLogin nodes**

  smw:/var/opt/cray/imps/config/sets/p0/files/roles/simple_sync
  ./classes/common/ # matches all nodes
  ./hostnames/<host_name>/ # matches specific hostname

- **Files, including directory structure below classes/common/ or hostnames/ will be replicated on the target node starting at /**

  smw:/var/opt/cray/imps/config/sets/p0/files/roles/simple_sync/
  ./classes/common/etc/myapplication.conf
  - will be placed on all eLogin nodes as /etc/myapplication.conf

  smw:/var/opt/cray/imps/config/sets/p0/files/roles/simple_sync/
  ./hostnames/elogin2/tmp/i_am_elogin2
  - will be placed on only eLogin node elogin2 as /tmp/i_am_elogin2

Reconfiguration – external NFS server

- **Configure for external NFS server**
  - Update network configuration on DVS node for network interface
    smw# **cfgset update -S all -s cray_net -l advanced -m interactive p0**
  - Configure DVS node to external NFS server
    smw# **cfgset update --state all --service cray_dvs -l advanced p0**
  - Configure LDAP
    smw# **cfgset update -S all -s cray_auth --level advanced p0**
  - Configure automount files on DVS node with Simple Sync
    smw# **cd /var/opt/cray/imps/config/sets/p0/files/simple_sync**
    smw# **mkdir –p hardwareid/c0-0c0s0n2/files/etc/auto.master.d**
    smw# **cd hardwareid/c0-0c0s0n2/files/etc**
    smw# **cp -p /home/crayadm/etc/auto.css .**
    smw# **cp -p /home/crayadm/etc/auto.master.d/css.autofs auto.master.d**
  - Validate config set
    smw# **cfgset validate p0**

Reconfiguration – netroot images

- **Matched pair of images**
  - Compute nodes
    - initrd-compute-large
    - compute-large
  - Login nodes
    - initrd-login-large
    - login-large
- **For compute netroot (similar for login netroot)**
  - NIMS boot image is set to the "initrd-compute-large” image
  - NIMS kernel parameter is set to "netroot=compute-large”
  - compute-large image root needs to be pushed to the boot node

Reconfiguration – netroot images

- **Add netroot images to cray_image_groups**
  - smw# **vi /var/opt/cray/imps/config/sets/global/config/cray_image_groups.yaml**

```
- recipe: "initrd-compute-large_cle_6.0up01_sles_12_x86-64_ari”
  dest: "initrd-compute-large{note}_cle_{cle_release}-build{cle_build}{patch}_sles_12-created{date}.cpio”
  nims_group: "netroot_compute”
- recipe: "initrd-login-large_cle_6.0up01_sles_12_x86-64_ari”
  dest: "initrd-login-large{note}_cle_{cle_release}-build{cle_build}{patch}_sles_12-created{date}.cpio”
  nims_group: "netroot_login"
```

- **Create netroot groups in NIMS**
  - Add compute nodes to group “netroot_compute”
    - smw# **cnode update --partition p0 -G compute -g netroot_compute --filter group=compute**
  - Add login nodes to group “netroot_login”
    - smw# **cnode update --partition p0 -G login -g netroot_login --filter group=login**

Reconfiguration – netroot images

- **Create new images (will assign to new NIMS groups)**
  smw# **imgbuilder --map**
- **Push netroot image to boot node**
  smw# **image push -d boot compute-large_cle_6.0.UP01-build6.0.68_sles_12-createdYYYYMMDD**
  smw# **image push -d boot login-large_cle_6.0.UP01-build6.0.68_sles_12-createdYYYYMMDD**
- **Reboot nodes with netroot image**
  - Test boot image on single compute node with a warm boot
    smw# **xtcli shutdown c0-0c0s15n3**
    smw# **xtnmi c0-0c0s15n3**
    crayadm@smw> **xtbootsys --reboot -r "testing netroot" c0-0c0s15n3**
  - Test boot image on single login node with a warm boot
    smw# **xtnmi c0-0c0s0n2**
    crayadm@smw> **xtbootsys --reboot -r "testing netroot" c0-0c0s0n2**
  - Next full system reboot will use the new image for all nodes with netroot
    crayadm@smw> **xtbootsys -a auto.pluto**

Reconfiguration – DAL

- **On DAL node, identify LUNs for Lustre**
- **Prepare Lustre fs_defs file**

  smw# **cp /opt/cray-xt-lustre-utils/default/etc/example.fs_defs /home/crayadm/dal.fs_defs**
  - Add the disk device names (LUNs) for MGT, MDT, OST
  - Tune any other Lustre settings
- **Install Lustre fs_defs file to CLE config set**

  smw# **lustre_control install -c p0 /home/crayadm/dal.fs_defs**
- **Reformat DAL filesystem (on boot node)**

  boot# **/etc/init.d/cray-ansible start**

  boot# **lustre_control reformat -f dal**
- **Mount DAL Lustre targets on DAL nodes**

  boot# **ssh nid00009 mount -t lustre**

  boot# **ssh nid00010 mount -t lustre**
- **Mount DAL filesystem on login node as test**

  login# **mkdir -p /lus/dal**

  login# **mount -t lustre 9@gni1:/dal /lus/dal**

Reconfiguration – DAL

- **Configure Lustre server for DAL**
  smw# **cfgset update -s cray_lustre_server -m interactive p0**
- **Configure Lustre client for DAL**
  smw# **cfgset update -s cray_lustre_client -m interactive p0**
- **Configure LMT (Lustre Monitoring Tool)**
  smw# **cfgset update -s cray_lmt -m interactive p0**
- **Validate config set**
  smw# **cfgset validate p0**
- **Update boot automation file for DAL**
  smw# **vi /opt/cray/hss/default/etc/auto.rhine**
  - After booting service nodes and before booting compute nodes
    - Start Lustre server on DAL nodes

lappend actions { crms_exec_on_bootnode "root" “lustre_control start -f dal" }

    - Mount Lustre filesystem on login nodes

lappend actions { crms_exec_on_bootnode "root" “lustre_control mount_clients -f dal -w login[1-8]" }

Reconfiguration – PE Fresh Install

- Clone compute image to PE image
- Mount PE ISO
- Install craype-installer rpm
- Configure PE installer YAML
- Install PE software
- Push PE image to Boot Node
- Update config set to include name of PE image
- Validate config set
- Reboot CLE system

Reconfiguration – PE Fresh Install

- **Clone compute image root to PE image root**
  smw# PEIMAGE=pe_compute_cle_6.0up01_sles_12
  smw# image create -r pe_image_cle_6.0up01_sles_12 $PEIMAGE
- **Mount PE ISO**
  smw# mkdir -p mount_iso logs
  smw# mount -o loop,ro CDT-15.01-51.dev.iso ./mount_iso
- **Install craype-installer rpm**
  smw# rpm -ivh craype-installer-1.10.00-11.x86_64.rpm
- **Configure PE installer YAML**
  smw# cp /opt/cray/craype-installer/1.10.00/conf/install-cdt.yaml .
  smw# vi install-cdt.yaml

```
IMAGE_DIRECTORIES :
  - /var/opt/cray/imps/image_roots/pe_compute_cle_6.0up01_sles_12
LOGS_DIR                : ./logs
ISO_MOUNT_DIR           : ./mount_iso
```

Reconfiguration – PE Fresh Install

- **Install PE software**

  smw# **module load craype-installer**

  smw# **craype-installer.pl --install --install-yaml-path ./install-cdt.yaml**

- **Push PE image to Boot Node**

  smw# **image push --dest boot $PEIMAGE**

  INFO - Remotely cloning Image '<name of image>' to 'boot'...

  INFO - Checking remote destination...

  INFO - Passwordless SSH not established; prompting for password for root@boot:

  Password:

  INFO - Transferring Image '<name of image>' to 'root@boot:/var/opt/cray/imps/image_roots/<name of image>'...

  Password:

  INFO - Cloned Image '<name of image>' to remote host 'root@boot:/var/opt/cray/imps/image_roots/<name of image>'.

Reconfiguration – PE Fresh Install

- **Update Config Set to include name of PE image**

  smw# **cfgset update -s cray_image_binding -m interactive p0**

- **Validate config set**

  smw# **cfgset validate p0**

- **Reboot CLE system with PE**

  smw# su - crayadm

  crayadm@smw> xtbootsys -s last -a auto.xtshutdown

  crayadm@smw> xtbootsys -a auto.rhine

Reconfiguration – PE Update

- Reuse previous PE image root
- Mount PE ISO
- Update craype-installer rpm
- Reuse PE installer YAML
- Install PE software
- Push PE image to Boot Node
- Restart Image Binding service

Reconfiguration – PE Update

- **Reuse previous PE image root**
  smw# PEIMAGE=pe_compute_cle_6.0up01_sles_12
- **Mount PE ISO**
  smw# mkdir -p mount_iso logs
  smw# mount -o loop,ro CDT-15.01-51.dev.iso ./mount_iso
- **Update craype-installer rpm (if needed)**
  smw# rpm -uvh craype-installer-1.10.00-11.x86_64.rpm
- **Reuse PE installer YAML**
- **Install PE software**
  smw# module load craype-installer
  smw# craype-installer.pl --install --install-yaml-path ./install-cdt.yaml

Reconfiguration – PE Update

- **Push PE image to Boot Node**

  smw# **image push --dest boot $PEIMAGE**

  INFO - Remotely cloning Image '<name of image>' to 'boot'...

  INFO - Checking remote destination...

  INFO - Passwordless SSH not established; prompting for password for root@boot:

  Password:

  INFO - Transferring Image '<name of image>' to 'root@boot:/var/opt/cray/imps/image_roots/<name of image>'...

  Password:

  INFO - Cloned Image '<name of image>' to remote host 'root@boot:/var/opt/cray/imps/image_roots/<name of image>'.

- **Restart Image Binding service by running cray-ansible again**

  sdb# **pcmd -r -n ALL_COMPUTE "/etc/init.d/cray-ansible start"**

  login# **/etc/init.d/cray-ansible start**

Reconfiguration – Extend an Image Recipe

- Create or clone repository
- Create or clone package collection
- Create or clone image recipe
- Add post-build actions
- Validate image recipe
- Build image recipe
- Show build history of image recipe
- Package boot image
- Test boot image on single node
- Deploy boot image to whole machine

Reconfiguration – Extend an Image Recipe

repo create	Create new empty repositories or clone existing repositories
repo diff	Show changes between repositories
repo list	List repositories and associated metadata
repo pull	Copies new packages from a repository path on the local filesystem to a named repository
repo push	Copies the contents of a repository to a local or remote destination path
repo remove	Removes repositories
repo show	Prints detailed information about a repository
repo update	Modify the attributes or contents of a repository
repo validate	Checks that a repository is valid

Reconfiguration – Extend an Image Recipe

- **Create repository called my_sles12_repo**
  - New repository must have arch and type

  smw# **repo create --arch x86-64 --type SLES12 my_sles12_repo**
  - Clone existing repository gets arch and type from source repository

  smw# **repo create --clone sles_12_x86-64 my_cloned_sles12_repo**
  - If new repository exists, overwrite it

  smw# **repo create --arch x86-64 --type SLES12 --force my_sles12_repo**
- **Verify new repository was created**

  smw# **repo list my***

  my_sles12_repo

Reconfiguration – Extend an Image Recipe

- **Show information about the repository**

```
smw# repo show my_sles12_repo
my_sles12_repo:
  name: my_sles12_repo
  arch: x86-64
  created: 2016-02-19T05:11:42
  path: /var/opt/cray/repos/my_sles12_repo
  type: SLES12
```

- **Add rpms to repository**

```
smw# repo update -a "/path/to/RPMs/*.rpm" my_sles12_repo
smw# ls -l /var/opt/cray/repos/my_repo
-rw-r--r-- 1 crayadm crayadm   485137 Nov 23 08:56 myrpm-11.13.1.1-4.x86_64.rpm
```

- **Validate the repository**

```
smw# repo validate my_sles12_repo
```

Reconfiguration – Extend an Image Recipe

pkgcoll create	Create new empty package collections or clone existing package collections
pkgcoll list	List package collections and associated metadata
pkgcoll remove	Removes package collections
pkgcoll show	Prints detailed information about a package collection
pkgcoll update	Modify the attributes or contents of a package collection
pkgcoll validate	Checks that a package collection is valid

Reconfiguration – Extend an Image Recipe

- **Create package collection**
  - Create new empty package collection named 'my_collection'.

  smw# **pkgcoll create my_collection**

  smw# **pkgcoll create myusertools**
  - Clone from existing package collection

  smw# **pkgcoll create --clone cle_sles_12_service my_collection**
- **Verify new package collection was created**

  smw# **pkgcoll list my***

  my_collection

  myusertools
- **Add rpms to package collection**

  smw# **pkgcoll update -p favorite -p otherrpm my_collection**

Reconfiguration – Extend an Image Recipe

- **Add another package collection to package collection**

  smw# **pkgcoll update -c myusertools -c cle_6.0up01_centos_6.5_base my_collection**

- **Show information about the package collection**

  smw# **pkgcoll show my_collection**

```
my_collection:
  name: my_collection
  package_collections:
    cle_6.0up01_centos_6.5_base
    myusertools
  packages:
    favorite
    otherrpm
```

Reconfiguration – Extend an Image Recipe

- **Remove rpm from package collection**
  smw# **pkgcoll update -P otherrpm my_collection**
- **Remove package collection from my package collection**
  smw# **pkgcoll update -C cle_6.0up01_centos_6.5_base** my_collection
- **Verify removals**
  smw# **pkgcoll show my_collection**
  my_collection:
    name: my_collection
    package_collections:
      myusertools
    packages:
      favorite

Reconfiguration – Extend an Image Recipe

recipe create	Create new empty recipes or clone existing recipes
recipe list	List recipes and associated metadata
recipe remove	Removes recipes
recipe show	Prints detailed information about a recipe
recipe update	Modify the attributes or contents of a recipe
recipe validate	Checks that a recipe is valid

Reconfiguration – Extend an Image Recipe

- **List existing recipes**

```
smw# recipe list
        compute-large_cle_6.0up01_sles_12_x86-64_ari
        compute_cle_6.0up01_sles_12_x86-64_ari
        dal_cle_6.0up01_centos_6.5_x86-64_ari
        elogin_cle_6.0up01_sles_12_x86-64_ari
        initrd-compute-large_cle_6.0up01_sles_12_x86-64_ari
        initrd-login-large_cle_6.0up01_sles_12_x86-64_ari
        login-large_cle_6.0up01_sles_12_x86-64_ari
        login_cle_6.0up01_sles_12_x86-64_ari
        service_cle_6.0up01_sles_12_x86-64_ari
```

Reconfiguration – Extend an Image Recipe

- **Clone an existing image recipe to new recipe custom_compute_cle**

  smw# **recipe create --clone compute_cle_6.0up01_sles_12_x86-64_ari custom_compute_cle**

  INFO - Locally cloning Recipe 'compute_cle_6.0up01_sles_12_x86-64_ari' to 'custom_compute_cle'.

  INFO - Successfully created new empty Recipe 'custom_compute_cle'.

  INFO - Successfully cloned to Recipe 'custom_compute_cle'.

- **Create a new image recipe for custom_compute_cle with recursive recipe**

  smw# **recipe create custom_compute_cle**

  - Add recursive recipe to new recipe (--add-recipe)

    smw# **recipe update -i compute_cle_6.0up01_sles_12_x86-64_ari custom_compute_cle**

Reconfiguration – Extend an Image Recipe

- **Extend image recipe with three additional rpms**
  smw# recipe update -p package1 package2 package3 custom_compute_cle
- **Extend image recipe to remove two rpms**
  smw# recipe update -P badrpm1 badrpm2 custom_compute_cle
- **Extend image recipe with two package collections**
  smw# recipe update -c my_collection bad_compute_packages custom_compute_cle
- **Extend image recipe to remove one package collection**
  smw# recipe update -C bad_compute_packages custom_compute_cle
- **Extend image recipe with subrecipe**
  smw# recipe update -i my_sles12_recipe custom_compute_cle
- **Extend image recipe to remove subrecipe**
  smw# recipe update -I bad_recipe custom_compute_cle
- **Extend image recipe with new repo**
  smw# recipe update -r my_sles12_repo custom_compute_cle
- **Extend image recipe to remove repo**
  smw# recipe update -R bad_repo custom_compute_cle

Reconfiguration – Extend an Image Recipe

- **Add post-build actions**
  - The JSON file containing the recipe needs to be edited
  - Locate the image recipe definition in the IMPS image recipe local edits file
    - /etc/opt/cray/imps/image_recipes.d/image_recipes.local.json
  - Add the postbuild_copy and/or postbuild_chroot sections to your image recipe

Reconfiguration – Extend an Image Recipe

```
"custom_compute_cle": {
        ...
        "package_collections": { ... },
        "packages": { ... },
        ”recipes": { ... },
        "postbuild_copy": [
            "/file/1",
            ...
            "/dir/2/content"
        ],
        "postbuild_chroot": [
            "chroot_command1",
            ...
            "chroot_commandN"
        ],
        "repositories": { ... }
    },
```

Reconfiguration – Extend an Image Recipe

- **Validate image recipe**
  - Ensure that the JSON syntax of the image recipe is correct

```
smw# recipe validate custom_compute_cle
INFO - Repository 'custom_repo' validates.
INFO - Recipe 'custom_compute_cle' is valid.
```

- **The validate command will also validate all repositories and package collections referenced by your image recipe and will ensure that it can access any files in the *postbuild_copy* section**

Reconfiguration – Extend an Image Recipe

image create	Create new image from recipe or clone existing image
image diff	Show changes between two images
image export	Export images to various formats default is cpio (for CLE) qcow2 for eLogin
image list	List images and associated metadata
image push	Copy contents of image to local or remote path
image remove	Removes images
image show	Prints detailed information about an image
image validate	Checks that an image is valid

Reconfiguration – Extend an Image Recipe

- **Build image recipe to create image root**
  - **image** builds the image recipe starting with the package manager installation and then proceeds to step through the postbuild copy and chroot commands (in that order)

```
smw# image create -r custom_compute_cle custom_compute_cle
INFO - Repository 'custom_repo' validates.
INFO - Recipe 'custom_compute_cle' is valid for building.
INFO - Calling Package manager to build new image root; this will take a few minutes.
INFO - Rebuilding RPM database for Image 'custom_compute_cle'.
INFO - RPM database does not need to be rebuilt.
INFO - Running post-build scripts for Image 'custom_compute_cle'.
INFO - Copying postbuild files to /tmp/tmpmAyzGl in Image 'custom_compute_cle'
INFO -          * Executing post-build chroot script: 'chroot_command1'
INFO - post-build chroot script output will be located in /tmp/custom_compute_cle-postbuild_out_20150713-15:55:11g4WA6p
INFO - Build of Recipe 'custom_compute_cle' has completed successfully.
```

Reconfiguration – Extend an Image Recipe

- **See build history of image root**

```
smw# image show custom_compute_cle
 custom_compute_cle:
        name: custom_compute_cle
        created: 2015-07-13T15:54:06
        history:
            2015-07-13T15:55:16: Successful build of Recipe
              'custom_compute_cle' into Image 'custom_compute_cle'.
            2015-07-13T15:55:17: Successful rebuild of RPM database.
         path: /var/opt/cray/imps/image_roots/custom_compute_cle
```

Reconfiguration – Extend an Image Recipe

- **Package image root into CLE boot image**

  smw# image export custom_compute_cle

  INFO - Copying kernel /var/opt/cray/imps/image_roots/custom_compute_cle/boot/bzImage-3.12.28-4.6_1.0000.8685-cray_ari_c into /tmp/temp_tempfs_50LJ93/DEFAULT

  INFO - Copying parameters file /var/opt/cray/imps/image_roots/custom_compute_cle/boot/parameters-ari_c into /tmp/temp_tempfs_50LJ93/DEFAULT

  INFO - Copying directory /var/opt/cray/imps/image_roots/custom_compute_cle/lib/modules/3.12.28-4.6_1.0000.8685-cray_ari_c into /tmp/temp_tempfs_50LJ93/DEFAULT/debug

  INFO - Copying in debug files /var/opt/cray/imps/image_roots/custom_compute_cle/boot/System.map-3.12.28-4.6_1.0000.8685-cray_ari_c, /var/opt/cray/imps/image_roots/custom_compute_cle/boot/vmlinux-3.12.28-4.6_1.0000.8685-cray_ari_c, /var/opt/cray/imps/image_roots/custom_compute_cle/boot/vmlinux into /tmp/temp_tempfs_50LJ93/DEFAULT/debug/boot

  INFO - Writing package information file /tmp/temp_tempfs_50LJ93/DEFAULT/package.info

  INFO - Packaging up image root /var/opt/cray/imps/image_roots/custom_compute_cle into /tmp/temp_tempfs_50LJ93/DEFAULT/initramfs

  INFO - Gzipping initramfs file /tmp/temp_tempfs_50LJ93/DEFAULT/initramfs

  INFO - Creating size-initramfs file

  INFO - Creating loadfile /tmp/temp_tempfs_50LJ93/DEFAULT.load

  INFO - Creating symlinks of Image 'custom_compute_cle' to DEFAULT

  INFO - Creating boot cpio /var/opt/cray/imps/boot_images/custom_compute_cle.cpio

  INFO - Image 'custom_compute_cle' has been packaged into /var/opt/cray/imps/boot_images/custom_compute_cle.cpio.

Reconfiguration – Extend an Image Recipe

- **Compare two images**

smw# **image diff service_cle_6.0.UP01-build6.0.84_sles_12-created20160428 service_cle_6.0.UP01-build6.0.90_sles_12-created20160429**

```
INFO - Calculating RPM differences between Image 'service_cle_6.0.UP01-build6.0.84_sles_12-created20160428'
   and 'service_cle_6.0.UP01-build6.0.90_sles_12-created20160429'...
WARNING - Cannot query installed RPMs, falling back to disk based diff.
INFO - Calculating path entry differences..
INFO - Unique entries to IMPS Image 'service_cle_6.0.UP01-build6.0.84_sles_12-created20160428' (first 10 of 1060):
      - etc/opt/cray/lustre-utils/collect_perf_stats.py
      - etc/opt/cray/lustre-utils/late_accsm.sh
      - etc/opt/cray/lustre-utils/late_perf.sh
      - opt/cray/alps/6.1.3-14.11/bin/apbasil
Unique entries to IMPS Image 'service_cle_6.0.UP01-build6.0.90_sles_12-created20160429' (first 10 of 1140):
      - etc/ansible/aeld.yaml
      - etc/ansible/apptermd.yaml
      - etc/ansible/ncmd.yaml
      - etc/ansible/roles/compute_node/tasks/xthostname.yaml
      - etc/ld.so.conf.d/cray-alpscomm_sn.conf
INFO - None
```

Reconfiguration – Extend an Image Recipe

- **Assign new boot image to a single compute node**

  smw# **cnode update -i /var/opt/cray/imps/boot_images/custom_compute_cle.cpio c0-0c0s15n3**

- **Test boot image on single node with a warm boot**

  smw# **xtcli shutdown c0-0c0s15n3**

  crayadm@smw> **xtbootsys --reboot -r "testing custom compute image" c0-0c0s15n3**

Reconfiguration – Extend an Image Recipe

- **Assign new image to all compute nodes**

  smw# **cnode update -i /var/opt/cray/imps/boot_images/custom_compute_cle.cpio --group compute**

- **Warm boot all compute nodes in c0-0 with new image**

  smw# **COMPUTENODES=$(xtcli status s0 | egrep -v "empty|service|disabled" | grep c0-0 | awk '{ FS=":"; print $1 }' | tr ':' ' ' | awk '{ printf "%s,", $1 }' | sed s'/.$//')**

  smw# **xtcli shutdown $COMPUTENODES**

  crayadm@smw> **xtbootsys --reboot -r "Booting custom compute image on all compute nodes in c0-0" $COMPUTENODES**

- **Next full system reboot will use the new image for all compute nodes**

  crayadm@smw> **xtbootsys -a auto.pluto**

Reconfiguration – Stage Changes in Snapshot

- **Change to snapshot for reconfiguration**

```
smw# snaputil chroot mysnapshot
Changing root to: /media/root-sv/snapshots/mysnapshot.
btrfs_object_id: 275
kernel: vmlinuz-3.12.48-52.27-default
smwha_version: None
name: mysnapshot
parent: smw-8.0.66_cle-6.0.66.20151004
updated: 2015-12-11 08:31:33.024004
initrd: initrd-3.12.48-52.27-default
cle_version: 6.0.20
path: /media/root-sv/snapshots/mysnapshot
smw_version: 8.0.20
storage_set: smwdefault
(mysnapshot) -> root@smw:/ #
```

Reconfiguration – Stage Changes in Snapshot

- **Load modules**
  (mysnapshot) -> root@smw:/ # **module load imps**
  (mysnapshot) -> root@smw:/ # **module load install-support**
- **Build new images from new repo content**
  (mysnapshot) -> root@smw:/ # **imgbuilder -map**
- **Update config sets**
  (mysnapshot) -> root@smw:/ # **cfgset update --no-scripts p0**
  (mysnapshot) -> root@smw:/ # **cfgset validate p0**
  (mysnapshot) -> root@smw:/ # **cfgset update --no-scripts global**
  (mysnapshot) -> root@smw:/ # **cfgset validate global**
- **Leave chroot**
  (mysnapshot) -> root@smw:/ # **exit**
  smw#

Reconfiguration – Stage Changes in Snapshot

- **When ready, switch SMW to new snapshot**

  smw# **snaputil default mysnapshot**

  crayadm@smw> **xtbootsys -s last -a auto.xtshutdown**

  smw# **reboot**
- **After SMW reboot, update config sets with pre/post-scripts**

  smw# **cfgset update global**

  smw# **cfgset validate global**

  smw# **cfgset update p0**

  smw# **cfgset validate p0**
- **Boot CLE**

  crayadm@smw> **xtbootsys -a auto.pluto**

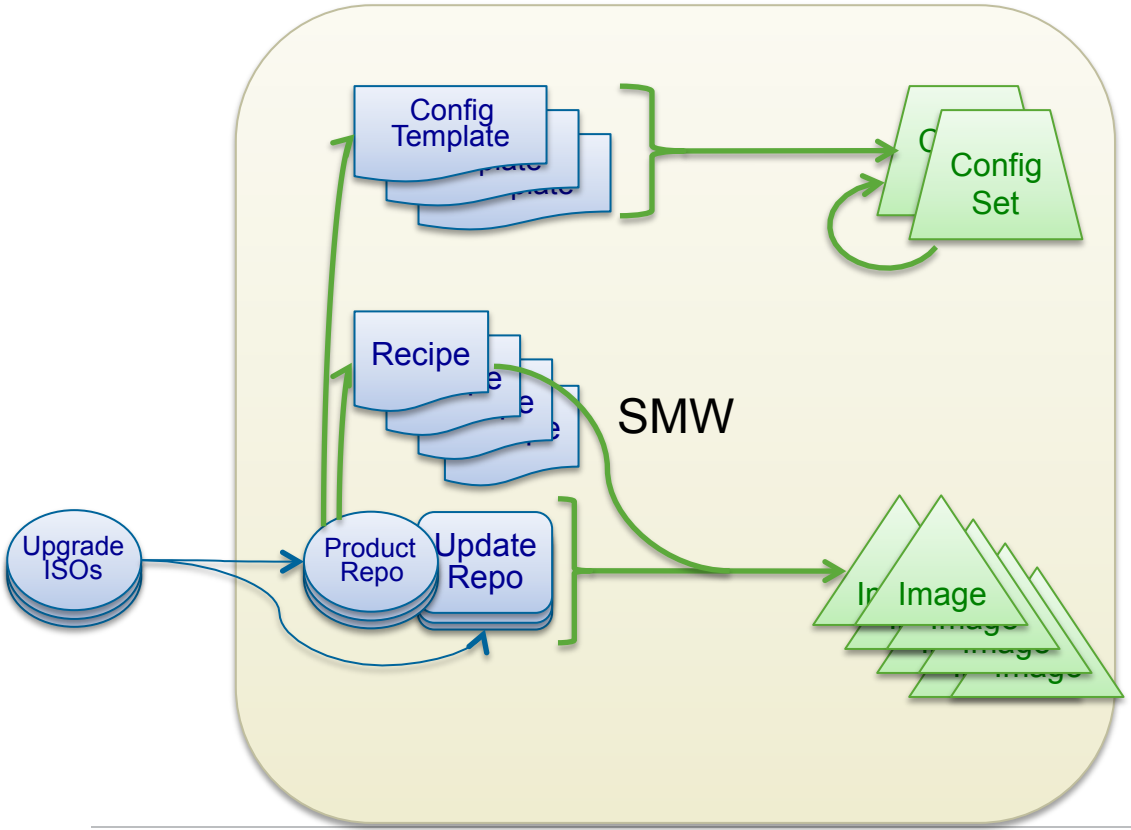
CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- Configuration
- Booting
- Reconfiguration
- **Software update**
- Summary
- Q & A

Software Update

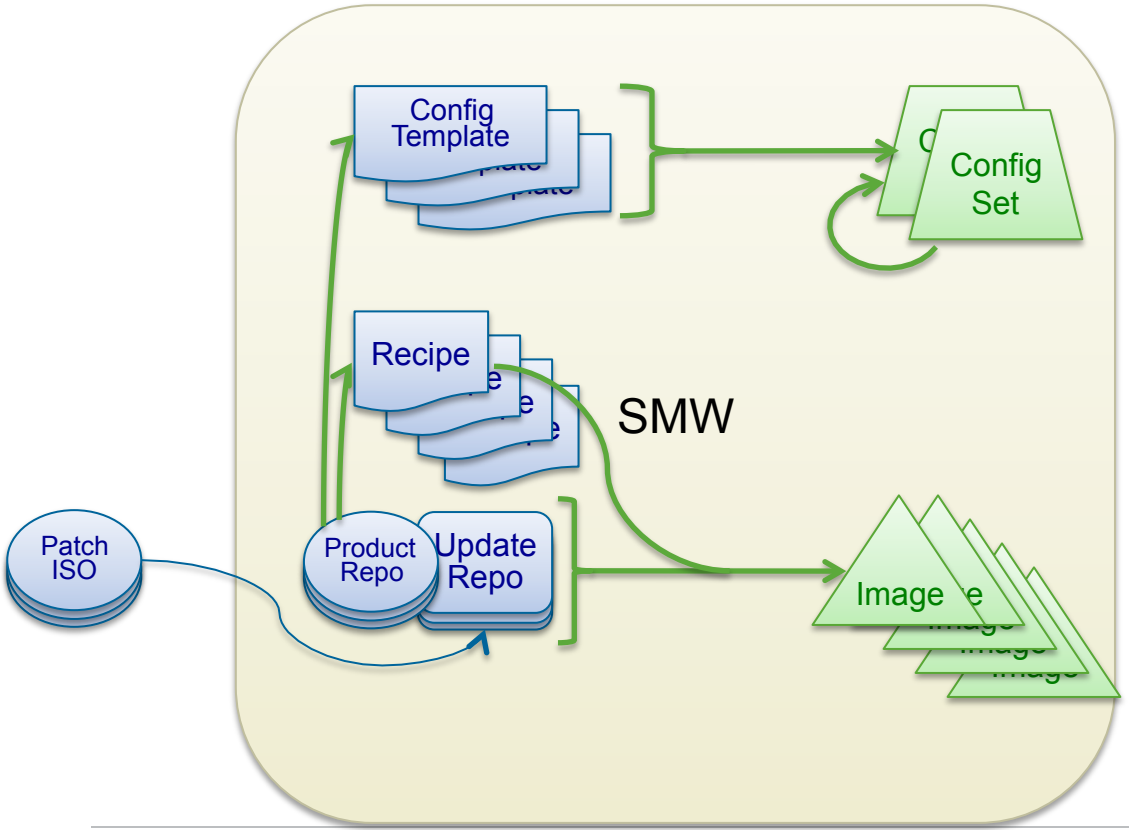
- Staged Upgrades
- Software Patches
- Security Updates
- Live Update
- Rolling Patches

Staged Upgrades



- **Focus:**
  - Minimal downtime
  - Safe: rollback possible
- **Snapshots and Versions**
  - SMW root gets btrfs snapshot
  - CLE objects are all explicitly versioned
- **While running current system in production…**
  - Update repositories
  - Apply new software to snapshots
  - Build new CLE images
  - Apply new configuration to config sets
- **Switch to new release**
  - Shut down system
  - Update firmware
  - Refresh snapshots config sets where necessary
  - Boot new system
- **Revert if necessary**

Software Patches



- **Safe**
  - SMW root can get btrfs snapshot
  - CLE objects are explicitly versioned
  - Clone global and CLE config sets
- **Process:**
  - Patchset README, INSTALL files in /var/opt/cray/patchsets
  - Recorded in /etc/opt/cray/release/pkginfo
  - Apply rpms to update repositories
  - Rebuild images and/or update configuration
  - Stage for booting
  - Reboot or live update or rolling patches, depending on the changes
  - Rollback if necessary

Software Patches – service nodes

- **Comparision of update procedures for service nodes**
  - Live updates (no reboot needed)
    - ssh, user commands, SUSE rpms
  - Warm reboot of patched service nodes
    - GNI driver, RSIP, RCA driver
  - System reboot (required)
    - Patches that change protocols
    - Patches that reboot the boot node
    - IDS changes
  - System reboot (recommended)
    - Security patches
    - DataWarp

Security Updates – SLEupdate

SLEupdate [--target=NAME] [options]

--live-update	Do not stage the update in a btrfs snapshot
--forceupdate	Force installation of packages, even if versions match or we're asking to downgrade packages, which zypper won't do by default
--target=NAME	Install software into btrfs snapshot
--media=DIR	Path to installation media (defaults to current working directory)

Security Updates – Run SLEupdate

- **Mount SLE security media**
  smw# **mkdir -p /media/SLE**
  smw# **mount -o loop,ro sleupdate-image-201510141357.iso /media/SLE**
- **If going to use "--live-update"**
  - Make an archival snapshot before installing security update
    smw# **snaputil create ${SNAPSHOT}-preupgrade**
- **Install Security Updates**
  smw# **/media/SLE/SLEupdate --media=/media/SLE --target=${SNAPSHOT}**
  - If no target on command line, a snapshot will be created
  - Changes made in snapshot
- **Logs created in /var/adm/cray/logs/install/ install.YYYYMMDD.log**
  - Very verbose log file with all zypper/rpm messages

Security Updates – reboot with new security

- **If "SLEupdate --live-update" was not used**
  - Use new snapshot to rebuild images
    smw# **snaputil chroot $SNAPSHOT**
    snapshot smw# **imgbuilder –map**
    snapshot smw# **exit**
  - If using netroot, push new netroot image roots to boot node
    smw# **image push –d boot $NETROOTCOMPUTE**
    smw# **image push –d boot $NETROOTLOGIN**
  - Shutdown CLE before rebooting the SMW
    crayadm@smw> **xtbootsys -s last -a auto.xtshutdown**
  - Set SMW to boot from new snapshot
    smw# **snaputil default $SNAPSHOT**
  - Reboot SMW
    smw# **reboot**
  - Reboot CLE system
    crayadm@smw> **xtbootsys -a auto.pluto**

Security Updates – reboot with new security

- **If "SLEupdate --live-update" was used**
  - Rebuild images
    smw# **imgbuilder –map**
  - If using netroot, push image root to boot node
    smw# **image push –d boot $NETROOTCOMPUTE**
    smw# **image push –d boot $NETROOTLOGIN**
  - Reboot CLE system
    crayadm@smw> **xtbootsys -s last -a auto.xtshutdown**
    crayadm@smw> **exit**
    smw# **reboot**
    crayadm@smw> **xtbootsys -a auto.pluto**

Live Updates

- **CLE service and compute nodes have zypper or yum repositories which use scalable services as http servers for software repos**
  - tier1 servers (boot node and SDB node) reference SMW
  - tier2 servers reference tier1 servers
  - All other nodes reference tier2 servers
- **On each node use zypper or yum commands to update rpms from those repos**
  - cle_node# **zypper --non-interactive install python3**
  - cle_node# **pcmd –r –n ALL_COMPUTE "zypper --non-interactive install python3"**
  - dal_node# **yum -y install python3**
- **Any rpms changed this way will disappear when the node reboots unless the boot image is rebuilt and the node rebooted from the new boot image.**

Rolling Patches – compute nodes

- **Rolling Patch**
  - Provides the ability to apply a qualified patch to a set of compute nodes without rebooting the system
  - Not for service nodes
- **Compute Node Patches**
  - Not all patches qualify
    - Some may have dependencies requiring a system reboot
  - Only allows for patches within a release
    - Upgrades between releases still require a system reboot
  - Requires the use of a workload manager (WLM)

Rolling Patches – compute nodes

- **Setup required**
  - Set up controlling batch queue in WLM
  - Install patch RPMs into a repository on the SMW
  - Create new image which contains patch
  - Test patched image on some node(s)
  - Use cnode to make patched image default for compute nodes
  - Invoke upgrade with cnat command
- **cnat (compute node administration tool) command**
  - Runs a batch script through a workload manager
  - Ensures that it runs successfully on each specified node
  - This allows administrative tasks to run on compute nodes without interfering with user jobs

cnat [-h] [-c CONFIG] [-o OUTPUT] [-l {debug,info,warning,error,critical}]
[-n NODE_LIST | -N NODE_LIST_FILE | -r RESUME] [--version] script

Rolling Patches – cnat-reboot

- **cnat-reboot is an included script which reboots the nodes assigned to it by the workload manager**
  - Must be run by crayadm to be able to reboot a node

crayadm@login> **cnat -n <node list> /opt/cray/cnat/default/bin/cnat-reboot**

  - Will control reboot of compute nodes to new image
  - Post invocation actions supported are
    - Suspend upgrade
    - Cancel upgrade
    - Rollback upgrade

Rolling Patches – cnat

```
crayadm@login> cnat -l debug -l info -n 32 hostname
Using script /bin/hostname
Creating output directory cnat-20160502101159
Setting up node features
Submitting 0 50-node jobs and a 1-node job
2016-05-02 10:11:59,652 INFO Submitted batch job 14632
2016-05-02 10:11:59,691 INFO Batch job 14632 started on node 32
2016-05-02 10:11:59,730 INFO Batch job 14632 succeeded
Results: 1 job, 1 succeeded, 0 failed
```

Rolling Patches – cnat-status

- **cnat-status gives further information about the status of a cnat run.**

```
cnat-status [-h] outputdir
crayadm@login> cnat-status cnat-20160502101159
cnat pid 3213 started Mon May  2 10:11:59 2016 by crayadm on opal-p2
    using config /etc/opt/cray/cnat/cnat.yaml
    using script /bin/hostname

Job 14632 submitted Mon May  2 10:11:59 2016
    started Mon May  2 10:11:59 2016 on nid 32
    ended Mon May  2 10:11:59 2016 exit code 0

No pending nodes
No active nodes
1 completed node: 32
No failed nodes
```

CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- Configuration
- Booting
- Reconfiguration
- Software update
- **Summary**
- Q & A

Summary

- **New management paradigm**
  - Very different from previous SMW/CLE software
  - Designed to reduce downtime for administrative activities
    - Staged Upgrade using snapshots on SMW
    - Rollback of software on SMW
    - Software and configuration operations can be staged to avoid impacting the running system
- **Separation of Software and Configuration**
  - Increasingly common and standard model in the Cloud, OpenStack, and Enterprise
  - Can update software images and configuration separately and independently without affecting the other
  - Software via IMPS (Image Management and Provisioning System)
  - Configuration via CMF (Configuration Management Framework)
    - Allows use of common configuration management tools like Ansible

Summary

- **Image Management and Provisioning System (IMPS)**
  - Prescriptive Image Creation
    - Use zypper/yum to satisfy rpm dependencies
      - Far more reliable and consistent
    - Use simple image recipes and package collections to build up image definitions
      - Provides more clear and flexible image definition
  - Ability to change and enhance existing image recipes
  - Ability to create new image recipes for different node types
  - tmpfs versus netroot images for login and compute nodes
  - Easy to build upgrade images with bug fixes or security updates
  - Easy to move images from system to system
  - PE software separate from base images
    - Provides more clear, defined encapsulation and management

Summary

- **Configuration Management Framework (CMF)**
  - Configuration centralized in one location
    - Not spread out across SMW, bootroot, and large sharedroot
  - Most configuration in YAML and applied to node via Ansible
    - Easily parsed by tools, viewable and editable by humans
  - Can clone configuration to try new options
  - Configurator tool to guide system administrators through configuring the system
  - Easy-to-use places to plug in site customizations
  - Easy to back up
  - Easy to move from system to system

Summary

- **Node Deployment**
  - NIMS (Node Image Mapping Service)
    - Can specify what image to boot on a node by node basis or on groups of nodes
    - Can provide alternate configuration for specific nodes or groups of nodes
    - Can try out new images or new configuration on select nodes prior to full deployment
  - Live updates (running zypper/yum live on a booted CLE node)
  - Rolling patches for compute nodes with WLM and cnat

Summary

- **New Installer**
  - Common installer for SMW, CLE, and future products
  - Completely redesigned, configuration driven, and written in Python
    - More maintainable, extensible, and reliable
  - Implements Staged Upgrades
    - More reliable, significantly less downtime, easy fallbacks
  - Sharing distribution repositories between SMW and CLE, reducing admin overhead and reducing disk space requirements
    - Prepares for Live Updates (using zypper/yum from CLE nodes)

CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- Configuration
- Booting
- Reconfiguration
- Summary
- Q & A

Q & A – Documentation

- **CrayPort http://crayport.cray.com and CrayDoc http://docs.cray.com**
  - What's New for CLE 6.0 and SMW 8.0
  - System Overview
  - XC Series System Software Installation and Configuration Guide
  - XC Series System Configurator User Guide
  - XC System Administration Guide
  - SMW HA Installation Guide
  - SMW HA XC Administration Guide
  - Lustre Administration Guide
  - CLE XC System Network Resiliency Guide
  - System Environment Data Collections (SEDC) Guide
  - Monitor and Manage Power Consumption on Cray XC Systems
  - Configure Cray SEC Software
  - DataWarp Installation and Configuration Guide
  - DataWarp Administration Guide
  - DataWarp User Guide
  - Cray Programing Environments Installation Guide

Q & A – Cray Training: Transferring Knowledge

- **Both online and instructor lead courses available to customers in Chippewa Falls, Wisconsin, USA or at a customer site starting June 2016**
- **Sign up for classes by starting at our Web site:**
  - cray.com → Support → Training
    - Or this link: http://www.cray.com/support/training/schedule
  - Contract Cray Training:
    - registrar@cray.com (or call: +1-715-726-4036)

Q & A

Harold Longley
htg@cray.com

Legal Disclaimer

Information in this document is provided in connection with Cray Inc. products. No license, express or implied, to any intellectual property rights is granted by this document.

Cray Inc. may make changes to specifications and product descriptions at any time, without notice.

All products, dates and figures specified are preliminary based on current expectations, and are subject to change without notice.

Cray hardware and software products may contain design defects or errors known as errata, which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Cray uses codenames internally to identify products that are in development and not yet publically announced for release. Customers and other third parties are not authorized by Cray Inc. to use codenames in advertising, promotion or marketing and any use of Cray Inc. internal codenames is at the sole risk of the user.

Performance tests and ratings are measured using specific systems and/or components and reflect the approximate performance of Cray Inc. products as measured by those tests. Any difference in system hardware or software design or configuration may affect actual performance.

The following are trademarks of Cray Inc. and are registered in the United States and other countries: CRAY and design, SONEXION, and URIKA. The following are trademarks of Cray Inc.: APPRENTICE2, CHAPEL, CLUSTER CONNECT, CRAYPAT, CRAYPORT, ECOPHLEX, LIBSCI, NODEKARE, THREADSTORM. The following system family marks, and associated model number marks, are trademarks of Cray Inc.: CS, CX, XC, XE, XK, XMT, and XT. The registered trademark LINUX is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a worldwide basis. Other trademarks used in this document are the property of their respective owners.