How to Automate and Not Manage under Rhine/Redwood

Paul Peltz Jr., Adam DeConinck, Daryl Grunau
High Performance Computing Division
Los Alamos National Laboratory
Los Alamos, NM, USA
Email: {peltz,ajdecon,dwg} at lanl.gov

***Abstract*—Los Alamos National Laboratory and Sandia National Laboratory under the Alliance for Computing at Extreme Scale (ACES) have partnered with Cray to deliver Trinity, the Department of Energy's next supercomputer on the path to exascale. Trinity, which is an XC40, is an ambitious system for a number of reasons, one of which is the deployment of Cray's new Rhine/Redwood (CLE 6.0/SMW 8.0) system management stack. With this release came a much-needed update to the system management stack to provide scalability and a new philosophy on system management. However, this update required LANL to update its own system management philosophy, and presented a number of challenges in integrating the system into the larger computing infrastructure at Los Alamos. This paper will discuss the work the LANL team is doing to integrate Trinity, automate system management with the new Rhine/Redwood stack, and combine LANL's and Cray's new system management philosophy.**

***Keywords*-system management; automation; Ansible; Cray; XC40; Rhine/Redwood; CLE 6.0/SMW 8.0**

I. What is Rhine/Redwood?

Rhine/Redwood (R/R) is the code name for Cray's new systems software stack, CLE 6.0/SMW 8.0. This release provides a much-needed update to the system management stack, and makes extensive changes to the tooling and overall system management philosophy on Cray systems. The inclusion of a new, more reproducible approach to image creation and deployment, and the tight integration of Ansible as a configuration management system, conforms more closely to industry standards for system management than the previous approach. However, while these changes eliminate many of the issues with the old stack, the complexity of this new system introduces a new set of challenges for the system administrator.

This paper is published under LA-UR-16-22395.

II. ACES/Cray Collaboration

ACES and Cray have been collaborating on the CLE 6.0/SMW 8.0 software release while it was still in active development and through several beta releases until the release of UP00 in December 2015. This collaboration is unprecedented for Cray and it has proven to be very successful. As new releases of CLE 6.0/SMW 8.0 became available, system administrators at LANL would install, test, and give feedback directly to the developers. Through this tightly coupled collaboration, Cray was able to deliver a more refined and administrator friendly version of both the CLE and the SMW management tools than what would have otherwise been released. The rapid customer feedback cycle enabled by the ACES/Cray collaboration uncovered a variety of bugs and design issues, as well as producing several new ideas which improved the eventual deployment on Trinity. This collaboration also provided invaluable experience to the system administrators at LANL, who would eventually support Trinity in production. The extensive changes from previous releases represent a daunting learning curve, and the early exposure to new system management concepts and tools allowed the administrators to learn the new system and begin devising new ways to automate and manage the system.

III. CLE 6.0/SMW 8.0 Management Philosophy

There are two important considerations administrators should think about for a given system. Is the system reproducible and is the management of the system automatable? Much of the evaluation and feedback to Cray of the new CLE 6.0/SMW 8.0 system directly addressed these issues. If there was a direct conflict with what is considered a best practice in system management, the LANL administrators would file a request for enhancement (RFE) to address the issue. However, not all of these processes can be automated, which is by design to allow certain aspects of the system to function in case of a failure or if there is planned downtime. Much of the initial work done by the LANL administrators was learning Cray's new system management philosophy and integrating that into LANL's system management philosophy so that they could co-exist and leverage each other's abilities. Sometimes these philosophies come into direct conflict and workarounds are in place to deal with these until

the larger issue can be resolved. This section details both system management philosophies and what LANL has done to implement them to create a reproducible and automatable system.

A. Cray Philosophy

Under CLE 6.0 there is no longer a shared root that nodes reference and therefore also no specializations for node classes or cnames. CLE 6.0 has a new system for managing images now which Cray calls the Image Management and Provisioning System (IMPS). IMPS now uses generic image types for classes of nodes (login, service, compute, and eLogin) and all of these images are built from a set of common SLES and Cray RPM repositories. This helps to provide a more homogenous environment between the external and internal system. In order to specialize these generic node images, Cray uses Ansible to customize the image post boot for its intended purpose. For example, if a DVS node boots the generic service image, Ansible runs on that node and configures it so that it will function as a DVS node. Another service node could also boot that same service image and be configured to be an LNET router, depending upon its Ansible configuration. The way in which Ansible determines what class a node belongs to in order to specialize it is through what Cray calls the config set. This is the source of truth for the system and defines everything from IP addresses to node classes, e.g. c2-3c0s0n2 is an LNET router, for the entire system. The Cray Ansible plays are specific for each node class and reference the config set information to generate configuration files, start services, and specialize the end nodes. In order to do this Cray has an Ansible play area that is run at boot, hereby referred to as Cray Ansible (CA). There are multiple Ansible areas that Cray and the administrators utilize, and differentiating them will be important later. The CA playbook runs twice during a nodes boot up. Once at the pre-init and then the second run of Ansible is after the completion of systemd's init (multi-user). See Figure 1.

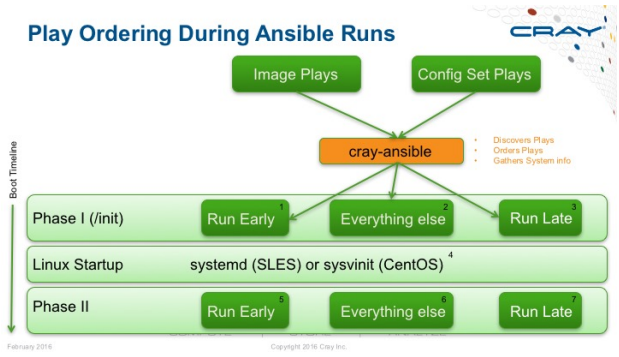

Figure 1. Ansible Play Ordering

Within each run of CA there is also the ability to order the plays so that they run in a specific order to fulfill any dependencies. Ordering is important and there are specific variables that can be set to define that order. While it is possible to modify the plays provided by Cray, this is not recommended. These plays are provided by RPMs and will be overwritten whenever new images are created. In order to address this issue, Cray has provided a hook into the CA area. This is what will be referred to as the Site Ansible (SA) area. This area allows a site to insert their own plays into Cray's playbook which runs at boot. This way it is possible to modify files, services, and the state of the machine at boot to ensure it boots into a known good configuration. The SA plays are coalesced with the CA plays to form one large playbook. Cray's `/etc/init.d/cray-ansible` python script, among other things, identifies all of the plays defined in `/etc/ansible` and `/etc/opt/cray/config/current/ansible` and builds this large playbook which is placed into `/etc/ansible/site.yaml`. The play ordering is explicitly defined in this file and Ansible executes each of these plays in order when executed against this playbook. This is an important point to remember, because if the ordering is incorrect a configuration file could be modified after a service is already started by another play earlier in the playbook. While it is possible to restart the service later, this is wasteful in terms of wasted time during boot and also there could be negative consequences to restarting the service later. One such consequence the administrators have seen is that restarting the `rsyslogd` service caused `apsys` to be stopped as well and not restarted. The SA area can contain any number of additional Ansible plays to further customize nodes which allows the administrator to specialize a class, group, or specific node to the site's specifications.

The config set and SA areas reside on the SMW in `/var/opt/cray/imps/config/sets/p0`. The config set data resides in `p0/config` and the SA resides in `p0/ansible`. Cray then exports the p0 config set directory read-only via NFS to both of the Tier 1 servers, boot and sdb. The Tier 1 servers utilizes the 9p network file system and `automount` to distribute the config set read-only to the Tier 2 servers. This system is known by the IMPS Distribution Service (IDS). IMPS is the implementation of the new system management philosophy by Cray and all of the tools associated with it. The mount location on the end node differs from the SMW, therefore on an end node the config set data will reside in `/etc/opt/cray/config/current`. The number of Tier 2 servers required is dependent on the scale of the system. Having too few Tier 2 nodes can lead to config set redistribution issues which will become apparent during system boots when the end nodes fail to mount the 9p file system and the CA run fails. Cray has guidelines on the required number of Tier 2 nodes and is actively working on minimizing the number of those nodes that are required.

As can be seen in Figure 2, Cray utilizes a multi tiered

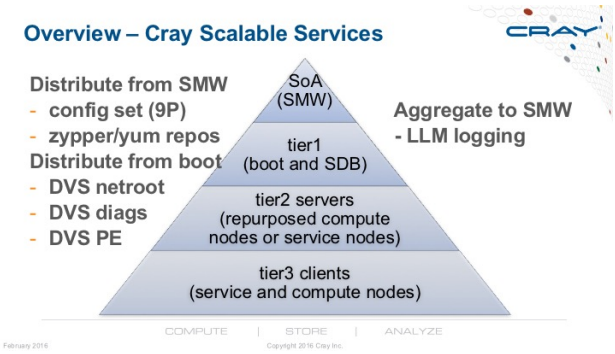

Figure 2. Scalable Services Diagram

approach to provide scalability of the configuration data. The use of the 9p file system allows for better caching and `autofs` provides failover in case one of the tiered nodes fail.

Image creation and management is another new feature of CLE 6.0. Images are built from recipes, recipes are built from packages and package collections, and the packages and package collections are provided by RPM repositories. This process builds generic images for the system to boot as mentioned previously. Images are then mapped to individual nodes by the Node Image Mapping Service (NIMS). This is diagrammed below in Figure 3.

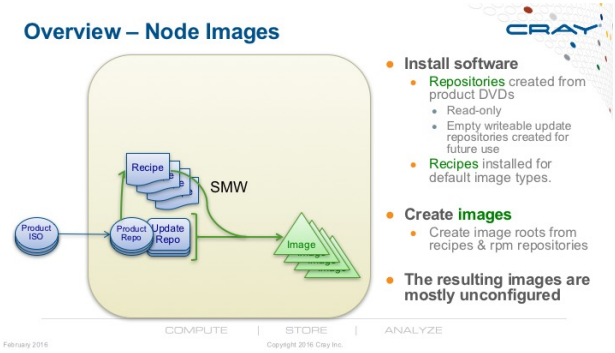

Figure 3. Node Images and Recipes

For most sites, external login nodes have been a source of frustration among admins and users when administering the Cray system. Choosing whether or not to use Bright or install from a SLES DVD and then manage it through a configuration management system was a decision that every site had to make. One of the problems with this system was that managing the Programming Environment (PE) and the external login node's software and keeping them identical between the internal system and external was a continuous challenge. Now under CLE 6.0, Cray is instead using OpenStack to manage the newly named eLogin nodes. Images are built from recipes on the SMW and reference the same repos that the other images reference. See Figure 4. This change in the provisioning process of the eLogin nodes goes a long way to solve the problem of package and package version skew between the internal and external systems. Once the eLogin image is built on the SMW, it is then exported to Glance, which is an OpenStack component. Glance runs on the Cray System Management Software (CSMS) node and OpenStack then distributes that image to be written to the eLogin node's disk and once the eLogin node is booted, use Ansible to customize and configure the node. Ansible references the same config set that the internal system does. The PE is pushed from the SMW to the CSMS node which guarantees it to be the same on the internal Cray system. The PE is then copied over to the eLogin node from the CSMS box while booting the eLogin node. Utilizing the same config set and PE provides consistency across the entire platform.

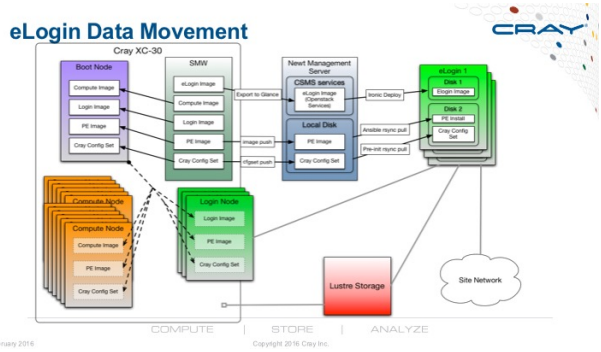

Figure 4. eLogin Data Movement

Cray's approach at deploying this new system management software philosophy was originally from the desire to no longer use Cray's unique shared root solution and move to leveraging Linux and more open source tools. In many ways they have done this, but most of these features are obfuscated behind Cray's helper commands. This isn't necessarily a bad thing, but it does add some confusion at times understanding what the Cray commands are doing. Understanding comes with time and continuous use of the new system.

B. LANL Philosophy

Over the past year the LANL administrators have spent a lot of time working with the new CLE 6.0/SMW 8.0 software. Much of that time was spent fixing the system once it was broken after attempting to influence the behavior of it somehow. Over time there have been a number of lessons learned, and what the LANL administrators are considering best practices which are shared here.

1) Use of the Cray Ansible Area: By default the `cray-ansible` script is only run at boot time to do an initial configuration and start all of the services that the node requires to provide its intended functionality. In order to maintain consistency across the system it is necessary to periodically run at least a subset of the Cray and Site

plays to avoid configuration skew and ensure services are still running. Presently there is no way to address this from a Cray perspective other than setting a cron job to run the cray-ansible script at a specified interval. This issue could be addressed by selecting a specific set of plays to run on a regular basis, especially the following: firewall, work load manager configs, syslog, ssh, and simple_sync plays.

2) When to use the Site Ansible Area: Initially the administrators pursued placing all of the site created plays into the SA area to leverage Cray's Ansible run at startup to configure the system, but there are a few considerations to make before deciding whether or not a task should be implemented within the SA area due to issues that were discovered by implementing this idea. Over time it was realized that over using the SA area was not desirable. One issue that was immediately noticed is that a failure in the Ansible playbook within the SA area could stop the boot of the machine. An end node would not return that it succeeded and the `xtbootsys` process would sit in wait until the timeout value was reached. The reason this is such a problem, especially if Ansible fails in the first Ansible phase, is because those logs are not written back to the SMW. The phase 1 Ansible play logs are only available on the end node. It is possible if the end node is far enough into the Ansible playbook to be configured for remote login to the node, then the Ansible playbook could be replayed to fix the failure, but this is not always possible. `xtcon` could also be used to log in to the node over the console, but this isn't always possible either because there could be a security policy that no password logins are allowed on an end node, or for a variety of other reasons. If this is the case where there is no mechanism for remote access, it would be required to do a debug boot session to monitor each stage of the boot process. One other issue is that every play is processed by Ansible during both the pre-init and multi-user Ansible runs. A large number of plays extends the boot time for a given system. To address these issues, the administrators have implemented their own local CM (LCM) area on a subset of nodes that require it. The LANL administrators are using Ansible as well to handle this, but isn't strictly necessary.

The LCM area is only deployed on a small set of nodes currently to address issues with the need for management of files on only one node. The SMW is a good example of this because even though the CA plays can be run on this host, there is no need to use the SA area for this. As mentioned previously these plays will be evaluated by Ansible on every node in the system during boot which lengthens the boot time. Therefore, all of the files that needed to be under configuration management were done so through the LCM area. This gives the administrator better fine grained control of what files to manage and the frequency in which they will need to be verified or changes pushed to the system. The state of the SMW and the configuration files on it need to be in a reproducible state in case of a system failure or user error. The LCM provides this ability without being a detriment to the boot time due to placement within the SA area.

With the previous generation of system management software 5.x (Pearl), an administrator could place a file in the shared root directory and immediately affect the entire system. There are pros and cons to this approach, but it certainly made it easy to affect a non-invasive change across the entire system. This was certainly beneficial when doing lightweight but important tasks such as account management. Previously this could be implemented by running a script on the boot node that would drop the appropriate files into the shared root space. This is no longer the case under CLE 6.0. In order to affect a change on the end nodes it is necessary to run a script or something equivalent to an Ansible play on each end node to accomplish this task. For instance, if the password/group/shadow files need to be updated on and end node there will have to be some mechanism to accomplish this. The LANL administrators implemented an Ansible play that runs periodically on the internal login nodes to keep the account files in sync. For the compute nodes however this is more of a challenge. An hourly play or script could be run to accomplish this as well, but this causes extra jitter on the compute node which could affect job performance. To avoid this issue we have implemented a resource manager (RM) prologue task that calls a local Ansible play to sync these files. This causes a slight delay (~3s) in job launches but is more desirable than job performance degradation. Therefore, considerations need to be made on when a file needs to be distributed to the system frequently. If it is a config file that needs to be distributed widely to the end nodes, consider leveraging the 9p file system. The `/var/opt/cray/imps` directory is redistributed via IDS and mounted at `/var/opt/cray/imps-distribution` on the end nodes. For instance, this is where LANL administrators distribute accounting files and other files that change frequently so that the LCM system can reference those to put them in place. The way this is accomplished is the SMW has an LCM area that generates the files and then places them into the directory structure that IDS utilizes.

There are some considerations to make before deciding how to distribute files throughout the system, and this is mostly applicable to compute nodes, because an hourly play on service nodes is not as disruptive to users. If it is a file that is updated regularly (more often than a typical DST/PM cycle) then it should be distributed through the 9p file system. However if it cannot be referenced/included by another file and has to overwrite a file in the root file system then a play may need to be written to accommodate this task which would also have to be included in the RM prologue. Also consider a soft link to the file in 9p space if the application supports this as this would remove the necessity of the play. This is more difficult if there is a service that

needs to be restarted/reloaded based on a change to the file. In this case it will likely require an action in the RM prologue to accomplish this task. Not everything will follow these guidelines however and careful consideration should be made when deciding on an implementation strategy.

3) Using Images and Recipes Effectively: An administrator can clone and modify site specific recipes based off of Cray provided default recipes, and then add new package collections, and repos. This gives the site the ability to take a preset Cray recipe and extend it to fulfill the needs of the site. The LANL administrators are using package collections that match the recipe/image names to make things more transparent. See Figure 5.

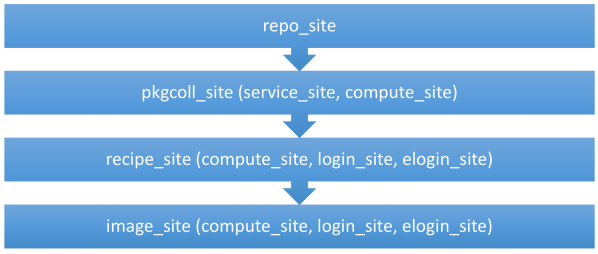

Figure 5. Naming Schemes

This makes it more obvious when inspecting a recipe to see what has been extended to differentiate between a site modification and what Cray includes in the recipe. Use of the package collection is advised rather than adding packages directly to recipes. It is easy for a single package to get lost in the long listing of packages that some recipes have. There are no restrictions on the number of site generated package collections and it makes sense to create another package collection if there are multiple recipes sharing a specific set of packages. For instance, the work load manager packages. If they do not serve a common purpose it is recommended to place them in the recipe's corresponding package collection. For instance, add the package X to the service_site package collection for use in the service_site recipe. It is also possible to include a package collection within a package collection, but this isn't obvious by inspecting the recipe. You would also have to look through the package collections to find where it was included and therefore not recommended.

Another nice feature of recipes and package collections is that it provides a mechanism for change tracking. There are fields for "rationale" associated with each collection and package. This will allow an administrator to associate a change with a ticket number to more easily trace back to why a particular package was added.

```
"compute_lanl": {
    "description": "Compute packages",
    "package_collections": {},
    "packages": {
        "bash-completion": {
            "rationale": "RT#104314"
        },
        "vtune": {
            "rationale": "RT#102090"
        }}
```

In the future it may be desirable to clone and customize more images for various node types. There may be the case where a node needs to have more memory available to it and it would be necessary to remove the excess packages from the recipe to reduce its memory footprint. Currently, each service node runs an approximately 1.5G image that lives in memory and it may be possible to reclaim some of that memory.

IV. Configuration Management Challenges

A. Image Management Challenges

The most obvious change in CLE 6.0/SMW 8.0 is the introduction of Cray's new Image Management and Provisioning System (IMPS), which eliminates the shared root environment that was present in previous versions. While the shared root provided a convenient mechanism for making changes across all of the service nodes on a large, running system, configuration management was difficult and it was nearly impossible to test new software or perform staged upgrades on a portion of the system. All service nodes ran the same essential Linux image, with limited differentiation possible only in `/etc` (where a complex system of symbolic links provided a mechanism to specialize different nodes), and in `/var` (where each service node had its own dedicated storage). Under CLE 6.0, each node boots into a RAM-based root filesystem which is not shared with any other node. Changes can be made live, allowing testing of alternative configurations and even package installs using `zypper`; but these changes must be reproduced in the original image to be persistent across reboots. While this is convenient, it does lend to the possibility of configuration skew across what would normally be a homogenous system. Live changes to images also risks the same kind of non-reproducible system administration which was risky under the shared root environment.

One of the challenges the administrators encountered early on was how to manage the contents of an image that is created by Cray's image creation tools. The easy process for this is to create an image and then modify the files in the newly created image root before packaging the image up for use by the boot process. There are commands to give you the controls to do this, but this can become burdensome every time an image is created and requires manual intervention in the image creation process. However, each subsequent image build creates a new image root and all of the changes to the previous image root are not transferred to the new image root location. There is also a tool that Cray uses, named `imgbuilder`, which will do the complete image

build, packaging, and NIMS mapping with one command and this is the tool that Cray uses in its patch sets when an image build is required after an update. Fortunately Cray has provided hooks within the recipe that the image creation commands execute to allow the administrator to copy in files and execute commands after the image is created. For example:

```
"example_image_recipe": {
  "package_collections": {},
  "packages": {},
  "postbuild_copy": [
    "/home/crayadm/post_conf.sh",
    "/home/crayadm/post_conf_files/" ],
  "postbuild_chroot": [
    "${IMPS_POSTBUILD_FILES}/post_conf.sh],
  "repositories": {}
},
```

This will copy the files in from the location specified and execute a script to do some actions to customize the image before it is packaged. One thing to note about this is that it is executing the script in a chroot environment, so if access to `/dev` or `/proc` would not be possible. To address this the administrators have copied the script in the image and then the script is set to run at reboot to configure the system. This process is all possible through the postbuild actions. This system will allow the site to fully customize the image, but only after the base image has been created. The LANL administrators also have a need to be able to pre-seed an image with specific files. This feature is under development and once it is implemented, it will allow the administrators to achieve full prescription of an image.

B. Fine Grained Control of the Ansible Playbook

One other challenge the administrators have dealt with is how to play nicely with Cray's Ansible plays. This was briefly mentioned in the Philosophy section, but it deserves more detail here. Cray has provided the ability to have mechanisms to control the CA playbook. As shown in Figure 1, there is an ordering to the Ansible playbook and the site is able to use the same variables to control the order that Cray uses. There are three run stages within each phase of the two Cray Ansible phases (pre-init and post-init). When the `cray-ansible` script evaluates these plays and orders them it will look at the `vars` section and see if there are any directives there for ordering. Below are the vars that are available to use to influence the playbook ordering.

- run_early – a set of plays, first run stage (Boolean)
- run_late – a set of plays, last run stage (Boolean)
- run_after – run after specific play (Multival)
- run_before – run before specific play (Multival)

If no `run_` variable is specified then the play is run between `early` and `late` in the middle stage. One other important note is that dependencies take precedence over `run_early` and `run_late` variables. The early and late variables are boolean so they only take true or false. The after and before variables accept a multival list of plays so there can be multiple dependencies. For example, if an administrator may want to put in additional `rsyslog` rules to forward to a remote server, but Cray also has in influence on the `rsyslog` configuration files as well and manages starting the service in the CA play. To leverage this and avoid another `rsyslog` restart after their play an administrator could do the following.

```
---

- hosts: localhost
  vars:
    run_early: true
    run_after:
    - persistent_data
    run_before:
    - llm

  roles:
    - syslog
```

This would ensure the play is run in the first stage, but after `persistent_data` and before the `llm` play. The `persistent_data` play is responsible for mounting persistent storage space from the boot RAID and the `llm` play is Cray's Lightweight Logging Manager which influences and controls the `rsyslog` service.

In order to control which of the two phases, pre- or post-init, the Ansible play can have a conditional to evaluate whether or not the system is in the first phase or not and execute an action. See the example below.

```
---

# Run only when not in init
- include: example.yaml
  when: ansible_local.cray_system is
        defined and not
        ansible_local.cray_system.in_init
```

This will run the example play if the system is not in init (second phase) and if "not" is left out it will only run at pre-init (first phase). If the `ansible_local.cray_system.in_init` variable is not included to evaluate, the play will be run in both phases. This `in_init` variable is one of the most important to include in all of the SA plays that are developed by the administrators. Here are some other handy variables to use for conditionals.

- All compute nodes
  - ansible_local.cray_system.platform == "compute"
- All service nodes

  - ansible_local.cray_system.platform == "service"
- All DataWarp nodes
  - 'nvme0n1' in ansible_devices
- All internal login nodes
  - ansible_local.cray_system.hostid in cray_login.settings.login_nodes.data.members
- All eLogin nodes
  - ansible_local.cray_system_elogin is defined
- The SDB node(s)
  - 'sdb' in ansible_local.cray_system.roles
- The SMW node(s)
  - 'smw' in ansible_local.cray_system.roles
- The boot node(s)
  - 'boot' in ansible_local.cray_system.roles

With these conditionals in place, a play can be isolated to a single node, a group of nodes, or any combination thereof.

C. Simple Sync vs. Configuration Management

Simple Sync is a play that Cray provides to manage files in a similar process in which they were managed under the previous shared root system. Within the config set area there exists the `p0/files/roles/simple_sync/` directory. Underneath that is the familiar subdirectories of classes, cnames, and hostnames. Classes can contain the following: common, compute, sdb, and service. Common will place the files on every node, and the others will place the files for those particular groups of nodes. The `cnames` area contains the cname of a specific node only, e.g. `c0-0c0s10n0`, but cname supersets do not work, e.g. `c0-0c0`. One thing to note about Simple Sync is that Cray also places files within this area as well to be distributed onto the system.

One issue with Simple Sync is that it starts in the second stage of the Ansible run. Therefore it is not always the best solution for distributing files if there are services that depend upon them. With the release of CLE 6.0 UP01, the Simple Sync play is supposed to be expanded and offer more fine grain control of the timing of the placement of files which should address some of the concerns the administrators have with this play as it is currently deployed.

D. eLogin Challenges

One of the issues with eLogin currently is the process to update the configurations on those systems. The issue is that it is not automatable because it involves running commands to push and pull data on different hosts to distribute the data around the system. The current workflow to push a change made on the SMW to the config set is as follows.

```
smw:# cfgset push -d csms p0
smw:# ssh csms
csms:# ssh cdl
cdl:# /etc/opt/cray/pre-pivot.d/20ConfigSync.sh
cdl:# ansible-playbook /etc/ansible/site.yaml
```

This can be addressed slightly by scripting part of this process on the CSMS box.

```
#!/bin/bash

ssh -t root@$1 "/etc/opt/cray/pre-pivot.d/20
    ConfigSync.sh && cd /etc/ansible &&
    ansible-playbook site.yaml"
```

Cray implemented this workflow by design in order to prevent any dependencies on system availability of any of the components. This allows either the CSMS or SMW to be down due to a system failure or due to a software update. While this is convenient the implementation of this leaves much to be desired. The config set, programming environment, and images must all be pushed via a Cray provided command to the CSMS. This entire process is automated within the internal Cray system, but the eLogins are not handled in this way. This non-automatable implementation makes it more difficult to manage because the administrator now has to remember extra steps in order to affect changes onto the eLogin nodes.

V. Revision Control of System Configuration

Large HPC systems usually remain in operation over a period of several years, during which the system's software stack, configuration, and environment can undergo extensive changes. For example, the ACES *Cielo* system at LANL has experienced such changes as OS upgrades; changing the parallel filesystem from Panasas to Lustre; renumbering of network interfaces due to changes in the site networking infrastructure; and hundreds of smaller configuration changes in response to day-to-day operations. Each of these changes must be tracked in such a fashion that they can be rolled back in response to unexpected problems, as well as maintaining a coherent history of the system that can be used in such situations as interpreting historical log files or tracking security-significant changes. A good mechanism for revision control of configuration files therefore becomes essential to long-term management of the HPC system.

The LANL administrator's objectives for a revision control mechanism include the following capabilities:

1) Keep a record of all changes to configuration files and management scripts, from initial installation to decommissioning
2) Attach commentary to each set of changes (e.g., "reconfigured network settings according to ticket 912")
3) Associate each change with the particular person
4) View the differences between two revisions of the system configuration
5) Roll the configuration back to any previous revision
6) Share common files and scripts between multiple HPC systems

This section will describe the built-in mechanisms provided by CLE 6.0 for revision control of configuration files, the changes and additions that have been made to this system

in order to satisfy LANL's objectives, and areas in which the implemented solutions still needs improvement.

A. Insufficiency of Built-In Tools for Change Management

CLE 6.0 provides some built-in mechanisms for change management, but these mechanisms fall short of the administrator's objectives for a revision control system. In most instances, the built-in change management tools take one of two forms: an automatically-generated changelog, and creation of back-up files.

In CLE 6.0, most changes to configuration files are made using tools such as `cfgset`, `recipe`, and `pkgcoll`. Some of these tools, such as `cfgset` and `recipe`, maintain some kind of change history within the data they manage; others, such as `pkgcoll`, make changes without recording any kind of log.

For example, each config set managed by `cfgset` includes a `changelog` directory. The files in this directory are named according to the date and time that `cfgset` was run, and contain a record of each setting which was changed. Changing the MTU on a network interface might generate lines in a changelog file which resemble:

```
cray_net.settings.hosts.data.<hostname>
.interfaces.eth0.mtu:
    previous: '1500'
    current: '9000'
```

`cfgset` also creates auto-save copies of the config sets it manipulates, so that the config sets directory will typically include several subdirectories with names resembling `p0-autosave-2016-01-01T12:00`. It also allows the administrator to clone an existing config set, in order to create a backup of the live config set before a disruptive operation such as an OS upgrade.

While automated changelogs and backups are potentially useful, these mechanisms are not truly a revision control solution. Where complete, changelogs do provide a history of changes to the system, but they do not provide a mechanism for attaching comments or tracking who made the change. In addition, these change logs cannot be used to back out undesirable changes, except by reading the value of the previous settings and reverting the changes manually. The backup files do provide a rollback mechanism, but they do not provide a straightforward way to "diff" these changes, and each subsequent copy occupies as much disk space as the original, limiting how much of the history can be made available at any given time. Perusal through each backed up config set directory is far from ideal and tracking which changes are made in each particular backup is overly cumbersome. There is also no mechanism to share configuration files and scripts between systems, except for manually copying the files from one system to another and editing them in place.

B. Revision control implementation

As the built-in tools were insufficient for the LANL administrator's needs, a new design was embarked upon to provide a revision control mechanism for the CLE 6.0 systems, based on similar requirements for other HPC systems managed at LANL. Currently, the administrator's standard systems management scheme makes use of a dedicated configuration management server ("CM server") which sits outside the HPC system and acts as the single "source of truth" for one or more systems. The CM server stores a collection of scripts and configuration files referred to as the "product area" for each HPC system, which fully describe the system's configuration. The product area is under revision control using an industry-standard source code management system such as RCS or Subversion, and provided to the HPC system read-only over the network, typically using rsync as a transport.

CLE 6.0 required an adjustment to the current revision control scheme in two important respects. First, because certain tools such as `cfgset` and `recipe` must be run on the SMW to generate configuration files, the SMW must have read-write access to the source repository. Second, because some configuration files such as image recipes and package collections are stored separately from the config set, it is necessary to use multiple check-outs of the repository from the CM server.

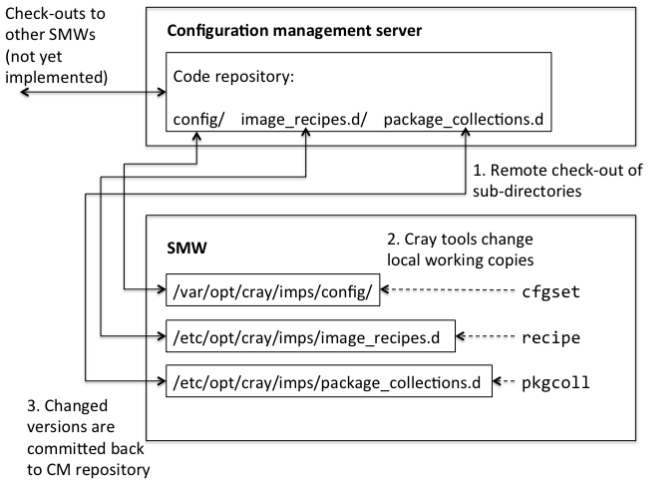

Figure 6. Configuration files and scripts are stored in a remote code repository on the configuration management server. Subdirectories of this repository are checked out on the SMW, replacing the original configuration directories.

A diagram of the revision control scheme on Trinity is shown in Figure 6. After the initial install of a new CLE 6.0 system, the administrators imported the config set, image recipe, and package collection directories into a source code repository on the CM server. Then, on the SMW,

each of the configuration directories were replaced with a remote check-out of the corresponding directory in the CM server's repository. An hourly cron job on the SMW was set up to periodically update the local copies of configuration directories from the CM server, and changes made on the SMW can be committed back to the CM server repository to update its state. For Trinity, Subversion is what was chosen as the revision control system, with remote checkouts being performed via HTTPS. However, this general scheme could easily be adapted to other revision control systems which support remote check-outs, such as Git and Mercurial.

At this point, the built-in management tools `cfgset`, `pkgcoll`, and `recipe` behave normally. They can make changes to the files in their area, and create their own changelogs and backups, without any awareness of or interference from the revision control software. The system administrator who makes the change must simply remember to commit their changes back to the repository to ensure it is tracked.

A major advantage to this scheme is that the system's configuration can be kept in sync with multiple copies off of the SMW, whether on the CM server or on other remote checkouts. This allows the administrator to make some types of changes in a context where they are not immediately presented to the rest of the system, avoiding the situation where a change is made "live" by accident. This allows restriction of some types of change to groups of administrators with different allowed levels of access. For example, LANL has a dedicated team for managing workload management and scheduling configuration; this arrangement allows permissions to be given to them to manipulate the workload manager's configuration files, without necessarily giving them administrative access on the HPC system itself.

A further advantage of this scheme is that the same source repository can be used by multiple Cray systems to share common configuration. While each system's "product area" must necessarily be customized to that system, site-local customizations such as Ansible playbooks and the addresses of external resources can be shared directly rather than having to be updated separately for each system. This feature has not yet been implemented in production, as there are currently not multiple CLE 6.0 systems on the same internal network, this ability could be utilized in the future.

C. Challenges for our approach

The use of an industry-standard revision control system to control LANL's CLE 6.0 systems allows the majority of objectives to be met quite easily. Revision Control systems keep a record of each change to the configuration files and scripts, including commit comments by the person who made the change; allows easy comparisons of revisions; makes it possible to roll back to previous versions; and potentially allows the sharing of the CM server repository with multiple remote SMWs. However, several issues were encountered implementing this initially, some of which have been addressed.

1) File Ownership and Permissions Within the Config Set: One of the first problems encountered was with respect to file ownership and permissions. Because they were designed for software development, none of the industry-standard revision control systems there were evaluated preserve ownership or permissions information. This is fine as long as the the configuration management system (i.e., Ansible) enforces these attributes when the configuration is applied to the system. However, it was discovered that several Cray-provided Ansible plays assume that the ownership and permissions of files within the config set will match their desired attributes when copied into place on the system. In several places, the Ansible plays will simply rsync a directory of files from the config set to its final location without checking any of their metadata. This led to immediate problems when the original config set directory was replaced on the SMW with a Subversion checkout, and attempted to boot the system, as several services failed to start because the files copied from the config set had the wrong permissions.

The current solution to this problem is to keep a list of files in the config set which are "sensitive" to permissions issues, and have the hourly "svn update" cron job enforce their ownership and permissions to prevent accidental changes. This solution works, but is potentially brittle as the list of files may change with new updates, and it needs to check the files in each of several possible config sets. Another possible option to explore is to use the "propset" option to Subversion which can enforce user/group/permissions on a given file. The moral equivalent could be used for other revision control systems.

2) Change Management Challenges on the SMW: One of the objectives for the implementation of revision control was to associate each change with a particular person. This is relatively easy to do with respect to changes made from the CM server, as Subversion includes the name of the committer in its logs. However, the Subversion checkout on the SMW must be owned by root for the Cray tools to work correctly, and all changes made from the SMW (such as with the `cfgset` tool) are made as root. This makes it impossible to tell with certainty who made a given change that originated on the SMW.

In addition, changes made on the SMW can immediately take effect on the larger system, whether they are committed to the repository or not. This has led to situations where a file is changed in the config set from the SMW to address some urgent issue, and then not committed to the repository for hours or days. At that point, the reason for the change might not be as immediately clear, making it difficult to write a good commit message.

For now, part of the commit process is to include the username of the committer in the commit message of all

changes from the SMW. A "nag script" runs periodically on the SMW and reports the presence of un-committed changes via e-mail to the team. In the future, "wrapper scripts" for the Cray tools may be explored which would prompt the user to perform an svn commit as soon as the Cray tool exits.

3) Requirement That Certain Changes be Made on the SMW: Currently Cray does not support running tools such as `cfgset` from a different host than the SMW. This breaks the administrator's usual workflow, which on other types of system requires that all changes be made from the CM server rather than being made "live" on the actual system. This is an important workflow because it prevents "in-progress" changes from accidentally taking effect on the running system. Many files can be edited without the use of these tools, allowing those changes to be made from the CM server; but `cfgset` performs several useful validations on this data, as well as auto-generating other files from user-supplied config data. However, it would be ideal if the changes would be prepared on the CM server in preparation for implementation. A solution to this issue is still being explored.

VI. Programming Environment Management

The elimination of the shared root under CLE 6.0 brings its own set of challenges to Programming Environment (PE) management. Consider the fact that each node's root filesystem is now RAM-based and the PE is pushing 50GB these days. Cray's legacy stack afforded a single PE release to be projected to back-end nodes via Data Virtualization Services (DVS) as an integral part of its shared root. Clearly it would be infeasible now to deliver over the wire a RAM-root, including a full PE revision, to every back-end node of the machine! CLE 6.0 rises to this challenge by factoring out the heavy-weight PE and projecting it to back-end nodes in traditional fashion.

CLE 6.0's PE management philosophy falls right in line with how it oversees its various other components. In fact the PE can simply be regarded as being just another specialized *image* type, though no node will actually boot it. CLE 6.0 also bakes more flexibility into what appears holistically on the back-end by permitting the administrator to *bind* any pre-built PE collection to that node's software stack. A nicety of this scheme is that the PE may be modified, or even built from scratch, and then {re-}bound without the necessity of taking the mainframe down. However to affect these changes on the system the PE needs to be "pushed" to the boot node and then a fairly heavy-weighted Anisbile play needs to be run on every node that has the PE mounted. Because of the impact this would have on running jobs, it would not be advisable to run on a system with running jobs. In order to address this issue, the site would need to implement an Ansible play that would be run by the resource manager's job prologue or epilogue to do the PE update. Missing here, however, is the opportunity to automate any fine-grained modification to the PE that is routinely performed by a CA/SA play at boot time; unfortunately such an alteration must still be made by hand. In order to address this issue, the administrators are using an LCM area to manage files within the PE image on the SMW that need to be under CM control.

Gone too is the notion of what static software versions comprise the PE. Cray's legacy mind set is to have sites *nuke and pave* PEs from one release to another. This, however, never works in practice as user requirements commonly demand off-rev versions of software. CLE 6.0 now makes it possible for each customer to tailor PE releases to the potpourri of software variants necessary to support its user base. And best of all: CLE 6.0 utilizes common RPM/Zypper repositories to build all of its image types so there need no longer be stack disparity between what runs on the mainframe and disk-full externals such as eLogin/CDL nodes.

The remaining cost associated with CLE 6.0's image schema that needs to be emphasized is that of disk bloat. The size of the system's boot RAID has historically been a limitation on Cray systems, leading to close scrutiny of all installed software - especially that of the PE. While the Trinity system does include the largest boot RAID device the administrators have seen to date, the variety of deployment images and flexibility in the PE management process will mandate relentless assessment of the disk space on this storage controller.

Adding or removing non-Cray software within the PE image can be a challenge too. For instance, installing the Intel Parallel Studio XE package requires some additional installation steps because the image must be `chroot'd` into. The installer requires that `dev` and `proc` be mounted. These can be mounted into the chroot'd image by the following commands.

```
mount -t proc none $PE_LOCATION/proc
mount -o bind /dev $PE_LOCATION/dev
chroot $PE_LOCATION
```

Now it will be possible to install other third party tools into the PE image if `proc` or `dev` are required. Another issue that is still under investigation is the Intel Parallel Studio will only allow for one version to be installed at a time. If it detects the presence of a previous version it will uninstall the older version and replace it. This is undesirable for sites that want to provide multiple versions of releases for its users.

VII. Update Testing and Rolling Upgrades

With the previous versions of Cray's system management software, almost all changes required the Cray back-end system to be down. The debugging and testing of the new changes lengthen the downtime of a system. 6.0/8.0 gives the administrators the ability to now test patches, field

notices, and configurations changes while the system is in a production mode without affecting users. The new image management and deployment system allows administrators to build images and only deploy them to a small set of nodes for testing purposes. This ability will reduce the amount of time required for dedicated system time (DST)/preventative maintenance (PM). Future versions of 6.0/8.0 also promise to provide support for rolling updates so that after a user job completes it will reboot the nodes within the allocation into the new image. There are exceptions to this update process of course, including if the updates require firmware, BIOS changes to the system, or Aries driver updates. Also, the testing of service node images will most likely be difficult, because most of the service nodes are actively in use on the system. An administrator could test the service image on a spare service node, but it is not a thorough test though because that node will not be providing the services that may need to be tested because it will be a generic image that is not configured for a specific service.

Even if the rolling updates are not being utilized by a site, the ability to prepare a system for an update is made significantly easier because of the new IMPS system. One thing to note that the LANL administrators have learned is not to map the new images using NIMS that are built ahead of time. The issue with this is that if a node is rebooted for some reason the node will boot with the newer image. This can lead to major issues, especially within service nodes. Another important point about testing updates is that configuration set mapping is also definable within NIMS. So not only can a new image be tested, but also a new configuration set. Within the configuration set the administrator can test new config changes (p0/config), the SA area (p0/ansible), and the files tree for simple sync (p0/files). This will allow testing of the major components of an upgrade on a test node to validate changes.

Staged Upgrades is another new feature Cray is debuting with 6.0/8.0. This allows the administrator to do an upgrade (UP01->UP02) while the system is still up. Cray is able to do this because the SMW is using the BTRFS file system. A snapshot can be created to capture the current running system's state and then `chroot` into that snapshot. All upgrades can be done within that snapshot and it can be ready for deployment at a later time by simply rebooting into that snapshot. Keep in mind though that any changes done to the SMW after modifying the snapshot will not be captured within that snapshot. Also, the snapshot does not necessarily have to be used at a later time. It can also be used to test the upgrade procedure and if it works as expected, then run the upgrade on the current snapshot the system is booted in to. While the Cray administrators have not had an opportunity to use this feature yet, they are eager to try this new procedure.

VIII. Conclusions

The project of upgrading from a previous CLE 5.X/SMW 7.X (Pearl/Pecos) software stack should not be undertaken without serious consideration of the time investment that will be required. The SMW and boot RAID will need to be formatted and reinstalled with SMW 8.0 and very little of the previous implementation of configuration management will be applicable to the new system. Learning Ansible is also essential to the administrator. The boot process, when it fails, is almost always due to an Ansible issue of some kind. If the site wants to take advantage of the Site Ansible area provided by Cray, the only option is to use Ansible in that space. Use of another configuration management system would be possible within the system, but considering the number of services and configuration files that Cray is managing it would be difficult to affect a change at the desired time in the boot process.

As CLE 6.0/SMW 8.0 matures through upgrades from Cray, the system will become more robust and admin friendly, but the initial release will still be a shock to those who are accustomed to the previous generations of the Cray system software. It will take time for Cray to improve the software so early adopters of 6.0/8.0 may have a harder time than sites who wait to upgrade. For those who are adventurous or have a need to upgrade to 6.0/8.0 with the initial release, there will be a steep learning curve, but the long term benefits of the new system software will eventually outweigh the early development cost of learning the new system.

IX. Acknowledgments

The authors would like to thank the entire LANL systems support team for their time and dedication to Trinity and helping ready it for production. We would also like to thank Cray for being responsive to our requests and helping drive the development of their product. Many long days were spent working closely with Cray's developers and support staff to help make the system operational and reliable. The authors would also like to thank Harold Longley at Cray as well for use of his slides.