# Next Generation Cray Management System for XC Systems

Harold Longley, Cray Inc.

John Hesterberg, Cray Inc.

John Navitsky, Cray Inc.

# Next Generation CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- Configuration
- Booting
- Reconfiguration
- Summary
- Questions

# Safe Harbor Statement

This presentation may contain forward-looking statements that are based on our current expectations. Forward looking statements may include statements about our financial guidance and expected operating results, our opportunities and future potential, our product development and new product introduction plans, our ability to expand and penetrate our addressable markets and other statements that are not historical facts. These statements are only predictions and actual results may materially vary from those projected. Please refer to Cray's documents filed with the SEC from time to time concerning factors that could affect the Company and these forward-looking statements.

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

# Next Generation CMS Disclaimer

- The software described in this presentation has not yet been released!
- What is shown in this presentation may still change prior to release to Cray customers

# Next Generation CMS Introduction

- **Why change?**
  - Refresh system software architecture
    - Previous generation tools were developed over 10 years ago for XT systems and then grew with add-on features for XE/XK and XC systems
  - Move away from Cray unique solutions
    - e.g. the shared root
  - Make system administration:
    - easier
    - more common
    - require less downtime
  - Provide common solutions for multiple Cray products

# Next Generation CMS Introduction

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

# Next Generation CMS Introduction

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

# Next Generation CMS Introduction

- How?
  - Separation of software and configuration
    - Prescriptive image creation
      - Create images based on “recipes”
      - Utilizes rpm dependencies
    - Centralization of configuration
      - Provide structure for configuration data
      - Provide a tool to manage the configuration data
      - Provide a framework for configuration
    - Node customization at boot time or after adjusting configuration

# Next Generation CMS Introduction

- **When?**
  - Next major releases for CLE and SMW
    - CLE Rhine (presumably 6.0)
    - SMW Redwood (presumably 8.0)
  - SMW and CLE based on SLES 12
  - HSS controllers based on OpenSUSE 13
  - New software installers
  - New system management
    - IMPS (Image Management and Provisioning System)
    - NIMS (Node Image Mapping System)
    - Ansible configuration management
  - Much software is mostly unchanged (HSS, ALPS, NHC, RUR, etc.)

# IMPS

- **Image Management and Provisioning System**
  - Separation of Software and Configuration
- **IMPS Prescriptive Image Creation**
  - Create and update standard or custom images
- **IMPS Centralized Configuration**
  - Create, change, or add new configuration information, including site specific configuration
- **IMPS Node Deployment**

# NIMS

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

# Fresh Install

- **Install SMW with SLES12**
  - SLES12 with RAID1 setup for root filesystem
- **Configure disk space on SMW and Boot RAID**
  - Everything uses LVM and btrfs, xfs, or ext4
- **Create and populate repositories**
  - Shared across multiple Cray products
- **Install & Configure HSS**
  - SLES12/OpenSUSE, but otherwise largely unmodified
- **Configure CLE**
  - The “configurator” part of IMPS
- **Build CLE images**
  - Build boot images using prescriptive image recipes
- **Boot system to configure CLE storage**

# Booting combines images and configuration

- **Nodes boot unconfigured boot image**
- **Early /init does:**
  - Basic discovery of node ID, etc.
  - Imports read-only config set
  - Runs early Cray Ansible plays
    - Ansible plays contained in the image based on included software
    - Consumes system facts from the discovery
    - Consumes config set data
    - Updates /etc configuration
    - Updates running system
- **Multi-user boot does:**
  - Normal multi-user boot continues after /init turns control over to systemd
  - Some additional Ansible plays are run

# Why Ansible?

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

# System upgrades

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

# CLE Software Updates (patches, security fixes)

- **Install and configure in a BTRFS snapshot on SMW**
  - Instead of contaminating your existing, working environment
- **Apply rpms to update repositories**
- **Rebuild images and stage for booting**
- **Rolling the update out:**
  - Compute Nodes
    - Reboot between jobs
  - Service Nodes
    - Live update service nodes if possible
    - Reboot service nodes if necessary

# Next Generation CMS Agenda

- Introduction
- **Overview of new concepts**
- Software installation
- Configuration
- Booting
- Reconfiguration
- Summary
- Questions

# Overview of New Concepts

- **Separate software and configuration**
- **Management of software**
  - Repositories
  - Image recipes
  - Package collections
  - Image root
  - Boot image
- **Centralized configuration**
  - How it gets created and managed
- **Boot process**
  - Customizing software with configuration during boot process
- **Software installation**

# Overview - separate software and configuration

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

# Overview - management of software

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

# File formats

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

# Repositories

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

# Image Recipes

- **Each default image type has an image recipe installed on SMW**
  - Compute, service, login, DAL (Direct Attached Lustre)
  - All Cray image recipes are named to avoid naming conflicts
- **Each image recipe is in a JSON file**
  - Has name and description
  - Includes package collections, packages (rpms), and repositories
- **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, or repository is listed
- **Custom image recipes can be created to serve specific purposes**
- **SMW location:**
  - /etc/opt/cray/imps/image_recipes.d/

# Package Collections

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

# Image Recipe Example 1

```
{
    "compute_cle_rhine_sles_12_x86-64_ari": {
        "description": “Compute image for SLES 12",
        "package_collections": {
            "cle-compute_rhine_sles_12_kernel_ari": {
                "rationale": "Provides the needed kernel and kernel drivers."
            },
            "cle_rhine_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": {},
        "repositories": {
…
        }
    },
}
```

# Image Recipe Example 2

```
"repositories": {
            "cle_rhine_sles_12_x86-64_ari": {
                "rationale": "A base set of Cray provided packages for SLES
12."
            },
            "cle_rhine_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_rhine_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_rhine_sles_12_compute": {
        "description": "Collection of packages for base SLES compute node
capabilities.",
        "package_collections": {
            "cle_rhine_sles_12_base": {
                "rationale": "compute nodes need base software"
            },
            "cle_rhine_sles_12_compute_cray": {
                "rationale": "Cray packages installed on a compute node"
            },
        },
        "packages": {
            "ksh": {
                "rationale": "Needed for Test group."
            },
            "tcsh": {
                "rationale": "Required by some applications and some customer sites."
            }
```

# Extended Image Recipe Support

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

# Extended Image Recipe Example

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

# Image Roots and Boot Images

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

# Boot Images

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

# Overview - centralized configuration

- Config sets
- IMPS Distribution Service (IDS)
- Configuration data
- Configurator

# 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
  - config set CLE
  - global config set which covers both the management domain (“SMW” and/or “CIMS”) as well as truly global data).

# Config sets – directory structure on node

- **From the end node's perspective, it's just a directory of config files for the current 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, etc.

# Config sets – directory structure on SMW

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

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

- **Other config sets on SMW**

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

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

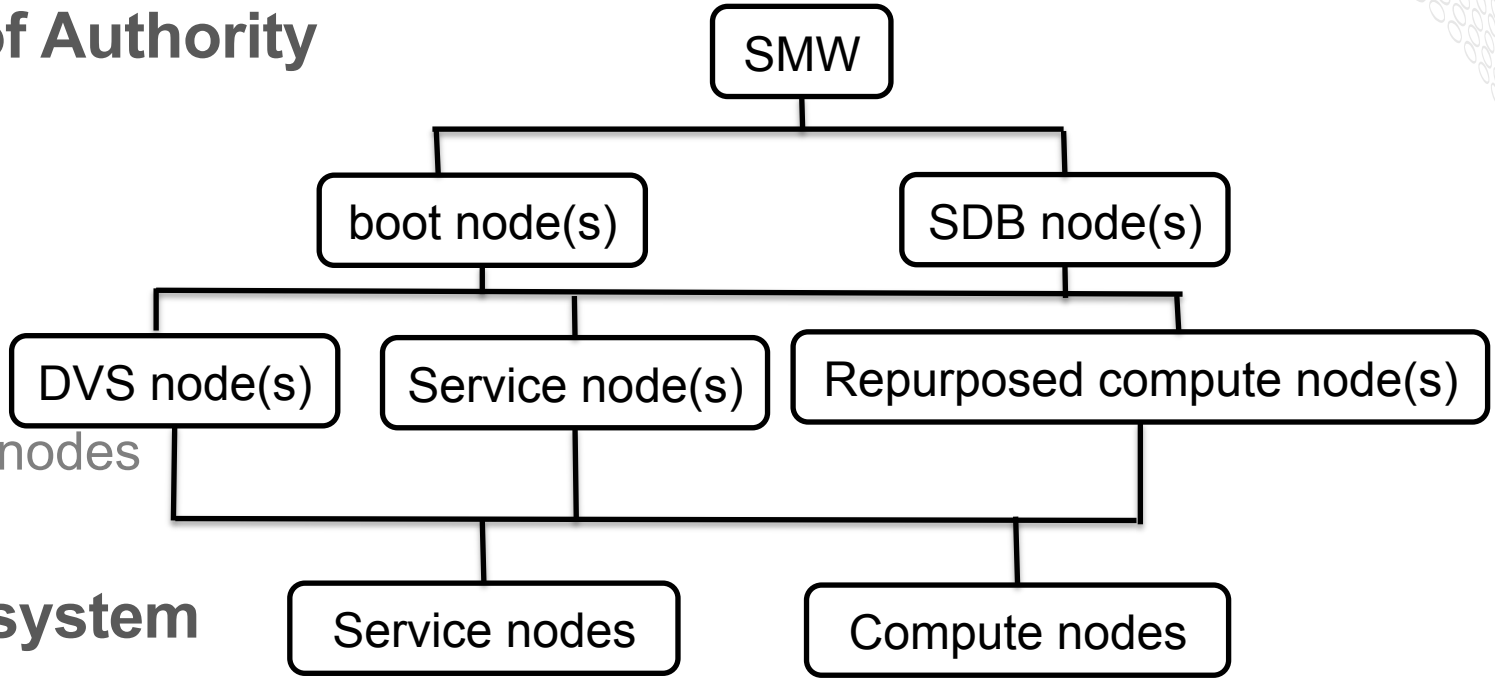
  `smw:/etc/opt/cray/config/global`

# Config set distribution - IDS

- **In order for the config set to be available on all nodes it is distributed by a service called the IMPS Distribution Service (IDS)**
- **IDS leverages the 9P network file system and the Linux automounter facility to share the files from the SMW to the entire XC system**
  - 9P can re-share a 9P mount
  - Read-only allows us to leverage caching
  - 9P support built into modern kernels
  - autofs allows for resiliency and failover
- **The content and use of the config set is independent of the distribution mechanism**

# Config set distribution – IDS scability

- **Server of Authority**
- **Tier 1**
- **Tier 2**
  Not login nodes
- **Rest of system**



# Config set data

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

# Configurator

- **The configurator**
  - Completely data driven by files called templates
  - Merge existing configuration data with new templates
- **Configuration templates**
  - Provide useful documentation for the value
  - Provide useful defaults
  - Provide value and syntax checking to be used by configurator
- **Run interactive configurator to collect new data**
  - Will automatically prompt/merge new data elements
  - System administrator's "answers" to questions become new default
- **Iterate on the configurator as necessary**
  - Admin can configure a specific service
    - impscli update config_set p0 with state unset service cray_alps level basic
    - impscli update config_set p0 with state all service cray_alps level advanced

# Config Templates - schema

- **Design of template schema drives how information is gathered**
  - YAML format
  - Cray-provided templates start with “cray_”
  - <service_name>_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
  - Changelog – history of changes to the configured value
  - 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
      changelog:
      - 2015-03-31T15:20:51 - Configured by IMPS
      comments: []
      configured: true
      config_after: []
      default_value: true
      guidance: "Cray Scalable Services allows you to define which servers (nodes)
         are used to define the scaling of your system.  \nOnce defined, these
         servers will be used for various services like DVS servers and other services
         to provide horizontal scaling of those services.\nScalable services defines
         a logical tree of servers, starting with the Server of Authority (SoA)
         followed by tier one and then tier two servers.\n                    SoA\n
         \                 / \\\n              tier1   tier1\n              /
         \  |   \\\n          tier2  tier2  tier2\nScalable services need to
         be properly defined in order for correct operation of your system. Would
         you like to configure Cray Scalable Services now?"
      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
  - Changelog - history of changes to the configured value
  - 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: 'The tier one servers must have a direct IP
              connection to the Server of Authority (SoA) which
              is typically the SMW.  Any node that is directly
              connected to the SoA can be a tier one server.
              In a typical Cray system, the boot node can always
              serve this role.  However, often the SDB node can
             also perform this role assuming it is properly
             configured to reach the SMW.  Adding additional
              tier one servers provides enhanced resiliency.
             Enter at least one tier one server by its cname.'
            level: required
            multival_key: false
            purge: false
            regex: ^c(\d+)-(\d+)c([0-2])s(\d[0-5]?)n([0-3])$
            title: tier one servers
            type: list
  scope_type: class
  changelog:
  - 2015-03-24T17:03:11 - Configured by IMPS
  comments: []
  guidance: null
  purge: false
```

# Config Templates – settings multival

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

# Config Templates Example – settings multival

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

# Configurator - impscli

- **Updating a config set – actions by IMPS configurator**
  - Clone the config set as a backup
  - Run pre-configuration scripts
  - Validate templates and configuration data
    - YAML syntax validation check
    - Schema validation check
    - Merge templates
  - Prompt for all information to be configured
  - Run post-configuration scripts
  - Remove backup config set

# Overview - Boot process configuration

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

# 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 optimal location for the site so they don't have to tinker with custom images just to get their stuff there
- **Integrating site Ansible plays**
  - smw:/var/opt/cray/imps/config/sets/<config_set>/ansible/myplay
- **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 – example.yaml

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

- hosts: localhost

  vars:
    run_after:
    - common

  roles:
    - example
```

# Ansible – ansible-playbook 1

```
boot# ansible-playbook –v /etc/opt/cray/config/current/ansible/example.yaml
PLAY [localhost]
*********************************************************************
GATHERING FACTS
*********************************************************************
ok: [localhost]
TASK: [example | task template, set variables]
*********************************
ok: [localhost] => {"ansible_facts": {"myservice_bar": "9999",
"myservice_baz": "turnip", "myservice_foo": "true"}}

TASK: [example | task template, create myservice.conf config]
*****************
ok: [localhost] => {"changed": false, "gid": 0, "group": "root", "mode":
"0644", "owner": "root", "path": "/etc/myservice.conf", "size": 199,
"state": "file", "uid": 0}
```

# Ansible – ansible-playbook 2

```
TASK: [example | task copy, check for file]
************************************

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

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

skipping: [localhost]
```

# Ansible – ansible-playbook 3

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

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

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

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

# Ansible – example tasks

```
boot# cd /etc/opt/cray/config/current/ansible/roles/example/tasks
boot# ls
copy.yaml  lineinfile.yaml  main.yaml  service.yaml  shell.yaml  template.yaml
boot# cat main.yaml
---
- name: task main, template example
  include: template.yaml
- name: task main, make copy of config file
  include: copy.yaml
- name: task main, customize for second instance
  include: lineinfile.yaml
- name: task main, turn on rsyncd
  include: service.yaml
- name: task main, run a shell script, but only once
  include: shell.yaml
```

# Ansible – example tasks

```
boot# cat template.yaml
---

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

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

```
boot# cat copy.yaml
---

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

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

# Ansible – example tasks

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

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

```
boot# cat shell.yaml
---

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

# Ansible references

- **Ansible web site:**
  - http://www.ansible.com/configuration-management
- **Wikipedia:**
  - http://en.wikipedia.org/wiki/Ansible_%28software%29
- **Source:**
  - https://github.com/ansible/ansible
- **Documentation:**
  - http://docs.ansible.com/

# 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
- **in_init flag indicates whether in the first phase or not**

# 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 file systems
    - When the system starts, these file systems 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
  - File systems 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

# Booting XC diskless nodes – method 1

- **Pure tmpfs root**
  - Use cases:
    - Compute nodes for well defined workloads
    - Service nodes for internal services
  - Put root filesystem directly into tmpfs
    - Very fast and standard, but consumes memory

# Booting XC diskless nodes – method 2

- **Netroot**
  - tmpfs on top of readonly network filesystem using overlayfs
  - Use cases:
    - Compute nodes for diverse workloads
      - Could be analogous to …
        - a default shared root or /dsl install (~12GB)
        - a default SLES12 install (~4GB)
    - Login nodes
    - Compute nodes with high memory footprint sensitivity
  - Leverage the network to minimize memory footprint
  - Overlayfs supported in SLES12, recently accepted upstream

# Overview of software installation

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

# SLES 12 - Differences

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

# SLES 12 btrfs (B-trees filesystem)

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

# SLES 12 - systemd

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

# SLES 12 - systemd

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

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

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

# Storage Layout – concepts

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

# Storage layout – SMW internal disks

- SMW boot disk is a RAID1 pair of drives (mirrored) for swap, /boot, and /
- Power Management requires separate SMW disk

| File system | Type | Description |
|---|---|---|
| /boot | ext3 | Booting area |
| swap | swap | Swap |
| / | btrfs | Root (/) file system with btrfs subvolumes |
| /var/lib/pgsql | ext4 | HSS postgresql database |

# Storage layout – SMW df

- **SMW disks are on device mapper named disk devices**

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

# Storage layout – SMW /etc/fstab

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

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

# Storage layout – SMW on Boot RAID

- SMW file systems in LVM Volume Group

| Owning node | File system | Type | Description |
|---|---|---|---|
| SMW | /home | xfs | Home directories on SMW |
| SMW | /var/lib/mysql | btrfs | HSS MySQL database |
| SMW | /var/opt/cray/disk/1 | xfs | Logs, dumps, debug |
| SMW | /var/opt/cray/repos | btrfs | IMPS repositories |
| SMW | /var/opt/cray/imps | btrfs | IMPS data |

# Storage layout – boot node on Boot RAID

- CLE boot node file systems in LVM Volume Group

| Owning node | File system | Type | Description |
|---|---|---|---|
| boot | /home | btrfs | Home directory for crayadm |
| boot | /var/opt/cray/imps | btrfs | IMPS data for PE |
| boot | /non_volatile | btrfs | Persistent data for service nodes, including /var if necessary |

# Storage layout – SDB node on Boot RAID

- CLE SDB node file systems in LVM Volume Group

| Owning node | File system | Type | Description |
|---|---|---|---|
| SDB | /var/lib/mysql | xfs | SDB database |
| SDB | /alps_shared | btrfs | ALPS data |

# Fresh install - preparation

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

# Fresh install

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

# Installer

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

# Staged upgrades 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
  - Update config set using IMPS configurator
  - Use NIMS to map boot images and kernel parameters to nodes
- **When ready to use the new software, use snaputil to choose snapshot**
  - 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)
  - Boot CLE

# Reverting to a previous software version

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

# Programming Environment

- **Same PE software content can be used for:**
  - Compute
  - Login
  - Cray Development and Login (CDL, or esLogin)
- **Installed and managed on the SMW**
  - Uses the craype-installer
  - Deployed to boot node for internal XC nodes
  - Deployed to Cray Integrated Management Server (CIMS) for CDL
- **PE will be a network filesystem on diskless XC nodes**
  - NFS for Login nodes
  - NFS->DVS for Compute nodes

# Next Generation CMS Agenda

- Introduction
- Overview of new concepts
- **Software installation**
- Configuration
- Booting
- Reconfiguration
- Summary
- Questions

# Software Installation

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

# Software Installation – Fresh Install Only

- **Install SLES 12 on SMW from bootable DVD**
  - Similar to past releases
  - Configure RAID1 on a pair of SMW internal disks
  - Create /, swap, and /boot filesystems
  - Install software after confirming installation choices
- **Gather software ISOs**
  - All SUSE and CentOS ISOs should be in /root/isos
- **Mount SMW media**

  smw# mkdir -p /media/SMW

  smw# mount -o loop,ro /root/isos/smw-redwood-image-sle12-8.0.0.YYYYMMDDhhmm.iso /media/SMW

# Software Installation – Storage Sets

- **Create storage set configuration**
  - Copy initial cray_storage_sets.yaml from SMW media
    smw# cp –p /media/SMW/cray_storage_sets.yaml /etc/opt/cray/config/global/config
  - Customize with disk devices from boot RAID for the SMW, boot, and SDB LVM volume groups
  - Check sizes for file systems, adjust as needed
    - Sizes shown in this presentation might have different recommendations in final release
    - Consult with Cray before changing any of the file system types
  - Nodes will “self configure” volume groups, volumes, and file systems from this file
    - SMW – while running installer during fresh install
    - boot and SDB nodes – during first boot after fresh install

# Software Installation – Storage Sets 1

```
cray_storage_sets:
  enabled: false
  settings:
    lvm_volume_groups:
      data:
      # Boot node volume group. This holds the IMPS, home, alps_shared, and non_volatile filesystems
      - key: boot_node_vg
        devices:
        - /dev/disk/by-id/<your_disk1>
        hostid: c0-0c0s0n1
      # SDB node volume group. This holds the SDB (mysql) filesystem
      - key: sdb_node_vg
        devices:
        - /dev/disk/by-id/<your_disk2>
        hostid: c0-0c0s1n1
      # SMW volume group. This holds the IMPS, home, log, HSS database (mysql), and repository filesystems.
      - key: smw_node_vg
      devices:
        - /dev/disk/by-id/<your_disk3>
        hostid: smw
```

# Software Installation – Storage Sets 2

```
lvm_volumes:
    data:
    # Default CLE Boot/SDB node storage set
    - key: cledefault_db
      description: volume for CLE system database (SDB) data
      fs_mount_point: /var/lib/mysql
      fs_size: 20
      fs_type: xfs
      lvm_volume: db
      lvm_volume_group: sdb_node_vg
      type: lvm
    - key: cledefault_home
      description: volume for CLE local home directories
      fs_mount_point: /home
      fs_size: 20
      fs_type: btrfs
      lvm_volume: home
      lvm_volume_group: boot_node_vg
      type: lvm
    - key: cledefault_alps
      description: volume for internal ALPS state
      fs_mount_point: /alps_shared
      fs_size: 20
      fs_type: btrfs
      lvm_volume: alps
      lvm_volume_group: sdb_node_vg
      type: lvm
    - key: cledefault_imps
      description: volume for local IMPS data such as PE images
      fs_mount_point: /var/opt/cray/imps
      fs_size: 250
      fs_type: btrfs
      lvm_volume: imps
      lvm_volume_group: boot_node_vg
      type: lvm
    - key: cledefault_non_volatile
      description: volume for persistent storage
      fs_mount_point: /non_volatile
      fs_size: 200
      fs_type: btrfs
      lvm_volume: nvolatile
      lvm_volume_group: boot_node_vg
      type: lvm
```

# Software Installation – Storage Sets 3

```
# Default SMW storage set
- key: smwdefault_home
  description: volume for SMW home directories
  fs_mount_point: /home
  fs_size: 300
  fs_type: xfs
  lvm_volume: home
  lvm_volume_group: smw_node_vg
  type: lvm
- key: smwdefault_db
  description: volume for HSS database instance
  fs_mount_point: /var/lib/mysql
  fs_size: 50
  fs_type: btrfs
  lvm_volume: db
  lvm_volume_group: smw_node_vg
  type: lvm
- key: smwdefault_log
  description: volume for logging directories
  fs_mount_point: /var/opt/cray/disk/1
  fs_size: 500
  fs_type: xfs
  lvm_volume: log
  lvm_volume_group: smw_node_vg
  type: lvm
- key: smwdefault_imps
  description: volume for IMPS image and repo storage
  fs_mount_point: /var/opt/cray/imps
  fs_size: 2000
  fs_type: btrfs
  lvm_volume: imps
  lvm_volume_group: smw_node_vg
  type: lvm
- key: smwdefault_repos
  description: volume for IMPS image and repo storage
  fs_mount_point: /var/opt/cray/repos
  fs_size: 300
  fs_type: btrfs
  lvm_volume: repos
  lvm_volume_group: smw_node_vg
  type: lvm
```

# Software Installation – Storage Sets 4

```
storage_sets:
    data:
    - key: cledefault
      description: The default CLE storage set
      member_volumes:
      - cledefault_db
      - cledefault_home
      - cledefault_alps
      - cledefault_imps
      - cledefault_non_volatile
    - key: smwdefault
      description: The default SMW storage set
      member_volumes:
      - smwdefault_home
      - smwdefault_db
      - smwdefault_log
      - smwdefault_imps
      - smwdefault_repos
```

# Software Installation – Installer

SMWinstall [--target=NAME] [--media=PATH] [--plus-media=PATH, --plus-media=PATH, ...] [options]

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

# Software Installation – Installer

- **Install SMW, CLE and Security Updates together**

  smw# /media/SMW/SMWinstall --plus-media=/root/isos/cle-rhine-image-sle12-6.0.0. YYYYMMDDhhmm.iso --plus-media /root/isos/sleupdate-image-rhine.2015-02-25.iso --target=${SNAPSHOT}
  - If no target on command line, a snapshot will be created
- **All tasks on SMW media done first, then tasks from the other media**
  - SMW VG created, file systems created
  - Changes made in snapshot
- **Logs created in /var/adm/cray/logs/install*.log**
  - Very verbose log file with all zypper/rpm messages

# Software Installation – snaputil

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

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

# Software Installation – snaputil list

- **List all snapshots**

smw# snaputil list

| Status | Name | Size (MB unshared) | Created |
|---|---|---|---|
| | @ | 20355.4 | 2014-11-07 11:10:12 |
| | SLES12 | 8.36 | 2014-11-07 11:58:38 |
| | SMW-8.0DV00_CLE-6.0DV00.20150304 | 12.53 | 2015-03-03 07:15:57 |
| cur,def | SMW-8.0DV00_CLE-6.0DV00.20150323 | 757.5 | 2015-03-23 08:54:40 |

# Software Installation – snaputil create

- **Create a new snapshot**

```
smw# snaputil create demo
Created subvolume demo in /media/root-sv/snapshots/demo
smw# snaputil list
```

| Status | Name | Size (MB unshared) | Created |
|---|---|---|---|
| | @ | 20355.4 | 2014-11-07 11:10:12 |
| | SLES12 | 8.36 | 2014-11-07 11:58:38 |
| | SMW-8.0DV00_CLE-6.0DV00.20150304 | 12.53 | 2015-03-03 07:15:57 |
| cur,def | SMW-8.0DV00_CLE-6.0DV00.20150323 | 757.5 | 2015-03-23 08:54:40 |
| | demo | 0.2 | 2015-03-26 13:32:57 |

# Software Installation – snaputil show

- **Show a snapshot**

```
smw# snaputil show demo
boot menu       : False
booted          : False
btrfs_object_id : 1177
cle_version     : 201503230201
created         : 2015-03-26 13:32:40
default         : False
initrd          : initrd-3.12.28-4-default
kernel          : vmlinuz-3.12.28-4-default
name            : demo
path            : /media/root-sv/snapshots/demo
read-only       : False
smw_version     : 7.3.0-1.0000.36198.499
smwha_version   : None
storage_set     : smwdefault
subvolumes      :
   /var/lib/mysql:MW-8.0DV00_CLE-6.0DV00.20150323
   /var/opt/cray/repos:SMW-8.0DV00_CLE-6.0DV00.20150323
total size      : 1729.60 MB
unshared size   : 0.02 MB
updated         : 2015-03-26 13:32:57.969610
```

# Software Installation – snaputil default

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

```
smw# snaputil default demo
subvolume demo is now default.
smw# snaputil list
```

| Status | Name | Size (MB unshared) | Created |
|---|---|---|---|
| | @ | 20355.4 | 2014-11-07 11:10:12 |
| | SLES12 | 8.36 | 2014-11-07 11:58:38 |
| | SMW-8.0DV00_CLE-6.0DV00.20150304 | 12.53 | 2015-03-03 07:15:57 |
| cur | SMW-8.0DV00_CLE-6.0DV00.20150323 | 757.5 | 2015-03-23 08:54:40 |
| def | demo | 0.2 | 2015-03-26 13:32:57 |

# Software Installation – snaputil diff

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

```
smw# snaputil diff SMW-8.0DV00_CLE-6.0DV00.20150323 demo
etc/motd
root/.bash_history
root/.viminfo
```

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

```
smw# snaputil diff demo SMW-8.0DV00_CLE-6.0DV00.20150323 etc/motd
--- /media/root-sv/snapshots/demo/etc/motd      2014-10-14
03:52:43.000000000 -0500
+++ /media/root-sv/snapshots/SMW-8.0DV00_CLE-6.0DV00.20150323/
etc/motd   2015-03-26 13:46:35.738501158 -0500
@@ -0,0 +1 @@
+test of change to /etc/motd
```

# Software Installation – snaputil rename/delete

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

| Status | Name | Size (MB unshared) | Created |
|---|---|---|---|
| | @ | 20355.4 | 2014-11-07 11:10:12 |
| | SLES12 | 8.36 | 2014-11-07 11:58:38 |
| | SMW-8.0DV00_CLE-6.0DV00.20150304 | 12.53 | 2015-03-03 07:15:57 |
| cur,def | SMW-8.0DV00_CLE-6.0DV00.20150323 | 757.5 | 2015-03-23 08:54:40 |

# Software Installation – Configure SMW

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

# Software Installation – Configure SMW

- **Update only - actions done in new snapshot before SMW reboot**
  - After software installed, chroot to new snapshot
    - Create new boot images with imgbuilder
    - Update config set for CLE
    - Update global config set
  - When new configuration is ready, reboot SMW
  - Discover XC hardware with xtdiscover
  - Discover routing configuration of HSN with rtr
  - Update firmware and BIOS with xtzap

# Software Installation – imgbuilder

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

imgbuilder ([--map [--nims-group=GROUP][--nims-map=MAP][--partition=PART,...]]|--bootstrap-nims [--partition=PART,...]) [options] [--<key>=<val>...]

Options:

| | |
|---|---|
| -c --config=FILE | Use the specified configuration YAML |
| -g --image-group=GROUP | Use the specified image group |
| --map | Add newly-built images to the NIMS |
| --nims-group=GROUP | Map the images to a specific NIMS group |
| --nims-map=MAP | Map the images to a specific NIMS map table |
| --partition=PART | When mapping images, update a partition's active map |
| --bootstrap-nims | Update NIMS table on new systems lacking node groups |

# Software Installation – imgbuilder

- **The imgbuilder configuration file lists the set of images to build**

  /etc/opt/cray/config/global/config/cray_image_groups.yaml

- **Image names defined in the above configuration file have runtime values available to them. This includes:**

  {date} includes the current system date (20140314)

  {time} includes the current system time. (134514)

  {host} includes the current system hostname

  {cle_release}

  {cle_build}

# Software Installation – imgbuilder

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

    - recipe: "compute_cle_rhine_sles_12_x86-64_ari"
      dest: "{compute_prefix}_{cle_release}-{cle_build}_my${date}.cpio”
- **Then when invoking imgbuilder, we can specify the value to use for {compute_prefix}:**

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

  my_compute_cle_rhine-201503210201_my20150425.cpio

# Software Installation – NIMS map

- **NIMS map associates a node with a boot image, a load file, a config set, and kernel parameters**
  - The node can be assigned to an arbitrary group
  - Each map can hold all of the nodes for a partition
  - Maps are associated with one partition and should not span partitions
- **There is only one active map per partition at a time**
- **The system administrator can control which map is the active map for a partition**
- **Use NIMS to control the aforementioned attributes which in turn control how the node boots**

# Software Installation – nimscli

- **nimscli [action] [action_options] for [filter_options]**
- **nimscli performs [action] on all nodes that are specified with the given filters**

view - Query nimsd for image mappings and print that information to the screen.
set - Set the values specified in [action_options] on all nodes specified with the given [filter_options]
unset - Unset the values specified in [action_options] on all nodes specified with the given [filter_options]
get - Get the specified attribute of the map [--version, --default_config_set, --partition, --path]
maps - List the available maps for the specified partition (lists all partitions if none is specified)
create - Create a new NIMS map
merge - Merge the specified map into the active map.

# Software Installation - nimscli

- **Filter options**
  - Specify which nodes have the given action performed on them
  - At least one filter option must be specified for set or unset action.

  -a, --all Every node in the system/partition

  -n, --node NODELIST Comma-separated list of nodes. Higher level names like 'c1-2c2' will include all nodes underneath

  -g, --group NAME Comma-separated list of groups

  -t, --type [hardware_type] Any nodes belonging to [hardware_type]. [hardware_type] can be one of compute, service, sdb, or boot
- **Action options**

  -i, --image image.cpio Bootable image cpio

  -p, --parameter key=value key value kernel parameter pair. If value is not included, this matches all nodes with 'key', regardless of 'value'

  -l, --loadfile NAME Name of loadfile

  -m, --map <my-map> Act on <my-map> rather than the active map for the specified partition

# Software Installation – nimscli

- **To see the set of image mappings:**
  smw# nimscli view
- **To set an image to boot on all service nodes:**
  smw# nimscli set --image /path/to/image.cpio for --group service
- **To set a group on some nodes, and then set the image for that group**
  smw# nimscli set --group dal for --node c0-0c0s1n1,c0-0c0s3n2
  smw# nimscli set --image /path/to/image.cpio for --group dal
- **To set kernel parameters for huge page sizes on compute nodes:**
  smw# nimscli set --duplicate --parameter hugepagesz=1M for --group compute
  smw# nimscli set --duplicate --parameter hugepagesz=512M for --group compute
  smw# nimscli set --duplicate --parameter hugepagesz=128M for --group compute
- **To unset kernel parameter for a particular hugepagesz key on one node:**
  smw# nimscli unset --parameter hugepagesz=512M for --node c1-2c0s4n3
- **To unset an image on some nodes:**
  smw# nimscli unset --image for --node c1-2,c1-3,c1-4

# Software Installation – Create/assign images

- **Fresh install**

  smw# imgbuilder --bootstrap-nims

  smw# nimscli set --group login for --node c0-0c0s0n2

  smw# nimscli set --group dal for --node c0-0c0s1n1,c0-0c0s3n2

  smw# imgbuilder --map

- **Software update**

  smw# imgbuilder --map

# Next Generation CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- **Configuration**
- Booting
- Reconfiguration
- Summary
- Questions

# Configuration

- **impscli**
- **Manipulate config sets**
  - Create or update CLE config set
  - Update global config set
  - Display or search config set data
  - Validate config sets
- **IMPS configurator**

# Configuration – impscli

- **impscli is the command line interface to IMPS with several subcommands which operate on different object types**
  - build, clone, create, diff, export, extend, import, list, package, prepare, push, remove, search, set, show, strip, sync, update, validate
- **Run single line commands**

  smw# impscli command operands

# Configuration – impscli config_set

impscli create config_set <name>
impscli update config_set <name>
impscli search config_set <name> [params]
impscli validate config_set <name>
impscli update config_set <config_set_name> state unset
impscli update config_set <config_set_name> state set
impscli update config_set <config_set_name> state all
impscli update config_set <config_set_name> state all service <service_name>
impscli create config_set <config_set_name> level required
impscli update config_set <config_set_name> service <service_name> level advanced

# Configuration – config set manipulations

- **Create CLE config set for partition p0 - fresh install**

  smw# impscli create config_set p0

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

  smw# impscli clone config_set p0 to p0-preupgrade-YYYYMMDD

  smw# impscli update config_set p0

- **Clone and update global config set**
  - Both fresh install and update

  smw# impscli clone config_set global to global-preupgrade-YYYYMMDD

  smw# impscli update config_set global

# Configuration – config set manipulations

- **Display or search config set data**
  - smw# impscli search config_set p0
  - smw# impscli search config_set global
- **Validate config sets**
  - smw# impscli validate config_set p0
  - smw# impscli validate config_set global
- **Clone config set to archive them post upgrade**
  - smw# impscli clone p0 p0-postupgrade-YYYYMMDD
  - smw# impscli clone global-postupgrade-YYYYMMDD

# Configuration – IMPS configurator

- **Updating or creating a config set with the IMPS configurator**
  - Any required questions will be asked
    - Guidance is displayed for the question
    - Current setting (which may be default) is shown
    - System administrator “answers” the “question” with appropriate data
    - Can “skip” questions or entire services on the first pass
    - Or disable service on the first pass and then selectively configure a previously unconfigured service on another pass
- **User interface for the IMPS configurator is still being updated and improved**
  - Simple example on next few slides shows sample interface

# Configuration – IMPS configurator – service

```
**********Service Configuration**********
Service: Cray Scalable Services
Guidance: Cray Scalable Services allows you to define which servers (nodes) are used to
      define the scaling of your system.
    * Once defined, these servers will be used for various services like DVS servers
      and other services to provide horizontal scaling of those services.
    * Scalable services defines a logical tree of servers, starting with the Server of
      Authority (SoA) followed by tier one and then tier two servers.
    *                SoA
    *                /   \
    *            tier1   tier1
    *             /      |      \
    *         tier2  tier2  tier2
    * Scalable services need to be properly defined in order for correct operation of
      your system. Would you like to configure Cray Scalable Services now?
Enable this service? [y/n/? show options] (default: y): y
```

# Configuration – IMPS configurator – setting

**********Setting Configuration**********
Service: Cray Scalable Services
Setting: server_of_authority
Guidance: The Server of Authority (SoA) is the holder of the authoritative configuration information for the whole Cray system.
* This is typically the SMW. The name used here must be reachable (pingable) by the tier one servers such as the boot node.
* Enter the hostname of the server of authority as known by the tier one servers.
Current Value: 'Server of Authority (SoA)' is set to:
'smw'
Press '<cr>' to keep the current value(s) for 'Server of Authority (SoA)' (? for options):

# Configuration – IMPS configurator – options

Press '<cr>' to keep the current value(s) for 'Server of Authority (SoA)' (? for options): ?

Options:
Enter the following at the prompt for more actions:
<cr> - Keep current value for this setting and continue
? - Print this menu
e - Edit this setting
f - Show filename and location of configuration schema for this setting
g - Show configuration guidance for this setting
s - Skip configuration of this setting and continue
u - Undo changes to the configured value(s) for this setting
v - Print configured value(s) for this setting

Press '<cr>' to keep the current value(s) for 'Server of Authority (SoA)' (? for options):

# Configuration – IMPS configurator – class

```
**********Class Configuration**********
Service: Cray Scalable Services
Setting: tier1
Guidance: The tier one servers must have a direct IP connection to the Server of Authority
      (SoA) which is typically the SMW. Any node that is directly connected to the SoA
      can be a tier one server.
    * In a typical Cray system, the boot node can always serve this role.  However,
      often the SDB node can also perform this role assuming it is properly configured
      to reach the SMW.
    * Adding additional tier one servers provides enhanced resiliency.
    * Enter at least one tier one server by its cname.
Current Values: 'tier one servers' is set to:
  [1] - c0-0c0s0n1
Press '<cr>' to keep the current value(s) for 'tier one servers' (? for options):
```

# Configuration – IMPS configurator – class

**********Class Configuration**********
Service: Cray Scalable Services
Setting: tier2
Guidance: The tier two servers provide the primary scaling burden. They must have a direct IP connection to the tier one servers.
* These nodes serve DVS file-systems such as the Cray Programming Environment, network root file-systems (netroot), the config set and other services to all compute nodes.
* Typically the number of servers listed here is proportional to the size of the system. For a small system, a single server may be sufficient, for a very large system, hundreds of servers may be needed. If more than one server is specified resiliency will be enhanced.
* Candidate nodes: Dedicated repurposed compute nodes are the optimal choice for tier two servers. Most service nodes are candidates although servers may belong to only one tier. It is better NOT to utilize login nodes and there is currently an advantage to make LNET routers also tier two servers.
* Enter at least one tier two server by its cname.
Curret Values: 'tier two servers' is set to:
[1] - c0-0c0s1n2
[2] - c0-0c0s0n2
Press '<cr>' to keep the current value(s) for 'tier two servers' (? for options):

# Configuration – global config set services

cray_global_net_config.yaml
cray_image_groups.yaml
cray_logging_config.yaml
cray_network_boot_packages_config.yaml
cray_storage_sets.yaml
cray_time_config.yaml

# Configuration – CLE config set services

cray_alps_config.yaml
cray_auth_config.yaml
cray_dvs_config.yaml
cray_image_layering_config.yaml
cray_lmt_config.yaml
cray_lnet_config.yaml
cray_local_users_config.yaml
cray_logging_config.yaml
cray_login_config.yaml
cray_lustre_client_config.yaml
cray_munge_config.yaml
cray_net_config.yaml
cray_network_boot_packages_config.yaml
cray_node_health_config.yaml
cray_persistent_data_config.yaml
cray_rsip_config.yaml
cray_rur_config.yaml
cray_scalable_services_config.yaml
cray_sdb_config.yaml
cray_service_node_config.yaml
cray_simple_shares_config.yaml
cray_simple_sync_config.yaml
cray_ssh_config.yaml
cray_storage_config.yaml
cray_time_config.yaml
cray_user_settings_config.yaml
cray_wlm_detect_config.yaml
cray_wlm_trans_config.yaml

# Next Generation CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- Configuration
- **Booting**
- Reconfiguration
- Summary
- Questions

# Booting

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

# Booting – What is New?

- **Boot the system**

  crayadm@smw> xtbootsys -a auto.pluto

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

# Booting – Run simple jobs

- **Check status on nodes**

```
crayadm@login> xtprocadmin
crayadm@login> xtnodestat
crayadm@login> apstat –v
```

- **Test basic aprun functionality**

```
crayadm@login> NUMNODES=$(($(apstat -v | grep XT | awk "{print \$3}"))); \
        echo NUMNODES is $NUMNODES
crayadm@login> cd /tmp; aprun -b -n $NUMNODES -N 1 /bin/cat /proc/sys/
kernel/hostname
crayadm@login> aprun -n $NUMNODES -N2 python -c "print 'hello world.'"
```

# Troubleshoot A Boot 1

- **Boot automation file starts nodes in a certain order**
  - boot, sdb, service, compute
  - Additional actions can run commands on certain nodes
- **Logs on the SMW in /var/opt/cray/log**
  - HSS daemons and rsyslogd daemon running on the SMW will log to files in this directory
    - nimsd, pmd, xtremoted, erfsd, nm, bm, sedc_manager, bm, sm,erdh, erd
- **The output from booting CLE will be /var/opt/cray/log/p0-current**
  - Several files with more detailed and specific information
    - bootinfo.* file is the output from running xtbootsys
    - console-YYYYMMDD is the combined console output from every node
- **Commands on SMW logged in /var/opt/cray/log/commands**

# Troubleshoot A Boot 2

- **As each node boots:**
  - Load the kernel and boot image into memory
  - /init will execute first phase of Ansible
    - If failure, then it will appear on the console log
  - Switch to multi-user with systemd
  - Execute second phase of Ansible
    - If failure, login to node
      - If node started sshd, login via ssh
      - If sshd not started, login via xtcon to connect to the console
- **Once on the node, look for the logs in /var/opt/cray/log/ansible**
  - sitelog-init has Ansible plays which were run in the first phase
  - sitelog-booted has Ansible plays from the second phase

# Dumping Cray XC System

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

# Next Generation CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- Configuration
- Booting
- **Reconfiguration**
- Summary
- Questions

# Reconfiguration

- **Configure for external Lustre server**
  - Configure LNet node
  - Configure Lustre client
  - Update network configuration
- **Update /etc/motd**
- **Install file with Simple Sync**
- **Configure for external NFS server**
  - Configure DVS node
  - Configure LDAP
  - Configure automount files with Simple Sync
- **PE (Programming Environment) Install and Update**
- **Extend an Image Recipe**
- **Stage Changes in Snapshot**

# Reconfiguration – external Lustre server 1

- **Clone config set**
  smw# impscli clone config_set p0 to p0beforeLustre
- **Configure LNet node**
  smw# impscli update config_set p0 with state all service cray_lnet level advanced
- **Configure Lustre client**
  smw# impscli update config_set p0 with state all service cray_lustre_client
- **Update network configuration**
  smw# impscli update config_set p0 with state all service cray_net level advanced

# Reconfiguration – external Lustre server 2

- **Validate config set**

  smw# impscli validate config_set p0

- **Shutdown CLE**

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

- **Boot CLE**

  crayadm@smw> xtbootsys -a auto.pluto

- **Test simple job**

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

# Reconfiguration - /etc/motd

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

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

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

  CUG tutorial

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

  boot# cat /etc/motd

  *** Welcome to IMPS service node c0-0c0s0n1 (nid 1) ***
  Running 1.4GB Suse 12 image service_cle_rhine_sles_12_x86-64_ari
  CLE release rhine, build 201503230201
  16 vcores, boot_freemem: 28868mb

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

  boot# cat /etc/motd

  *** Welcome to IMPS service node c0-0c0s0n1 (nid 1) ***
  Running 1.4GB Suse 12 image service_cle_rhine_sles_12_x86-64_ari
  CLE release rhine, build 201503230201
  16 vcores, boot_freemem: 28868mb
  CUG tutorial

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

# Reconfiguration – Simple Sync

- **Files in these config set locations are copied to the target nodes at boot time:**
  - /var/opt/cray/imps/config/sets/p0/config/files/roles/simple_sync/…
    **…/{classes,cnames}/…**
    **…/{common,compute,service}/…**
- **For example, the setup below:**
  - …/simple_sync/classes/common/tmp/i_am_common
  - …/simple_sync/classes/service/tmp/i_am_service
  - …/simple_sync/cnames/c0-0c1s2n0/tmp/i_am_c0-0c1s2n0
- **Would copy the file so it would appear on the node as:**
  - /tmp/i_am_common on all nodes
  - /tmp/i_am_service on all service nodes
  - /tmp/i_am_c0-0c1s2n0 on cname c0-0c1s2n0

# Reconfiguration – Simple Sync

- **Simple Sync service provides a simple and easy to use generic mechanism for administrators to copy files onto their system without resorting to writing an Ansible play**

  smw# cd /var/opt/cray/imps/config/sets/p0/files/roles
- **Make a file for all nodes**

  smw# touch simple_sync/classes/common/pluto.common
- **Make a file for all service nodes**

  smw# touch simple_sync/classes/service/pluto.service
- **Make a file for all compute nodes**

  smw# touch simple_sync/classes/compute/pluto.compute
- **Content will be delivered during a boot, but you can deliver it immediately to the nodes**

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

  boot# pcmd -r -n ALL_NODES_NOT_ME "/etc/init.d/cray-ansible start”
  - ALL_NODES, ALL_COMPUTE, ALL_SERVICE, ALL_SERVICE_NOT_ME

# Reconfiguration – external NFS server

- **Configure for external NFS server**
  - Update network configuration on DVS node for network interface
    smw# impscli update config_set p0 with state all service cray_net level advanced
  - Configure DVS node to external NFS server
    smw# impscli update config_set p0 with state all service cray_dvs level advanced
  - Configure LDAP
    smw# impscli update config_set p0 with state all service cray_auth level advanced
  - Configure automount files on DVS node with Simple Sync
    smw# cd /var/opt/cray/imps/config/sets/p0/files/roles
    smw# mkdir -p simple_sync/cnames/c0-0c0s0n2/etc/auto.master.d
    smw# cd simple_sync/cnames/c0-0c0s0n2/etc
    smw# cp -p /home/crayadm/etc/auto.css .
    smw# cp -p /home/crayadm/etc/auto.master.d/css.autofs auto.master.d

# Reconfiguration – PE Fresh Install

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

# Reconfiguration – PE Fresh Install

- **Clone compute image root to PE image root**
  smw# PECOMPUTE=pe_compute_cle_rhine_sles_12
  smw# impscli clone image $COMPUTE to $PECOMPUTE
- **Mount PE ISO**
  smw# mkdir -p mount_iso logs
  smw# mount -o loop,ro CDT-15.01-51.dev.iso ./mount_iso
- **Install craype-installer rpm**
  smw# rpm -ivh craype-installer-1.10.00-11.x86_64.rpm
- **Configure PE installer YAML**
  smw# cp /opt/cray/craype-installer/1.10.00/conf/install-cdt.yaml .
  smw# vi install-cdt.yaml

```
IMAGE_DIRECTORIES :
  - /var/opt/cray/imps/image_roots/pe_compute_cle_rhine_sles_12
LOGS_DIR                : ./logs
ISO_MOUNT_DIR           : ./mount_iso
```

# Reconfiguration – PE Fresh Install

- **Install PE software**

  smw# module load craype-installer

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

- **Push PE image to Boot Node**

  smw# impscli push image $PECOMPUTE to boot

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

  INFO - Checking remote destination...

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

  Password:

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

  Password:

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

# Reconfiguration – PE Fresh Install

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

  smw# impscli update config_set p0 state all service cray_image_layering

- **Validate config set**

  smw# impscli validate config_set p0

- **Restart Layering Services**

  boot# pcmd -r -n ALL_COMPUTE "ansible-playbook /etc/ansible/cray_image_layering.yaml”

  login# ansible-playbook /etc/ansible/cray_image_layering.yaml

# Reconfiguration – PE Update

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

# Reconfiguration – PE Update

- **Reuse previous PE image root**

  smw# PECOMPUTE=pe_compute_cle_rhine_sles_12
- **Mount PE ISO**

  smw# mkdir -p mount_iso logs

  smw# mount -o loop,ro CDT-15.01-51.dev.iso ./mount_iso
- **Update craype-installer rpm (if needed)**

  smw# rpm -uvh craype-installer-1.10.00-11.x86_64.rpm
- **Reuse PE installer YAML**
- **Install PE software**

  smw# module load craype-installer

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

# Reconfiguration – PE Update

- **Push PE image to Boot Node**

  smw# impscli push image $PECOMPUTE to boot
  INFO - Remotely cloning Image '<name of image>' to 'boot'...
  INFO - Checking remote destination...
  INFO - Passwordless SSH not established; prompting for password for root@boot:
  Password:
  INFO - Transferring Image '<name of image>' to 'root@boot:/var/opt/cray/imps/image_roots/<name of image>'...
  Password:
  INFO - Cloned Image '<name of image>' to remote host 'root@boot:/var/opt/cray/imps/image_roots/<name of image>'.

- **Restart Layering Services**

  boot# pcmd -r -n ALL_COMPUTE "ansible-playbook /etc/ansible/cray_image_layering.yaml”
  login# ansible-playbook /etc/ansible/cray_image_layering.yaml

# Reconfiguration – Extend an Image Recipe

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

# Reconfiguration – Extend an Image Recipe

- **Clone an existing Image Recipe**

```
smw# impscli list image_recipes
INFO - Image_Recipes:
        compute_cle_rhine_sles_12_x86-64_ari
        dal_cle_rhine_centos_6.5_x86-64_ari
        login_cle_rhine_sles_12_x86-64_ari
        service_cle_rhine_sles_12_x86-64_ari

smw# impscli clone image_recipe compute_cle_rhine_sles_12_x86-64_ari to
custom_compute_cle
INFO - Locally cloning Recipe 'compute_cle_rhine_sles_12_x86-64_ari' to
'custom_compute_cle'.
INFO - Successfully created new empty Recipe 'custom_compute_cle'.
INFO - Successfully cloned to Recipe 'custom_compute_cle'.
```

# Reconfiguration – Extend an Image Recipe

- **Create a new image recipe**
  smw# impscli create image_recipe custom_compute_cle
  INFO - Successfully created new empty Recipe 'custom_compute_cle'.
- **Extend image recipe with three rpms**
  smw# impscli extend image_recipe custom_compute_cle with packages package1 package2 package3
  INFO - Successfully extended Recipe 'custom_compute_cle' with 3 items.
- **Extend image recipe with two package collections**
  smw# impscli extend image_recipe custom_compute_cle with package_collections custom_compute_packages_base custom_compute_package_latest
  INFO - Successfully extended Recipe 'custom_compute_cle' with 2 items.
- **Extend image recipe with new repo**
  smw# impscli extend image_recipe custom_compute_cle with repository custom_repo
  INFO - Successfully extended Recipe 'custom_compute_cle' with 1 item.

# Reconfiguration – Extend an Image Recipe

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

# Reconfiguration – Extend an Image Recipe

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

# Reconfiguration – Extend an Image Recipe

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

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

  - The validate command will also validate all repositories and package collections referenced by your image recipe and will ensure that it can access any files in the *postbuild_copy* section
    - If IMPS cannot read the local edits image recipe, it will not proceed to validating the other IMPS objects

# Reconfiguration – Extend an Image Recipe

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

```
smw# impscli build image_recipe custom_compute_cle
INFO - Repository 'custom_repo' validates.
INFO - Recipe 'custom_compute_cle' is valid.
INFO - Calling Package manager to build new image root; this will take a few minutes.
INFO - Rebuilding RPM database for Image 'custom_compute_cle'.
INFO - RPM database does not need to be rebuilt.
INFO - Running post-build scripts for Image 'custom_compute_cle'.
INFO - Copying postbuild files to /tmp/tmpmAyzGl in Image 'custom_compute_cle'
INFO -          * Executing post-build chroot script: 'chroot_command1'
INFO - post-build chroot script output will be located in /tmp/custom_compute_cle-postbuild_out_20140929-11:38:11g4WA6p
INFO - Build of Recipe 'custom_compute_cle' has completed successfully.
```

# Reconfiguration – Extend an Image Recipe

- **See build history of image recipe**

```
smw# impscli show image_recipe custom_compute_cle
INFO - Recipe 'custom_compute_cle':
created: 07.28.14 03:26:00 AM
history:
   ...
- '07.28.14 03:26:00 AM: Successful build of Image custom_compute_cle.'
- '07.28.14 03:26:00 AM: An error occurred while executing a post-build chroot script.
    Output was stored in /tmp/custom_compute_cle-postbuild_out_20140728-03:26:001t3WIx'
   ...
package_collections: ...
packages: ...
path: /etc/opt/cray/imps/image_recipes.d/image_recipes.local.json
postbuild_chroot:
- chroot_command1
postbuild_copy: ...
repositories: ...
```

# Reconfiguration – Extend an Image Recipe

- **Package image root into boot image**

smw# impscli package image custom_compute_cle with destination /var/opt/cray/imps/boot_images/custom_compute_cle.cpio

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

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

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

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

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

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

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

INFO - Creating size-initramfs file

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

INFO - Creating symlinks of Image 'custom_compute_cle' to DEFAULT

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

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

# Reconfiguration – Extend an Image Recipe

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

  smw# nimscli set --image /var/opt/cray/imps/boot_images/custom_compute_cle.cpio for --node c0-0c0s15n3

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

  smw# xtcli shutdown c0-0c0s15n3

  smw# xtbootsys --reboot -r “testing custom compute image” c0-0c0s15n3

# Reconfiguration – Extend an Image Recipe

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

  smw# nimscli set --image /var/opt/cray/imps/boot_images/custom_compute_cle.cpio for --group compute

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

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

  smw# xtcli shutdown $COMPUTENODES

  smw# xtbootsys --reboot -r “Booting custom compute image on all compute nodes in c0-0” $COMPUTENODES

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

  smw# xtbootsys -a auto.pluto

# Reconfiguration – Stage Changes in Snapshot

- **Change to snapshot for reconfiguration**

```
smw# snaputil chroot mysnapshot
Changing root to: /media/root-sv/snapshots/mysnapshot.
btrfs_object_id: 275
kernel: vmlinuz-3.12.28-4-default
smwha_version: None
name: mysnapshot
updated: 2015-02-24 08:17:30.206783
initrd: initrd-3.12.28-4-default
cle_version: 201502240201
path: /media/root-sv/snapshots/mysnapshot
smw_version: 7.3.0-1.0000.36035.467
storage_set: smwdefault
(mysnapshot) -> root@smw:/ #
```

# Reconfiguration – Stage Changes in Snapshot

- **Load modules**
  (mysnapshot) -> root@smw:/ # module load imps
  (mysnapshot) -> root@smw:/ # module load install-support
- **Build new images from new repo content**
  (mysnapshot) -> root@smw:/ # imgbuilder -map
- **Update config sets**
  (mysnapshot) -> root@smw:/ # impscli update config_set p0
  (mysnapshot) -> root@smw:/ # impscli update config_set global
- **Leave chroot**
  (mysnapshot) -> root@smw:/ # exit
  smw#

# Reconfiguration – Stage Changes in Snapshot

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

  smw# snaputil default mysnapshot

  smw# xtbootsys -s last -a auto.xtshutdown

  smw# reboot

- **Boot CLE**

  smw# xtbootsys -a auto.pluto

# Security Updates

- Process for security updates under development
- Process for security updates under development
- Process for security updates under development
- Process for security updates under development

- One method
  - Same installer engine used for security updates as for other installations on SMW
    - SLEupdate

# Security Updates – SLEupdate

SLEupdate [--target=NAME] [options]

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

# Security Updates – Run SLEupdate

- **Mount SLE security media**

  smw# mkdir -p /media/SLE

  smw# mount -o loop,ro sleupdate-image-rhine.2015-02-25.iso /media/SLE

- **Install Security Updates**

  smw# /media/SLE/SLEupdate –media=/media/SLE --target=${SNAPSHOT}

  - If no target on command line, a snapshot will be created
  - Changes made in snapshot

- **Logs created in /var/adm/cray/logs/install*.log**
  - Very verbose log file with all zypper/rpm messages

# Security Updates – reboot with new security

- **If –live-update was not used**
  - Shutdown CLE before rebooting the SMW
    smw# xtbootsys –s last –a auto.xtshutdown
  - Set SMW to boot from new snapshot
    smw# snaputil default $SNAPSHOT
  - Reboot SMW
    smw# reboot
- **Rebuild images**
  smw# imgbuilder –map
- **Reboot CLE system**
  - If --live-update was used, CLE wasn't shutdown yet and SMW was not rebooted
    smw# xtbootsys -s last -a auto.xtshutdown
    smw# reboot
    smw# xtbootsys --a auto.pluto
  - If --live-update not used, then SMW was rebooted
    smw# xtbootsys -a auto.pluto

# Next Generation CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- Configuration
- Booting
- Reconfiguration
- **Summary**
- Questions

# Summary

- **Separation of Software and Configuration**
  - Increasingly common and standard model in the Cloud, OpenStack, and Enterprise
  - Allows natural use of common configuration management tools like Ansible
  - Can update software images and configuration separately and independently without affecting the other

# Summary

- **IMPS Prescriptive Image Creation**
  - Use zypper/yum to satisfy dependencies instead of manually creating rpm lists
    - Far more reliable and consistent
  - Use simple image recipes and package collections to build up image definitions
    - Provides more clear and flexible image definition
  - Ability to change and enhance existing image recipes
  - Ability to create new image recipes for different node types
  - Easy to build new images with bug fixes or security updates
  - Operations can be staged to avoid impacting the running system

# Summary

- **IMPS Centralized Configuration**
  - Configured centralized in one location
    - Not spread out across large shared root
  - Most configuration in YAML
    - Easily parsed by tools, viewable and editable by humans
  - Can clone configuration to try new options
  - Easy-to-use places to plug in site customizations
  - Easy to back up
  - Easy to move from system to system
  - Easy to upgrade
  - Configurator tool to guide system administrators through configuring the system

# Summary

- **IMPS Node Deployment**
  - Can specify what image to boot on a node by node basis or on groups of nodes
  - Can provide alternate configuration for specific nodes or groups of nodes
  - Can try out new images or new configuration on select nodes prior to full deployment
  - Enablement for live updates (running zypper/yum live on a node)
  - Simple tmpfs root filesystems for small image
    - More compatible, standard and common, faster
  - Overlayfs root filesystem (nfs/dvs read-only plus small tmpfs read-write) for large images
    - More image flexibility and smaller memory footprint
  - PE separate from base images
    - Providing more clear, defined encapsulation and management
    - More general and forward looking model

# Summary

- **New Installer**
  - Common installer for SMW, CLE, and future products
  - Completely redesigned, configuration driven, and written in Python
    - More maintainable, extensible, and reliable
  - Implements staged upgrades
    - More reliable, significantly less downtime, easy fallbacks
  - Sharing distribution repositories between SMW and CLE, reducing admin overhead and reducing disk space requirements

# Summary

- **New SMW and boot RAID structure**
  - Everything on LVM and using xfs/btrfs
    - More flexible and maintainable
  - Mirroring SMW root filesystem disk
    - More reliable with less admin overhead
  - Consolidating SMW HA and non-HA disk space requirements to reduce redundancy and complexity

# Next Generation CMS Agenda

- Introduction
- Overview of new concepts
- Software installation
- Configuration
- Booting
- Reconfiguration
- Summary
- **Questions**

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