Migrating, Managing, and Booting Cray XC and CMC/eLogin

Jeff Keopp, Joel Landsteiner, Harold Longley
Cray Inc.

CUG.2017.CAFFEINATED COMPUTING
Redmond, Washington May 7-11, 2017

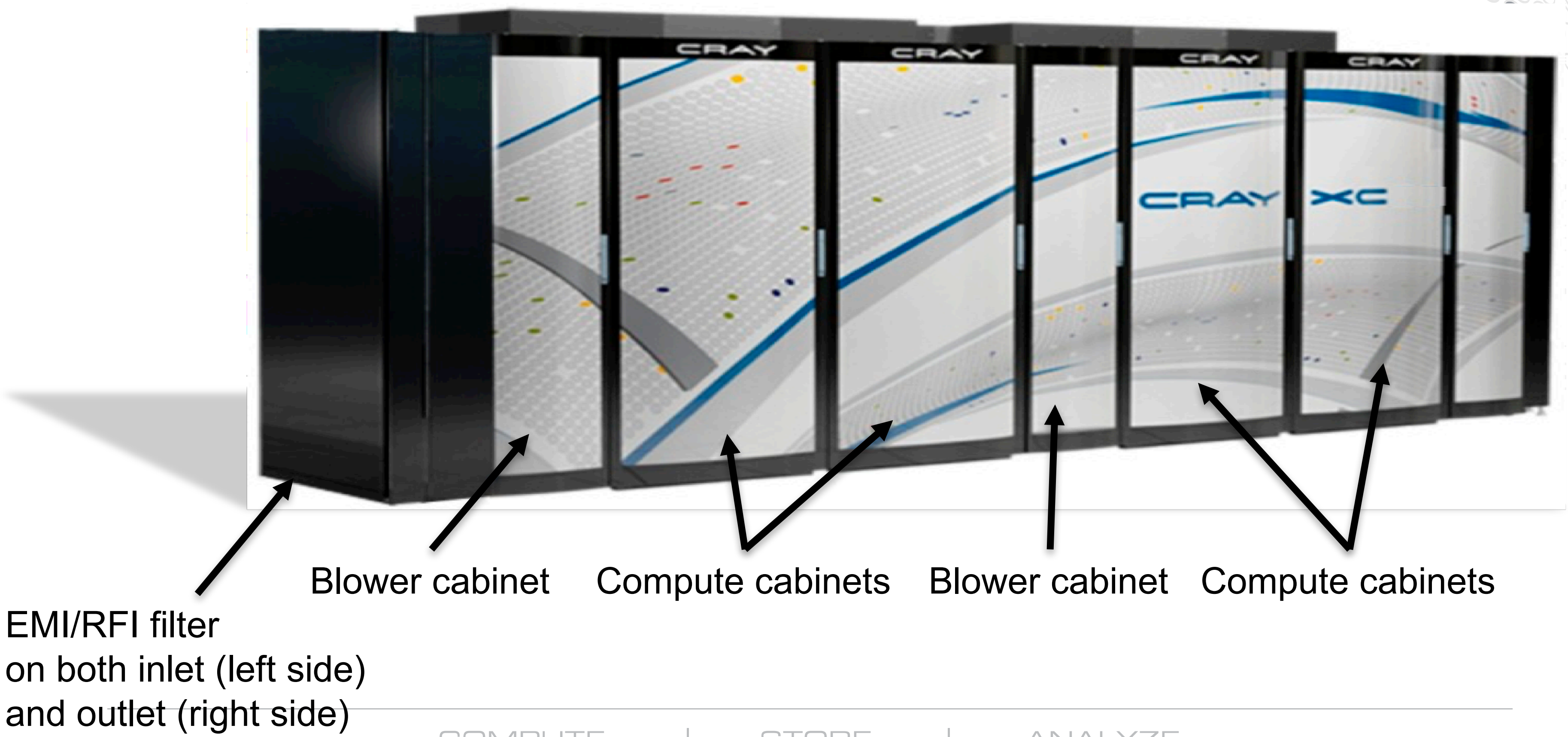
Agenda

- **Introduction to SMW/CLE system management**
- **New system management features since UP01**
- **Best practices for using Ansible**
- **Troubleshooting XC system booting problems**
- **Migrating SMW/CLE software from 7.2/5.2 to 8.0/6.0**
- **Intro to CMC/eLogin system management**
- **Migrating CIMS/CDL to CMC/eLogin**
- **CLE Boot Performance and Reliability**
- **Q & A**

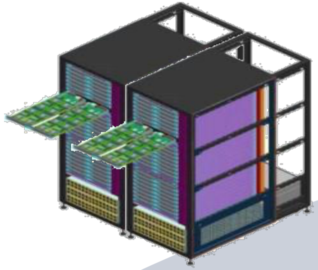
Introduction to SMW/CLE system management

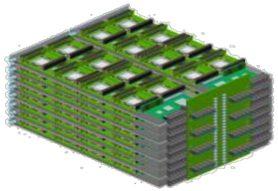
- Cray XC System
- Management features
- Management of software images
- Configuration Management Framework
- Node to image mapping
- Boot process (software plus configuration)

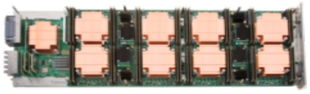
Cray XC System



Cray XC System Building Blocks







Compute Blade

4 Nodes

Service (I/O) Blade

2 or 4 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

Identifying Components

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

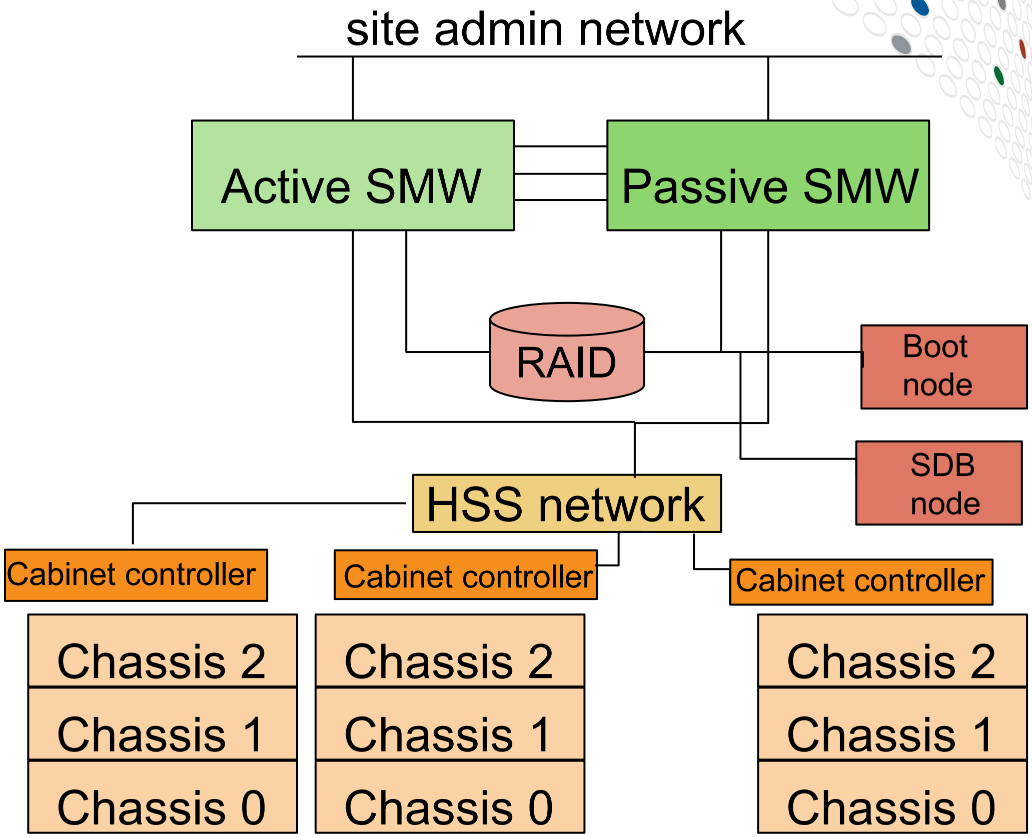
Component	Format	Description
System	s0, p0 all	All components attached to the SMW.
Cabinet	cX-Y	Cabinet number and row; this is the cabinet controller (CC) host name.
Chassis	cX-Yc#	Physical chassis in cabinet: 0, 1, 2. Chassis are numbered from bottom to top.
Blade or slot	cX-Yc#s#	Physical blade slot 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)

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:
    - site-admin (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

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



Service Node Roles and Functions

- **Service nodes are defined by the service they provide**
  - The “service” includes hardware and software components
    - Service nodes can be I/O nodes or repurposed compute nodes (RCN)
      - I/O Nodes are configured with Fibre Channel, InfiniBand, Ethernet, or SAS cards
      - I/O Nodes could be configured with PCI SSDs in the case of DataWarp nodes

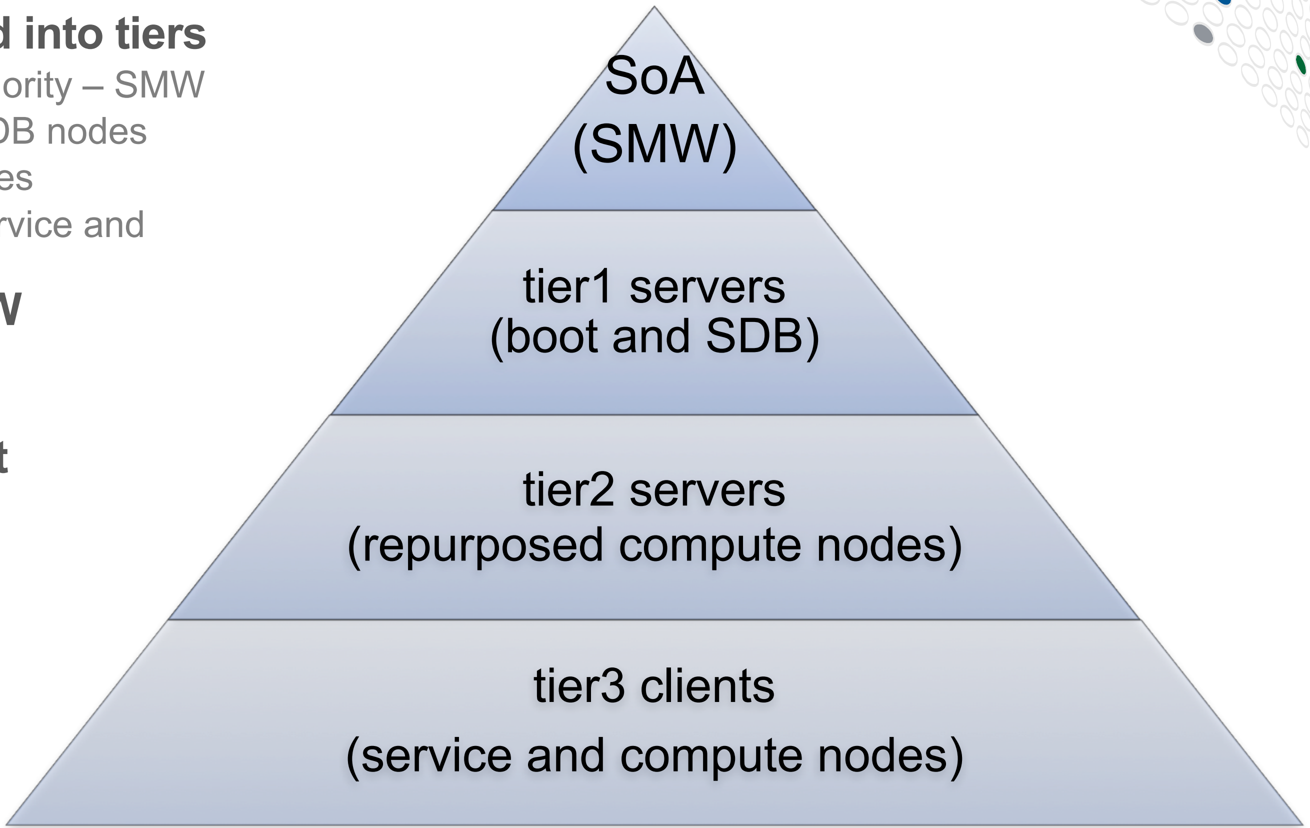
Service Node	Role
Boot	Tier 1 – Boots other nodes and serves images to Tier 2 nodes
SDB	Tier 1 - Service Database node
Login	Allows users to control their applications. Configured with a GigE or 10-GigE card
LNET	Lustre Network Router – provides access to external Lustre filesystems
DVS	Data Virtualization Servers used to project external file systems (NFS, GPFS, and more) to other nodes
RSIP	Realm-specific IP – Provides access to external IP addresses
DataWarp	DataWarp-managed nodes with SSD hardware or DataWarp API gateway nodes
WLM	Nodes providing a role for workload management
DAL	Direct Attached Lustre nodes (MGS, MDS, or OSS)
RCN	Used for Tier 2, MOM, or MAMU nodes

Data Virtualization Service (DVS)

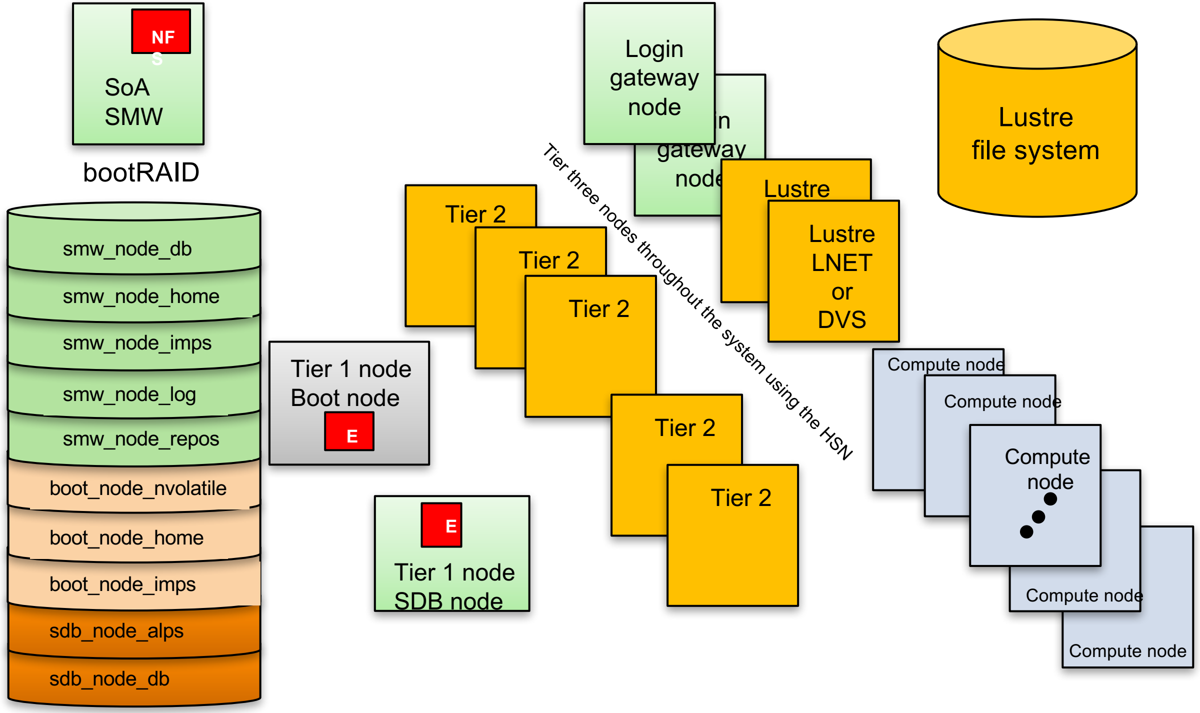
- **Cray DVS is a distributed network service that provides transparent access to filesystems residing on the service I/O nodes and/or remote servers in the data center**
  - Projects local filesystems resident on service nodes or remote file servers to compute and service nodes within the Cray system
    - *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

Cray Scalable Services

- **Nodes are classified into tiers**
  - SoA - Server of Authority – SMW
  - Tier 1 – Boot and SDB nodes
  - Tier 2 – Special nodes
  - Tier 3 – Clients – service and compute nodes
- **Distribute from SMW**
  - config sets (9P)
  - zypper/yum repos
- **Distribute from boot**
  - DVS netroot
  - DVS diags
  - DVS PE
- **Aggregate to SMW**
  - LLM logging



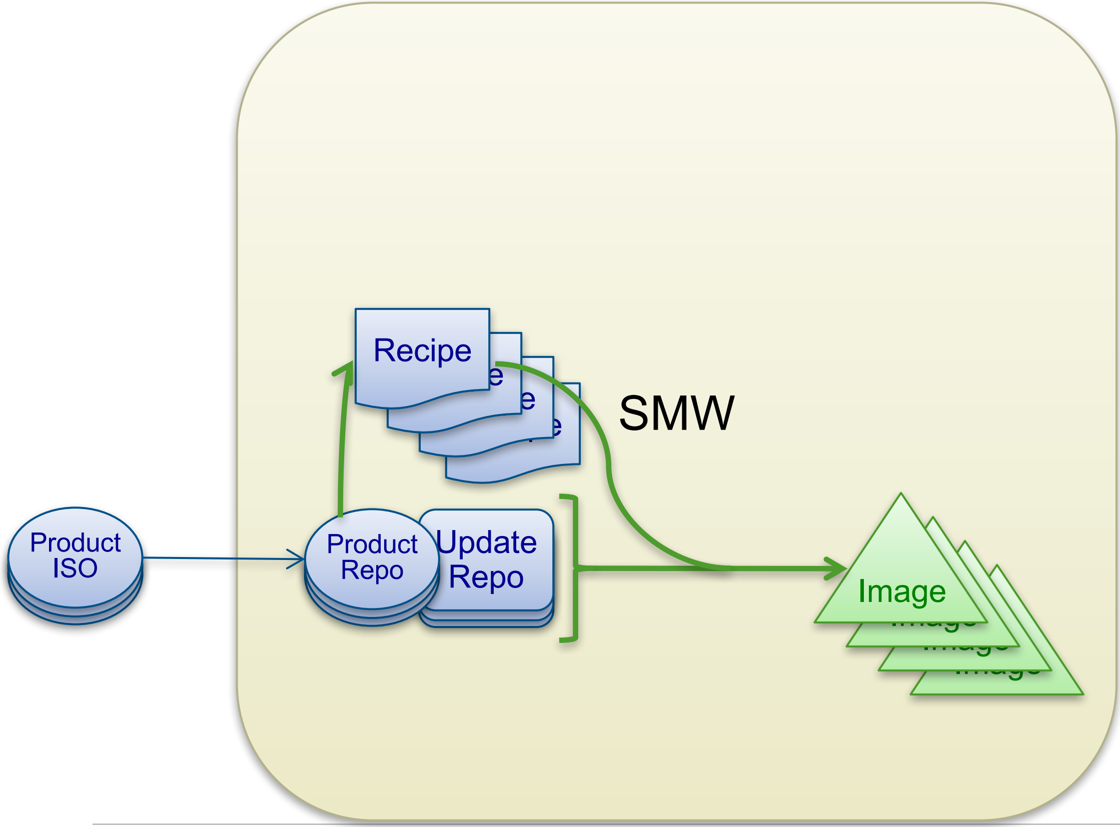
Basic System Configuration



Separate Software and Configuration

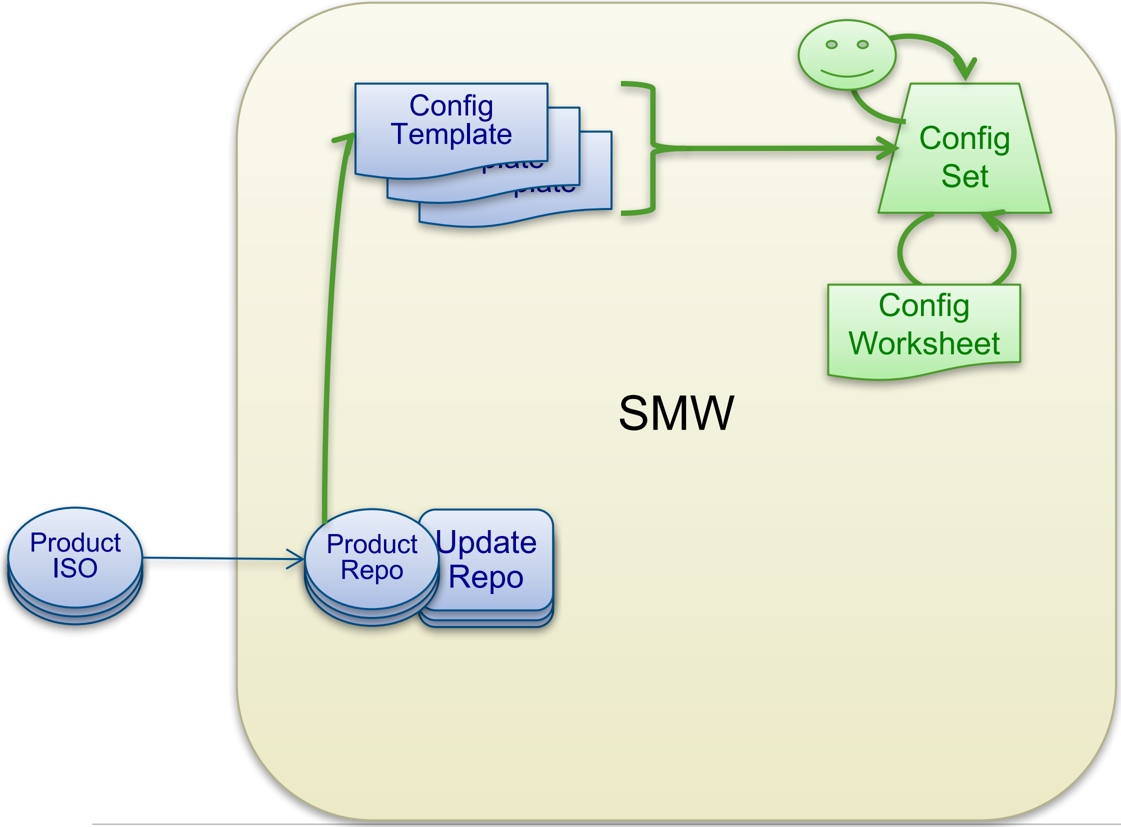
- **Node Images contain [unconfigured] code**
  - Different images for admin, compute, service, login, DAL, …
- **Config sets contain centralized configuration**
  - Global config set used by SMW and CLE
  - CLE config set used by CLE
- **Putting software and configuration together at boot**
- **Some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

Node Images



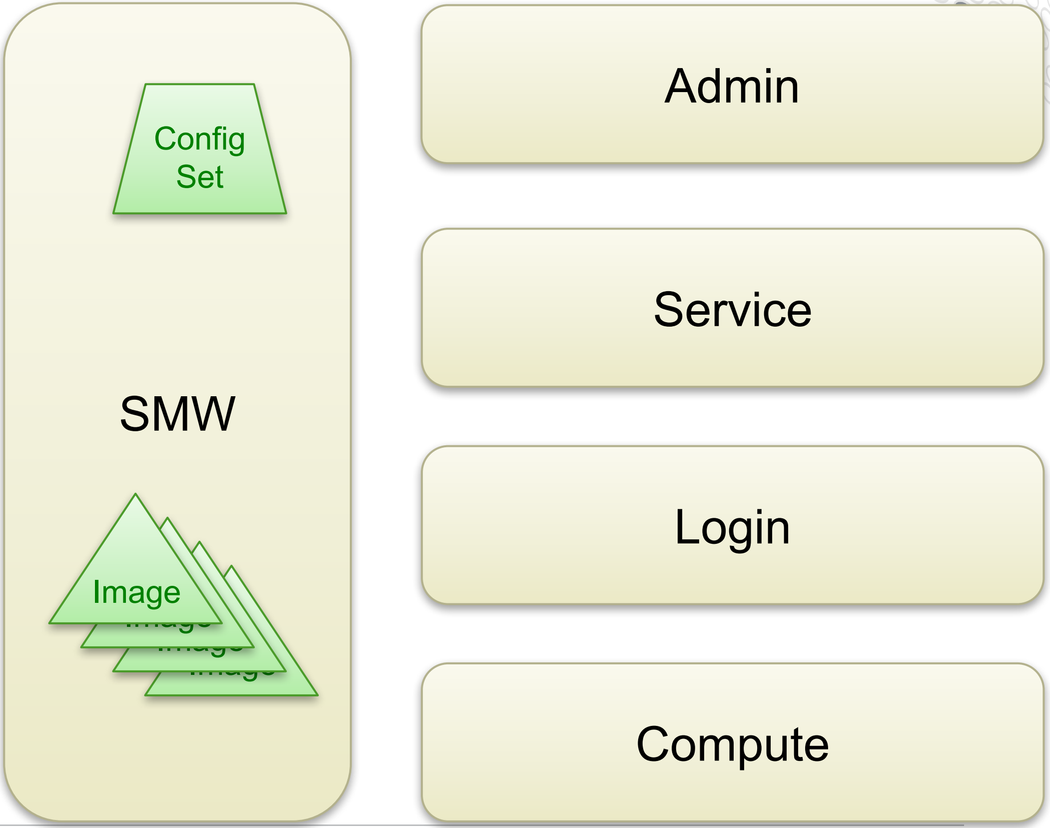
- **Install software**
  - Repositories created from product DVDs
    - Read-only
    - Empty writeable update repositories created for future use
  - Recipes installed for default image types.
- **Create images**
  - Create image roots from recipes and rpm repositories
- **The resulting images are mostly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

Centralized Configuration



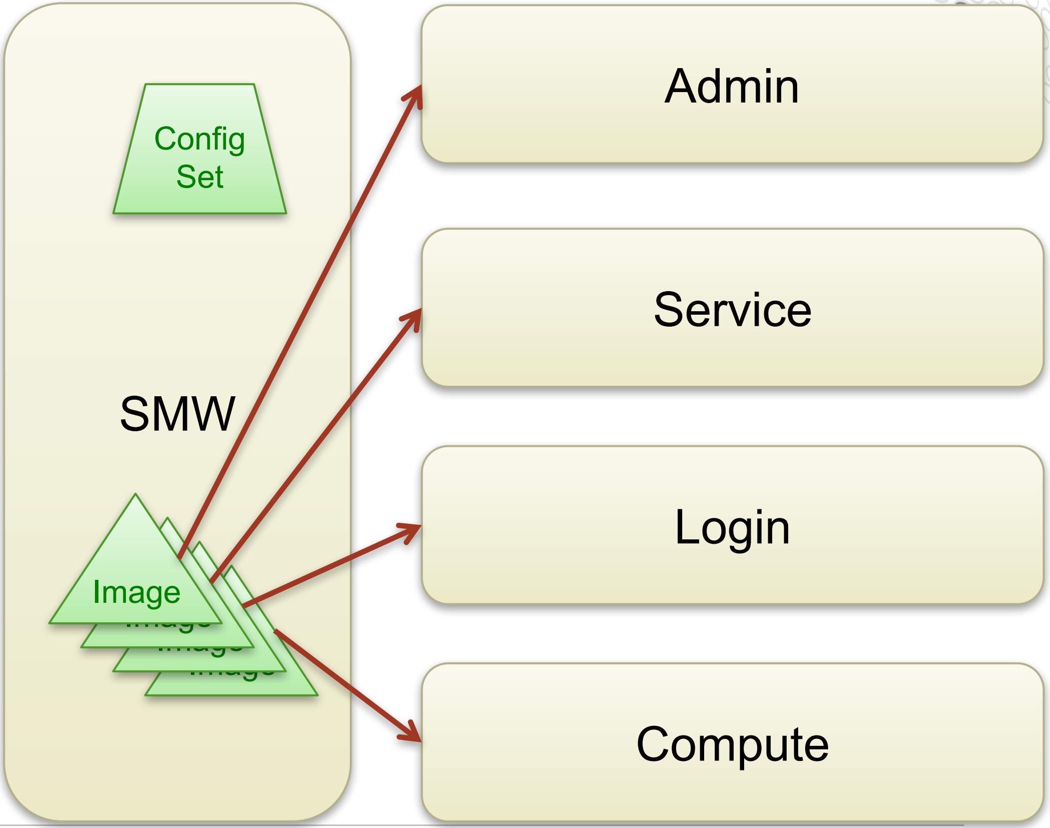
- **Config Set container**
  - Centralized configuration
  - Contains multiple configuration files, for different areas.
  - Well defined schema facilitates tools
  - YAML format allows direct editing
  - Other (non-YAML) content can be added as necessary.
- **Configurator tool**
  - Understands the 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

Boot Software With Configuration



Boot Software With Configuration

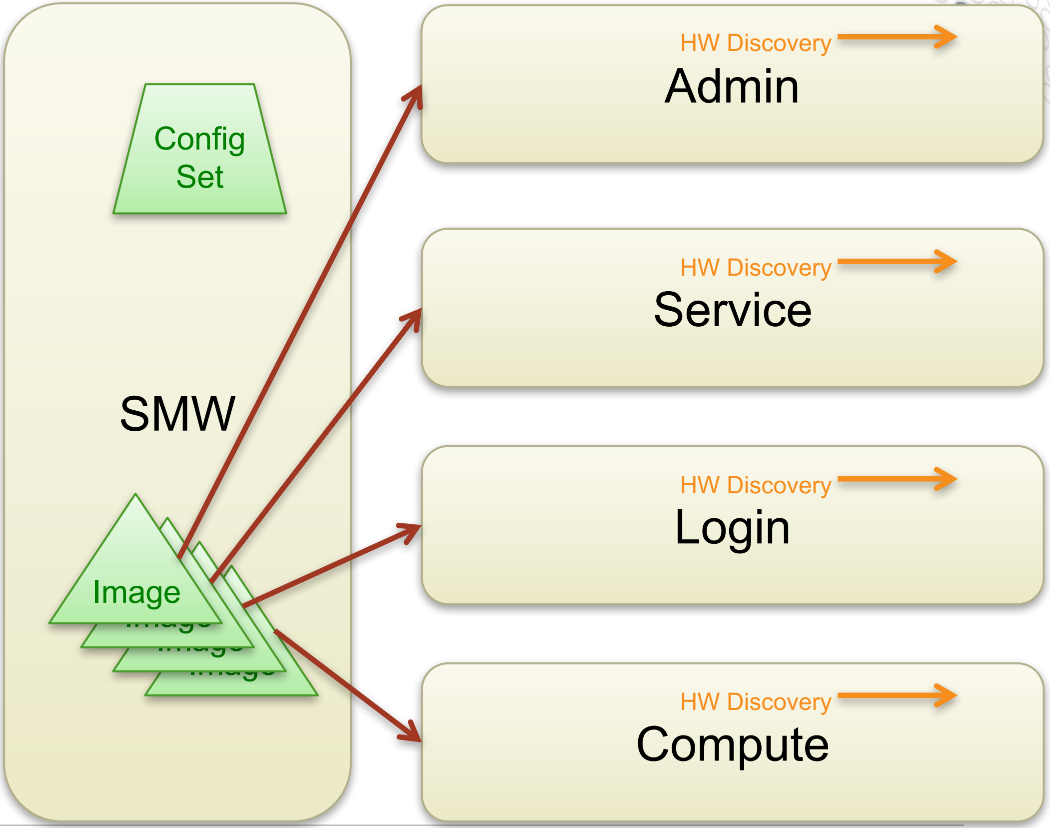
- Nodes boot unconfigured image



Boot Software With Configuration

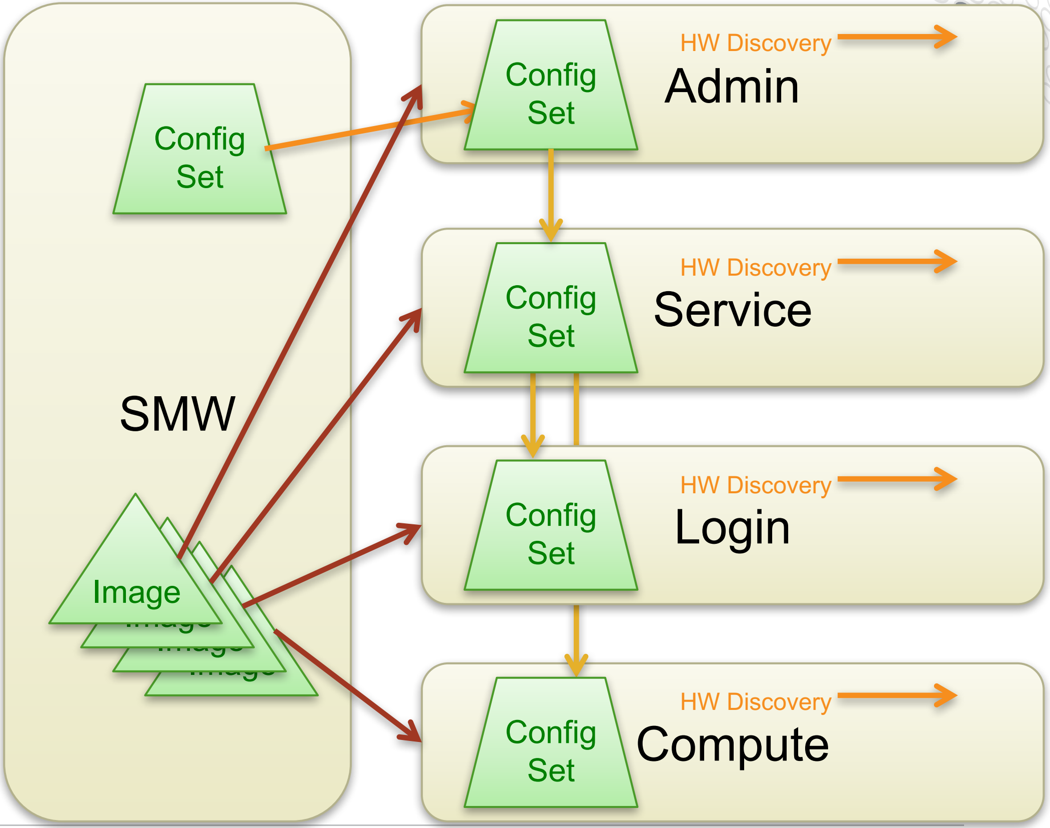
- Nodes boot unconfigured image

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



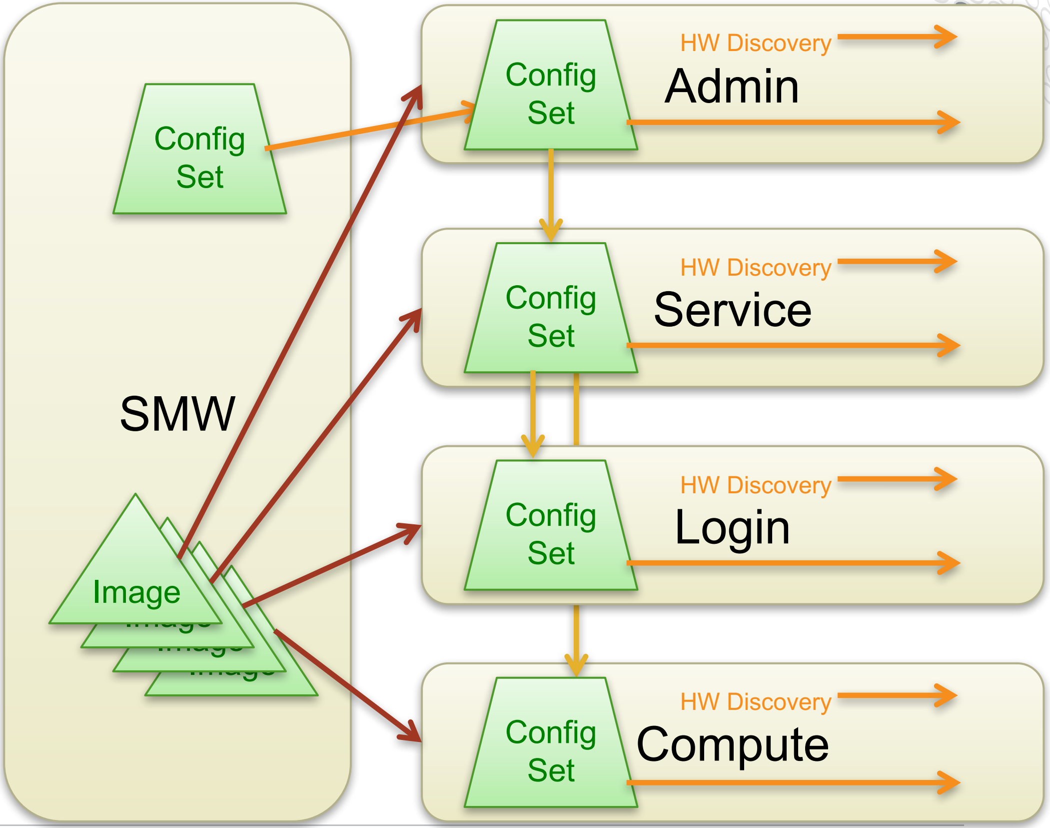
Boot Software With Configuration

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



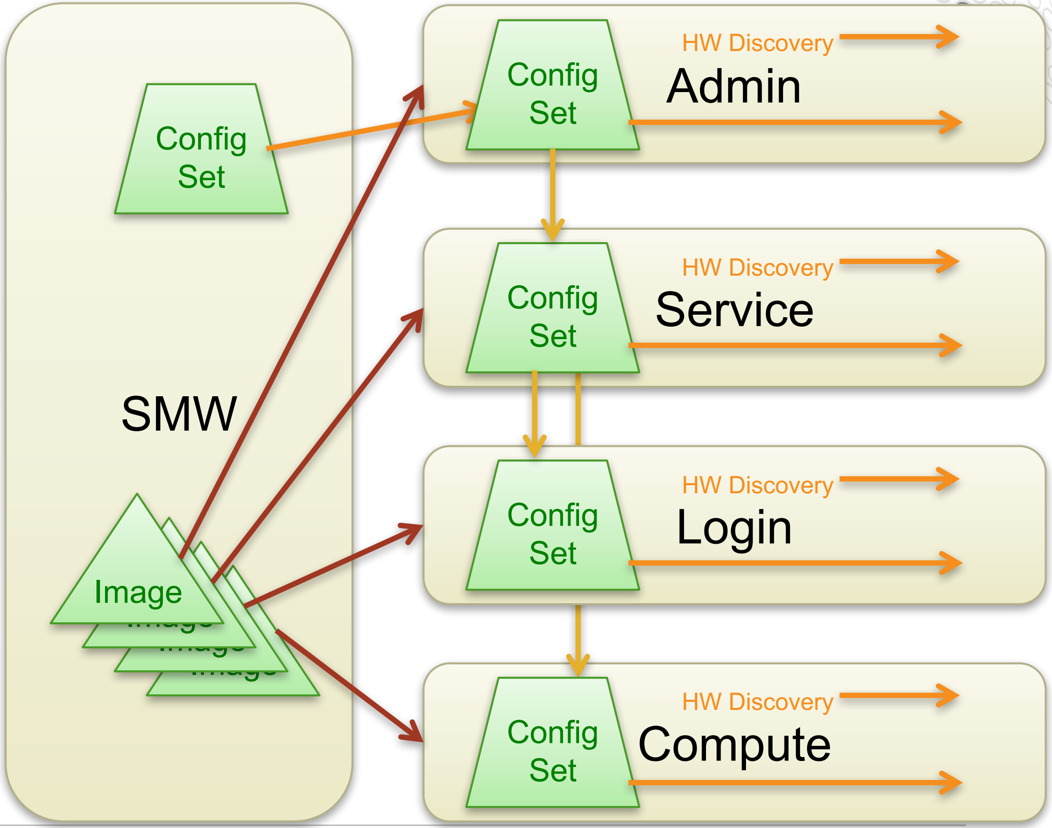
Boot Software With Configuration

- 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

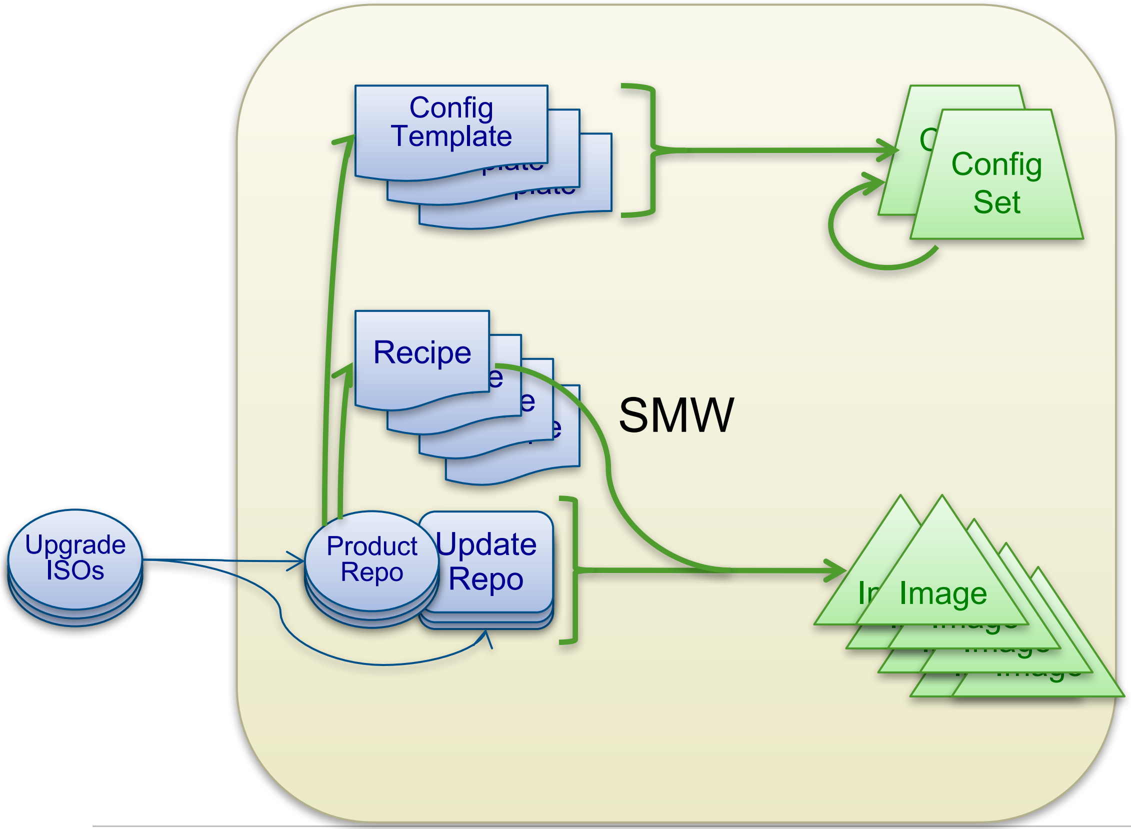


Boot Software With Configuration

- 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



Staged Upgrades

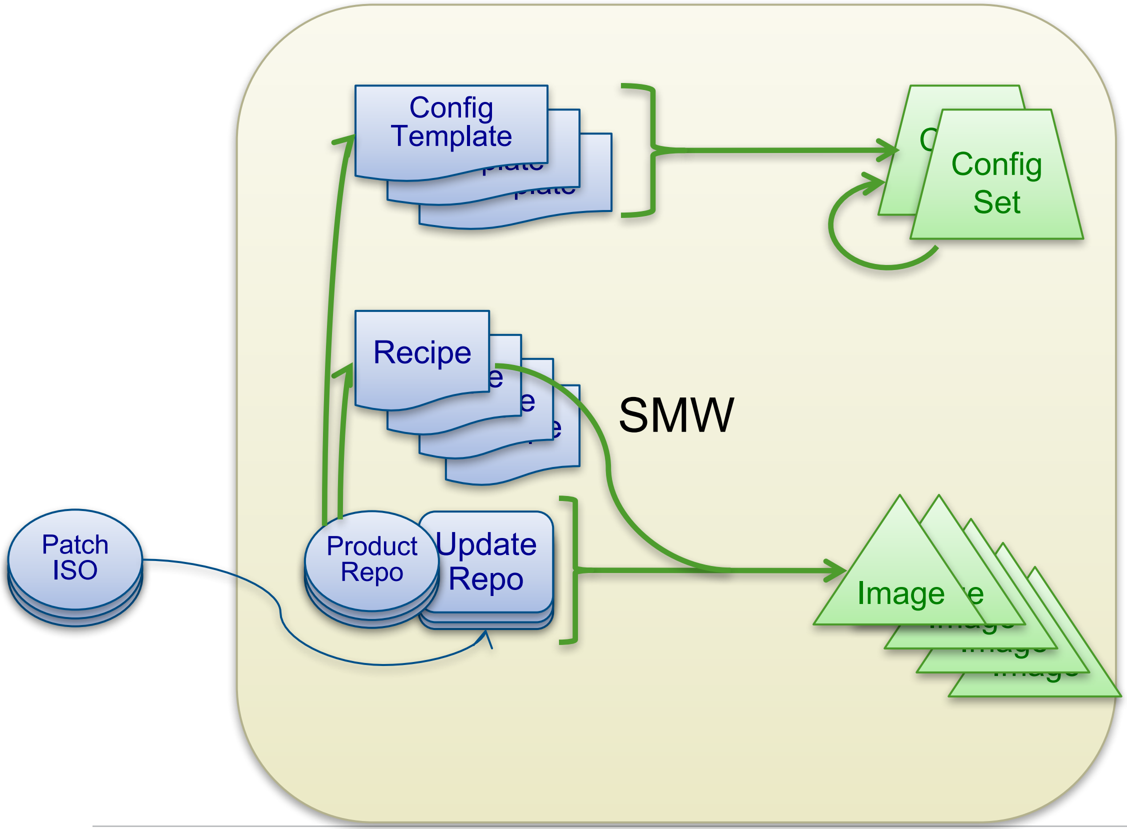


- **Focus:**
  - Minimal downtime
  - Safe: rollback possible
- **Snapshots and Versions**
  - SMW root gets btrfs snapshot
  - CLE objects are all explicitly versioned
- **While running current system in production…**
  - Update repositories
  - Apply software to snapshots
  - Build new CLE images
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

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

- **Comparison 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

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

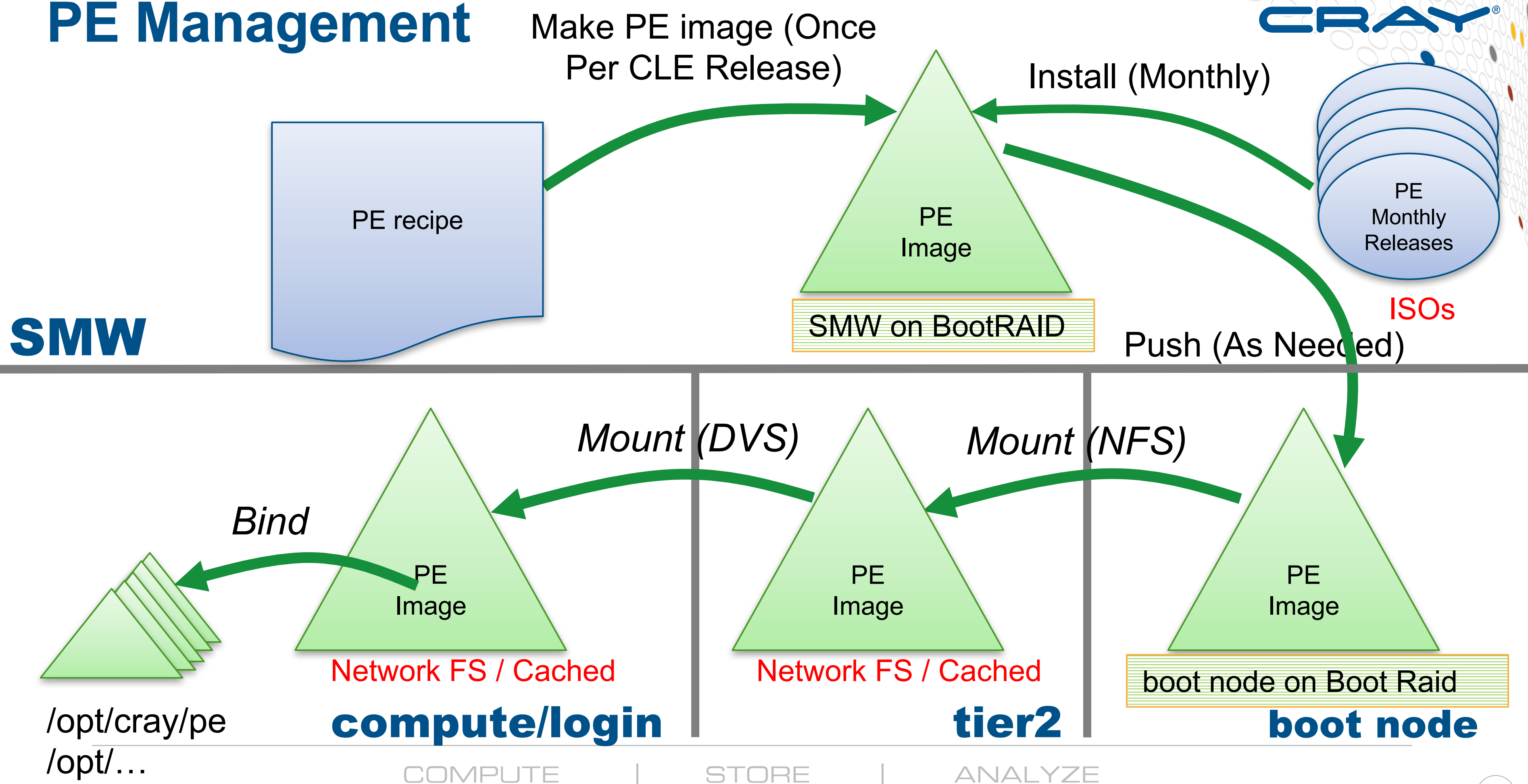
- **cnat-status gives further information about the status of a cnat run.**

```
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
```

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 file system for diskless XC nodes**

PE Management



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
  - Cray commands for changing, searching, displaying, validating
    - Ensure files stay in correct format
- **JSON (JavaScript Object Notation)**
  - Open standard format
    - A proper subset of YAML
  - Data objects consist of attribute–value pairs
- **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
- **Site can create their own repositories**

Image Recipes

- **Each default image type has an image recipe installed on SMW**
  - Compute, admin, 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
- **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**
  - Site can create their own recipes
- **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
- Site can create their own 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."
            }
```

Customizing 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

Customizing 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": { ... }
    },
```

Customizing 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": { ... }
    },
```

Customizing 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**
  - Admin node boot image – used by boot and SDB
  - 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

tmpfs versus netroot

- `tmpfs`
  - Default location of the root file system on Cray XC series systems
  - Always used for service nodes (except login nodes) and DAL (direct-attached Lustre) nodes
  - Efficient and fast root file system access
  - Large memory footprint
  - File system content is limited to reduce memory footprint
  - Typically used when minimal commands and libraries required
  - Works well for compute nodes with well defined workloads and for service nodes that are used primarily for internal services
- `netroot`
  - Alternative approach that mounts the root file system from a network source
  - Used only for compute and login nodes, never for service nodes (except login nodes)
  - Uses `overlayfs` to layer `tempfs` on top of a read-only network file system
  - Slower root file system access
  - Increased node boot time
  - Minimized memory footprint due to leveraged network
  - No restriction on file system content
  - Typically used when a robust set of commands and libraries required (`netroot` enables large network-based images, formerly enabled through the DSL feature)
  - Works well for compute nodes with diverse workloads and for compute nodes requiring a small memory footprint

tmpfs comparison with netroot

	Memory consumption	Number of rpms
Admin image root – tmpfs	1400MB	600
Service image root – tmpfs	1700MB	670
Login image root – tmpfs	3600MB	1100
Compute image root – tmpfs	1500MB	745
Login image root – netroot	125MB	2500
Compute image root – netroot	150MB	2380

Note: Numbers are approximate for UP03. Use of netroot preload may change memory consumption.

Cray image recipes for tmpfs and netroot can be used as sub-recipes in a site recipe then customized with additional rpms needed or reduced to remove rpms not needed

Configuration Management Framework

- **Config sets**
- **IDS (IMPS Distribution Service)**
- **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, config, dist, files

- **/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.

  - Configurator processes only files ending in “config.yaml”
  - cray-ansible makes all files ending in “yaml” available as local vars
    - A site can include config data without using a configurator template

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

  - Subdirectories

    ansible, **changelog**, config, dist, files, **worksheets**

  - Worksheet files

    cray_alps_worksheet.yaml, cray_logging_worksheet.yaml, cray_net_worksheet.yaml, cray_scalable_services_worksheet.yaml, etc.

- **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/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, a cache is distributed by the IMPS Distribution Service (IDS)**
- **IDS leverages the 9P network filesystem and the Linux automounter facility to transport the cache 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
  - 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
- **Full copy of the config set is on the node**
  - Updated at boot time or upon demand

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

- XC™ Series System Configurator User Guide (CLE 6.0.UPxx S-2560)
- **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 setting

Configurator

- **Iterate on the configurator as necessary**
  - Admin can configure a specific service

    smw# **cfgset update -s cray_alps -S unset -l basic p0**

    smw# **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
```

Note: This example is before UP02. In UP02 tier1 comes from a nodegroup not a list of cnames

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 (for this run of the configurator)
  - 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
  - Regenerate configuration worksheets
  - Write changelog entry

    /var/opt/cray/imps/config/sets/p0/changelog/changelog_2015-09-29T12:00:37.yaml
  - Run post-configuration scripts
  - Remove backup config set (from this run of the configurator)

Simple Sync

- **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

- **Avoid copying files from one OS type (SLES) to another OS type (CentOS) when using common, platform service, and customized node groups**

Simple Sync

- **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

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 cmap and cnode

NIMS

- **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
  - Use “cmap” command to manipulate NIMS maps
- **NIMS Node**
  - Represents the physical, bootable node on the XC system
  - Use “cnode” command to manipulate information about a node in a given NIMS map

NIMS - cnode

- **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
```

Booting – What is New?

- **Boot the system**

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

- **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

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)
  - Ensures the config set cache on the node is current
  - 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

Agenda

- Introduction to SMW/CLE system management
- **New system management features since UP01**
- Best practices for using Ansible
- Troubleshooting XC system booting problems
- Migrating SMW/CLE software from 7.2/5.2 to 8.0/6.0
- Intro to CMC/eLogin system management
- Migrating CIMS/CDL to CMC/eLogin
- CLE Boot Performance and Reliability
- Q & A

New System Management Features since UP01

- **Boot Performance and reliability improvements (UP01) (details later)**
  - Config set caching
  - Netroot preload
- **Boot troubleshooting guide released (between UP01 and UP02)**
  - XC™ Series Boot Troubleshooting Guide CLE 6.0 UPxx S-2565)
- **Boot performance and reliability improvements (UP02) (details later)**
  - Ansible filtering
  - Fact collection tweaks
  - Sparse looping adjustments
  - Boot profiler
- **Admin recipe (UP02)**
- **Ansible log (init_netroot_setup, init, booted) for log and file-changelog (UP02)**
- **eLogin nodes now use Simple Sync v2 (UP02)**
- **Node groups (UP02)**
- **Improved config set validation (UP03)**
- **Boot performance and reliability improvements (UP03) (details later)**
  - Greater boot concurrency support
  - ARP table initialization
  - PE ldconfig caching
  - ntpd improvements
  - Node groups optimization
  - DVS read only optimizations
- **Ansible Play Writing guide (UP03)**
  - XC™ Series Ansible Play Writing Guide (CLE 6.0 UP03 S-2582)
- **Migration guides (UP03+)**

UP02 changes

- **Admin recipe**
  - For boot and SDB node
  - Smaller version of service recipe
    - May be small enough that when workload manager (WLM) content is added, the resulting image is still small enough to PXE boot SDB node
- **Ansible log change for netroot**
  - For nodes which use netroot, the logs from "init" phase of cray-ansible have been split into "init_netroot_setup" and "init"
    - file-changelog-init_netroot_setup
    - sitelog-init_netroot_setup
- **eLogin nodes now use Simple Sync v2**
  - All nodes now use same version of Simple Sync
    - Same directory structure, same Ansible plays

Node groups (UP02)

- **Only used for Simple Sync in UP01 to transfer files to groups of nodes**
- **Define and manage logical groupings of system nodes**
  - Nodes can be grouped arbitrarily, though typically they are grouped by software functionality or hardware characteristics, such as login, compute, service, DVS servers, and RSIP servers
- **Can be referenced by name within all CLE services in that config set**
  - Eliminates the need to specify groups of nodes for each service individually and greatly streamlining service configuration
- **Used in Cray-provided Ansible plays and can be also used in site-local Ansible plays**
- **Similar to, but more powerful, than the class specialization feature of CLE 5.2**
  - For example, a node can be a member of more than one node group but could belong to only one node class in CLE 5.2
- **Sites are encouraged to define their own node groups and specify their members**
- **Defined in cray_node_groups service of the config set**

Node groups (UP02) Characteristics

- **Node group membership is not exclusive**
  - a node may be a member of more than one node group.
- **Node group membership is specified as a list**
  - cnames for CLE nodes
  - hostid (command output) for SMW
  - hostnames for eLogin nodes
- **All compute nodes and/or all service nodes can be added as node group members by including the keywords “platform:compute” and/or “platform:service” in a node group**
- **Any CLE configuration service is able to reference any defined node group by name**
- **CMF exposes node group membership of the current node through the local system "facts" provided by the Ansible runtime environment**
  - Each node knows to which node groups it belongs and that knowledge can be used in Cray and site-local Ansible playbooks

Node groups (UP02) Default Node Groups

- **compute_nodes**
  - Defines all compute nodes for the given partition
- **service_nodes**
  - Defines all service nodes for the given partition
- **smw_nodes**
  - Add the output of the hostid command for the SMW
  - For an SMW HA system, add the host ID of the second SMW also
- **boot_nodes**
  - Add the cname of the boot node
  - If there is a failover boot node, add its cname also
- **sdb_nodes**
  - Add the cname of the SDB node
  - If there is a failover SDB node, add its cname also
- **login_nodes**
  - Add the names of internal login nodes on the system.
- **all_nodes**
  - Defines all compute nodes and service nodes on the system
  - Add external nodes (eLogin nodes), as needed.
- **tier2_nodes**
  - Add the cnames of nodes that will be used as tier2 servers in the cray_scalable_services configuration.

Node groups (UP02) Additional Platform Keywords

- **Cray uses these two platform keywords to create default node groups that contain all compute or all service nodes.**
  - platform:compute
  - platform:service
- **Sites that need finer-grained groupings can use these additional platform keywords to create custom node groups that contain all compute or service nodes with a particular core type.**
  - platform:compute-XXNN
  - platform:service-XXNN
- **For XXNN, substitute a four-character processor/core designation, such as KL64 or KL68, which designate the two Intel® Xeon Phi™ processors (Knights Landing) with different core counts.**

Processor (XX)	Core (NN)	Intel Code Name
BW	12, 14, 16, 18, 20, 22, 24, 28, 32, 36, 40, 44	Broadwell
HW	04, 06, 08, 10, 12, 14, 16, 18, 20, 24, 28, 32, 36	Haswell
IV	02, 04, 06, 08, 10, 12, 16, 20, 24	Ivy Bridge
KL	60, 64, 66, 68, 72	Knights Landing
SB	04, 06, 08, 12, 16	Sandy Bridge

Improved config set validation (UP03)

- **Our goal is to detect configuration errors before system boot**
- **Focus areas**
  - Validation Rules plugin functionality for configurator
    - Cray use only – not open for customer additions (yet)
  - New configurator core data types
    - `cname`
    - `hostname`
      - Fully qualified domain name (FQDN) allowed
    - `ipv4_address`
  - Configurator bug fixes
  - Configurator template updates found via audit
    - Implement Validation Rules
    - Use new configurator core data types
    - Update regular expressions
    - Update guidance, default values, and level

Improved config set validation (UP03)

- **Validation Rules plugin functionality for configurator**
  - Automatically runs all rules

    smw# **cfgset validate p0**
  - List all rules

    smw# **cfgset list-rules p0**
  - List all rules for a service

    smw# **cfgset list-rules -s cray_net p0**
  - List a specific rule

    smw# **cfgset list-rules -n system-config.cray_networking.CLENetworksValidGateway p0**

    ```
    - name: system-config.cray_networking.CLENetworksValidGateway
      location: /opt/cray/imps_config/system-config/default/configurator/rules/cray_networking.py
      description:
          The gateway address for each network defined in cray_net must be an ip address that is in the network, and it
           must not be the highest or lowest address in the network, which is reserved for broadcast.
    ```
  - Skip all the rules

    smw# **cfgset validate --no-rules global**

Agenda

- Introduction to SMW/CLE system management
- New system management features since UP01
- **Best practices for using Ansible**
- Troubleshooting XC system booting problems
- Migrating SMW/CLE software from 7.2/5.2 to 8.0/6.0
- Intro to CMC/eLogin system management
- Migrating CIMS/CDL to CMC/eLogin
- CLE Boot Performance and Reliability
- Q & A

Best Practices for Using Ansible

- Ansible Introduction
- Cray specific techniques, practices, and processes
- Ansible references

Ansible – Terms

- **Playbook**
  - One or more plays
- **Play**
  - For XC, a collection of tasks to run on "localhost"
- **Task**
  - For XC, perform a specific sequence of actions on "localhost"
- **Modules**
  - Large Ansible library of common code
    - Control system resources like services, packages, or files
    - Execute system commands
- **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 – Local System Facts

- **Boot process provides access to a group of Ansible facts that describe the running node**

  node# **ansible -m setup localhost**

- **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

Ansible – Best Practices 1

- **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 – 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
```

Jinja2 template file

boot# **cat ../templates/myservice.conf.j2**

```
# myservice.conf
# {{ ansible_managed }}

foo={{ "yes" if myservice_foo else "no" }}
bar={{ myservice_bar }}
baz={{ myservice_baz }}
```

Ansible – example tasks

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
```

boot# **cat lineinfile.yaml**

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

Ansible – example tasks

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 – Running a Play 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 – Running a Play 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 – Running a Play 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 – Best Practices 2

- YAML whitespace is “space” characters and not tab characters
  - http://www.yaml.org/spec/1.2/spec.html
- Write comments (starting with #)
  - Bad comments are worse than no comments at all
- Always name tasks
  - Provide a description about why something is being done
  - This name is shown when the playbook is run
- Use debug module to display the value of a variable
  - With a when clause this can be used to show some collected data if data is not defined
- Always mention the state
  - The ‘state’ parameter is optional to a lot of modules
  - Whether ‘state=present’ or ‘state=absent’, it’s always best to leave that parameter in your playbooks to make it clear, especially as some modules support additional states
- Keep It Simple
  - When you can do something simply, do something simply
  - Do not reach to use every feature of Ansible together, all at once
  - If something feels complicated, it probably is, and may be a good opportunity to simplify things

http://docs.ansible.com/playbooks_best_practices.html

Cray specific Ansible techniques, practices, and processes

- Basic guidelines for Ansible on SMW, CLE, and eLogin
- When are Ansible plays run?
- Which Ansible plays are run?
- How to specify order of plays
- When will plays execute?
- What data is available to plays?
- How to distribute Ansible plays using config set
- Testing Ansible plays
- How to search in image root and config set for Ansible plays
- Ansible Limitations

Ansible – 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
  - Enable/disable services
  - Change crontab entries
  - Modify files
  - Copy files
  - Run shell programs when node meets certain conditions

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 – 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

Configuring MySQL Example

- **cray-ansible runs Ansible twice (sandwiches Linux startup)**
  - *Init phase* - during Linux init
    - “pre-configures” Linux
    - Shut off Linux default services we don’t want
    - Setup HSN network interfaces
  - …Linux/systemd takes over
  - *Booted phase* – 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

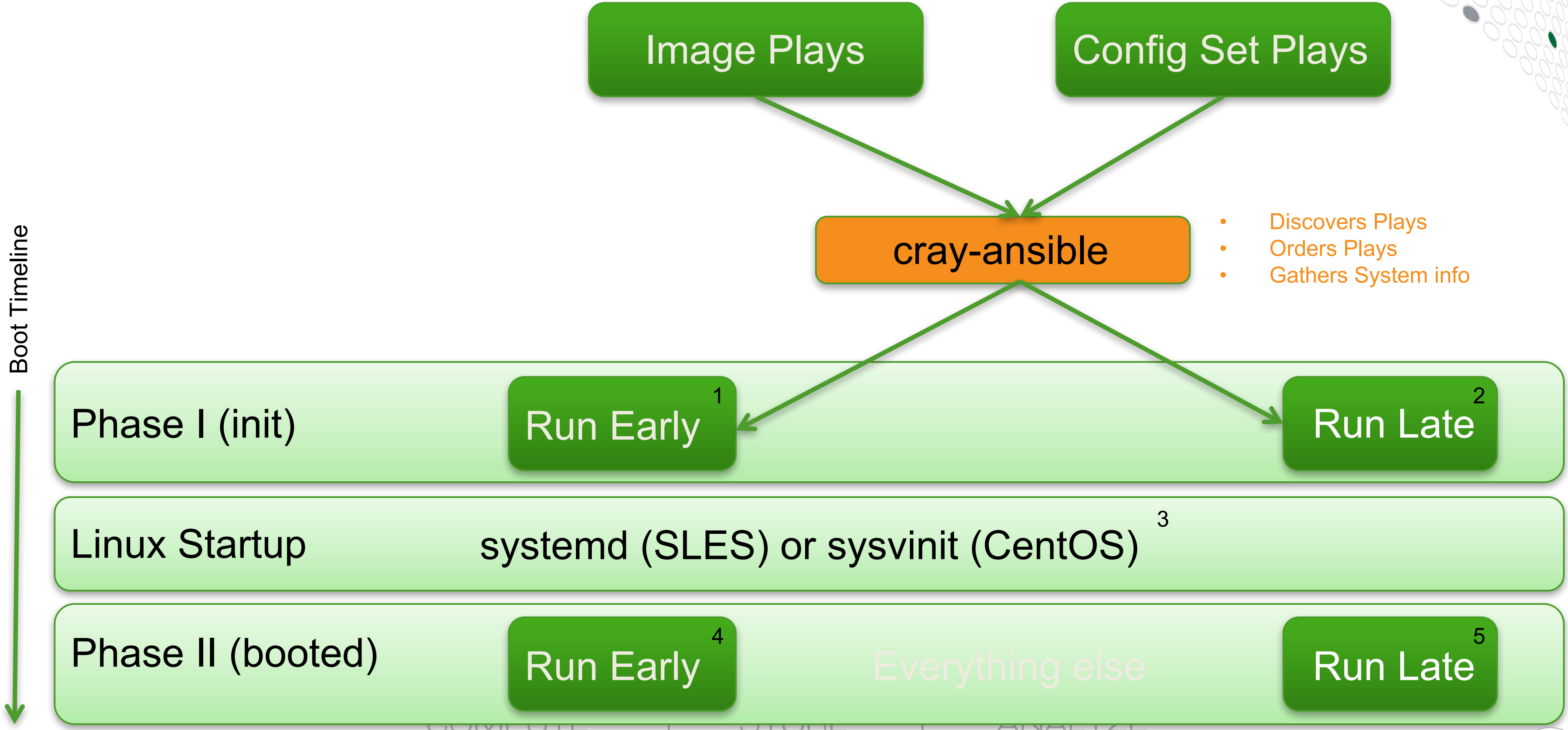
Play Ordering During Ansible Runs

- **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 in “run_early” container
  - *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



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”.
  - **This is deprecated now that node groups exist. Do not use “roles” in UP02 or later.**
- **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).
  - **This is deprecated now. Use hostid instead in UP02 or later.**
- **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 nodes
- **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 |
               ismember(cray_login.settings.login_nodes.data.members_groups)
    is_sdb:    ansible_local.cray_system.hostid |
               ismember(cray_sdb.settings.node_groups.data.sdb_groups)
    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 |
          ismember(cray_login.settings.login_nodes.data.members_groups)

- 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 |
          ismember(cray_login.settings.login_nodes.data.members_groups)
```

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:
      (not ansible_local.cray_system.hostid |
            ismember(cray_login.settings.login_nodes.data.member_groups)
      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: msg=“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 Facts and Config Set

- **Use ansible command to list facts available on a node**

  node# **ansible -m setup localhost | grep ansible_kernel**

  "ansible_kernel": "3.12.60-52.49.1_2.2-cray_ari_s"

- **View cray system facts on a node**

  node# **/etc/ansible/facts.d/cray_system.fact**

- **Explore the config set data from the SMW**

  smw# **cfgset search --level advanced -s cray_global_net -t admin global**

- **cfgsetquery searches for variable name and sub path matches, but provides the namespace path**

  node# **/opt/cray/cfgutils/bin/cfgsetquery networks.data**

Testing Ansible plays in config set

- **Switch between production and development configurations**
  - Clone the target CLE config set

    smw# **cfgset create --clone p0 p0.proving**
  - Clone the active NIMS map

    smw# **cmap list | grep -i true**

    smw# **cmap create --clone <grepped-map> p0.proving.map**
  - Set the new NIMS map active and update a test node to use new config set

    smw# **cmap setactive p0.proving.map**

    smw# **cnode update -c p0.proving <test-node>**
  - Adjust your CLE config set **AND/OR** add your site Ansible play to new config set

    smw# **cfgset update -m interactive -l advanced p0.proving**

    smw# **cp -pr myansibleplay /var/opt/cray/imps/config/sets/p0.proving/ansible/**
  - Test Ansible on node with or without rebooting it
  - Switch back to production by updating the active NIMS map

    smw# **cmap setactive <grepped-map>**

    smw# **xtbootsys --reboot <test-nodes>**

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 Limitations

- **Service configuration when “in_init”**
  - cray-ansible runs before systemd starts so that plays can influence which services will be started at boot by systemd
  - Because systemd not running, cannot use the Ansible 'service' module
  - Because systemd not running, cannot use systemctl enable/disable
  - Must make symbolic links
    - /etc/systemd/system/multi-user.target.wants
- **References are parsed even if skipped**
  - It is common for roles to use the set_fact module to update the data available for plays at runtime
  - This can lead to problems if the fact is referenced in some contexts later
  - If a constraint is placed on the role that causes the set_fact to be skipped, and a later task references the fact in a when clause, for instance, the fact will be undefined and cause the play to fail even though the same constraint that skipped the set_fact will skip the failing task
  - It is not always easy to tell whether a fact reference will be parsed by Ansible, but in cases where it does occur using the Jinja filter “|default(true)” will avoid the error by providing a value
  - Thorough testing on uninvolved nodes will help identify such issues

Ansible References

- **Cray publication:**
  - XC™ Series Ansible Play Writing Guide (CLE 6.0.UPxx S-2582)
- **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

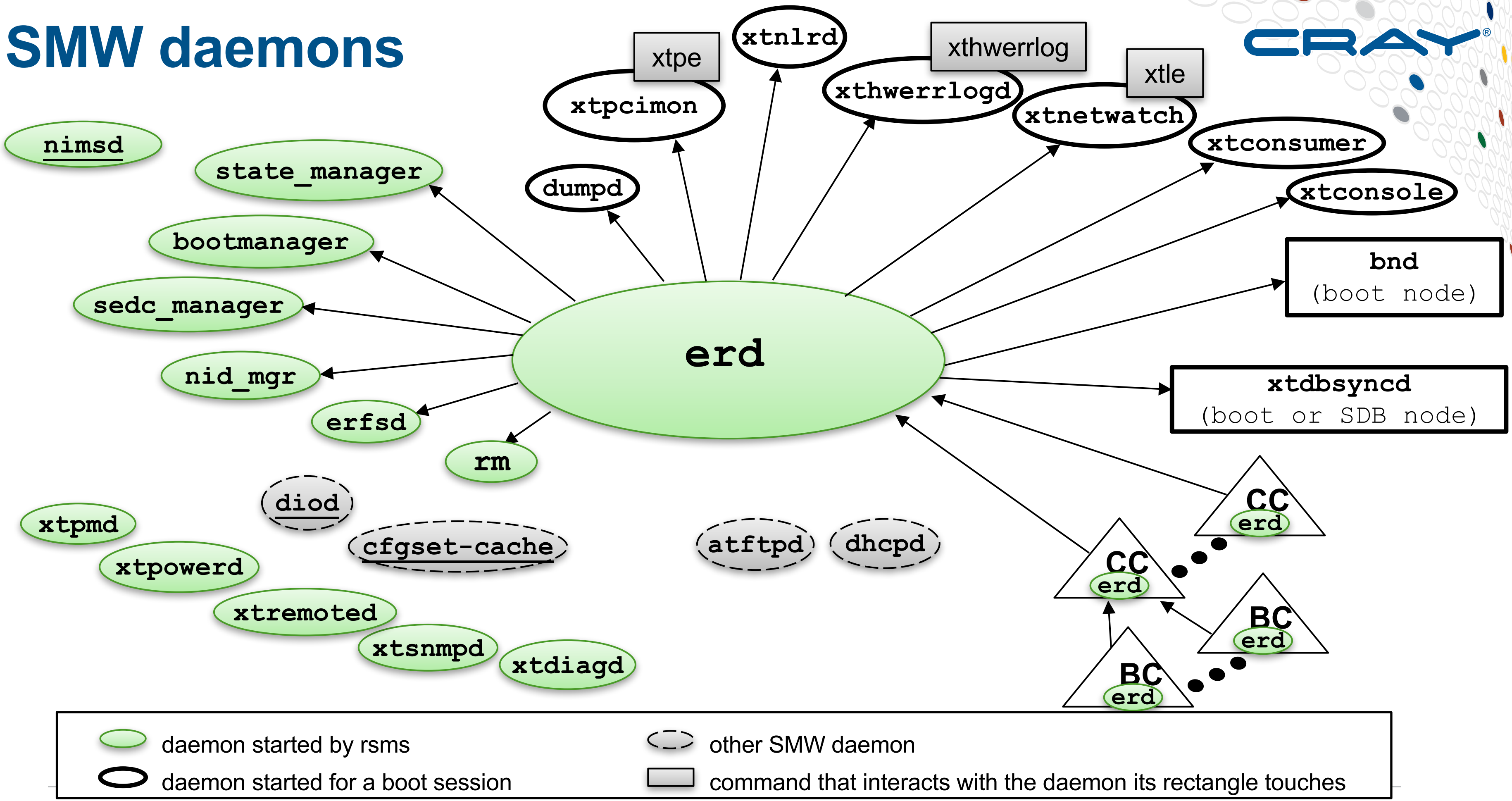
Agenda

- Introduction to SMW/CLE system management
- New system management features since UP01
- Best practices for using Ansible
- Troubleshooting XC system booting problems
- Migrating SMW/CLE software from 7.2/5.2 to 8.0/6.0
- Intro to CMC/eLogin system management
- Migrating CIMS/CDL to CMC/eLogin
- CLE Boot Performance and Reliability
- Q & A

Troubleshooting XC System Booting Problems

- SMW daemons, processes, and logs
- Anatomy of XC system boot
- Booting process from CLE node view
- Troubleshooting Commands
- SMW HA daemons
- SMW HA commands
- Troubleshooting Techniques
- DEBUG shell
- Further information:
  - XC™ Series Boot Troubleshooting Guide (CLE 6.0.UPxx S-2565)

SMW daemons



xtpe
xtnlrd
xthwerrlog
xtle
xtpcimon
xthwerrlogd
xtnetwatch
xtconsumer
xtconsole
nimsd
state_manager
dumpd
bootmanager
sedc_manager
nid_mgr
erfsd
rm
erd
bnd
(boot node)
xtdbsyncd
(boot or SDB node)
diod
cfgset-cache
atftpd
dhcpd
xtpmd
xtpowerd
xtremoted
xtsnmpd
xtdiagd
CC
erd
CC
erd
BC
erd
BC
erd
daemon started by rsms
daemon started for a boot session
other SMW daemon
command that interacts with the daemon its rectangle touches

SMW Logs

- **System-wide /var/opt/cray/log**
  - bootmanager: bm.out and bm.out.1 (previous)
  - erd: event-YYYYMMDD
  - nid_manager: nm.out and nm.out.1 (previous)
  - nimsd: nimsd.out and nimsd.out.1 (previous)
  - sedc_manager: sedc_manager.out and sedc_manager.out.1 (previous)
  - state_manager: sm.out and sm.out.1 (previous)
  - xtdiagd: xtdiagd.out and xtdiagd.out.1 (previous)
  - xtpmd: pmd.out and pmd.out.1 (previous)
  - xtpowerd: xtpowerd.out and xtpowerd.out.1 (previous)
  - xtremoted: xtremoted.out and xtremote.out.1 (previous)
  - xtsnmpd: xtsnmpd.out and xtnsnmpd.out.1 (previous)

SMW Logs

- **Boot session (started by xtbootsys)**
  - /var/opt/cray/log/session-ID
  - /var/opt/cray/log/p0-current links to current session-ID
    - xtbootsys log: bootinfo.session-ID
    - xtconsole: console-YYYYMMDD
    - xtconsumer: consumer-YYYYMMDD
    - xthwerrlogd: hwerrlog.session
    - xtnetwatch: netwatch-YYYYMMDD
    - xtpcimon: pcimon-YYYYMMDD
    - xtnlrd: nlrd.session-ID

SMW Logs

- **CC/BC**
  - On controller: /var/log
  - On SMW: /var/opt/cray/log/controller
    - cX-Y directory for each cabinet
      - Directory for each CC and BC
        - messages-YYYYMMDD
        - bios-n[0,1,2,3]-YYYYMMDD
- **Commands executed on SMW**
  - /var/opt/cray/log/xtdiscover – Only xtdiscover
  - /var/opt/cray/log/commands – most HSS commands

systemd Log

- **Dealing with systemd (new in 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
    - Display all kernel messages and other available information

      node# **journalctl –a**
    - Display all messages, but augment log lines with explanation from message catalog

      node# **journalctl -ax**
    - Display updates as they happen

      node# **journalctl -f**

Anatomy of XC System Boot

- **Booting XC system with xtbootsys**

  crayadm@smw> **xtbootsys -a auto.hostname.start**
- **xtbootsys runs several tasks before working on the tasks in the boot automation file**
- **Boot automation file initiates boot of nodes in a certain order**
  - Default for systems without DAL:
    1. Boot + SDB (if SDB image small enough to PXE boot)
    2. SDB (if SDB image too large to PXE boot)
    3. Service + Compute
  - Default for systems with DAL:
    1. Boot + SDB (if SDB image small enough to PXE boot)
    2. SDB (if SDB image too large to PXE boot)
    3. Service
    4. Compute
- **Additional actions in boot automation file can run commands before or after any of the above steps**
- **bootinfo log file shows every task executed and any output from each task**
  - Once the boot is complete, a summary will be added to the bootinfo log file with the names and duration of al tasks
- **bootinfo log**
  - /var/opt/cray/log/p0-current/bootinfo-YYYYMMDD
- **When analyzing a failed boot, check the bootinfo file for that failed boot session**

bootinfo-YYYYMMDD Boot Time Statistics

TASK	CONCURRENT	DURATION
initialization		0m0s
xtcli_part_cfg_show		0m4s
user_input		0m4s
analyze_archive		1m12s
xtcli_status_a		0m0s
xtcli_status_lcb		0m0s
verify_nodelists		0m0s
clean_up_old_daemons		0m1s
Internal		0m22s
disable_flood_control		0m1s
start_xtconsole		0m0s
config_bcsysd		0m0s
config_bcbwtd		0m0s
xtbounce		3m53s

Timings are representational only. Sample from a small system without KNL and without DAL.

bootinfo-YYYYMMDD Boot Time Statistics

TASK	CONCURRENT	DURATION
cable_check		0m1s
xthwinv		0m2s
xthwinv_X		0m1s
xtsdbhwcache		0m1s
xtclear_alert		0m0s
xtclear_warn		0m1s
route_setup		0m0s
start_xtconsole_1		0m0s
start_xtnetwatch		0m0s
start_xtpcimon		0m0s
start_dumpd		0m0s
start_xthwerrlogd		0m0s
start_xtnlrd		0m0s
start_xtwatcher		0m0s
crms_exec		0m4s

Timings are representational only. Sample from a small system without KNL and without DAL

bootinfo-YYYYMMDD Boot Time Statistics

TASK	CONCURRENT	DURATION
crms_set_failed_option		0m0s
crms_set_failed_timeout		0m0s
boot_bootnode_sdbnode		0m22s
wait_for_bootnode_sdbnode	YES	4m32s
extract_debug	YES	0m6s
extract_debug_1	YES	0m3s
extract_debug_2	YES	0m5s
extract_debug_3	YES	0m6s
crms_set_failed_option_1		0m0s
crms_set_failed_timeout_1		0m0s
boot_all		1m4s
wait_for_all		6m5s
crms_exec_via_bootnode		0m1s
gather_ko		0m4s
gather_fstab		0m3s
clean_up YES		0m1s
enable_flood_control	YES	0m1s
Total		17m55s

Timings are representational only. Sample from a small system without KNL and without DAL

Booting Process from CLE Node View – Boot Image

- **xtbounce triggers the node power on**
  - node runs node BIOS
  - Successful completion of node BIOS leaves the message "Wait4boot" on the console
- **xtbootsys calls "xtcli boot" for the node**
  - Nodes PXE booting (boot and sometimes SDB)
    - The node requests an IP address from the SMW via DHCP
    - The boot image is transferred via the TFTP over the SMW's eth3 to the node's eth0 network connection
  - Nodes HSN booting (all other nodes)
    - bnd on the boot node will extract files from the boot image to transfer to the node's memory

Booting Process from CLE Node View - cray-ansible

- /init from the boot image executes
- /init calls cray-ansible in init_netroot_setup (ONLY if netroot node)
  - If this fails, then the node will drop into the DEBUG shell
  - If it succeeds, then /init continues
- /init calls cray-ansible in the init phase
  - If this fails, then the node will drop into the DEBUG shell
  - If it succeeds, then /init continues
- /init finishes and transfers control to systemd
- systemd mounts file systems from /etc/fstab, starts all enabled services, and so forth
- cray-ansible runs in the booted phase
  - If it fails, then a cray-ansible failed message to console
  - If it succeeds, then a cray-ansible succeeded and a boot succeeded message to console

Node Image at Boot time

kernel loaded	
/init bootstrapping	
cray-ansible in /init (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

cray-ansible and Ansible Logs on a CLE Node

- **/init calls cray-ansible in the init netroot setup phase (only if node is using netroot)**
  - /var/opt/cray/log/ansible/sitelog-init-netroot_setup
  - /var/opt/cray/log/ansible/file-changelog-init-netroot_setup
  - /var/opt/cray/log/ansible/file-changelog-init-netroot_setup.yaml
- **/init calls cray-ansible in the init phase**
  - /var/opt/cray/log/ansible/sitelog-init
  - /var/opt/cray/log/ansible/file-changelog-init
  - /var/opt/cray/log/ansible/file-changelog-init.yaml
- **systemd runs cray-ansible in the booted phase**
  - /var/opt/cray/log/ansible/sitelog-booted
  - /var/opt/cray/log/ansible/file-changelog-booted
  - /var/opt/cray/log/ansible/file-changelog-booted.yaml

Troubleshooting Commands – see Guide

- RSMS daemons
- diod daemon
- cray-cfgset-cache daemon
- DHCP and TFTP daemons
- Console messages
- xtcon
- xtalive
- stonith
- xtcablecheck
- xthwinv
- xtcli part_cfg
- xtcli status
- Changing node between service and compute
- NIMS map (cmap)
- NIMS node information (cnode)
- Bootimages and image roots
- Tools to check network traffic
- Firewall/iptables
- Searching a config set
- Searching Ansible playbooks in config set and image root
- Searching Ansible plays on a node
- Checking for warnings, alerts, reservations
- Checking for locks
- Checking for PCIe link errors
- Checking for hardware errors
- Checking for LCB and router errors

SMW HA Daemons

- **SLE HA Pacemaker daemons**
  - pacemakerd – Pacemaker cluster resource manager
  - cib: cluster information base daemon
  - stonithd: STONITH daemon (reset or power down failed node)
  - lrmd: local resource manager daemon
  - attrd: attribute daemon
  - pengine: policy engine daemon
  - crmd: cluster resource manager daemon
- **Logs**
  - /var/log/pacemaker.log
  - /var/log/smwha.log

SMW HA Commands

- **Is HA cluster configured and healthy?**

  smw# **/opt/cray/ha-smw/default/sbin/ha_health**

  Cluster State

  Health State : Healthy

  Active Node : minnie

  Node-1 : mickey (online)

  Node-2 : minnie (online)

  Number of Resources : 33

  Number of Resources Running : 33

  Number of Resources Stopped : 0

  Maintenance Mode : disabled

  Stonith Mode : enabled
- **Check status of cluster resources**

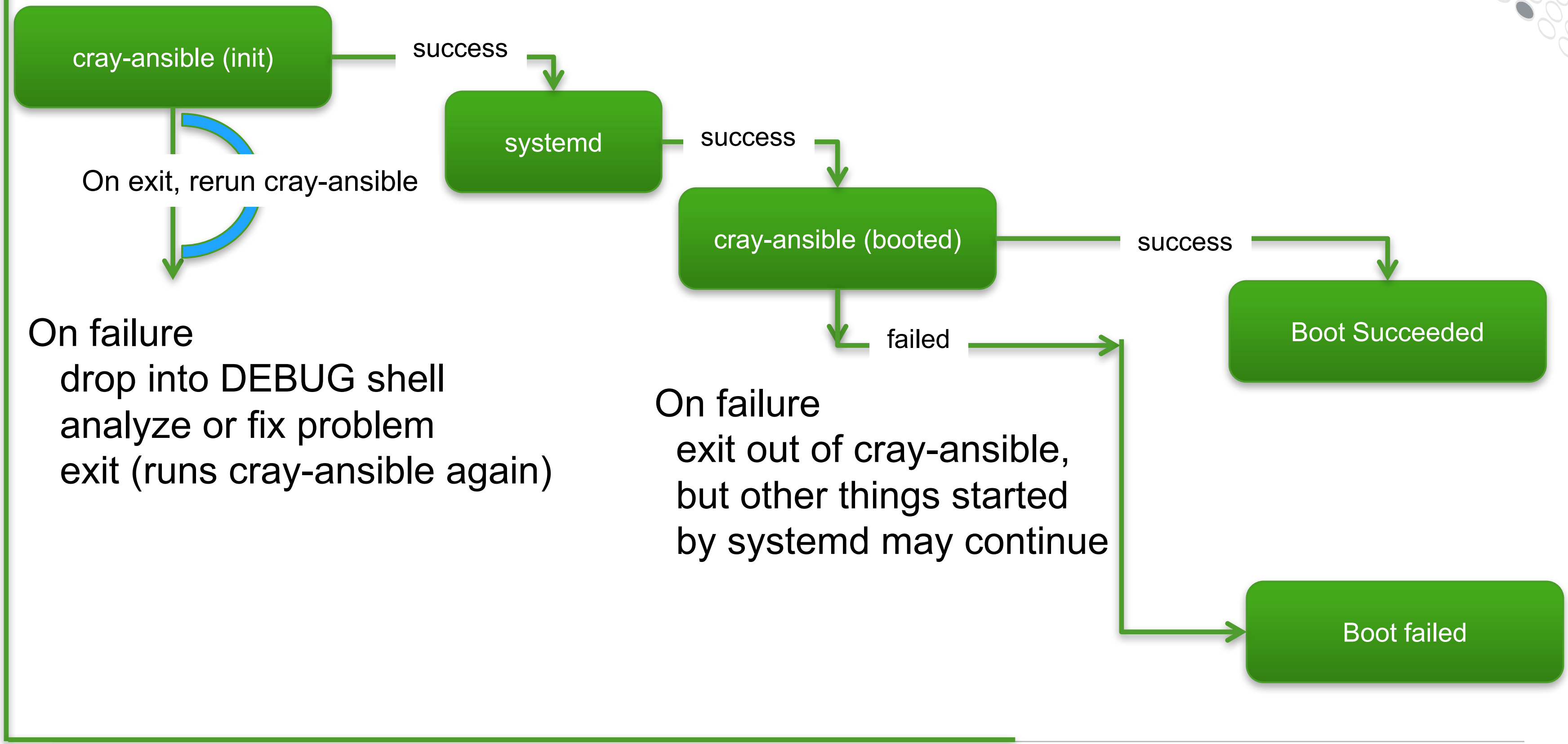
  smw# **crm_mon -1r**
  - All services should be in "Started" state and on the active node, except for stonith (one on each SMW)
- **Check health of DRBD sync of PostgreSQL database between smw1 and smw2**

  smw# **cat /proc/drbd**

Troubleshooting Techniques

- **XC™ Series Boot Troubleshooting Guide (CLE 6.0.UPxx S-2565)**
  - List of over 30 potential errors and where to look further
    - xtbootsys related failures
    - cray-ansible related failures
    - Node mount failures
    - Node network interface failures
    - Content from Netroot, diags, and PE image root failures
    - IDS failures

Troubleshoot A Boot – Boot Flow and Exits



Troubleshoot A Boot

- **Detect Ansible failure in /init for a node**
  - Check the console log for the node

    smw# **grep FAILED /var/opt/cray/log/p0-current/console-***

    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 and rerun cray-ansible, 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

- **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, xtattr
  - 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
```

Troubleshoot A Boot – Check Setting in Config Set

- **Use search to print out the entire config**
  - To narrow a search, use state and level filters

  smw# **cfgset search --level advanced --state set p0**

- **Use search to locate settings in a config set**

  smw# **cfgset search --term myvalue CONFIGSET**

- **Search tips:**
  - To broaden a search, use multiple search terms (a logical OR)
  - Unlike the create and update subcommands, the search subcommand has a default value of all for the state filter

Troubleshoot A Boot – Check Setting in Config Set

- **Search for the terms c0-0c0s1n1 and lus/ in settings of any level in config set p0:**

  smw# **cfgset search --term c0-0c0s1n1 --term lus/ --level advanced p0**
  # 1 match for 'c0-0c0s1n1' from cray_scalable_services_config.yaml
  #--
  cray_scalable_services_data.settings.scalable_service.data.tier1: c0-0c0s0n1, c0-0c0s1n1
  # 1 match for 'lus/' from cray_node_health_config.yaml
  #--
  cray_node_health_.settings.filesys_plugins.data.Default Filesystem.path: /lus/case1
  ...(more matches not included in example)

- **To output more information about the fields and values that match the search term(s) with level, state, and default value**
  - smw# **cfgset search --term myvalue –format full CONFIGSET**

Troubleshoot A Boot – List Playbooks

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

```
smw# module load system-config
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
```

Troubleshoot A Boot – Search Playbooks

- Search the Ansible playbooks in a config set and image root for the set_hostname.yaml playbook

```
smw# module load system-config
smw# ansible_cfg_search -p set_hostname.yaml p0 custom_compute_cle
```

- Search the Ansible playbooks to find which plays do something with the setting cray_alps.settings.common.data.xthostname

```
smw# module load system-config
smw# ansible_cfg_search p0 service_cle_6.0.UP01-build6.0.96_sles_12-created20160614 -s cray_alps.settings.common.data.xthostname
```

DEBUG Shell

- /init script has breakpoints that enable a user to examine various system values and files during the boot
- Check /init for the image root related to the node's boot image on the SMW in /var/opt/cray/imps/image_roots/custom_compute_cle

```
${DEBUG} && echo "DEBUG SHELL: in setup_netroot; exit will init vars" > ${con_debug}
${DEBUG} && echo "DEBUG SHELL: in setup_netroot; exit will contruct netroot" > ${con_debug}
${DEBUG} && echo "DEBUG SHELL: prior to DVS lower mount; exit will resume" > ${con_debug}
${DEBUG} && echo "DEBUG SHELL: prior to chroot prep; exit will resume" > ${con_debug}
${DEBUG} && echo "DEBUG SHELL: post netroot preload debug; exit will resume" > ${con_debug}
${DEBUG} && echo "DEBUG SHELL: prior to mounting merge layer tmpfs" > ${con_debug}
${DEBUG} && echo "DEBUG SHELL: prior to chroot Ansible; exit will resume" > ${con_debug}
```

DEBUG Shell

- Connect to console of node (in 1st window)

  smw# **xtcon c8-0c2s8n1**
- Set DEBUG kernel parameter for a node (in 2nd window)

  smw# **cnode update -k DEBUG=true c8-0c2s8n1**

  smw# **cnode list c8-0c2s8n1**
- Reboot node (in 2nd window)

  crayadm@smw> **xtbootsys --reboot -r "testing init" c8-0c2s8n1**
- Interact with DEBUG shell in 1st window
- Remove DEBUG kernel option when done

  smw# **cnode update -K DEBUG c8-0c2s8n1**

  smw# **cnode list c8-0c2s8n1**

Agenda

- Introduction to SMW/CLE system management
- New system management features since UP01
- Best practices for using Ansible
- Troubleshooting XC system booting problems
- **Migrating SMW/CLE software from 7.2/5.2 to 8.0/6.0**
- Intro to CMC/eLogin system management
- Migrating CIMS/CDL to CMC/eLogin
- CLE Boot Performance and Reliability
- Q & A

SMW/CLE Migration

- **SMW 8.0/CLE 6.0 requires a fresh install of software on the SMW**
  - Leaves behind
    - SLES11SP3
    - bootroot and sharedroot filesystems
    - xtopview for specialized /etc (default, class, node) views in sharedroot
    - persistent /var filesystem for service nodes
  - Changes/adds
    - Based on SLES12 operating system
    - New filesystem layout on SMW
    - New filesystem layout on boot RAID
    - LVM with BTRFS and XFS filesystems
    - Persistent (nonvolatile) storage for nodes not just in /var
    - Uses repositories for zypper/yum on SMW and CLE images

SMW/CLE Migration

- **Migration goal**
  - Minimize system downtime while preserving configuration and operational data
- **Migration SMW used for preparation**
  - Additional physical SMW with additional boot RAID
- **Scope**
  - SMW 7.2.UP04/CLE 5.2.UP04 migration to SMW 8.0.UP03/CLE 6.0.UP03
  - XC series systems only—not XE/XK
  - SMW HA system from SLEHA11.SP3.UP02 to SLEHA12.SP0.UP03
- **Migration caveats**
  - Process to preserve DAL (Lustre) filesystem untested, but should not require reformatting
  - DAL LMT database not migrated
  - DataWarp Intel P3608 SSDs may need to be reformatted unless FN6121a was already applied
  - DataWarp Fusion IO ioMemory3 (SX300) are supported, but not other versions from Fusion IO
  - DataWarp Fusion IO ioMemory3 SSDs will be flashed to newer firmware for CLE 6.0.UP03 which is incompatible with old CLE 5.2.UP04

SMW/CLE Migration – Migration Service Offering

- Communication on Cray's Migration Service offering will be available through your Account Manager
- See Cray Field Notice (FN6149) for details
- Cray will open cases against each XC asset running CLE 5.2 to determine/track:
  - Long-term plans (e.g. no migration to 6.0 or desired timeframe)
  - Migration planning including hardware needs
- Necessary hardware required will be determined on a site-by-site basis and will be covered under the Migration Service Offering

SMW/CLE Migration – Process Phases

- Training
- Planning
- Install migration SMW
- Prepare config data and images
- Preserve other data before shutdown
- Shutdown and switch

SMW/CLE Migration– Cray Training

- **Cray training with lectures and hands-on experience in labs**
  - 4 day course on Cray XC Series System Administration with SMW 8.0/CLE 6.0 content
  - 2 day course on eLogin
- **Both online and instructor lead courses available to customers in Chippewa Falls, Wisconsin, USA or at a customer site**
- **Sign up for classes and see pricing by starting at our Web site:**
  - cray.com → Support → Training
    - Or this link: http://www.cray.com/support/training/schedule
  - Contract Cray Training for scheduling and pricing:
    - registrar@cray.com (or call: +1-715-726-4036)

SMW/CLE Migration – Training

- **New system management topics highlighted in Migration guide**
  - Cray Scalable Services
  - Cray XC System Configuration
  - Config Sets
  - Config Set Caching
  - Variable Names in the Configurator and Configuration Worksheets
  - Node Groups
  - Simple Sync
  - Boot Automation Files
  - Snapshots and Config Set Backups during a Migration
  - Install Third-Party Software with a Custom Image Recipe

SMW/CLE Migration – Documentation

- **CrayPort http://crayport.cray.com and CrayDoc https://pubs.cray.com**
  - What's New for CLE 6.0 and SMW 8.0 (CLE 6.0 UPxx S-2573)
  - XC™ Series Ansible Play Writing Guide (CLE 6.0.UPxx S-2582)
  - XC™ Series Boot Troubleshooting Guide (CLE 6.0.UPxx S-2565)
  - XC™ Series CLE 5.2 to CLE 6.0 Software Migration Overview (CLE 6.0.UP03 S-2574)
  - XC™ Series CLE 5.2 to CLE 6.0 Software Migration Using a Physical SMW (CLE 6.0.UP03 S-2580)
  - XC™ Series eLogin Administration Guide (CLE 6.0.UPxx S-2570)
  - XC™ Series eLogin Installation Guide (CLE 6.0.UPxx S-2566)
  - XC™ Series esLogin to eLogin Migration Guide (CLE 6.0.UP03 S-2584)
  - XC™ Series SMW HA Installation Guide (CLE 6.0 UPxx S-xxx)
  - XC™ Series SMW HA XC Administration Guide (CLE 6.0 UPxx S-0044)
  - XC™ Series System Administration Guide (CLE 6.0.UPxx S-2393)
  - XC™ Series System Configurator User Guide (CLE 6.0.UPxx S-2560)

New commands in SMW 8.0/CLE 6.0

- **IMPS commands**
  - recipe, pkgcoll, repo, image
- **CMF**
  - cfgset, cray-ansible, ansible_cfg_search
- **NIMS**
  - cmap, cnode
- **General**
  - imgbuilder, snaputil, cnat, cnat-status

SMW/CLE Migration - Planning

- **Cray Service will work with customer to plan hardware considerations for migration**
  - Additional SMW and boot RAID
  - Is SDB node Ethernet connected to SMW?
  - How many tier2 nodes will be needed?
  - Is the system SMW HA?
  - Does the XC system have KNC nodes?
    - Not supported with SMW 8.0/CLE 6.0

SMW/CLE Migration - Planning

- **Plan 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

SMW/CLE Migration – Migration SMW

- **Additional SMW and Boot RAID**
  - The CLE 6.0/SMW 8.0 software is installed on this SMW before any of the other preparation work is done
  - Much of the configuration can be done without being connected to XC hardware

SMW/CLE Migration – Preparation

- **Prepare config data and images**
  - Archive SMW 7.2.UP04/CLE 5.2.UP04 configuration
    - Save files and output from running probe commands
  - Transform data to config set
    - Translation tables for config worksheets
    - Load and validate worksheets
  - Manipulate images
    - Choose image recipes to build
    - Build image roots and boot images
    - Assign kernel parameters to nodes
    - Check NIMS data
  - Identify and port site-local scripts

SMW/CLE Migration – Translation Tables

Column in table	Description
Setting/Field Name	Name of setting in config service
Default	Default value for the setting
Level	required, basic, or advanced
Probe	Suggested command to probe SMW 7.2/CLE 5.2 system and where it should be run
Files/Installer	Context of where to find file (SMW, bootroot, default shareroot, class sharedroot, or node sharedroot); the path to file; variable within file OR A variable in SMWinstall.conf or CLEinstall.conf

SMW/CLE Migration – Preparation

- **No tools for SMW/CLE config translation**
  - All data transformation must be done by documented procedures
- **Config set data entry for migration**
  - global config set – About 450 settings in 9 config services
  - CLE config set – About 850 settings in 49 config services
    - Plus 21 settings for each CLE node with a network interface
    - 200 network nodes = 4200 settings
  - Total settings – About 1300-5500

SMW/CLE Migration – Final Shutdown

- **Preserve other data before shutdown**
  - Run final accounting reports
  - Save operational data
  - Save site user data
  - Drain WLM queues

SMW/CLE Migration – Shutdown and Switch

- **Additional SMW and boot RAID**
  - Shut down the CLE system
  - (SMW HA only) Put the SMW HA cluster in maintenance mode
  - Switch cabling to the migration SMW and boot RAID
  - Discover XC system hardware and update firmware on components
  - Complete CLE configuration with hardware connected
  - Complete first boot of CLE nodes with new software
  - Configure other SMW 8.0/CLE 6.0 features and services and install additional software (including SMW HA)
  - Restore any SMW 7.2/CLE 5.2 operational data (files, database exports, site user data, and site-local scripts)

Agenda

- Introduction to SMW/CLE system management
- New system management features since UP01
- Best practices for using Ansible
- Troubleshooting XC system booting problems
- Migrating SMW/CLE software from 7.2/5.2 to 8.0/6.0
- **Intro to CMC/eLogin system management**
- Migrating CIMS/CDL to CMC/eLogin
- CLE Boot Performance and Reliability
- Q & A

Intro to CMC/eLogin System Management

- **What is CMC/eLogin?**
- **Key d**ifferences between CIMS/esLogin and CMC/eLogin
- **System topology**
- **eLogin image management**
- eLogin image configuration
- Cray System Management Software (CSMS)
  - eLogin Node configuration
  - eLogin Node inventory
  - eLogin Node deployment
  - Troubleshooting

What is CMC/eLogin?

- **CMC: Cray Management Controller**
  - Replaces CIMS/esMS server
  - Manages eLogin nodes
  - Uses Cray System Management Software (CSMS)
    - Replaces Bright Cluster Manager software
    - Built on OpenStack

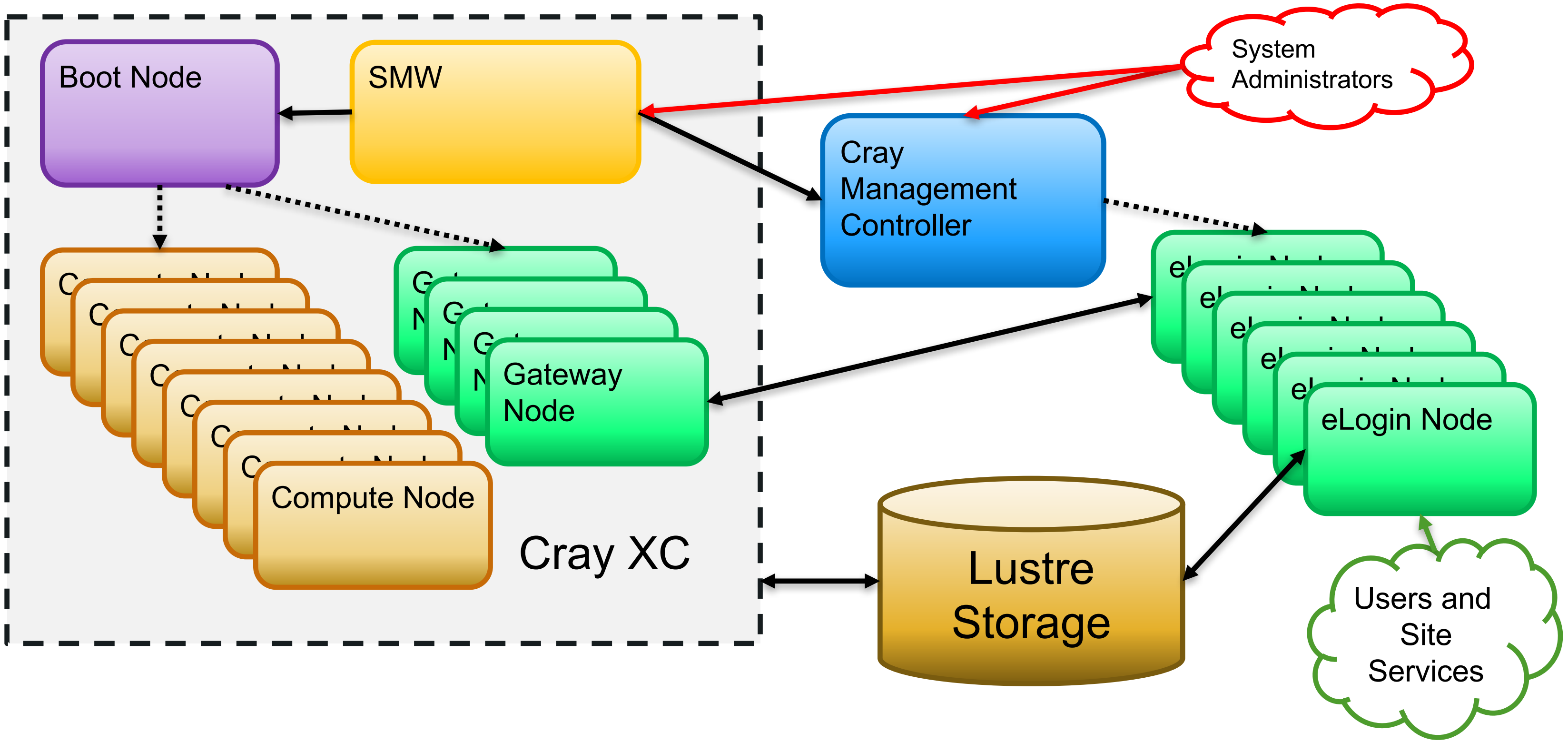
- **eLogin: Cray Development and Login server (CDL)**
  - Provides a login, job submission, and development environment for CLE 6.0.UP03
  - Replaces esLogin CDL which supported CLE 5.x
  - Available to users independent of the availability of THE CRAY® XC™ SERIES

CMC/eLogin and CIMS/esLogin Compared

- **Key Differences**

	CMC/eLogin	CIMS/esLogin
Image Management	• Prescriptively built on the SMW from CLE, eLogin, and any customer required sources. • Image is exported to the CMC for deployment to eLogin nodes.	• ESL image ISO released by Cray. • Installed on the CIMS for deployment to esLogin nodes.
Image Configuration	• eLogin configuration is provided by the Cray CLE Configuration Set created on the SMW. • Configuration Set is pushed to CMC for use by eLogin nodes.	• Bright Cluster Manager
Programming Environment	• Shared Cray PE image synchronized to each eLogin node from the CMC.	• Cray PE installed to the ESL image on the CIMS.
System Management	• Cray System Management Software	• Bright Cluster Manager

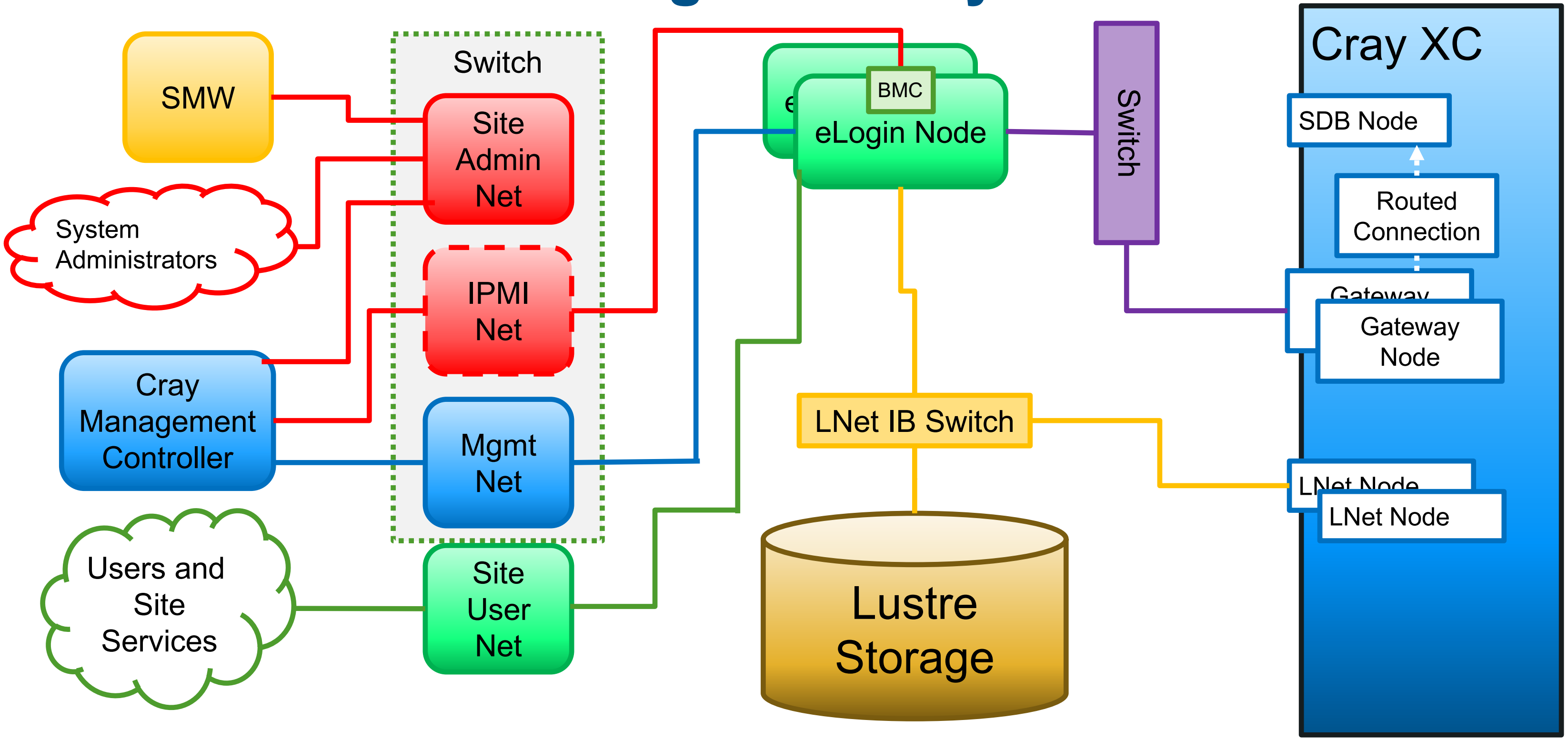
CMC/eLogin System Topology: It is the same as CIMS/esLogin



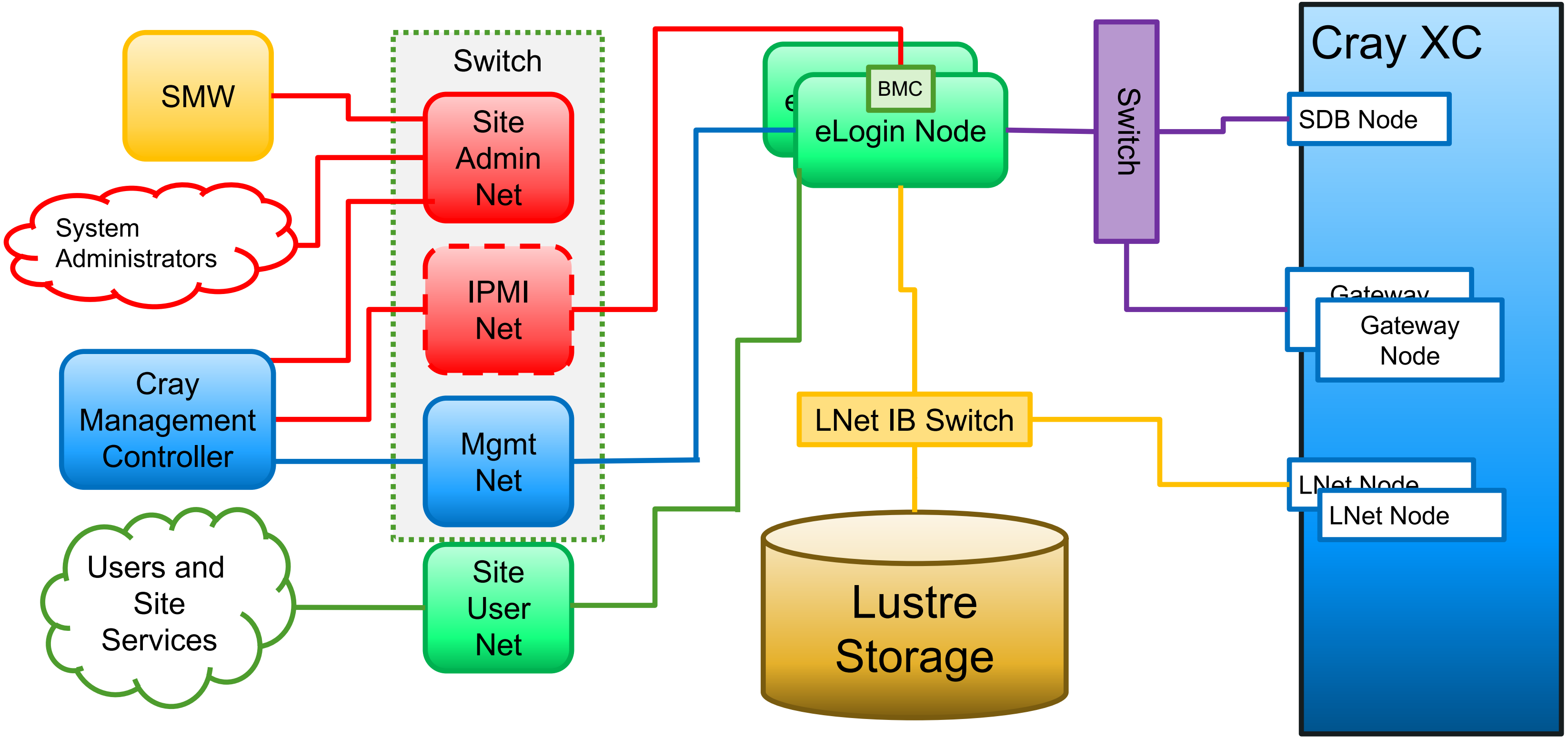
eLogin Networking to the Cray XC

- **4 supported topologies**
  - Differ in connection to the Gateway and SDB nodes of the THE CRAY® XC™ SERIES system
  - Gateway Node connection
    - Direct connection over private network
    - Connect over the Site User network
  - SDB node connection
    - Routed via Gateway Node
    - Direct connection over private network

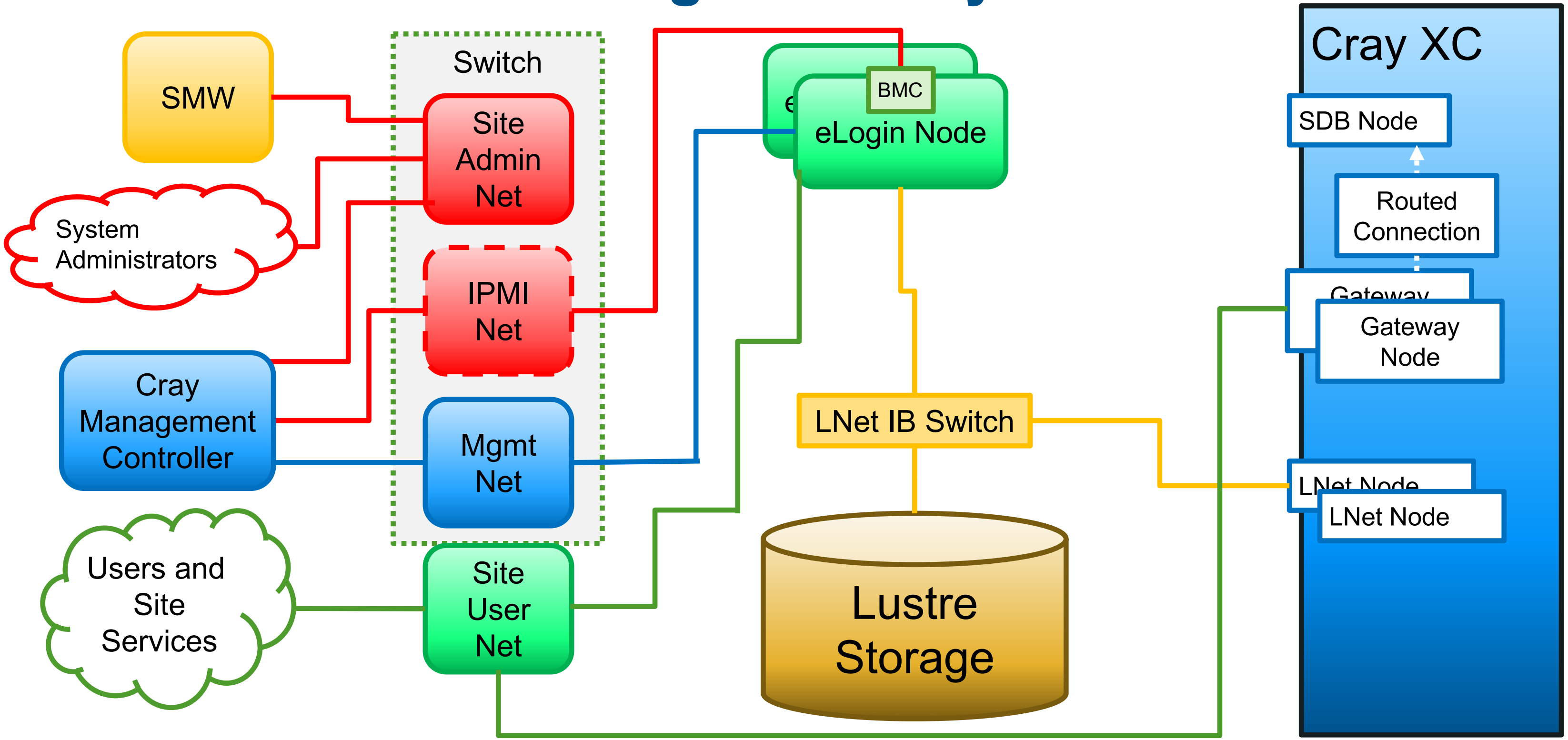
Gateway Nodes on a private network
SDB Node routed through Gateway Nodes



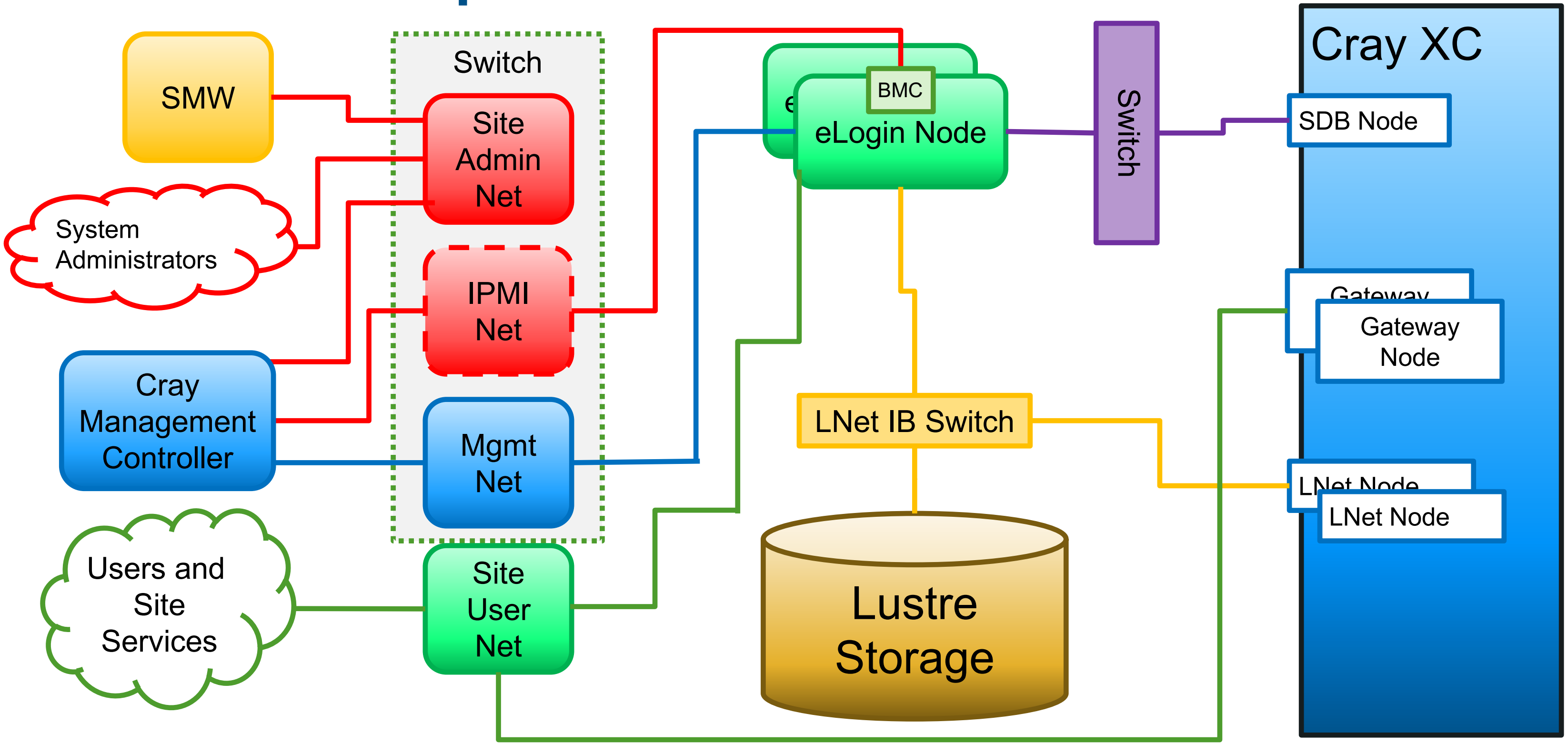
Gateway and SDB Nodes on a private network



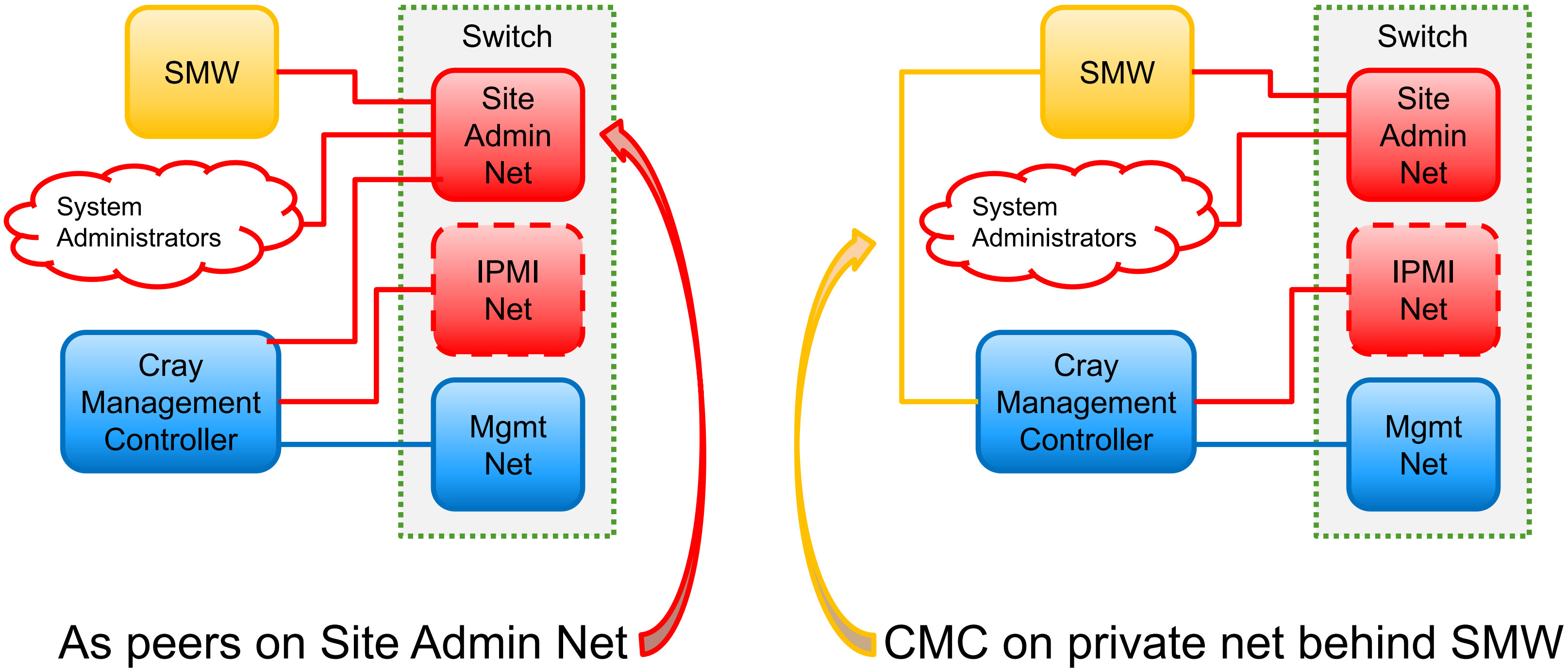
Gateway Nodes on Site User Net
SDB Node routed through Gateway Nodes



Gateway Nodes on Site User Net
SDB Node on a private network



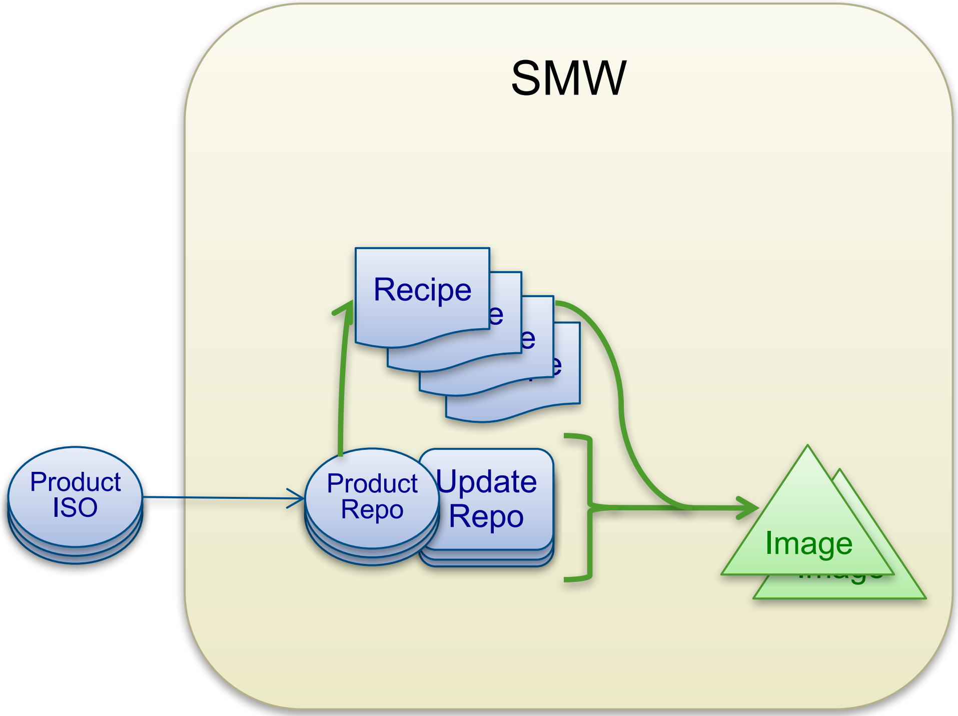
System Topology:
SMW to Cray Management Controller (CMC) options



eLogin Image Management

- **Same as Cray CLE images – IMPS**
- **Image Components**
  - SLES 12
  - Lustre Client
  - Cray eproxy (formerly eswrap)
    - Allows a set of Cray XC and WLM commands to be executed from the eLogin
  - Any customer required packages
- **Cray Programming Environment (Cray PE)**
  - Separate from the eLogin OS image
  - Same Cray PE as the Cray XC
  - Synchronized to local persistent storage on the eLogin node

eLogin Image Management: Image Creation



- **Images are built on the SMW**
  - Package repositories are under: /var/opt/cray/repos
  - All rpm dependencies are resolved from these repositories
- **Prescribed by Image Recipes**
  - Images are rooted under: /var/opt/cray/imps/image_roots
  - eLogin recipes based on CLE recipes plus eLogin specific packages
  - Recipes may be cloned and modified
  - Images must be exported to the CMC for eLogin deployment

eLogin Image Management: Image Export to the Cray Management Controller

- **Images must be exported to the CMC any time they are changed**

```
smw:~ # image export elogin_image_20170319 --format qcow2 \
        -d glance:example-cmc:elogin_image_20170319
```

  - The “**elogin_image_20170319**” image will be converted to **qcow2** format and exported to the **glance** image service on the CMC named “**example-cmc**”
  - The image will be registered in **glance** as “**elogin_image_20170319**”

- Raw image format is also supported
  - For raw images, it is suggested to add ”.raw” to the end of the image name

eLogin Image Management: Image Export to the Cray Management Controller

- **NOTE: Pushing the same image more than once**
  - You **MUST** login to the CMC and delete the existing image from glance or you will have two images with the same name registered!
  - cmc: # **source /root/admin.openrc**
    cmc: # **glance image-delete <image_name>**
  - In the case of images exported in qcow2 format, there are three images to delete
    - <image_name>.qcow2
    - <image_name>.initramfs
    - <image_name>.kernel

eLogin Image Configuration

- **Same as Cray CLE images – Cray Management Framework (CMF)**
  - Configuration data is separate from the image
  - eLogin configuration data is in the same configuration set as Cray CLE
- **Configuration sets are created and maintained on the SMW**
  - Must be pushed to the CMC and registered any time they are changed
  - `smw|cmc:/var/opt/cray/imps/config/sets/<config_set_name>`
- **More than one configuration set can exist to support alternative configurations**
  - Only one configuration set is active on a given eLogin node at a time

eLogin Image Configuration

- **Pushing configuration sets to the CMC**
  - Both the global and cle config sets must be pushed to the CMC

```
smw # cfgset push -d <cmc-name> global
smw # cfgset push -d <cmc-name> <config_set_name>
```

  - Config set is filtered for eLogin and stored in an OpenStack Swift container for deployment
    - add_configset **must** be run any time the config set is modified

```
cmc # add_configset –c <config_set_name> -e
/etc/opt/cray/elogin/exclude/<exclude_list>
```

eLogin Image Configuration

- **add_configset default exclude list**

```
worksheets
config/cray_sdb_config.yaml                # sdb configuration
files/roles/common/etc/ssh                 # ssh keys
files/roles/common/root                    # ssh and nodehealth
#files/roles/munge                         # munge
files/roles/common/etc/opt/cray/xtremoted-agent
files/roles/merge_account_files   # site provided user account info
```

eLogin Image Configuration

- **Config set on an eLogin node is at:**

```
elogin:/etc/opt/cray/config/current
elogin:/etc/opt/cray/config/global
```

- **/etc/opt/cray/config/current contains the following subdirectories**
  - ansible, config, dist, files
    - config subdirectory contains the configuration files

Configuration Sets: Services Shared with Cray CLE

- **eLogin references several Cray CLE services:**
  - cray_local_users - local users configuration
  - cray_time - settings for various aspects of time including ntp, timezone
  - cray_user_settings - user environment settings
  - cray_auth - user authentication settings - LDAP, NIS, etc.
  - cray_ssh - SSH settings
  - cray_lustre_client - Lustre Client settings
  - cray_net - management, site, and LNet network settings
  - cray_simple_sync - a generic method of distributing files to targeted locations on the eLogin nodes

Configuration Sets: eLogin Specific Services

- **eLogin references the following eLogin specific services:**
  - cray_elogin_lnet – Lustre Network (LNet) settings for eLogin
  - cray_elogin_networking – eLogin postfix configuration
  - cray_eswrap – settings for executing certain Cray XC and WLM commands

Agenda: CMC/eLogin Overview

- ~~**Intro to CMC/eLogin System Management**~~
  - ~~What is CMC/eLogin?~~
  - ~~Key differences between CIMS/esLogin and CMC/eLogin~~
  - ~~System topology~~
  - ~~eLogin image management~~
  - ~~eLogin image configuration~~
  - Cray System Management Software (CSMS)
    - eLogin Node configuration
    - eLogin Node inventory
    - eLogin Node deployment
    - Troubleshooting

Introduction to the Cray Management Controller
Cray System Management Software (CSMS)

- **Overview**
  - Three main software components
    - Base Operating System is CentOS 7
      - Installed via Cray Bootable CentOS ISO image
    - Cray System Management Software
      - Installed via Cray System Management Software ISO image
    - eLogin support software
      - Installed via eLogin Installation ISO image

Introduction to the Cray Management Controller Cray System Management Software (CSMS)

- Leverages OpenStack services
  - Keystone – Authentication between OpenStack services
  - Nova – eLogin lifecycle management
  - Ironic with Cray Fuel – eLogin Bare metal provisioning
  - Glance - Image service
  - Swift – Object storage (backs Glance) – storage for eLogin config set and other config data
  - Neutron – Networking service for CMC
  - Heat – Orchestration – used for deploying eLogin nodes, calls Nova and Ironic
- Provides remote console and console logging
- eLogin syslogs are forwarded to the Cray Management Controller

Managing eLogin Nodes: Image Registration

- **eLogin images are registered on the CMC (*Glance*)**
  - Images can be in raw or qcow2 format
    - "image export" command takes care of image registration from the SMW

- **Local Disk partitioning is separate from the image**
  - eLogin nodes have two disk devices
    - One for the eLogin image
    - One for persistent storage – config set data and Cray PE
  - Partitioning information is contained in the *deploy_config_elogin.json* file
    - Registered in the Glance service
    - Installed as part of the eLogin installation process
    - Can set to verify partitioning, or clean and repartition on every deployment

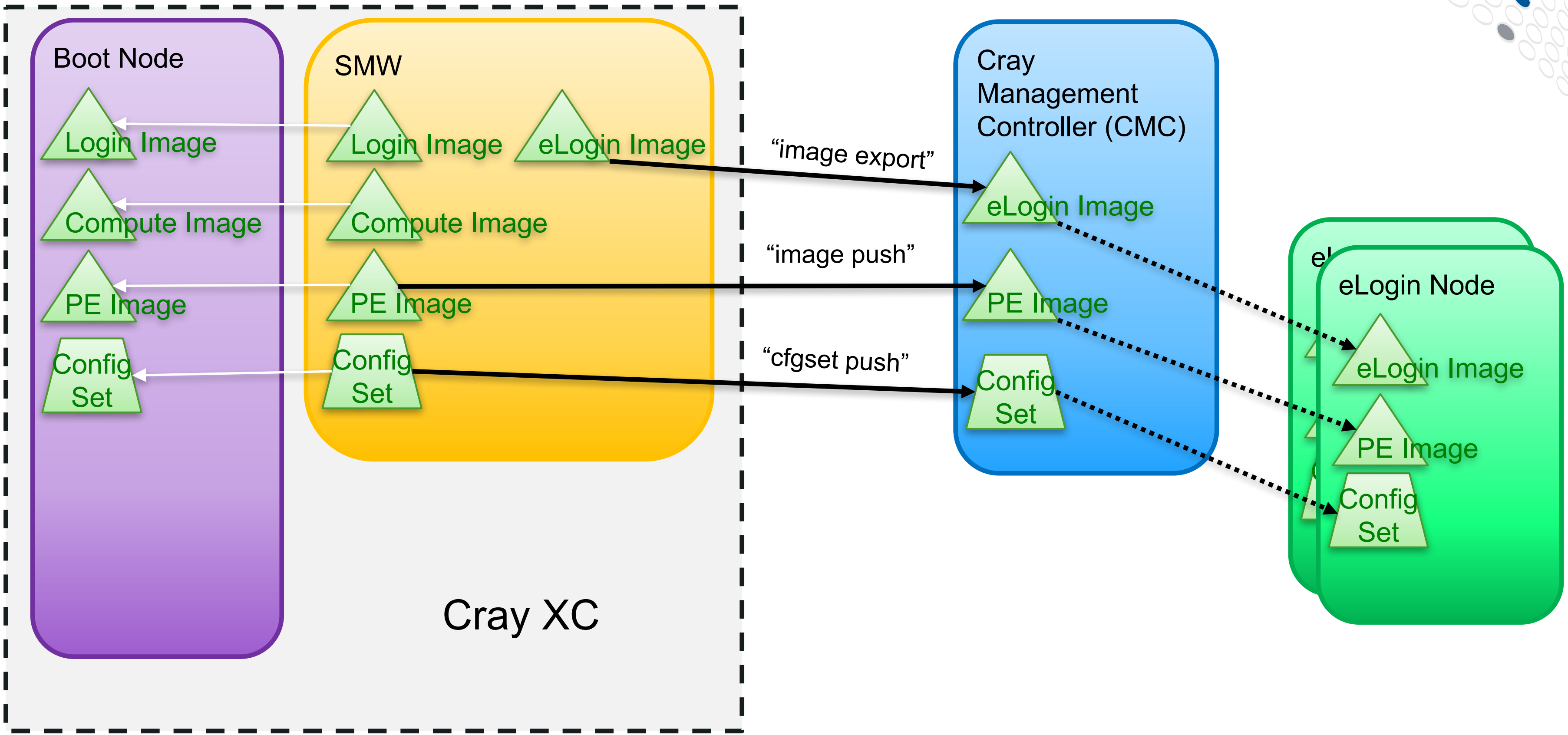
Managing eLogin Nodes: Cray PE

- **Cray Programming Environment (Cray PE)**
  - Pushed to the CMC from the SMW
  - Same exact Cray PE image that the XC is using!

```
smw # image push -d <cmc-name> <pe_compute_image>
```

- **eLogin syncs Cray PE from the CMC server**
  - Differs from esLogin where Cray PE was installed directly to the esLogin image which caused these images to grow very large
  - Cray PE syncs to the persistent disk, not the eLogin image disk

Managing eLogin Nodes: Image Flow



Managing eLogin Nodes: Image Configuration

- **Configuration performed locally on the node**
  - Config set is retrieved from the OpenStack Swift container by an ”action script”
  - Sites may add their own Ansible plays to the config set
  - **cray-ansible** finds all Ansible plays and executes them
  - Ansible plays are packaged with their application software
    - eLogin plays get packaged with the eLogin software

- **cray-ansible runs in the pre-pivot init phase of the eLogin image boot process**

Managing eLogin Nodes: Extending Config

- **Cray's Configuration Management Framework (CMF) allows additional configuration tasks**
  - Add site-specific tasks in concert with Cray-provided Ansible boot-time execution

- **As with CLE, site Ansible plays may perform the following tasks on eLogin**
  - Start/stop services
  - Change crontab entries
  - Modify files
  - Copy files

Managing eLogin Nodes: Simple Sync

- **Alternative method of installing files to eLogin nodes other than using a site Ansible play**
  - Simple rsync of files from a root in the config set container to "/" on the eLogin node
  - Files can be installed by hostname or by node_group membership

  - To install a file to a specific eLogin node, place it under the following directory on the CMC

```
/var/opt/cray/imps/config/sets/<config_set>/files/simple_sync/hostnames/<host_name>/files/<path_to_file_on_elogin>
```

- To install a file to a group of eLogin nodes, place it under the following directory on the CMC. The <node_group> must be created in the config set prior to being able to use it.

```
/var/opt/cray/imps/config/sets/<config_set>/files/simple_sync/nodegroups/<node_group>/files/<path_to_file_on_elogin>
```

Managing eLogin Nodes: Node Inventory

- Nodes are registered with CSMS using an inventory file

  /etc/opt/cray/openstack/ansible/inventory.csv

- Template can be found at:

/etc/opt/cray/openstack/ansible/roles/ironic_enrollment/files/example_inventory.csv

- Node inventory example:

```
NODE_NAME,   BMC_IP,       MAC_ADDR,          N_CPUs, ARCH,   RAM_MB, DISK_GB, NODE_DESC
elogin1, 10.142.0.5, 14:fe:b5:ca:b7:00, 32, x86_64, 131072, 550, elogin1
elogin2, 10.142.0.6, e0:db:55:0a:25:a8, 32, x86_64, 131072, 550, elogin2
elogin3, 10.142.0.7, 78:2b:cb:33:4b:b7, 32, x86_64, 131072, 550, elogin3
```

Managing eLogin Nodes: Node Inventory

- **Register nodes with the csms_ironic_enrollment script**

```
example-cmc # cd /etc/opt/cray/openstack/ansible/
example-cmc # ./csms_ironic_enrollment.sh
```

- Registers each node with the ironic service (bare-metal nodes)
- Assigns the node installer image to the node
- Creates a nova “flavor” named “ironic_flavor” matching the hardware configuration
  - Nova is the scheduler for tenant instances (deployed nodes)
  - Flavors define the minimum required hardware specifications
- Nodes must be connected to the IPMI and management networks
- Nodes must be powered off

Managing eLogin Nodes: Node Inventory

- List registered eLogin nodes

```
example-cmc # ironic node-list
+--------------------------------------+---------+---------------+-------------+--------------------+-------------+
| UUID                                 | Name    | Instance UUID | Power State | Provisioning State | Maintenance |
+--------------------------------------+---------+---------------+-------------+--------------------+-------------+
| 692e40b2-ff8c-4842-b9bf-8c6957256b23 | elogin1 | None          | power off   | available          | False       |
| 3c0a3cd6-eb76-4ab1-85e3-615d867a66a0 | elogin2 | None          | power off   | available          | False       |
| c0386c4d-9410-4113-a71b-2a770b6239df | elogin3 | None          | power off   | available          | False       |
+--------------------------------------+---------+---------------+-------------+--------------------+-------------+
```

Managing eLogin Nodes: Node Inventory

- **List node "ports" (network interfaces) for elogin3**
  - This is the interface that elogin3 will boot over
  - All other ports are configured by cray-ansible during deployment

```
example-cmc # ironic node-port-list elogin3
+--------------------------------------+-------------------+
| UUID                                 | Address           |
+--------------------------------------+-------------------+
| e23aff8f-25af-4b9d-89ff-a68f9ed54bf8 | 78:2b:cb:33:4b:b7 |
+--------------------------------------+-------------------+
```

Managing eLogin Nodes: Nova Flavors

```
example-cmc # nova flavor-list
+--------------------------------------+---------------+-----------+------+-----------+-------+-------+-------------+-----------+
| ID                                   | Name          | Memory_MB | Disk | Ephemeral | Swap  | VCPUs | RXTX_Factor | Is_Public |
+--------------------------------------+---------------+-----------+------+-----------+-------+-------+-------------+-----------+
| 7b26a552-d079-4635-a4a6-1bec1cd7b902 | ironic_flavor | 131072    | 550  | 0         |       | 32    | 1.0         | True      |
| aac33543-4d49-44c0-8e22-22a02537f2ee | eloginflavor  | 65536     | 100  | 0         | 16384 | 16    | 1.0         | True      |
+--------------------------------------+---------------+-----------+------+-----------+-------+-------+-------------+-----------+
```

Managing eLogin Nodes: Node Deployment

- **Heat Orchestration service used for node deployment**
  - Orchestrates node stack deployments
  - Uses a template plus an environment file to deploy eLogin stacks
    - Template file is common to all eLogin nodes
    - Environment file is node specific
      - Provides values for the heat stack template
    - CMC location:
      - `/etc/opt/cray/openstack/heat/templates`

Managing eLogin Nodes: Node Deployment

- **Heat stack template files for eLogin**
  - 2 templates provided
  - elogin_template.yaml
    - Dynamic management IP address assigned by the CMC (neutron)
  - elogin_template_fixed_ip.yaml
    - Static management IP address

Managing eLogin Nodes: Node Deployment

- **Heat environment template files for eLogin**
  - 2 environment templates to provide parameters to the stack templates
    - Use the one matching the eLogin heat stack template chosen
    - elogin-env.yaml.template
      - For use with elogin_template.yaml
    - elogin-env-fixed-ip.yaml.template
      - For use with elogin_template_fixed_ip.yaml

Managing eLogin Nodes: Node Deployment

- Heat environment template file for eLogin

```
# cat elogin-env-fixed-ip.yaml.template
---
# An example env.yaml file to use with the Heat templates when a fixed
# management IP address is desired.
# Copyright 2016 Cray Inc. All Rights Reserved.

parameters:
  image_id: elogin_image.qcow2
  host_name: elogin1
  fixed_ip: 10.142.0.100
  instance_flavor: eloginflavor
  cray_config_set: p0
  cims_host_name: example-cmc
  ironic_id: 7baa31f0-07c8-42e9-8743-ae5f147f78c3
  actions_list: copy_p0
```

Managing eLogin Nodes: Node Deployment

- **Deploy script for eLogin**
  - Copy deploy_elogin.sh.template to deploy_<elogin_name>.sh
  - Edit with appropriate settings – TEMPLATE_FILE, ENV_FILE, STACK_NAME

```
# cat deploy_elogin1.sh
#!/bin/bash
#
# A template eLogin deploy script.
# Copyright 2015 Cray Inc. All Rights Reserved.
# Edit these values to the correct values for the node to be deployed
# Use full paths only
TEMPLATE_FILE=/etc/opt/cray/openstack/heat/templates/elogin-env-fixed-ip.yaml
ENV_FILE=/etc/opt/cray/openstack/heat/templates/elogin1-env.yaml
STACK_NAME=elogin1

source ~/admin.openrc
heat stack-create -f $TEMPLATE_FILE -e $ENV_FILE $STACK_NAME
```

Managing eLogin Nodes: Node Deployment Checking heat, nova and ironic data

- Deploy elogin1 by running deploy_elogin1.sh

```
# ./deploy_elogin1.sh
# heat stack-list
+--------------------------------------+------------+-----------------+----------------------+
| id                                   | stack_name | stack_status    | creation_time        |
+--------------------------------------+------------+-----------------+----------------------+
| 956e7974-7557-4091-b749-2833827718f3 | elogin1    | CREATE_COMPLETE | 2016-03-01T22:45:33Z |
+--------------------------------------+------------+-----------------+----------------------+
# nova list
+--------------------------------------+---------+--------+------------+-------------+------------------------+
| ID                                   | Name    | Status | Task State | Power State | Networks               |
+--------------------------------------+---------+--------+------------+-------------+------------------------+
| d72227dc-bfd6-4c60-988b-b152a7bd821a | elogin1 | ACTIVE | -          | Running     | management=10.142.0.176 |
+--------------------------------------+---------+--------+------------+-------------+------------------------+
# ironic node-list
+--------------------------------------+---------+--------------------------------------+-------------+--------------------+-------------+
| UUID                                 | Name    | Instance UUID                        | Power State | Provisioning State | Maintenance |
+--------------------------------------+---------+--------------------------------------+-------------+--------------------+-------------+
| 50698545-ce51-41b3-b873-a3d2dbaf8d79 | elogin1 | d72227dc-bfd6-4c60-988b-b152a7bd821a | power on    | active             | False       |
+--------------------------------------+---------+--------------------------------------+-------------+--------------------+-------------+
```

Managing eLogin Nodes: Node Deployment

- Link between nova and ironic

```
# ./deploy_elogin1.sh
# heat stack-list
+--------------------------------------+------------+-----------------+----------------------+
| id                                   | stack_name | stack_status    | creation_time        |
+--------------------------------------+------------+-----------------+----------------------+
| 956e7974-7557-4091-b749-2833827718f3 | elogin1    | CREATE_COMPLETE | 2016-03-01T22:45:33Z |
+--------------------------------------+------------+-----------------+----------------------+
# nova list
+--------------------------------------+---------+--------+------------+-------------+-------------------------+
| ID                                   | Name    | Status | Task State | Power State | Networks                |
+--------------------------------------+---------+--------+------------+-------------+-------------------------+
| d72227dc-bfd6-4c60-988b-b152a7bd821a | elogin1 | ACTIVE | -          | Running     | management=10.142.0.176 |
+--------------------------------------+---------+--------+------------+-------------+-------------------------+
# ironic node-list
+--------------------------------------+---------+--------------------------------------+-------------+--------------------+-------------+
| UUID                                 | Name    | Instance UUID                        | Power State | Provisioning State | Maintenance |
+--------------------------------------+---------+--------------------------------------+-------------+--------------------+-------------+
| 50698545-ce51-41b3-b873-a3d2dbaf8d79 | elogin1 | d72227dc-bfd6-4c60-988b-b152a7bd821a | power on    | active             | False       |
+--------------------------------------+---------+--------------------------------------+-------------+--------------------+-------------+
```

Managing eLogin Nodes: Node Deployment
Useful heat stack commands

- **heat stack-list**
  - Lists all heat stacks
- **heat stack-show <stack_name>**
  - Displays information about the prescribed software stack
- **heat stack-delete <stack_name>**
  - Tears down the software stack and powers off the node

Managing eLogin Nodes: Node Console

- **Remote console can be accessed by the ironic_conman command**

```
# ironic_conman <hostname>
```

- **Console traffic is logged to /var/log/conman on the CMC**
  - Logs are named by ironic node UUID
    - ironic-c0386c4d-9410-4113-a71b-2a770b6239df.log

Managing eLogin Nodes: Deploy Process

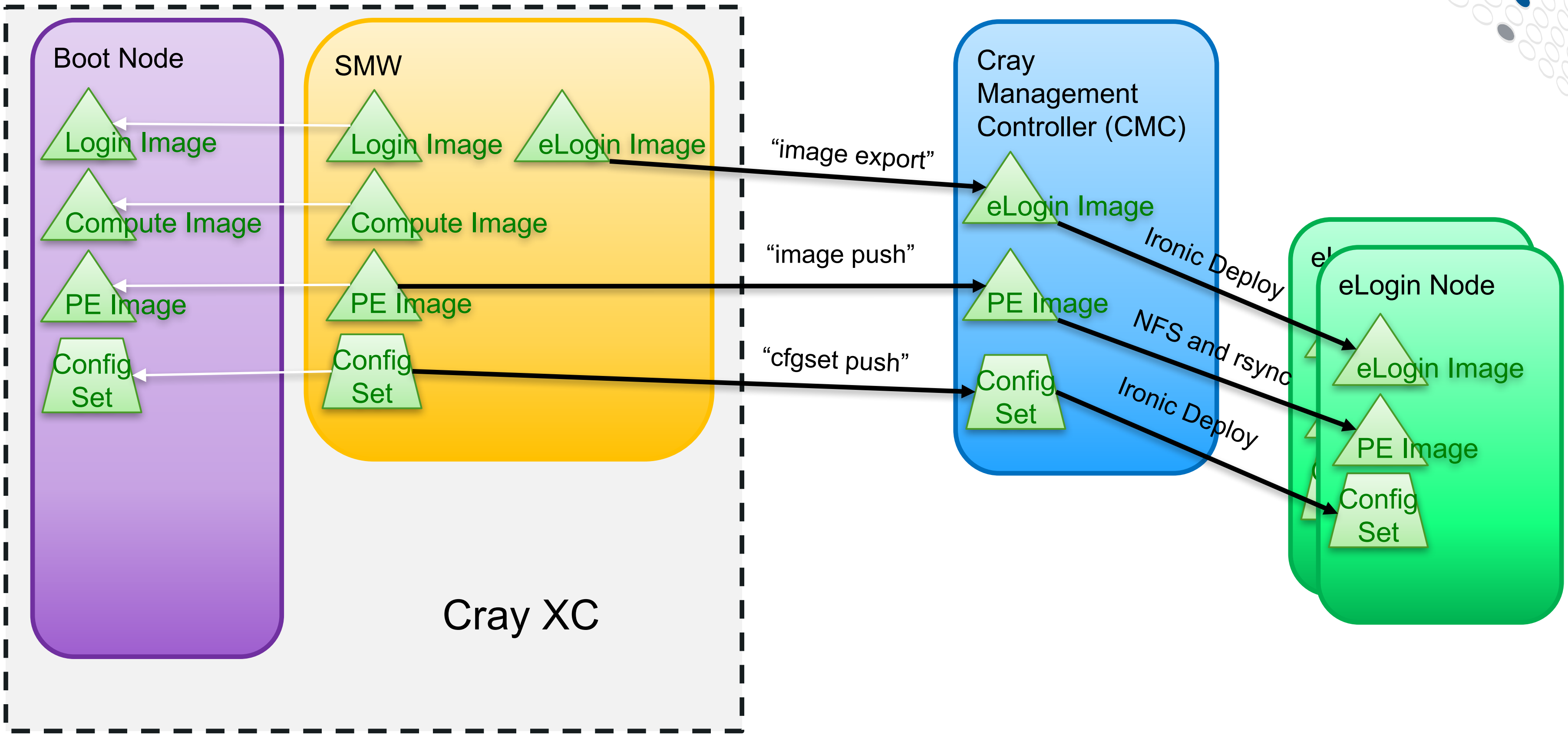
- **Node deployment first boots a common deploy image**
  - Heat calls Nova which looks to see if your eLogin node is available for deployment
  - Nova tells Ironic to power on the eLogin node
  - eLogin node PXE boots a common deployment image
    - Cray Fuel driver checks local disk partitioning and repartitions to match that described in the deploy_config_elogin.json file
    - Mounts the local disk and rsyncs the eLogin image to the node
  - Config Set is retrieved by the action script
  - Reboots the node from the eLogin image on the local disk
  - Heat and Nova are now done

Managing eLogin Nodes: Boot Process

- **Node boots into a dracut pre-init environment**
  - cray-ansible is run to configure the node
  - Non-root users are not allowed to login during this time
  - Cray PE is synchronized to persistent storage on the eLogin node
    - First sync can be long (all data transferred)
    - Subsequent syncs are image diffs only
      - Sync progress may be monitored on the eLogin node

```
elogin # tail -f /var/opt/cray/persistent/pe_sync.log
```

Introduction to eLogin: Node Deploy Flow



Managing eLogin Nodes: Node Shutdown

- **Shutdown with ’heat stack-delete <stack_name>’**
  - Tears down the stack
  - Powers off the node
  - Removes the nova instance
  - Removes the heat stack

```
# heat stack-delete elogin1
```

  - It is important to use this method when shutting down eLogin nodes
- **Reboot nodes with ‘nova reboot <hostname>’**

Managing eLogin Nodes: Things to Know

- **CSMS will enforce the power state of the node**
  - Manually powering “on” a node that is set to “power off” in CSMS (*ironic*) will result in the node being powered off. The reverse is also true.
  - Set the node to “maintenance mode” in ironic to avoid this enforcement
- **eLogin images must be exported to the CMC after being created or edited on the SMW**
- **Config Sets must be pushed to the CMC after being created or edited on the SMW**
  - “add_configset” must also be run after a config set is pushed
  - An action script must exist in glance for each config set

Managing eLogin Nodes: Things to Know

- **Changing eLogin hardware requires updating the inventory.csv file and the elogin-env.yaml file**
  - Copy original inventory.csv file to inventory.csv.back
  - Delete all entries except for the hardware being updated
  - Run `csms_ironic_enrollment.sh`
  - Update the environment template for this node with the new ironic node UUID

Managing eLogin Nodes: Troubleshooting

- **cray_dumpsys**
  - Gathers data to help debug Cray System Management Software (CSMS) problems
  - Dumps the state of the OpenStack services, various configuration and log files plus background information about the system
  - Files are compressed and the results are stored in /var/tmp/
  - By default, only recent logs are dumped
    - Use --all-logs option to dump all rotated logs
    - Use --days option to dump logs up to a certain number of days

Managing eLogin Nodes: Troubleshooting

- **cray_dumpsys eLogin plugin**
  - Includes information from eLogin nodes in the cray_dumpsys report
  - Edit /etc/cray_tools/cray_tools.conf
    - Add elogin to the list of enabled plugins, and a space-separated list of eLogin node names in the elogin.nodes option.
  - To override the configured node list, use the cray_dumpsys option --extra-option elogin.nodes="elogin1 elogin2"

Managing eLogin Nodes: Troubleshooting cray_tools.conf example enabling eLogin plugin

```
[cray_dumpsys]
# List of enabled Cray dumpsys plugins.
plugins =
    process,
    networking,
    memory,
    openvswitch,
    mysql,
    openstack_cinder,
    openstack_horizon,
    openstack_keystone,
    openstack_neutron,
    openstack_nova,
    openstack_swift,
    newtplugin,
    elogin

# List of Cray dumpsys plugin options.
options =
    openstack_cinder.log=off,
    openstack_horizon.log=off,
    openstack_keystone.log=off,
    openstack_nova.cmds=on,
    openstack_nova.log=off,
    openstack_swift.log=off,
    elogin.nodes="elogin1 elogin2"
```

Managing eLogin Nodes: Troubleshooting

- **Logs**
  - /var/log/messages

  - OpenStack Logs (on the CMC node)
    - /var/log/cinder – block storage service
    - /var/log/glance – image service
    - /var/log/heat – orchestration service
    - /var/log/ironic – bare metal provisioning service
    - /var/log/keystone – identity and authentication service
    - /var/log/neutron – networking service
    - /var/log/nova – node scheduling service
    - /var/log/swift – object storage service (backs glance in CSMS)

  - Ansible Install Logs (on the eLogin node)
    - /var/opt/cray/log/ansible/ansible-init
    - /var/opt/cray/log/ansible/ansible-booted

Managing eLogin Nodes: Troubleshooting Diagnostics: Heat Stacks (1 of 2)

- **Heat Stacks**
  - heat stacks are used for deploying eLogin nodes

```
example-cmc # heat stack-show elogin1
+----------------------+------------------------------------------------------------------------------------------------------------------+
| Property             | Value                                                                                                            |
+----------------------+------------------------------------------------------------------------------------------------------------------+
| capabilities         | []                                                                                                               |
| creation_time        | 2015-06-11T20:52:39Z                                                                                             |
| description          | Simple deploy template with parameters                                                                           |
| disable_rollback     | True                                                                                                             |
| id                   | 4452df3e-46f1-4345-8b61-c489bbbc863f                                                                             |
| links                | http://172.30.50.129:8004/v1/acc067874bfd45dcbce9f44d1516910a/stacks/elogin1/4452df3e-46f1-4345-8b61-c489bbbc863f (self) |
| notification_topics  | []                                                                                                               |
| outputs              | [                                                                                                                |
|                      |   {                                                                                                              |
|                      |     "output_value": {                                                                                            |
|                      |       "management": [                                                                                            |
|                      |         "10.142.0.156"                                                                                           |
|                      |       ]                                                                                                          |
|                      |     },                                                                                                           |
|                      |     "description": "IP assigned to the instance",                                                                |
|                      |     "output_key": "instance_ip"                                                                                  |
|                      |   }                                                                                                              |
|                      | ]                                                                                                                |
```

Managing eLogin Nodes: Troubleshooting Diagnostics: Heat Stacks (2 of 2)

- ## Heat Stacks

```
example-cmc # heat stack-show elogin
| parameters            | {                                                                                |
|                       |   "network_id": "management",                                                    |
|                       |   "OS::stack_id": "4452df3e-46f1-4345-8b61-c489bbbc863f",                        |
|                       |   "OS::stack_name": "elogin1",                                                   |
|                       |   "cray_config_set": “p0-elogin",                                                |
|                       |   "key_name": "default",                                                         |
|                       |   "instance_flavor": "eloginflavor",                                             |
|                       |   "cray_cims_ip": "10.142.0.1",                                                  |
|                       |   "image_id": "elogin1.qcow2",                                                   |
|                       |   "host_name": "elogin1"                                                         |
|                       | }                                                                                |
| parent                | None                                                                             |
| stack_name            | elogin1                                                                          |
| stack_owner           | admin                                                                            |
| stack_status          | CREATE_COMPLETE                                                                  |
| stack_status_reason   | Stack CREATE completed successfully                                              |
| template_description  | Simple deploy template with parameters                                           |
| timeout_mins          | None                                                                             |
| updated_time          | None                                                                             |
+-----------------------+----------------------------------------------------------------------------------+
```

Managing eLogin Nodes: Troubleshooting Diagnostics: Nova (1 of 2)

- **Nova (active servers)**
  - Use nova show to look for details about the eLogin in question

```
example-cmc # nova show elogin1
+--------------------------------------+----------------------------------------------------------+
| Property                             | Value                                                    |
+--------------------------------------+----------------------------------------------------------+
| OS-DCF:diskConfig                    | MANUAL                                                   |
| OS-EXT-AZ:availability_zone          | nova                                                     |
| OS-EXT-SRV-ATTR:host                 | csms                                                     |
| OS-EXT-SRV-ATTR:hypervisor_hostname  | e63ffc33-029f-44ac-8808-c55909f85f2f                     |
| OS-EXT-SRV-ATTR:instance_name        | instance-00000050                                        |
| OS-EXT-STS:power_state               | 1                                                        |
| OS-EXT-STS:task_state                | -                                                        |
| OS-EXT-STS:vm_state                  | active                                                   |
| OS-SRV-USG:launched_at               | 2015-06-11T21:01:16.000000                               |
| OS-SRV-USG:terminated_at             | -                                                        |
| accessIPv4                           |                                                          |
| accessIPv6                           |                                                          |
| config_drive                         |                                                          |
| created                              | 2015-06-11T20:52:40Z                                     |
```

Managing eLogin Nodes: Troubleshooting Diagnostics: Nova (2 of 2)

- **Nova (active servers)**
  - Use nova show to look for details about the eLogin in question

```
| hostId                              | 9e184dc6993ac9954652611f13f3faaaa797b5ff1625869be0edeb80             |
| id                                  | ac6384e2-4ca0-421f-9e6e-4c9e138f8785                                 |
| image                               | elogin1.qcow2 (1cc535c0-9f71-446a-8f4e-66aacc2617fe)                 |
| key_name                            | default                                                              |
| management network                  | 10.142.0.156                                                         |
| metadata                            | {"cray_config_set": “p0-elogin", "cray_cims_ip": "10.142.0.1",       |
|                                     |  "cray_cims_rsync_password": "daf5ba09-6be4-4e50-bf43-7ba54394aca4", |
|                                     |  "cray_cims_rsync_username": "elogin"}                               |
| name                                | elogin1                                                              |
| os-extended-volumes:volumes_attached | []                                                                  |
| progress                            | 0                                                                    |
| security_groups                     | default                                                              |
| status                              | ACTIVE                                                               |
| tenant_id                           | acc067874bfd45dcbce9f44d1516910a                                     |
| updated                             | 2015-06-11T21:01:16Z                                                 |
| user_id                             | 762d33ecbeb64356a933e27bce688579                                     |
+-------------------------------------+----------------------------------------------------------------------+
```

Managing eLogin Nodes: Troubleshooting Diagnostics: Ironic

```
example-cmc # ironic node-show elogin3
+------------------------+------------------------------------------------------+
| Property               | Value                                                |
+------------------------+------------------------------------------------------+
| target_power_state     | None                                                 |
| extra                  | {u'description': u'elogin3'}                         |
| last_error             | None                                                 |
| updated_at             | 2016-03-31T21:18:16+00:00                            |
| maintenance_reason     | None                                                 |
| provision_state        | available                                            |
| uuid                   | c0386c4d-9410-4113-a71b-2a770b6239df                 |
| console_enabled        | True                                                 |
| target_provision_state | None                                                 |
| maintenance            | False                                                |
| inspection_started_at  | None                                                 |
| inspection_finished_at | None                                                 |
| power_state            | power off                                            |
| driver                 | fuel_rsync_ipmi                                      |
```

Managing eLogin Nodes: Troubleshooting Diagnostics: Ironic

```
| reservation            | None                                                            |
| properties             | {u'memory_mb': 131072, u'cpu_arch': u'x86_64', u'local_gb': 550, |
|                        | u'cpus': 32}                                                    |
| instance_uuid          | None                                                            |
| name                   |                                                                 |
| driver_info            | {u'ipmi_password': u'******', u'ipmi_address': u'10.142.0.7',   |
|                        | u'deploy_ramdisk': u'e59491e5-d4da-4956-99c8-be662f6ea8c7',     |
|                        | u'deploy_kernel': u'60c99670-7512-4372-8102-84d94bdb5b50',      |
|                        | u'ipmi_username': u'root'}                                      |
| created_at             | 2016-03-17T18:59:18+00:00                                       |
| driver_internal_info   | {u'clean_steps': None, u'is_whole_disk_image': False}           |
| chassis_uuid           |                                                                 |
| instance_info          | {}                                                              |
+------------------------+-----------------------------------------------------------------+
```

Managing eLogin Nodes: Troubleshooting

- **There are multiple places where there may be pressure on the file system on the management server**
  - Images may fill up space in `/var/lib/glance`
    - Remove these using Glance commands only
  - Images may fill up space in `/var/lib/tftpboot`
    - These are removed automatically following a successful deployment
      - If they remain, remove manually
  - PE, config sets and repositories may fill up space in subdirectories of `/var/opt/cray`
    - Remove manually

Managing eLogin Nodes: Troubleshooting

- **The eLogin node is partitioned into two disk devices:**
  - /dev/sda contains the OS, and other data that can be rewritten
    - If an image is re-deployed, all data on sda will be overwritten
    - There should be no space concerns.
  - /dev/sdb is configured as persistent storage for the node
    - Config sets, PE, and some job submission details for workload managers are stored here
    - If the partition is destroyed, all config set data is resynchronized upon reboot. Administrators can safely delete data here.

Managing eLogin Nodes: Troubleshooting

- **Problem:** Lack of disk space on the management server to store the glance images used to boot.
- **Signature:** A 'heat stack-create' fails. 'heat stack-show' displays "No valid host was found".
- **Log messages:** From /var/log/nova/nova-conductor.log

```
2015-06-16 11:37:34.171 5202 ERROR nova.scheduler.utils [req-d7f3e9fa-f87f-44e0-b615-b288f3de02cd
None] [instance: 43de8ff4-d746-49e6-9226-2a3c159552db] Error from last host: example-cmc (node
e63ffc33-029f-44ac-8808-c55909f85f2f): [u'Traceback (most recent call last):\n', u'  File
"/usr/lib/python2.7/site-packages/nova/compute/manager.py", line 2053, in
_do_build_and_run_instance\n    filter_properties)\n', u'  File "/usr/lib/python2.7/site-
packages/nova/compute/manager.py", line 2184, in
_build_and_run_instance\n    instance_uuid=instance.uuid, reason=six.text_type(e))\n',
u"RescheduledException: Build of instance 43de8ff4-d746-49e6-9226-2a3c159552db was re-scheduled:
Failed to provision instance 43de8ff4-d746-49e6-9226-2a3c159552db: Failed to deploy. Error: Disk
volume where '/var/lib/ironic/master_images/tmpr4qVSW' is located doesn't have enough disk space.
Required 3646 MiB, only 784 MiB available space present.\n"]
```

- **Action:** Free up space for the file system that provides `/var/lib/tftpboot/`, which is where ironic copies glance images prior to deploy. Perhaps there are leftover ISO files in `/root/isos/` that can be deleted.

Managing eLogin Nodes: Troubleshooting

- **Problem:** Inability to communicate with a BMC prevents the admin from performing any operations on the ironic node as well as the corresponding nova server and heat stack. There may be other ways to get into this same state.
- **Signature:** A 'heat stack-delete' fails with a "Provision state still 'deleting'" message.
- **Log messages:** From `/var/log/heat-engine.log`:

```
2015-06-19 08:01:20.911 1743 TRACE heat.engine.resource Error: Server
elogin1 delete failed: (500) Error destroying the instance on node
e79e85cd-57f5-4fcd-ba43-14ccea0375e7. Provision state still 'deleting'.
```

- **Action:** Determine why the BMC fails to respond and address that issue.
  - Use 'ironic node-set-provision-state $UUID delete' to clear the Provisioning State.
  - Use 'nova reset-state $SERVER' to clear the server state. At this point, you should be able to delete the heat stack.

Recap of CMC/eLogin

- **eLogin differences over the previous esLogin product.**
  - Prescriptive image builds based on the same image recipes used for CLE nodes
  - Package collections are shared between CLE and eLogin images
  - Software releases with CLE
  - Cray Programming Environment is separate from the base eLogin image
    - Exactly the same Cray PE image as used on the XC
  - Managed by the new Cray System Management Software

Agenda

- Introduction to SMW/CLE system management
- New system management features since UP01
- Best practices for using Ansible
- Troubleshooting XC system booting problems
- Migrating SMW/CLE software from 7.2/5.2 to 8.0/6.0
- Intro to CMC/eLogin system management
- **Migrating CIMS/CDL to CMC/eLogin**
- CLE Boot Performance and Reliability
- Q & A

Migrating CIMS/CDL to CMC/eLogin

- **esLogin to eLogin Migration Overview**
- **Restrictions/Limitations**
- **Migration Process**
  - Gathering configuration data
    - eLogin Migration tool
  - Installing the CMC
  - Testing eLogin deployment and operation

esLogin to eLogin Migration Overview

- **What is being migrated?**
  - The CIMS is being migrated to a CMC
  - esLogin nodes are being migrated to eLogin nodes
  - Configuration data will be gathered from the CIMS and esLogin nodes to assist in migration
- **Management server (CMC) is a fresh installation**
  - OS changes from SLES 11 to CentOS 7
  - Management software changes from Bright Cluster Manager to CSMS
- **esLogin images are replaced by prescriptively built eLogin images**
  - Requires the SMW to be migrated to SMW 8.0.UP03/CLE 6.0.UP03
    - eLogin images and config sets are created on the SMW
  - OS changes from SLES 11 to SLES 12
- **Cray Service has a Migration toolkit**
  - Toolkit contains a data gathering tool, test config set, and test eLogin image

Restrictions/Limitations

- **Only esLogin nodes will be migrated**
  - Other external service nodes (data movers, visualization servers, CLFS, etc.) will **NOT** be migrated
- **CIMS/esMS servers managing more than esLogin nodes will NOT be migrated**
  - CIMS/esMS servers are required for continued management of the non-esLogin nodes
  - New CMC server hardware is required to manage eLogin nodes in these environments

Restrictions/Limitations

- **High Availability CMC configuration is not available**
  - Sites with HA CIMS/esMS will become a single CMC configuration
- **Multiple management networks are not available**
  - Sites using separate management networks for isolating esLogin node groups must manage eLogin nodes on a single management network

Migration Process: Gathering Configuration Data

- **Migration tool collects configuration data**
  - Bright Cluster Manager database
  - CIMS/esMS server
  - esLogin nodes
- **For systems with a CIMS/esMS**
  - The migration tool executes on the active CIMS/esMS
- **For systems without a CIMS/esMS**
  - The migration tool executes on each esLogin node
- **Data will be saved to another server for later use**
  - The CIMS/esMS will be reinstalled and all CIMS/esMS data will be lost

Migration Process

- **Disconnect the esLogins from the CIMS/esMS**
  - Allows the esLogins to continue operation while the CIMS/esMS is repurposed as a CMC.
  - Minimizes esLogin downtime to the time it takes to deploy an eLogin node

Migration Process: Migration Tool

- **Generates sections of the following config set worksheets**
  - cray_net
  - cray_node_groups
  - cray_eswrap
  - cray_lustre_client
  - cray_elogin_networking
  - cray_elogin_lnet

- **These sections must be merged into the worksheets generated on the SMW for THE CRAY® XC™ SERIES system**

Migration Process: Migration Tool

- **Generates the CSMS inventory.csv file for node enrollment**
- **Gathers data for use in configuring CSMS**
  - Network information
  - User account information
  - RPM list
    - CIMS/esMS and esLogin images
  - Other configuration data

Migration Process: Migration Tool

- **Files gathered from CIMS and esLogin**
  - /etc/hosts
  - /etc/bash.bashrc.local
  - /etc/csh.cshrc.local
  - /etc/resolv.conf
  - /etc/ldap.conf
  - /etc/openldap/ldap.conf
  - /etc/nsswitch.conf
  - /etc/ntp.conf
  - /etc/ssh/*
  - /etc/ssl/*
  - /etc/hosts.allow
  - /etc/hosts.deny
  - /etc/passwd
  - /etc/shadow
  - /etc/security/access.conf
  - /etc/aliases
  - /etc/fstab
  - /etc/group
  - /etc/rsyslog.conf
  - /etc/pam.d/*
  - /etc/sysctl/*
  - /etc/sysconfig/*
  - /etc/yp.conf
  - /etc/modprobe.d/cray-lnd.conf
    - (esLogin only)
  - /etc/modprobe.d/cray-lnet.conf
    - (esLogin only)

Migration Process: Installation and Testing

- **Install and configure the CMC using standard eLogin installation documentation**
  - Previously gathered data is used to help with configuration
- **Configure the test config set for eLogin nodes**
  - For initial testing without a SMW, a minimal config set is available for sanity checking
- **Enroll the eLogin nodes**
  - This requires the eLogin nodes be powered off and connected to the CMC IPMI and management networks
  - For initial testing, just one eLogin node could be used
- **Deploy and boot the default test image**
  - For initial testing without a SMW, a default test image is available

Migration Process: Post SMW migration

- **Install eLogin support on the SMW**
  - Instructions are in the eLogin installation guide
- **Build the production eLogin image on the SMW**
  - Requires a SMW supporting CLE 6.0.UP03
  - Instructions are in the eLogin installation guide
- **Create the production config set**
  - Requires a SMW supporting CLE 6.0.UP03
  - Instructions are in the eLogin installation guide
- **Deploy and boot the eLogin nodes using the production image and config set**
  - Instructions are in the eLogin installation guide
- **Verify eLogin operation**
  - Ensure the eLogin functions normally

Agenda

- Introduction to SMW/CLE system management
- New system management features since UP01
- Best practices for using Ansible
- Troubleshooting XC system booting problems
- Migrating SMW/CLE software from 7.2/5.2 to 8.0/6.0
- Intro to CMC/eLogin system management
- Migrating CIMS/CDL to CMC/eLogin
- **CLE Boot Performance and Reliability**
- Q & A

CLE Boot Performance and Reliability

- Introduction
- Time to boot – component analysis
- Measuring boot performance
- **Boot performance and reliability improvements**
  - Config set caching
  - Netroot preload
  - Ansible filtering
  - Fact collection tweaks
  - Sparse looping adjustments
  - Boot profiler
  - Greater boot concurrency support
  - ARP table initialization
  - PE ldconfig caching
  - ntpd improvements
  - Node groups optimization
  - DVS read only optimizations

Background

- Changing deployment models introduces additional cost and complexity
- Cray, Inc. wants to have performant products that can scale to all customer needs, now and in the future
- Product offerings are expanding to offer hardware architectures with more cores that run at slower speeds
- Proactively making CLE faster now reaps immediate benefits later
- Newer distribution software can provide better user experience, when leveraged correctly
- Boot performance is a marker for overall system responsiveness

CLE Boot Performance Project Charter

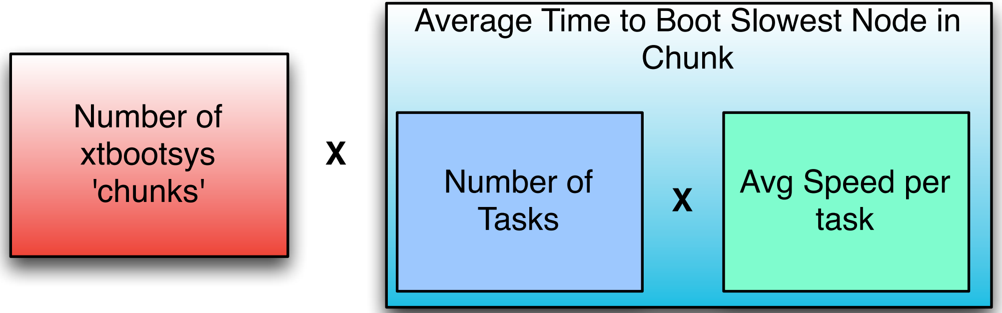
- Reduce overall downtime during install, updates, and node reboots
- Develop best practices for configuration, hardware resource allocation and feature development direction to achieve best scale
- Provide tooling for measurement of overall boot performance
- Prioritize faster component boots for nodes which are booted more frequently
- Reduce overall variability in deployment to increase overall reliability and reproducibility of behavior
- Optimize for configurations most common with customers
- Partner with customers to better understand scalability concerns
- Predict scale requirements ahead of time

Most Common Site Configurations

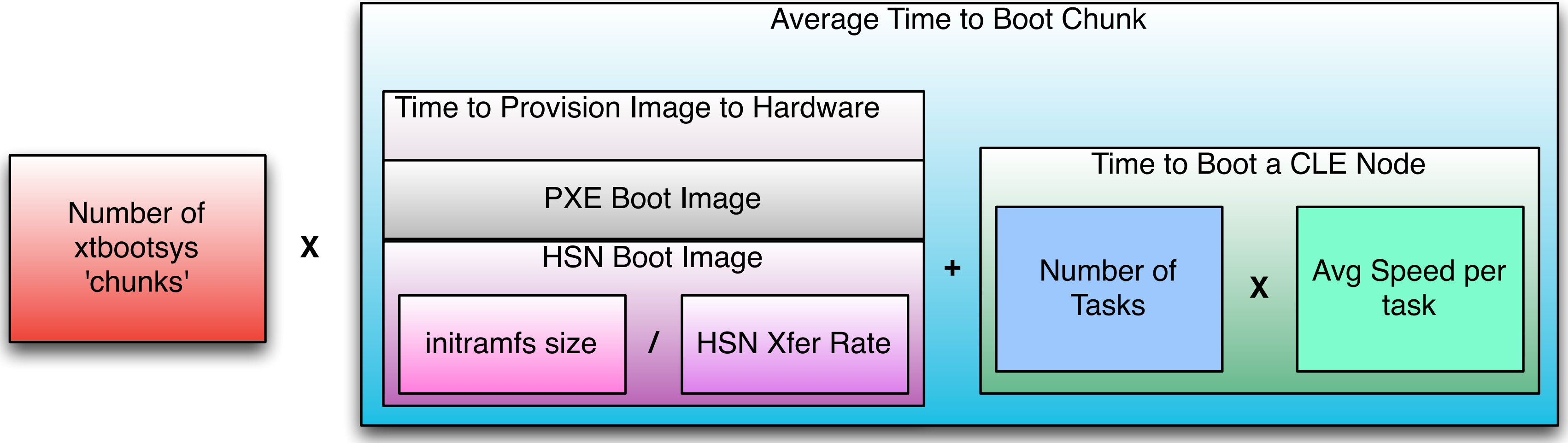
- Prefers netroot enabled images instead of tmpfs images
  - Larger images offer greater amount of content
  - Smaller memory footprint
  - Network filesystems are slower than filesystems in local memory
- Adds in a WLM of their choice
- Desires integration with external filesystem
- Enables one or more authentication systems
- Relatively homogenous compute hardware configuration

Time To Boot Gross Overview

- Two primary avenues to better boot performance
- Cannot reduce number of chunks to less than two
- Computes are most important target for improvement
  - Rebooted More Frequently
  - Largest scale requirements
- Improving slowest node in ‘chunk’ provides greatest benefit



Time To Boot Gross Overview

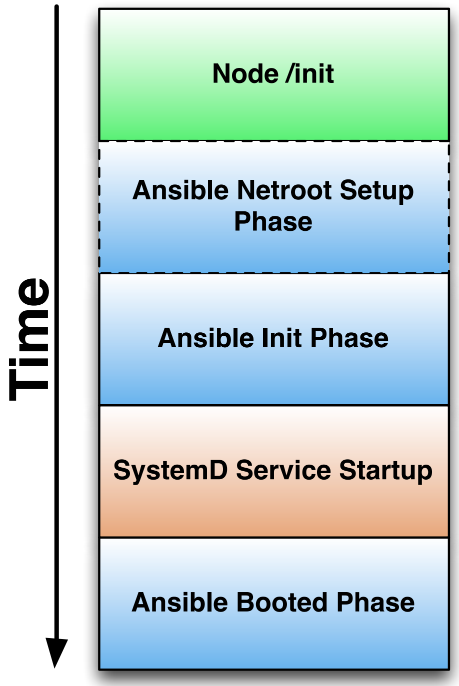


Xtbootsys ‘chunk’

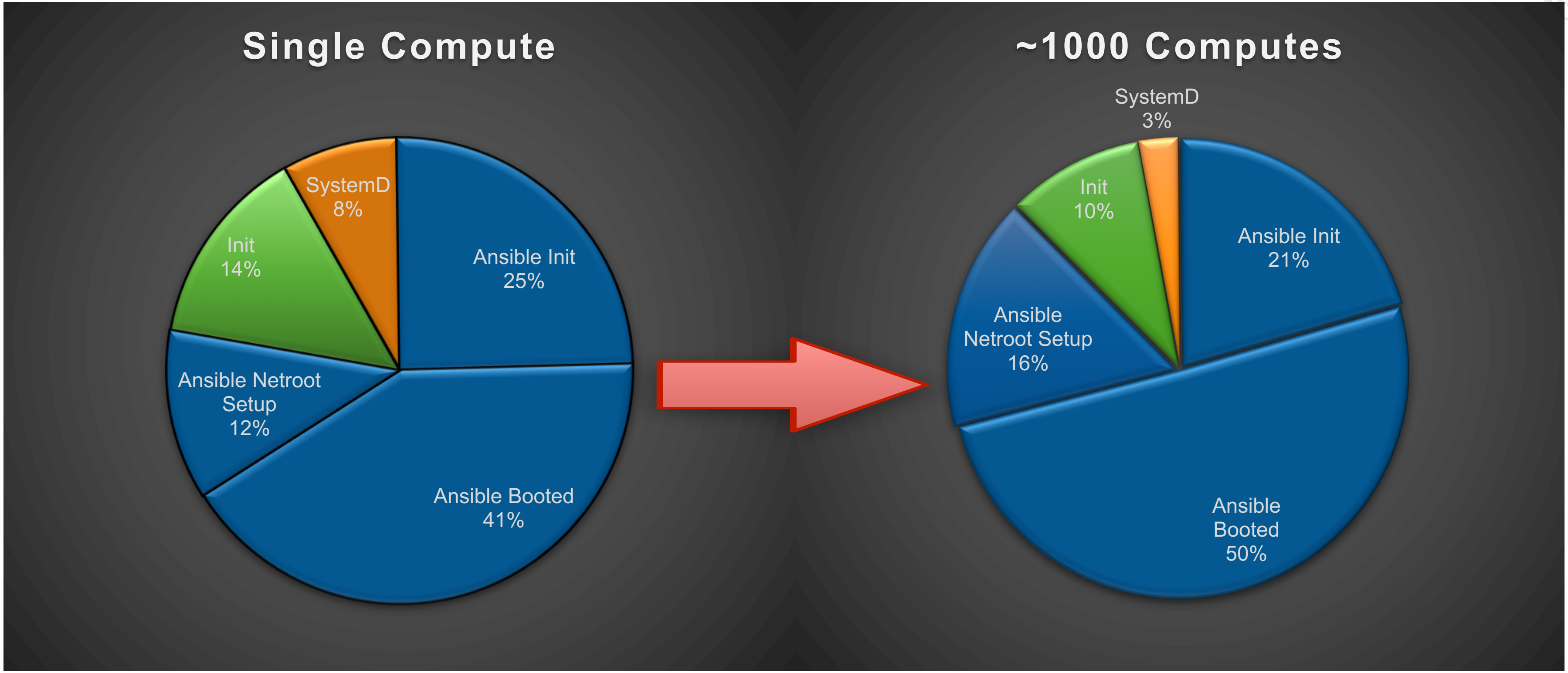
- **Typical deployments boot_ in 4 chunks**
  - Boot, SDB, Service, Compute
- **Two types of booting**
  - PXE Booting
    - Limited to boot, sdb, and backup boot & sdb
  - HSN Booting
    - Requires BND on a booted boot node
- **Each ‘chunk’ of HSN boot requires image fanout**
  - boot_* events
- **Each ’chunk’ waits for completion of all nodes within chunk**
  - wait_for_* events
  - Actual CLE Time to boot

Time to Boot Single CLE Node (simplified)

- 5 primary phases
  - Netroot setup phase for netroot nodes only
- Largely affected by performance of underlying filesystem
- Affected by larger amounts of data within cfgset fields
- DAL nodes have a SysVInit Phase instead of SystemD



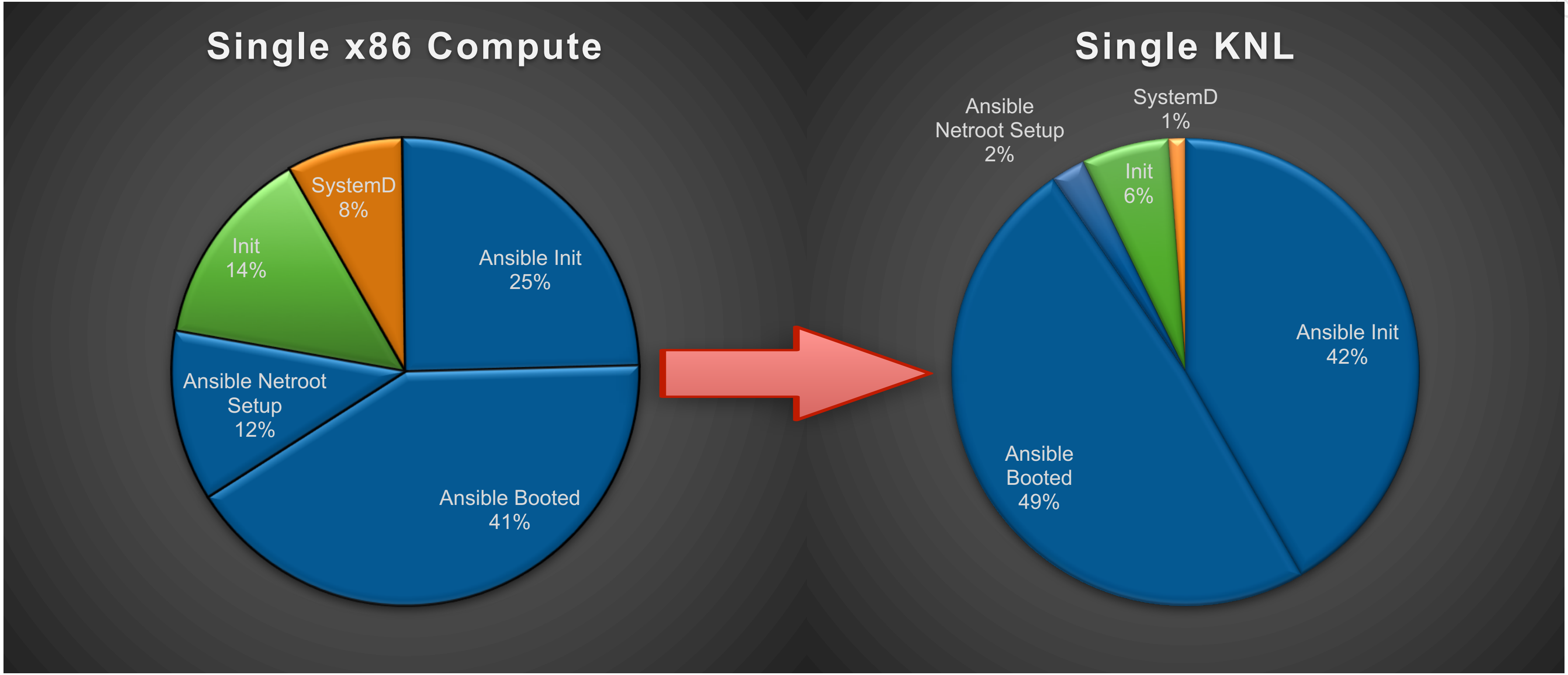
CLE Node Boot as a function of configuration load (x86-64 Netroot Compute, up03)



Intel® Xeon Phi™ 7250 (KNL) Boot Performance

- **KNL nodes must be rebooted in order to switch between memory modes**
- **Only one core is effectively used for the majority of configuration process**
  - Slower individual core clock speeds affect serial configuration tasks
- **Excels at systemd service startup (as effective as x86 service initialization)**
- **Ansible tends to run significantly slower**
  - Unloaded ansible-booted phase 4-6x slower than x86
  - Memory mode does not appear to be significant contributor to boot speed

Intel® Xeon Phi™ 7250 (KNL) CLE Boot Breakdown



Why Does Ansible Take So Much Time?

- **Ansible is optimized for remote configuration**
  - Normally done through central host
  - Not optimized for self/localhost configuration
- **Ansible drives file operations for each task**
  - Each task is designed to run against a remote
  - CLE tasks simply specify the target host as 'localhost'

```
nid00035:~ # ansible-playbook -vvv /etc/ansible/alps.yaml

PLAY [local alps playbook] ****************************************************

GATHERING FACTS ***************************************************************
<localhost> REMOTE_MODULE setup
<localhost> EXEC ['/bin/sh', '-c', 'mkdir -p $HOME/.ansible/tmp/ansible-tmp-14919
<localhost> PUT /tmp/tmpq_3HcE TO /root/.ansible/tmp/ansible-tmp-1491938579.46-18
<localhost> EXEC ['/bin/sh', '-c', u'LANG=C LC_CTYPE=C /usr/bin/python /root/.ans
ok: [localhost]
```

Why Does Ansible Take So Much Time? (pt2)

- **Ansible is written in python, python import operations walk, stat and open files for library resolution**
  - Zhao, Davis, Antypas, Yao, Lee, Butler, CUG 2012; Canon, Jacobson, CUG 2016
  - Modules append content to PATH and PYTHONPATH for release switching
    - CLE
    - PE
- **Ansible is single threaded/single process, and does not take advantage of multiple cores**
- **Ansible conditional looping is inefficient**
  - Task ‘when’ clause is evaluated for each iteration of ‘with_items’
- **Skipped tasks are not free**
  - Disabling playbooks still cost time to parse and exercise

Why Does Ansible Take So Much Time? (pt3)

- **Playbooks start services within ansible-init or ansible-booted**
  - Requirements are provided to playbooks too late in some cases
  - Faster to enable services during init phase and let systemd start them
- **Distributed configuration can become overloaded (UP01 and earlier)**
  - Parsing YAML (multiple times) is expensive compared to JSON or INI formats
  - File operations in and out of cfgset transferred over IDS fanout (9p/diod network).
  - Silent corruption possible when very stressed
- **Ansible collects 'facts' multiple times per run**
  - Out of the box facts for querying volume label information is slow

What happens when I/O is saturated?

- **Time to boot increases**
  - Global increase in time to completion of ansible tasks
  - Relatively unchanged performance of systemd and init processing
- **Systemd operations can fail**
  - systemctl related options timeout after 25 seconds
  - Typically within ansible booted phase, however occasionally in other services which reload other services
- **Affects overall system responsiveness, job launch performance**

Legacy SysVInit Service Infrastructure

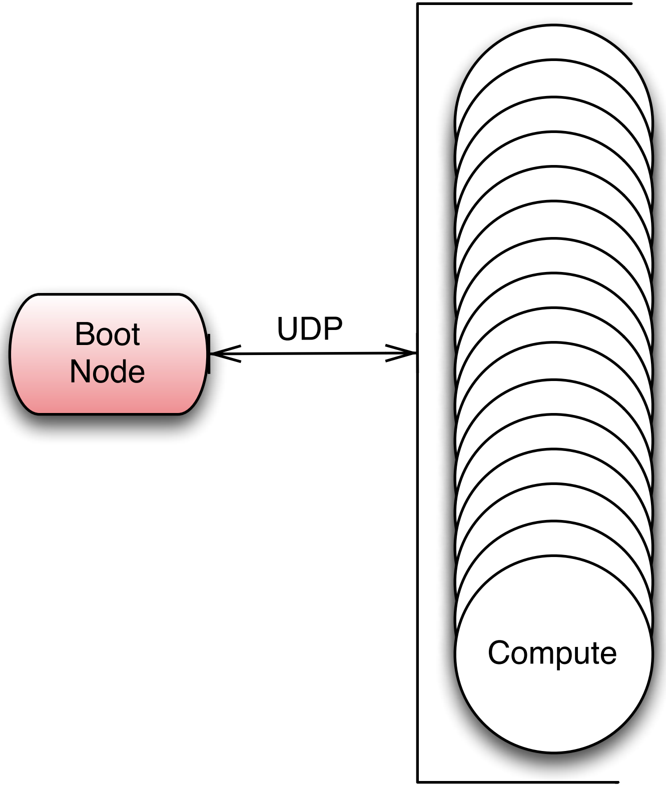
- **CLE 6 is based on two primary upstream distributions:**
  - SLES 12
  - CentOS 6.5 (DAL)
- **SLES 12 Nodes leverage systemd as a native service management paradigm**
- **CentOS 6.5 still uses SysVInit services**
- **Existing CLE services cannot easily integrate with systemd infrastructure**
  - Service initialization 'notify' support
- **Additional Ansible logic must be written to be conditional in order to cover both**
  - Skipped tasks are not free! Even when we don't have DAL nodes.

Reliability and Repeatability

- **Variability decreases reliability and repeatability**
  - Services are configured slightly different even when cfgset remains the same
    - Use of random client/server selection
    - Injection of random waits/timeouts
  - Services internally load balance non-deterministically
    - First come, first balanced
  - Variability creates variability
  - Often tied to timing of events
  - High variability exposes more edge cases
  - Variability increases overall time to boot ‘chunk’
- **Low repeatability hinders debuggability**

Service Failures Introduce Delays

- **Ansible Booted waits for clean state**
  - Single node service failure prolongs entire boot chunk
  - NTP cleanly started
  - /etc/fstab mounts made available
- **Slow to boot services increase critical-chain length**



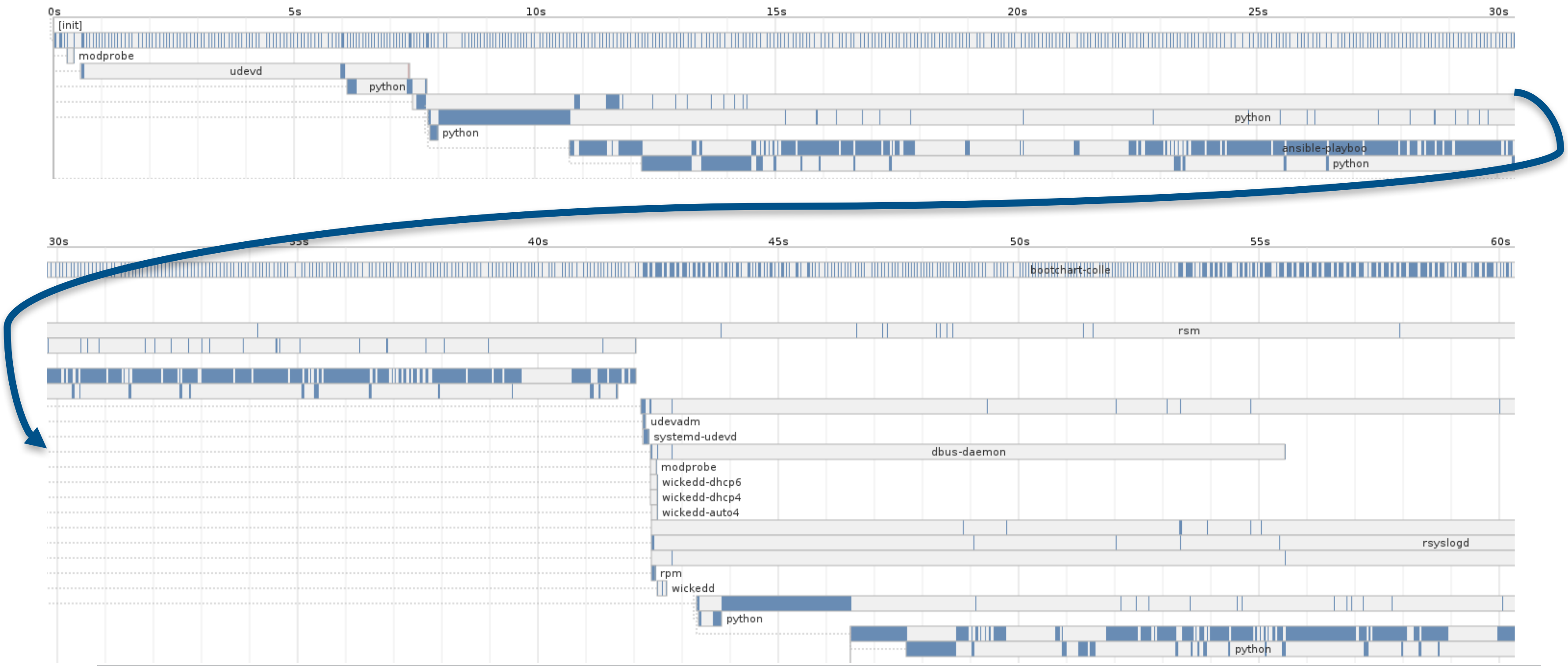
Why are we booting in 4 ‘chunks’?

- **PXE Boot Size Limitation of ~500Mb**
  - WLMs often install additional content into the SDB image, which increases overall image size beyond 500Mb
- **Client/Server Ordering Issues**
  - Exports before mounts
  - Services before clients
- **WLM Integration can be challenging**
  - MOM nodes
  - Licensing/external connectivity (RSIP)

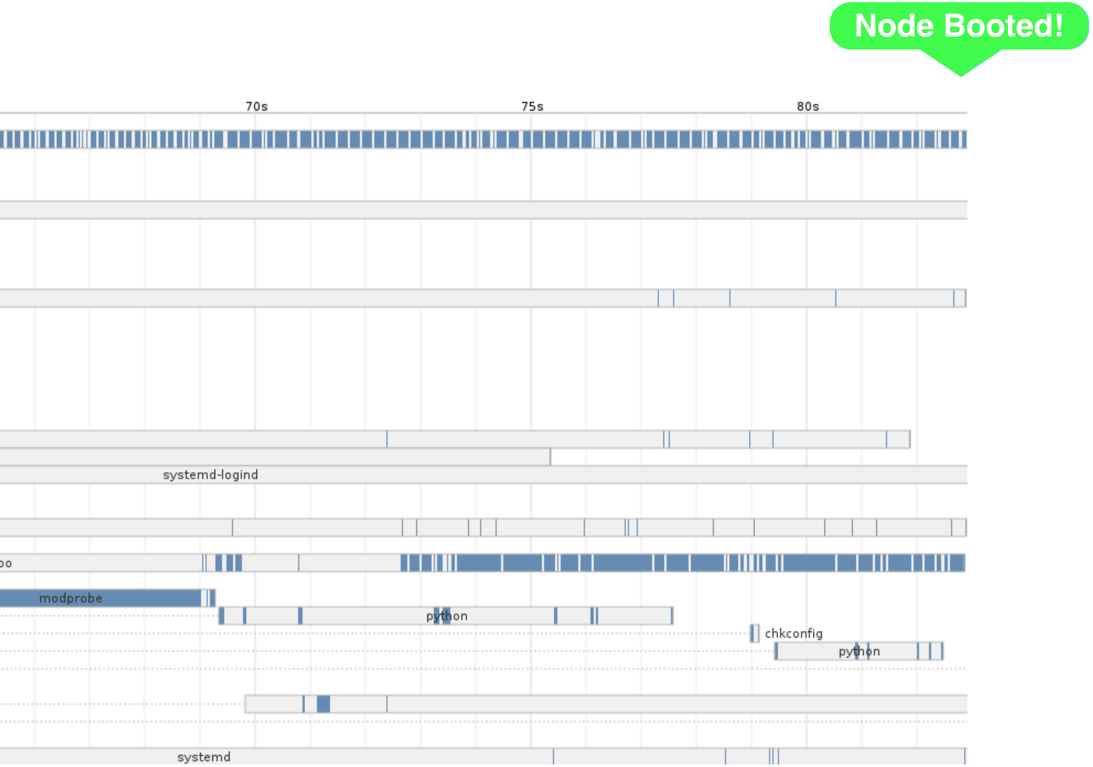
Measuring Boot Performance

- **Requirements**
  - Need a way to quickly and reliably provide root cause of slowest setup operations to the whole of the boot
  - Needs to be run on both SMW and CLE Node
  - Needs to be able to summarize events on both
    - Some information only available on CLE nodes
  - Capture records to file for historical comparison
  - Provided initially as a developer diagnostic tool
    - Eventual support in CLE product
  - May not artificially increase time to boot during measurement
    - Bootchart
    - Post-mortem timing information from logs

Bootchart UP01 (60 seconds of tmpfs boot)



Bootchart UP01 tmpfs compute (animated)



Boot Profiler

- **CLI on top of a ‘boot profiler’ API**
- **Runs from SMW or XC node**
  - No support yet for eLogin nodes
- **Modeled after ’systemd-analyze blame’**
- **Allows SMW to “introspect” slow nodes**
  - Currently only supports serial introspection
  - SSH Proxying through boot node
  - Requires Passwordless SSH (root)
- **’—blame’ allows for wider breadth of introspection**

Event Classification

- **Singleton Event**
  - Duration
    - From A to B
    - Total seconds self-report
  - Sparce Events
    - Represents period of time between two events that have reported
- **Parent Events**
  - Comprised by one or more child events
  - Serial Event
    - All children are run in one after the other
    - Duration defined as summation of all child events
  - Parallel Event
    - An event where all children are run in parallel
    - Duration defined as longest running child event

Available Option Set

```
bubble-smw:~ # bootprofiler --help
usage: bootprofiler [-h] [--blame BLAME] [-p] [-l] [-c]
                    bootsession [bootsession ...]

Tool to extract system boot profiling data from a boot session or an active
CLE node.

positional arguments:
  bootsession           Boot session to bootprofiler. May either be a path to
                        a boot session directory, or the name of a boot
                        session directory under /var/opt/cray/logs.

optional arguments:
  -h, --help            show this help message and exit
  --blame BLAME, -b BLAME
                        The number of problematic sub-events that should be
                        reported as a flagged by maximum duration.
  -p, --passwordless    Leverage passwordless ssh to query profiling
                        information from individual problematic slow nodes.
  -l, --last            Show only the results from the last xtbootsys
                        invocation
  -c, --csv             Output the results of the profiler to a csv file. If
                        multiple boot sessions used, all will be written to
                        the same file.
```

Demo on Virtual SMW

- **Commands Used in Demo:**
  - From SMW:
    - bootprofiler –blame 20 p0-current
    - bootprofiler –blame 3 –passwordless p0-current
  - From CLE Node:
    - bootprofiler –blame 5

Reducing Required Number of Xtbootsys Chunks

- **‘sres’ – distributed semaphore**
  - Command line tool
  - Works in either server or client capacities
  - Advertises available functionality over a unique TCP port
- **Allows asynchronous fulfillment of requirements during boot**
- **Node services are required to wait for their underlying dependencies to be met before progressing**
- **Notable uses**
  - DVS Client/Server
  - NFS Export/Mount
  - LiveUpdates Advertisement

Reducing Required Number of Xtbootsys Chunks (pt 2)

- **Combined boot of ‘service’ and ‘compute’ (up03)**
  - Leverages distributed semaphore controlled park and wait for DVS Mounts, Netroot, NFS/Export
- **Simultaneous boot of boot, sdb, backup boot, backup SDB (up02-up03)**
  - Limited by PXE image size limitation
  - Possible to dynamically install additional content during boot with LiveUpdates in order to get under PXE size limitation
- **DAL system may require xtbootsys commands or manual automation**

Making Ansible Faster

- Address upstream issue requiring multiple fact collection passes (1.9.2) (up02)
- Remove unused facts which probe local attached volume label information
- Creation and leverage of Ansible filters reduces 'no-op' looping
- Use Node Group filters to determine when and how a task should configure (up02)
  - Significantly reduces looping/set fact behavior

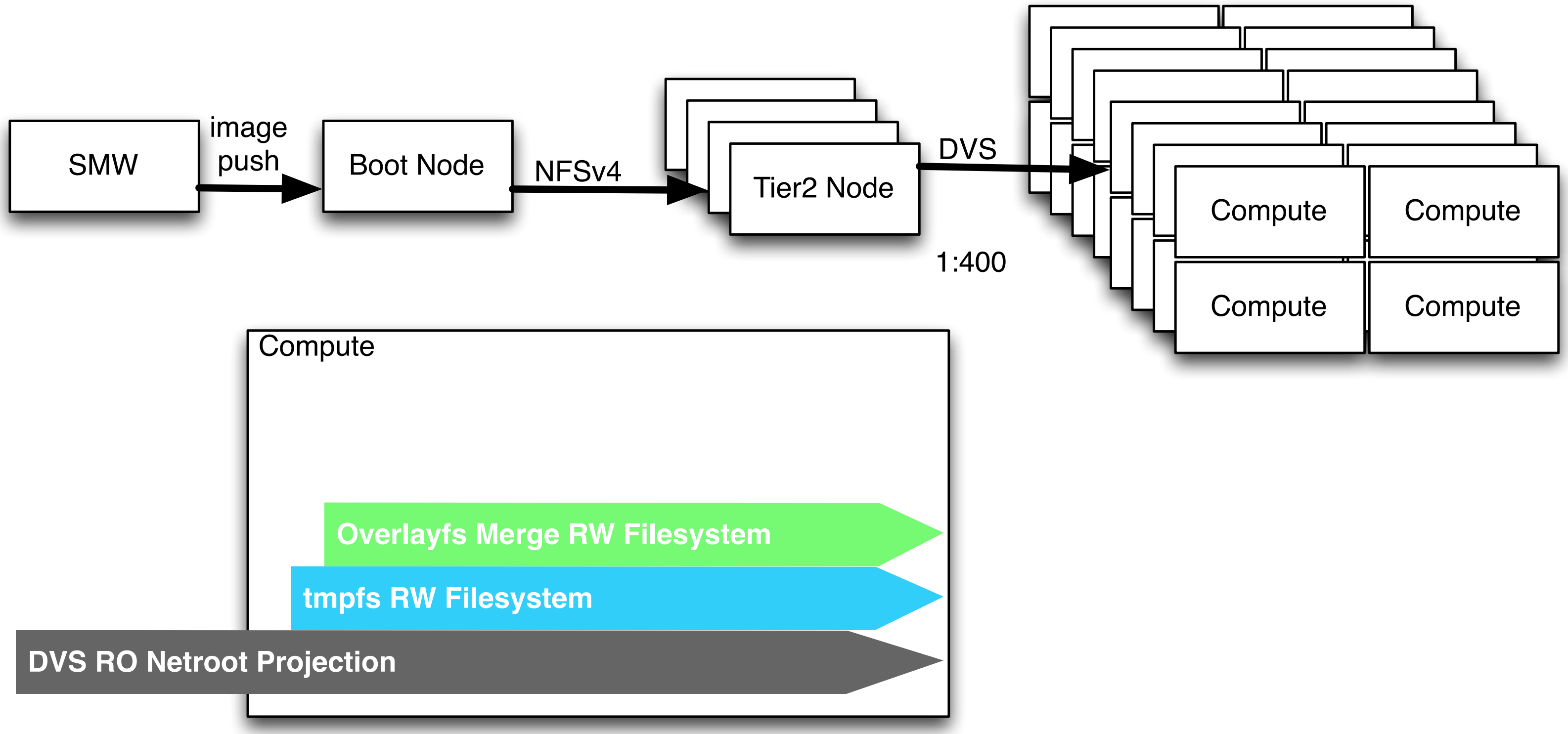
Making Ansible Faster (pt2)

- **Move common setup behavior to facts (up03)**
  - Reduces total number of tasks run
- **Split playbook packaging to be node type specific (up02)**
  - Computes no longer no-op service only plays
  - Allows better ways of integrating playbooks common to XC and eLogin

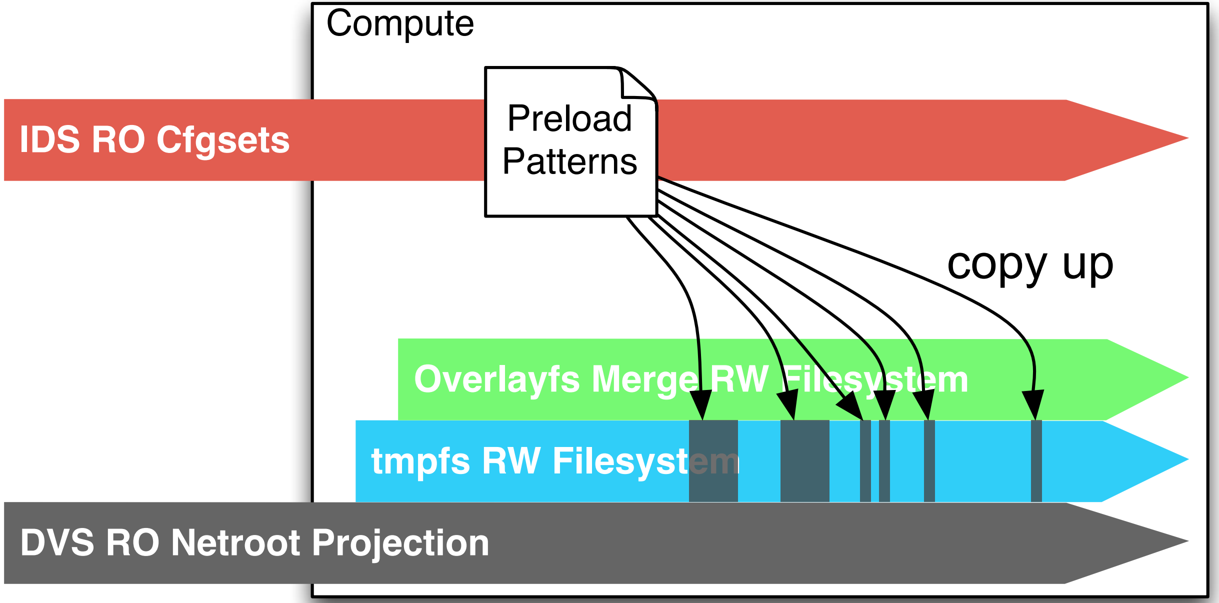
Increased Reliability through decreased variability

- **Services which use random selection now use a hashing function**
  - Seeded against node identity information
  - Ansible filters (not random filter) cray_hashselect
- **Progressive, low-initial latency timeout/retry in services**
  - Keeps behavior of clients closer synchronized
  - Nodes complete task earlier when services are under extra load
  - Decreases cache thrashing behavior

Netroot Overview



Netroot Preload (up01)



Netroot Preload Behavior

- **Provided preload packaged pattern file with XC**
- **Different preload patterns for computes login nodes**
- **Generated using DVS request patterns from logged files**
  - Files critical to boot sequence
  - Files opened most frequently
  - Agnostic to configuration where possible
  - Avoid files which will be promoted naturally during write configuration
- **Only alleviates load associated with readpages, open & close operations**
  - Does not preload directories
  - Listdir, getattr, stat requests still honored by the underlying mount
- **Consumes ~100Mb per compute (up01-up03); ~50Mb (up04)**
- **Generating your own preload file:**
  - XC™ Series DVS Administration Guide CLE6.0up03 s-0005
  - https://pubs.cray.com xctm-series-dvs-administration-guide-cle-60up03-s-0005

Boot FS Traffic Behavior

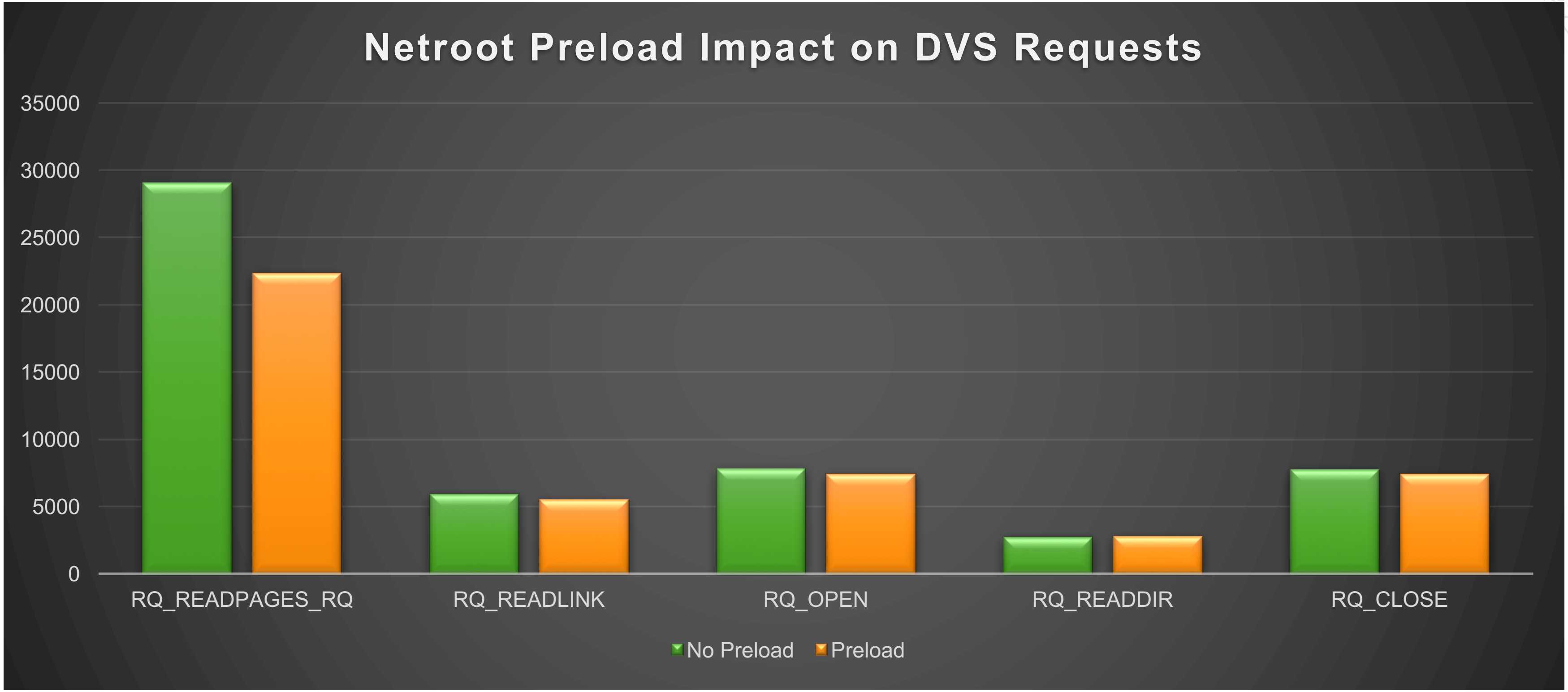
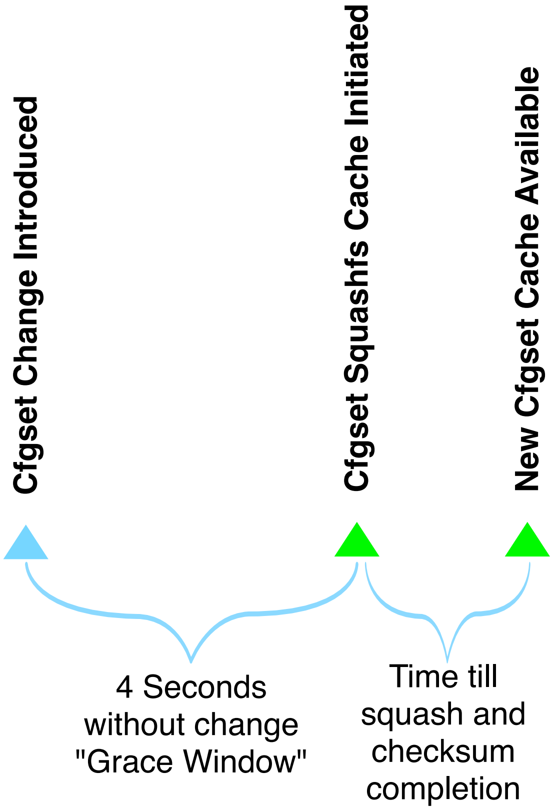


Image Binding: PE's ldconfig (up03)

- Cost of running ldconfig prohibitively long
- Impacts tmpfs and netroot images
  - Largest impact on netroot computes
- Reduce overall time of setup by booting once and preserving ld.so.cache
  - Instructions within /var/opt/cray/pe/ldconfig_cache_command
- Version of ld.so.cache saved is unique to PE image identity and CLE image identity
- New versions of PE or CLE require new save of ldconfig

Config Set Caching (up01)

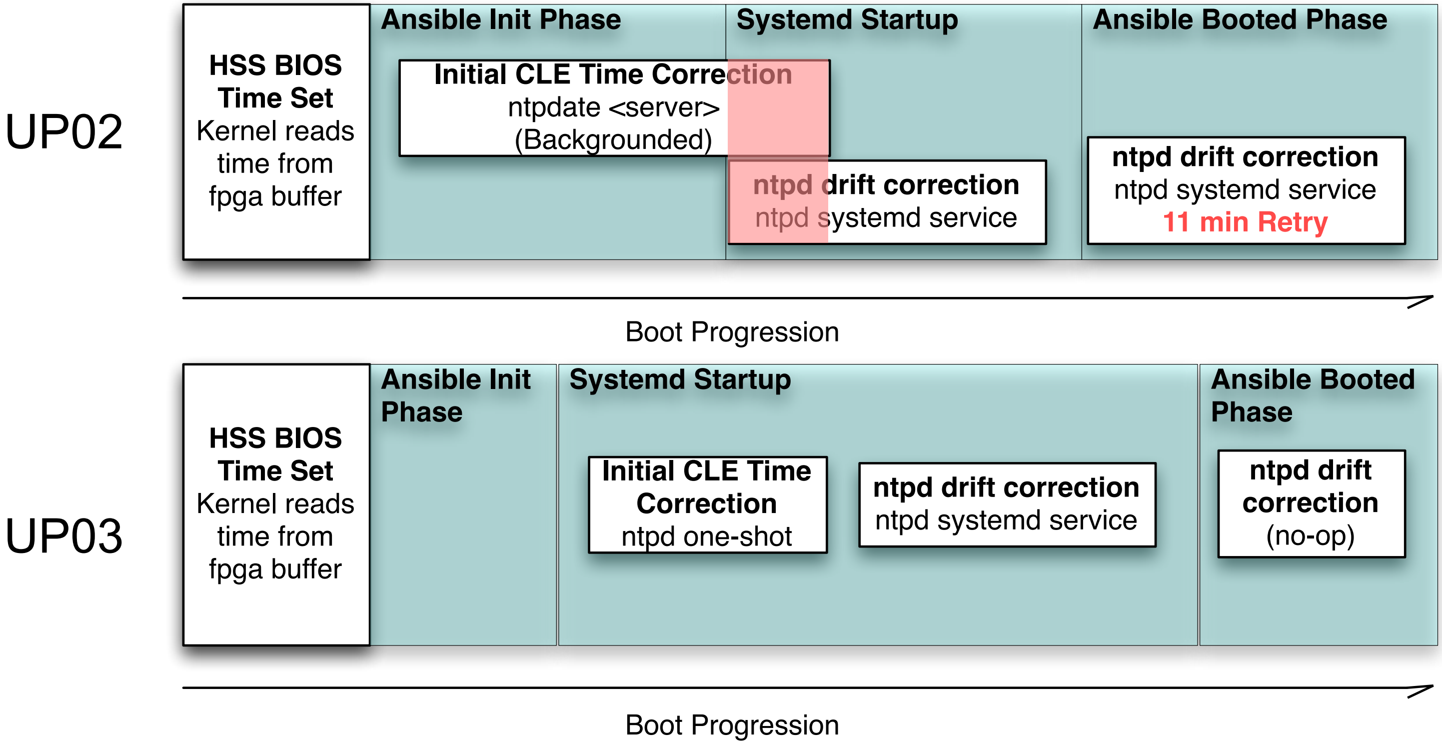
- **SMW service 'cray-cfgset-cache'**
  - After change, generates new cache after 4 seconds of inactivity
  - Uses kernel inotify watch descriptors
- **Allows for compression of entire cfgsets for distribution to client nodes**
- **Integrity of distribution/checksums**
- **Nodes have 'last known good' cfgset local copy**
- **Speeds up YAML parsing operations**
- **Checksums allow quick stale cfgset detection**
- **Faster transfer of content from cfgset to node**
  - Simple Sync
  - Dist/key transfer



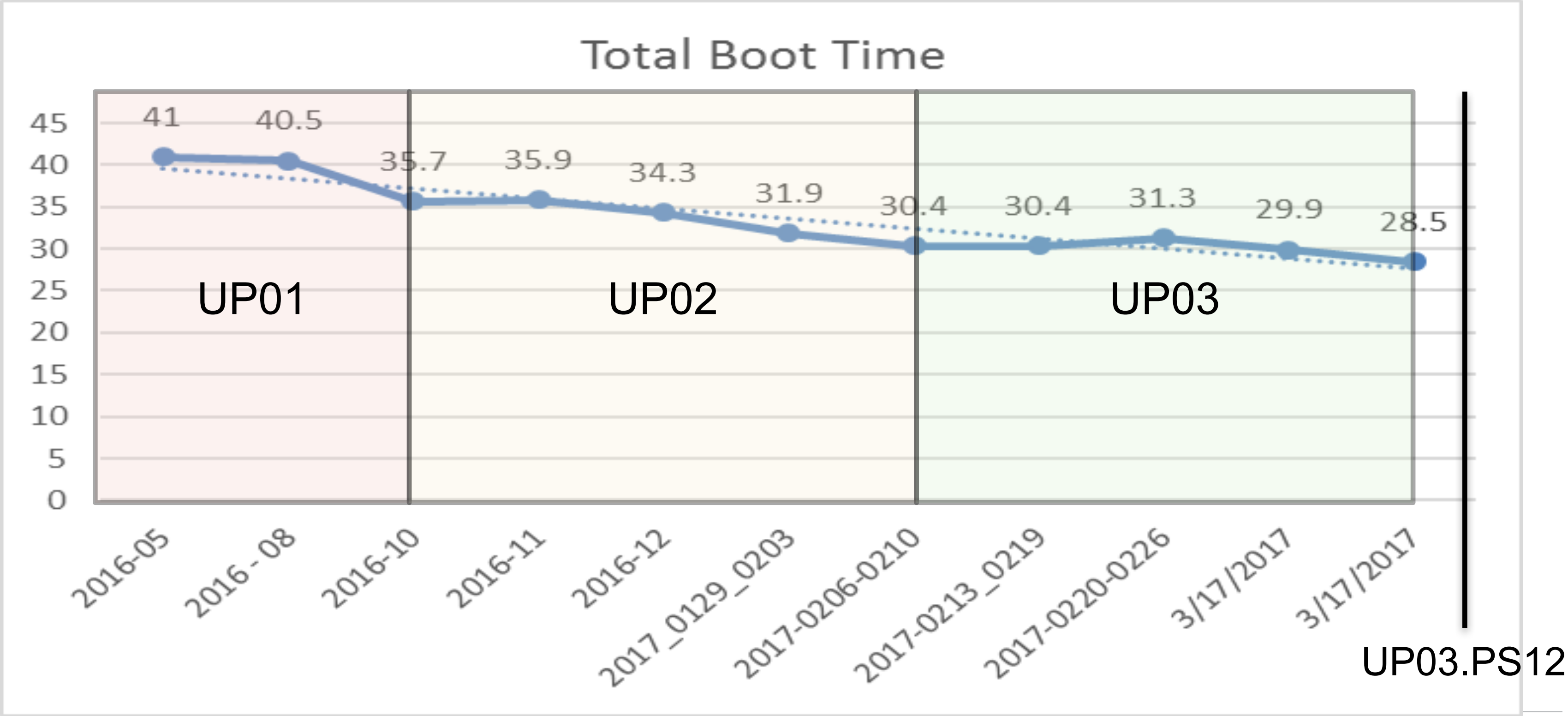
Faster Service Initialization

- **HSN ARP table initialization handled by rca_arpd service (up03)**
  - Previously was handled using individual calls to arpd -a for each node in system
- **LLM uses full tiering fanout for message aggregation (up01)**
- **NTP Fanout using all members of tier1 and tier2 (up03)**
  - Low rate of failure introduced significant boot staggering when used with distribution defaults
  - Greater Resilience
    - Added peer configuration
    - Fewer single points of failure

CLE NTP UP02 to UP03

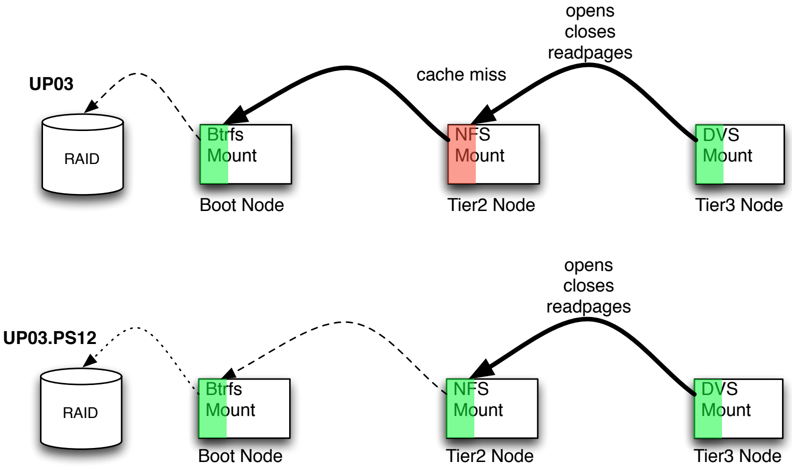


Results: XC mixed x86/KNL Full System Netroot Cold Boot Times, ~1000 computes (minutes)

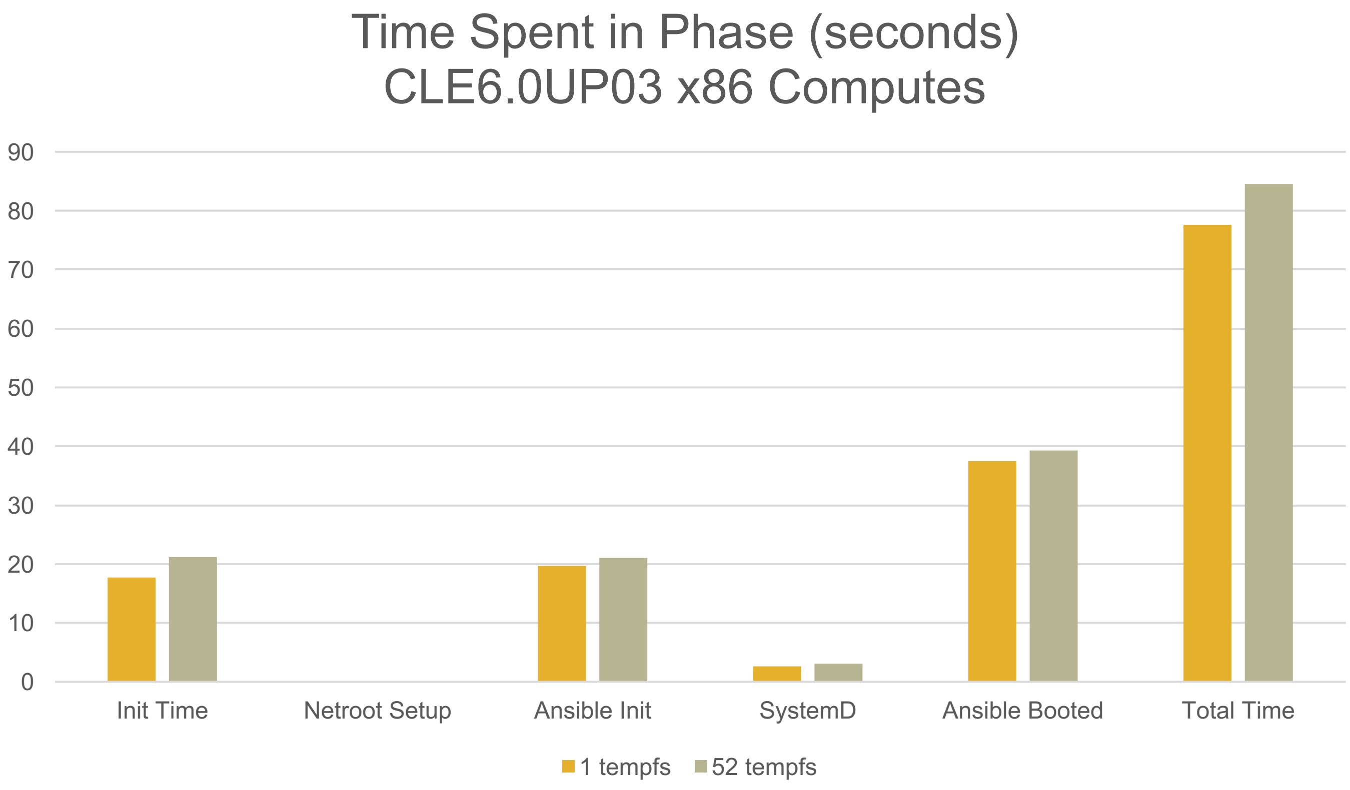


What is CLE 6.0UP03 PS12?

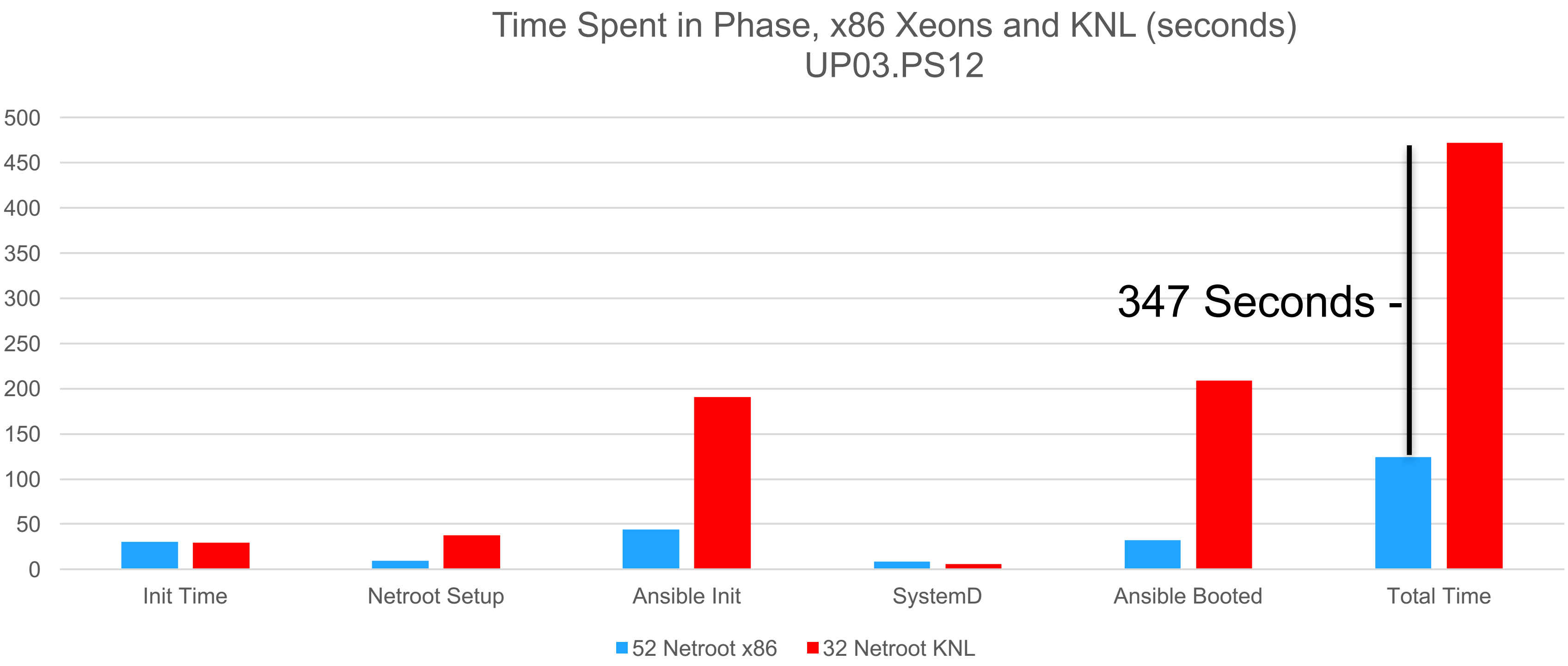
- Fixes how DVS Servers manage RO NFS client mounts
- Inode cache invalidated, forcing all traffic to be re-requested from the boot node
- Tier2 nodes must have RO NFS mount to boot node
  - cray_simple_shares /var/opt/cray/imps/ filesystem



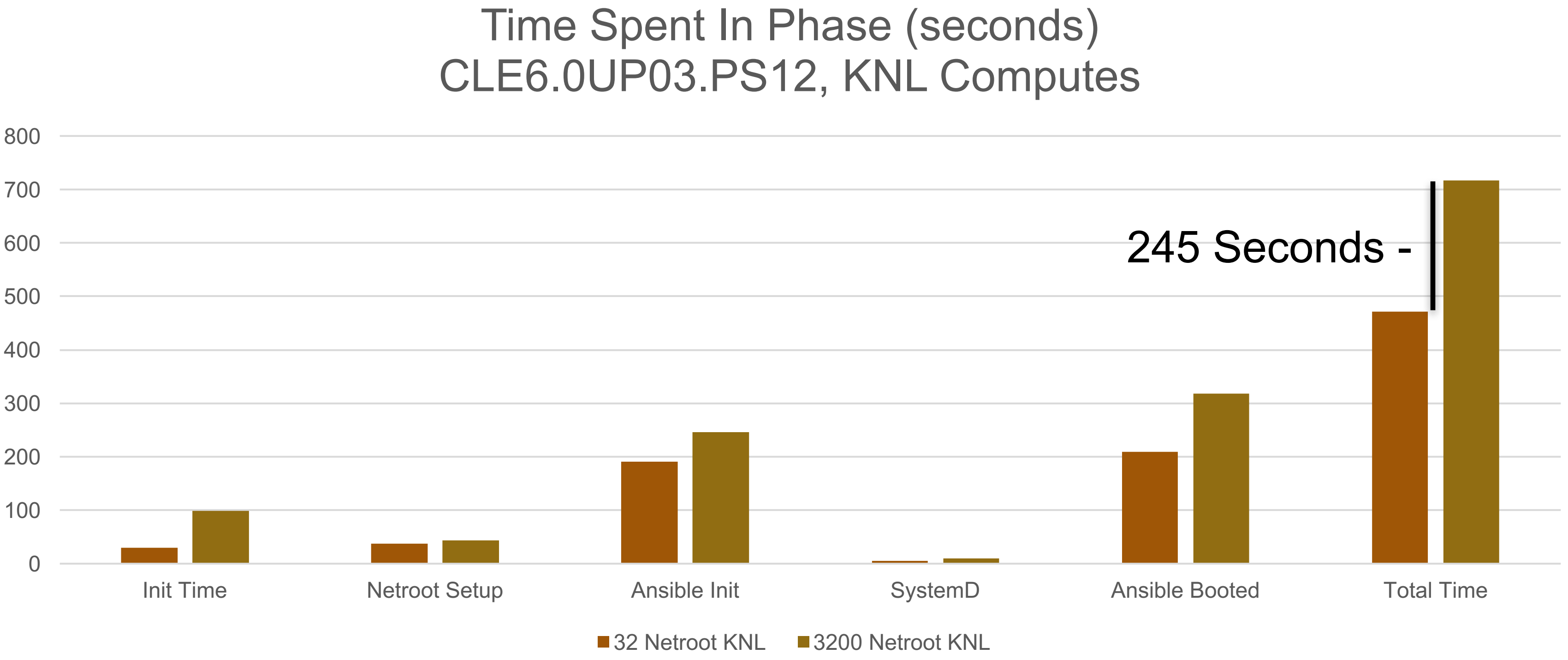
Scaling Characteristics of tmpfs



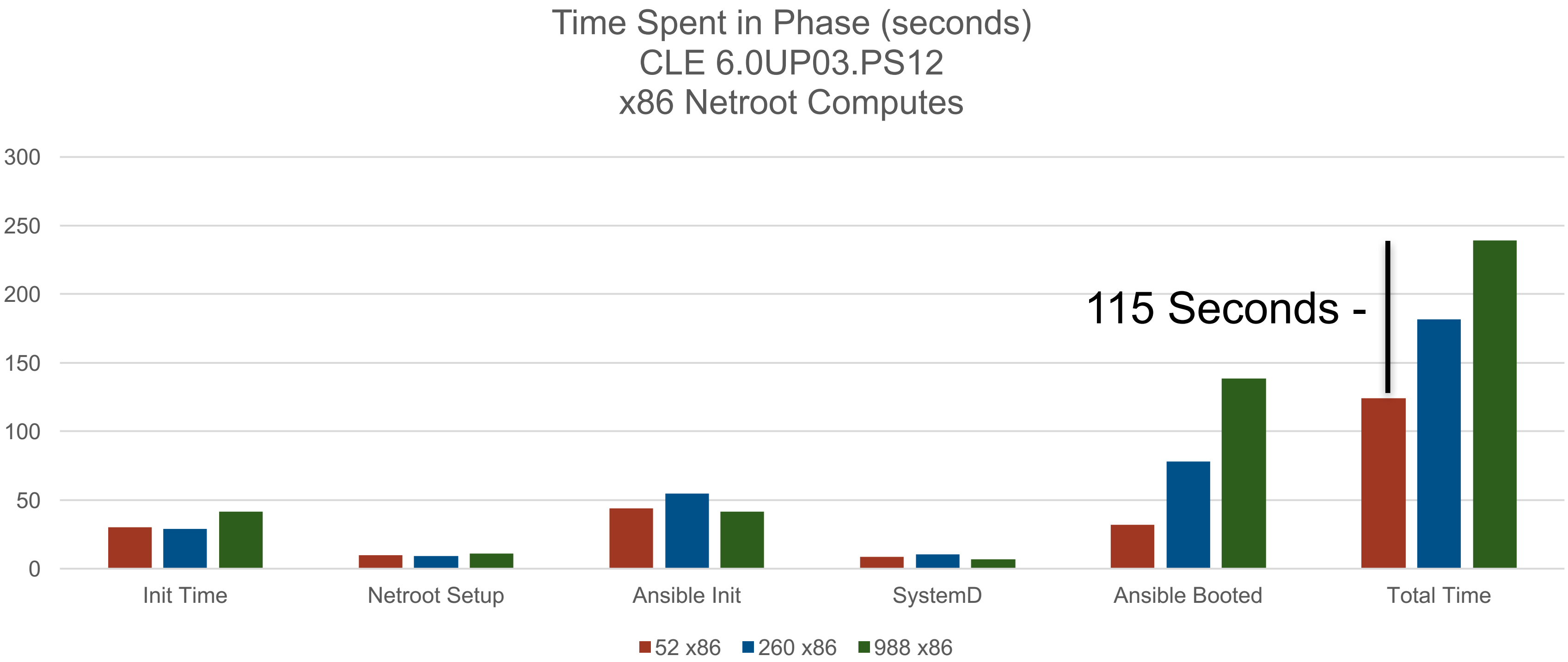
Xeon vs. KNL, Netroot, Low Load



KNL Netroot at Scale



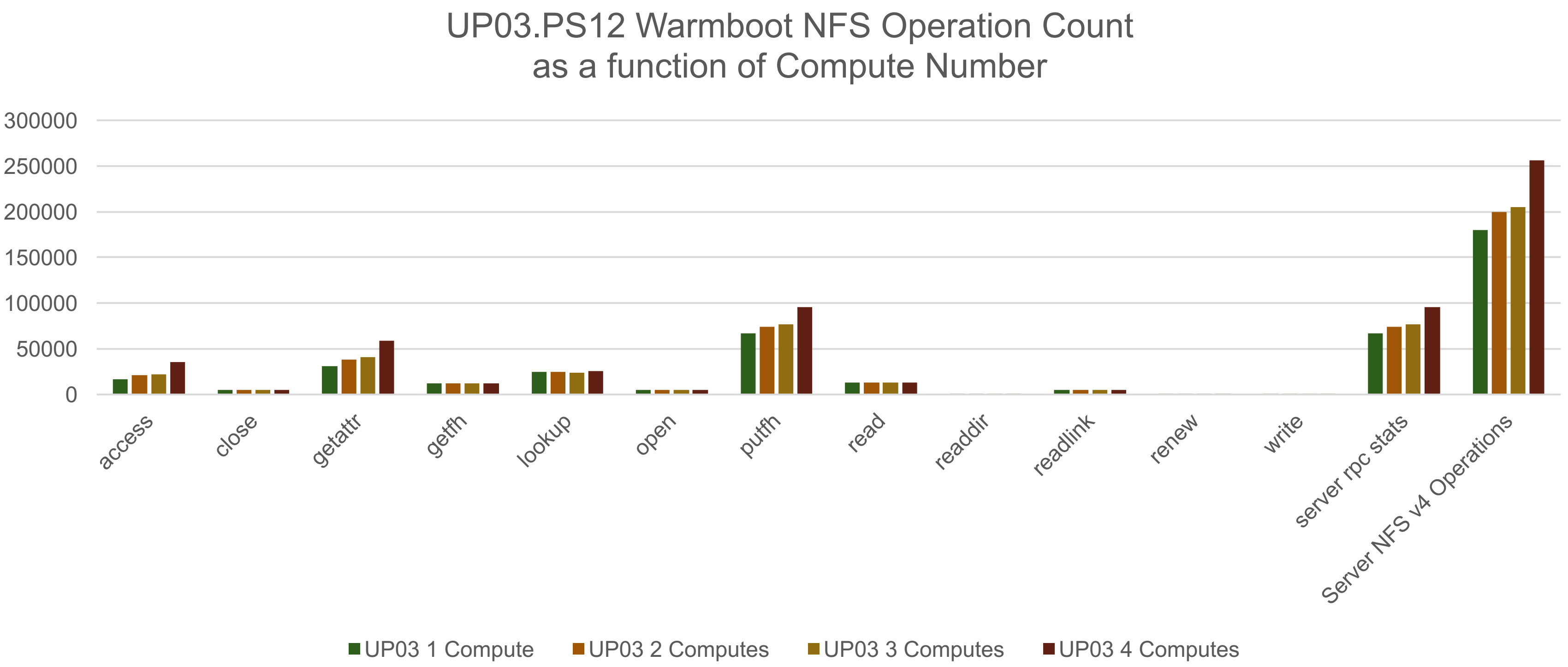
Scaling Characteristics of x86 Netroot Computes



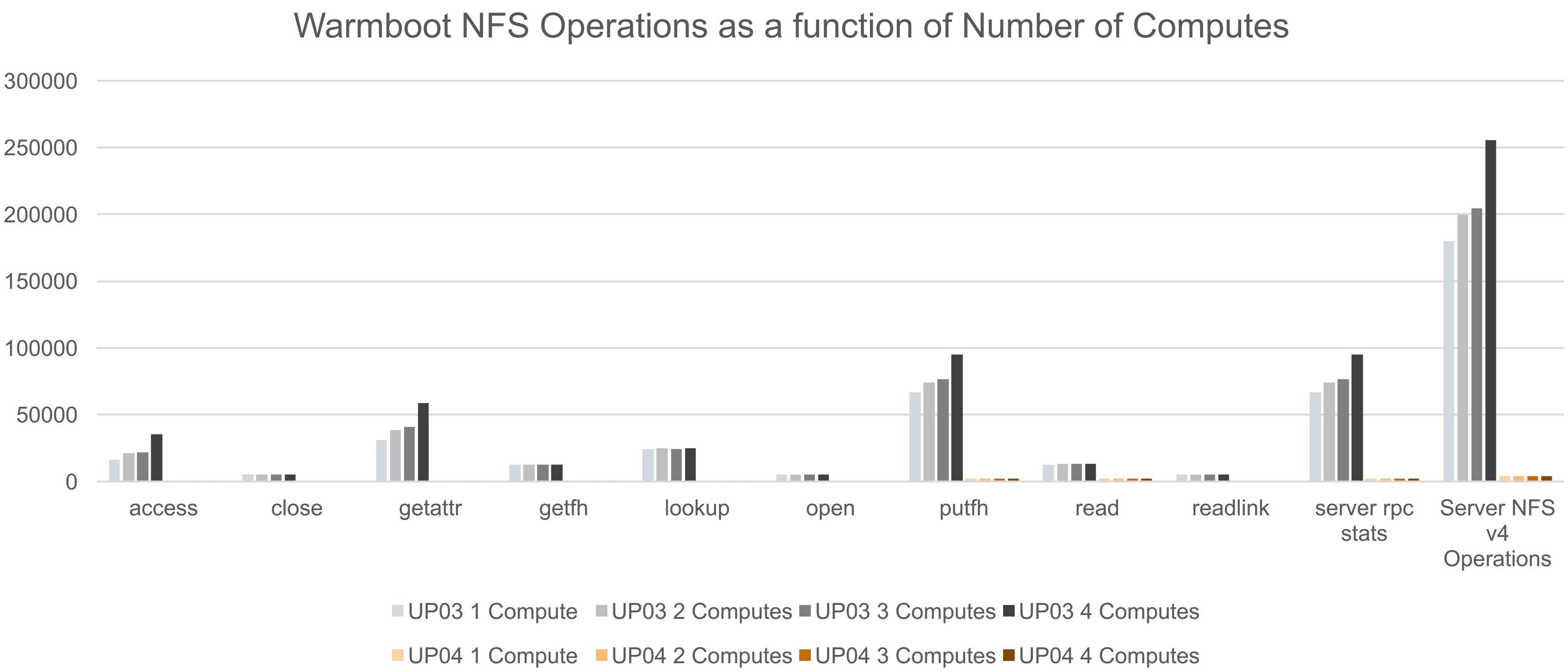
What's Coming in UP04?

- **SquashFS for Netroot, PE, Diags Images**
  - Performance
    - Reduce size of image transfer
    - Allow client nodes to resolve listdir, getattr, getxattr locally
    - Improvements outside of booting
  - Reduce overall size of image on boot node volume group
  - Focuses on improving t1:t2 ratio performance by reducing number of operations bootnode NFS server must honor
  - Most significant benefit on larger XC systems

Why You Should Care About Squashfs Images

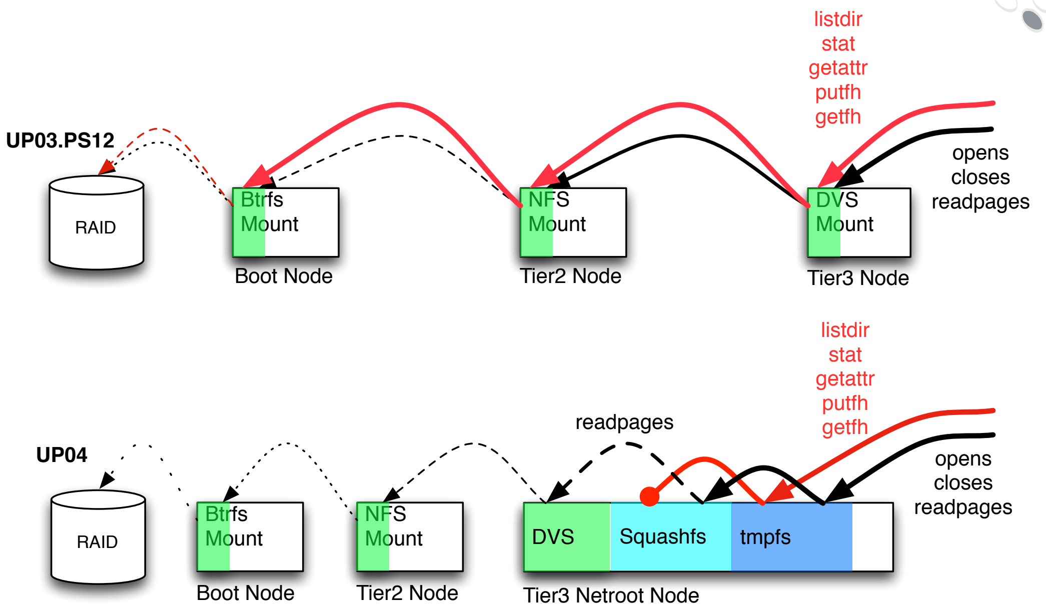


Why You Should Care About Squashfs Images (pt2)



Squashfs Images Changes Fundamentals of Load Requests

- **CLE 6.0UP03PS12**
  - 160,000 per tier2
  - 19,000 per client
  - 160,000*tier2+19,000*client
- **CLE 6.0UP04**
  - 3,735 per tier2
  - 35 per client
  - 3,735*tier2+35*client
- **10,000 compute system**
  - Recommended 400:1 ratio
  - UP03PS12: 198,320,000
  - UP04: 443,375
  - 447 fold reduction



Beyond UP04 (uncommitted short list of ideas)

- **t1:t2 ratio analysis for extreme scale**
  - Primary bottleneck for large XC systems
  - Boot performance data from partnering sites
- **Additional ansible filtering and modules**
  - Allows faster handling of large cfgset data structures
  - Simplifies and speed up ansible roles common to multiple deployments
- **Make sres windowing behavior configurable**
- **All-in-one jinja2 templates for service configuration**
  - Does not allow for easy site-additions to configuration
  - Removes upstream defaults
- **Heavier reliance on systemd drop-ins**
- **Better leverage of ansible**
  - Traditional push mode for reconfiguration, inventory management
  - Cray Ansible Modules
- **Decrease role of custom /init where possible**
- **More systemd native services**
- **BND Fanout Improvement**
  - Reduction in netroot initrd sizes
  - Parallelization of fanout
- **NIMS Speed Improvement**

Summary

- Primary speedup efforts target compute nodes
- KNL serial performance is particularly noticeable when running Ansible
- UP03.PS12 DVS patch has large overall impact to boot time
- Netroot performance is directly related to boot performance; largest bottleneck is at the boot node NFS server
- Configuration behavior is more consistent between deployments in newer releases
- Greater flexibility for booting multiple kinds of nodes simultaneously
- More speed improvements on the horizon

Special Thanks

- **Customer 'Boot-a-thon' sites**

Argonne NATIONAL LABORATORY

Los Alamos NATIONAL LABORATORY EST.1943

NERSC National Energy Research Scientific Computing Center

- **Crayons**
  - Richard Halkyard
  - Jane Kagan
  - Kelly Mark
  - Michael Primm
  - Dean Roe

Agenda

- Introduction to SMW/CLE system management
- New system management features since UP01
- Best practices for using Ansible
- Troubleshooting XC system booting problems
- Migrating SMW/CLE software from 7.2/5.2 to 8.0/6.0
- Intro to CMC/eLogin system management
- Migrating CIMS/CDL to CMC/eLogin
- CLE Boot Performance and Reliability
- Q & A

BOFs at CUG 2017

- **XC System Management Usability BOF**
  - Tuesday, May 9, 4:40pm-5:30pm
  - BoF 10C, Salon 3
- **eLogin Usability and Best Practices BOF**
  - Wednesday, May 10, 5:10pm-6:20pm
  - BoF 20B, Salon 2

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, REVEAL,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.

Q&A

Jeff Keopp keopp@cray.com

Joel Landsteiner jsl@cray.com

Harold Longley htg@cray.com