UCLV
Universidad Central
"Marta Abreu" de Las Villas

FIE
Facultad de
Ingeniería Eléctrica

Departamento de Electrónica y Telecomunicaciones

Trabajo de Diploma

Automatización del proceso de despliegue de un clúster para computación de alto rendimiento.

Autor: Orlando Ernesto Abreu Cruz

Tutor: Ingeniero Javier Antonio Ruíz Bosch

Santa Clara, junio de 2018

"Año 60 de la Revolución."

Departamento de Electrónica y Telecomunicaciones

Trabajo de Diploma

Automatización del proceso de despliegue de un clúster para computación de alto rendimiento.

Trabajo de Diploma presentado en opción al Título Académico de

Ingeniero en Telecomunicaciones y Electrónica

Autor: Orlando Ernesto Abreu Cruz
email: oabreu@uclv.cu

Tutor: Ingeniero Javier Antonio Ruíz Bosch
Ingeniero
Dpto. de Electrónica y Telecomunicaciones, Facultad de Ing. Eléctrica, UCLV
email: jrbosch@uclv.cu

Santa Clara, junio de 2018

"Año 60 de la Revolución."

Hago constar que el presente Trabajo de Diploma fue realizado en la Universidad Central "Marta Abreu" de Las Villas como parte de la culminación de estudios de la especialidad de Ingeniería en Telecomunicaciones y Electrónica, autorizando a que el mismo sea utilizado por la Institución, para los fines que estime conveniente, tanto de forma parcial como total y que además no podrá ser presentado en eventos, ni publicados sin autorización de la Universidad.

Orlando Ernesto Abreu Cruz
Autor

Para cualquier información contacte con:

Dirección de Información Científico Técnica. Universidad Central "Marta Abreu" de Las Villas. Carretera a Camajuaní. Km 5 1/2. Santa Clara. Villa Clara. Cuba. CP. 54830

PENSAMIENTO

"*Imagination is more important than knowledge. Knowledge is limited but imagination encircles the world.*"

Albert Einstein.

DEDICATORIA

A Sophie.

Lux mea mundi.

AGRADECIMIENTOS

A mami y a papi. ¿Qué agradecimiento es suficiente para los padres? Siempre para mí. Muchas gracias a ambos. A Yanet, que al momento de escribir esto me acaba de hacer el regalo más maravilloso. Que sé que hasta cuando no lo ha parecido, me ha estado queriendo. A Carlos Alberto. No importa qué conceptos pueda haber adquirido para tí esa palabra. Eres mi *hermano*. A Miguel Ángel. Mi otro hermano por elección. A Anabel. A tí no te elegí, pero es como si te hubiesen hecho para que fueras la hermana más entrañable. Ahora mismo no se me ocurre un modo mejor de ser para mi hermana del alma. A mis abuelos. Blanca, Dignora, Orlando y Justo. Todos ellos me moldearon de alguna forma. Y a todos los amo. A tía Rosy. Mi Willy querida. Mi tía c. Mi tía Rosy. A mi tía Isabelita. He tratado de parecerme a tí y la imitación es la expresión más sincera de la admiración. A mi tía Xiomy. Si los ángeles guardianes existieran (que puede que existan) deben aprender de tí. A Omar y a Elías por tanta y tanta ayuda. A Olgui y a Guillermo. Por tratarme como padres. A Odalys y a Toledo. Por lo mismo. Y por el amor que le profesan a mi hija. Gracias. A mis primas adoradas. Mariangel, María de Jesús, Elianet. A ese primo siempre serio poniéndome metas que es Osvaldo. A David Beltrán. La mejor mano en el hombro que hubiera podido pedir. A David Ernesto por dejarme estar ahí a su lado para compartir aula con la mente más brillante de mi generación. A Juan Carlos por sus estadísticas torcidas y la mejor aritmética (no tanto el football). A Anabel, la más paciente, gracias por explicarlo todo una y otra vez. A Rolando, porque estuvo a mi lado desde el minuto 1. A Sandy, por su persecución de la perfección (logrando algo parecido a veces, otras no tanto). A mi tutor Javier. Gracias. A Héctor y a todos los que son ya mis compañeros (así lo veo yo) en la Dirección de Informatización. A quien se me pueda estar olvidando a la hora de escribir esto, que sepa que a nadie lo olvido realmente. Gracias.

TAREA TÉCNICA

Diseñar e implementar un sistema que permita la automatización del proceso de despliegue de un clúster para computación de alto rendimiento.

Se establecieron las *Tareas de investigación.*

La realización de un estudio sobre los tipos de clústeres.

La comparación de las herramientas de instalación y configuración desatendidas.

La selección de las herramientas necesarias y el diseño de un sistema.

El desarrollo y la implementación de un sistema para automatizar la puesta a punto del clúster.

Orlando Ernesto Abreu Cruz
Autor

Ingeniero Javier Antonio Ruíz Bosch
Tutor

RESUMEN

Resumen

Es un hecho que la puesta a punto de un clúster puede ser tediosa y propensa a errores. En esta investigación se analizaron los tipos de clústeres poniendo énfasis en los clústeres para computación de alto desempeño y el software que necesitan. Se analizaron las herramientas para la instalación y la configuración desatendidas seleccionando de entre ellas las que suponen una mejor opción. Se diseñó un sistema que integrase Puppet, Foreman y Gitlab (las herramientas seleccionadas) y garantizase la escalabilidad, la fiabilidad y la rapidez. Se desarrolló e implementó el sistema diseñado, arribándose a resultados exitosos en cuanto a lo pretendido.

Summary

It is a fact that setting up a cluster could be tedious and bring errors. In this research were analyzed the kinds of clusters emphasizing on high performance computing clusters and the software they need. The tools for unattended installing and configuring were analyzed picking from among them the ones that were suppossed to be a better option. It was designed a system that could integrate Puppet, Foreman y Gitlab (the selected tools) and grant scalability, reliability and speed. The designed system was developed and implemented, getting successful results according to what was pretended.

TABLA DE CONTENIDO

Página

LISTA DE FIGURAS

Figura Página

INTRODUCCIÓN

Los problemas actuales de modelación y simulación requieren gran capacidad de cómputo. Si bien las computadoras personales son cada vez más rápidas y eficientes, para obtener un buen resultado en este tipo de aplicaciones se requiere mucha mayor capacidad de procesamiento y memoria. Para poder resolver problemas de modelación numérica, mecánica de fluídos, mecánica de sólidos, solución de ecuaciones diferenciales parciales, entre otros, es necesario disponer de la capacidad de cómputo ofrecida por un clúster de computadoras. (Rocha Quesada, 2011)

El clúster como concepto o técnica se trata de extender las capacidades de las clases de componentes existentes. Así, en los términos más generales, un clúster es un grupo de elementos que pueden ser operacionales independientemente, y que se integran por algún medio para lograr un comportamiento coordinado y cooperativo. Esto es así tanto en sistemas biológicos como en organizaciones humanas y estructuras de computadoras.

En el campo de los sistemas de cómputo el concepto de clúster está siendo aplicado a la producción de nuevas estructuras de sistemas partiendo de elementos existentes que permitan obtener poder de cómputo que a través de otro enfoque podría costar fácilmente 10 veces más. En años recientes el hardware y el software en torno a los clúster han evolucionado de manera tal que las instituciones que suponen usuarios potenciales tienen variedad de opciones en cuanto a forma, escala, entornos y medios de implementación para la satisfacción de sus necesidades. Algunas de las computadoras más grandes del mundo son clústeres, pero los clústeres están jugando un papel importante también en cómputo comercial y técnico de mediana escala. Esto quiere decir, por tanto, que existen varios tipos de clústeres de computadoras que van desde las computadoras más grandes del mundo hasta grupos de computadoras en desuso. Varios clústeres han sido implementados en el ámbito industrial y académico. (Sterling, 2001)

La tecnología clúster permite a las organizaciones incrementar su capacidad de procesamiento usando tecnología estándar, tanto en componentes de hardware como de software que pueden adquirirse a un costo relativamente bajo.

Para que un clúster funcione como tal, no basta solo con conectar entre sí los computadores, sino que es necesario proveer un sistema de manejo del clúster, el cual se encargue de interactuar con el usuario y los procesos que corren en él para optimizar el funcionamiento. (Abarca, 2008).

Actualmente este proceso, que suele ser largo y tedioso para una persona y por ende propenso a errores, se ha estado realizando parcialmente de manera manual en nuestra universidad. Esto puede afirmarse dado que actualmente la instalación y configuración del clúster HPC se puede realizar de forma desatendida utilizando herramientas como Foreman, Cobbler (configurador de Kickstart), Anaconda (utiliza Kickstart) o Preseed. Además, para el manejo de configuraciones existen herramientas como Puppet o Ansible. Sin embargo, como se explicaba según (Abarca, 2008), para el funcionamiento de un clúster para la computación de alto rendimiento se necesita no solo de la instalación de un sistema operativo en sus nodos sino de la instalación y configuración de software adicional tal como gestor de colas, planificador de recursos, sistema de archivos y un software que se encargue de gestionar los procedimientos de instalación de aplicaciones y la gestión de sus dependencias. Precisamente esta segunda parte del despliegue de un clúster es la que se ha estado realizando de manera manual, teniendo los inconvenientes de la demora y ser propenso a errores, y repercutiendo también en la escalabilidad del sistema.

Esto nos hace plantearnos una interrogante que expresa la *situación del problema de investigación.*

¿Cómo automatizar el proceso de instalación y puesta a punto de un clúster computación de alto rendimiento?

Se plantea pues el siguiente *Objetivo general.*

Automatizar el proceso de despliegue de un clúster de computadoras para la computación de alto rendimiento.

Para desarrollar el *Objetivo general* se plantean los siguientes *Objetivos específicos.*

- *Comparar las herramientas de instalación y configuración desatendidas.*
- *Diseñar una estructura que permita la escalabilidad del sistema desarrollado.*
- *Desarrollar e implementar un sistema basado en las herramientas seleccionadas que permita automatizar la puesta a punto del clúster.*

Surgen pues las siguientes *Interrogantes de investigación.*

¿Qué tipos de clústeres existen y cuáles son sus propósitos?

¿Qué software es necesario instalar y configurar para que el clúster HPC funcione como tal?

¿Qué herramientas de instalación y configuración desatendida existen y cuáles utilizar?

¿Cómo integrar las herramientas seleccionadas para automatizar la instalación y la configuración del clúster?

Este proyecto pretende proveer a la Universidad Central "Marta Abreu" de Las Villas de un sistema que automatice completamente y permita agilizar y librar de errores el proceso de puesta a punto de un clúster de computadoras para computación de alto rendimiento respondiendo a la parcial inexistencia de un sistema para este propósito específico. Esto redundaría en mejores posibilidades para el desarrollo científico de la universidad. La implementación de esta herramienta brindará a las instituciones interesadas en el uso de HPC (así como a los técnicos e ingenieros de las mismas) un sistema que contribuirá al ahorro de tiempo, personal y capacitación del mismo y a la prevención de errores producto de la interacción humana derivando en una mayor fiabilidad. Este proyecto será también una contribución al uso de software libre que es importante en nuestro país.

La relativa modernidad de este tema hace que sea importante para el país. Esto se ve ampliamente argumentado en el ahorro que podrían experimentar empresas e instituciones del país por concepto de contratación y capacitación de personal así como en la agilización de este proceso. Los resultados de esta investigación poseen una aplicación práctica y teórica para especialistas tanto que tengan que ver con el proceso de puesta a punto del clúster como que sean usuarios finales del mismo. Existe la posibilidad de hacer uso de herramientas libres de pago para la puesta a punto desatendida y la orquestación de estos procesos, se puede afirmar que es posible la realización de este proyecto sin gastos mayores por concepto de licenciamiento de software y utilidades. También se cuenta con documentación suficiente para hacer uso de dichas herramientas.

Para el desarrollo de esta investigación se hará uso de métodos que permitirán abordar el problema. El método *histórico-lógico* permitirá ver el desarrollo histórico de los temas abordados por la investigación. Mientras, el método *analítico-sintético* permitirá procesar la bibliografía llegando a conclusiones respecto al estado del arte y de la práctica de los temas abordados. El método *inductivo-deductivo* permitirá seleccionar qué herramientas y métodos existentes serán utilizados en el sistema que se desea desarrollar. Luego, el método *sistémico-estructural* permitirá elaborar un sistema de contenido que vaya de lo general a lo particular.

Para la realización eficaz del informe se le dará la siguiente estructura.

Introducción. La introducción plasmará la relevancia del tema objeto de investigación, así como antecedentes y actualidad del mismo.

Capítulo 1. El capítulo primero se encargará de recoger las características esenciales de los medios y las tecnologías disponibles respecto al tema de investigación estableciendo comparaciones entre ellos.

Capítulo 2. El segundo capítulo recogerá la selección de herramientas así como su modo de empleo y la vía en que se utilizarán integradas en el desarrollo del sistema.

Capítulo 3. El capítulo 3 describirá el desarrollo y la implementación propiamente dicha del sistema y discutirá los resultados derivados de la misma.

Conclusiones.

Recomendaciones.

Referencias bibliográficas.

Anexos.

CAPÍTULO 1

1
MARCO TEÓRICO

Introducción

Este capítulo se dividirá en tres secciones. En una primera sección se expondrán los tipos de clśteres y sus características fundamentales haciendo énfasis en la computación de alto rendimiento, tema al que tributa este *Trabajo de diploma*. La segunda sección del capítulo se dedicará al software necesario para que un clúster HPC funcione como tal para así informar sobre lo que habrá de configurarse mediante las herramientas descritas en el capítulo siguiente. Así, la tercera sección tratará de las herramientas disponibles para la automatización de los procesos de instalación y configuración. Estará así explicado el estado del arte y de la práctica para establecer eficazmente un hilo conductor para la investigación.

1.1 Clústeres

1.1.1 Generalidades de los clústeres

De acuerdo con la interpretación brindada por (Sterling, 2001), los clústeres de computadoras son precisamente varias computadoras que pueden ser operativas individualmente, y que son integradas por medio de una interconexión de red y que soportan software accesible por el usuario para organizar y controlar tareas de computación concurrente que pueden cooperar en un programa o carga de trabajo común.

Los clústeres son usualmente empleados para mejorar el rendimiento y/o la disponibilidad por encima de la que es provista por un solo computador típicamente siendo más económico que computadores individuales de rapidez y disponibilidad comparables. De un clúster se espera que presente combinaciones de los siguientes servicios.

Alto rendimiento. Un clúster de alto rendimiento es un conjunto de computadores que está diseñado para dar altas prestaciones en cuanto a capacidad de cálculo.

Alta disponibilidad. Es un conjunto de dos o más máquinas que se caracterizan por mantener una serie de servicios compartidos y por estar constantemente monitorizándose entre sí.

Equilibrio de la carga. Un clúster de balanceo de carga o de cómputo adaptativo está compuesto por uno o más computadores (llamados nodos) que actúan como front-end del clúster, y que se ocupan de repartir las peticiones de servicio que reciba el clúster a otros computadores del clúster que forman el back-end de éste.

Escalabilidad. La escalabilidad es la propiedad deseable de un sistema, una red o un proceso, que indica su habilidad para, o bien manejar el crecimiento continuo de trabajo de manera fluida, o bien para estar preparado para hacerse más grande sin perder calidad en los servicios ofrecidos.

Por supuesto, utilizar un clúster puede aportar importantes ventajas en gran variedad de aplicaciones y ambientes, tales como incremento de velocidad de procesamiento ofrecido por los clústeres de alto rendimiento, incremento del número de transacciones o velocidad de respuesta ofrecido por clústeres como los de balanceo de carga e incremento de la confiabilidad y la robustez ofrecido por los clústeres de alta disponibilidad. (Abarca, 2008)

1.1.2 Componentes de los clústeres

En general, un clúster necesita de varios componentes de software y hardware para poder funcionar. Que son los siguientes.

Nodos. Pueden ser simples computadores, sistemas multiprocesador o estaciones de trabajo (workstations). En informática, de forma muy general, un nodo es un punto de intersección o unión de varios elementos que confluyen en el mismo lugar. Bajo el contexto de clúster tenemos dos tipos de nodos, que son.

Nodos dedicados. Los nodos no disponen de teclado, mouse ni monitor y su uso está exclusivamente dedicado a realizar tareas relacionadas con el clúster.

Nodos no dedicados. Los nodos disponen de teclado, mouse y monitor y su uso no está exclusivamente dedicado a realizar tareas relacionadas con el clúster, el clúster hace uso de los ciclos de reloj que el usuario del computador no esta utilizando para realizar sus tareas.

Almacenamiento. El almacenamiento puede consistir en una NAS, una SAN, o almacenamiento interno en el servidor. El protocolo más comúnmente utilizado es NFS (Network File System), sistema de ficheros compartido entre servidor y los nodos. Sin embargo existen sistemas de ficheros específicos para clústeres como Lustre (CFS) y PVFS2.

Sistemas operativos. Debe ser multiproceso, multiusuario. Otras características deseables son la facilidad de uso y acceso y permitir además múltiples procesos y usuarios.

Conexiones de red. Los nodos de un clúster pueden conectarse mediante una simple red Ethernet con placas comunes (adaptadores de red o NICs), o utilizarse tecnologías especiales de alta velocidad como Fast Ethernet, Gigabit Ethernet, Myrinet, Infiniband, SCI, etc.

Middleware. Es un software que generalmente actúa entre el sistema operativo y las aplicaciones con la finalidad de proveer a un clúster lo siguiente.

- Una interfaz única de acceso al sistema, denominada SSI (Single System Image) la cual genera la sensación al usuario de que utiliza un único ordenador muy potente.
- Herramientas para la optimización y mantenimiento del sistema. Migración de procesos, checkpoint-restart (congelar uno o varios procesos, mudarlos de servidor y continuar su funcionamiento en el nuevo host), balanceo de carga, tolerancia a fallos, etc..

- Escalabilidad. Debe poder detectar automáticamente nuevos servidores conectados al clúster para proceder a su utilización.

Existen diversos tipos de middleware, como por ejemplo: MOSIX, OpenMOSIX, Condor, OpenSSL, etc..

Protocolos de comunicación y servicios.

Aplicaciones.

Ambientes de programación distribuida. Los ambientes de programación distribuida permiten implementar algoritmos que hagan uso de recursos compartidos. CPU (Central Processing Unit), memoria, datos y servicios.

(Abarca, 2008)

1.1.3 Clasificaciones de los clústeres

La elección de los computadores del clúster es más fácil y económica debido a su flexibilidad. Pueden tener todos la misma configuración de hardware y sistema operativo (clúster homogéneo), diferente rendimiento pero con arquitecturas y sistemas operativos similares (clúster semi-homogéneo), o tener diferente hardware y sistema operativo (clúster heterogéneo), lo que hace más fácil y económica su construcción.

El término clúster tiene diferentes connotaciones para diferentes grupos de personas. Los tipos de clústeres, establecidos en base al uso que se de a los clústeres y los servicios que ofrecen, determinan el significado del término para el grupo que lo utiliza. Los clústeres pueden clasificarse con base en sus características. Se pueden tener clústeres de alto rendimiento (HPC - High Performance clústeres), clústeres de alta disponibilidad (HA - High Availability) o clústeres de alta eficiencia (HT - High Throughput).

Alto rendimiento (HPC). Son clústeres en los cuales se ejecutan tareas que requieren de gran capacidad computacional, grandes cantidades de memoria, o ambos a la vez. El llevar a cabo estas tareas puede comprometer los recursos del clúster por largos periodos de tiempo.

Alta disponibilidad (HA). Son clústeres cuyo objetivo de diseño es el de proveer disponibilidad y confiabilidad. Estos clústeres tratan de brindar la máxima disponibilidad de los servicios que ofrecen. La confiabilidad se provee mediante software que detecta fallos y permite recuperarse frente a los mismos, mientras que en hardware se evita tener un único punto de fallos.

Alta eficiencia (HT). Son clústeres cuyo objetivo de diseño es el ejecutar la mayor cantidad de tareas en el menor tiempo posible. Existe independencia de datos entre las tareas individuales. El retardo entre los nodos del clúster no es considerado un gran problema.

Los clústeres pueden también clasificar como clústeres de IT Comerciales (Alta disponibilidad, Alta eficiencia) y clústeres Científicos (Alto rendimiento). A pesar de las discrepancias a nivel de requerimientos de las aplicaciones, muchas de las características de las arquitecturas de hardware y software, que están por debajo de las aplicaciones en todos estos clústeres, son las mismas. Más aún, un clúster de determinado tipo, puede también presentar características de los otros. (Abarca, 2008)

1.1.4 High Performance Computing

El término High Performance Computing (HPC) no posee, al momento de la redacción del presente documento, una definición universalmente aceptada en el ámbito académico. A menudo, los actores que conforman la disciplina de la computación moderna tienen la tendencia a utilizar la sigla HPC ya sea para referirse a ciertas prácticas en ingeniería de software que para hablar de algunos tipos de infraestructuras hardware.

Según (Mrozek, 2014), el Profesor Jack Dongarra de la Universidad de Tennessee (USA), considerado como uno de los referentes a nivel mundial en la materia, propone la siguiente definición:

"High-performance computing generalmente se refiere a la práctica de agregar poder de cómputo en una manera en que entregue mucho más alto rendimiento que el que se

puede obtener de una computadora de escritorio o estación de trabajo típica para resolver grandes problemas en ciencia, ingeniería o negocios."

Sin embargo, una definición más precisa de la HPC debería incluir todas las tecnologías de procesamiento en paralelo de un gran volumen de información, de manera rápida y fiable. Completando la definición de Dongarra, podemos afirmar que la HPC es una práctica que consiste en concebir, desarrollar e instalar sistemas de computación dedicados al tratamiento de problemas industriales o académicos complejos que poseen la capacidad de:

- realizar varios miles de millones de operaciones de coma flotante por segundo - FLoating-Point Operations Per Second (FLOPS);
- almacenar varios terabytes de datos, pudiendo acceder a los mismos en todo momento con un rendimiento cercano al de un dispositivo de memoria masiva de un ordenador personal;
- paralelizar las tareas entre los procesadores disponibles y comunicar los cambios de estado en tiempo real de manera eficiente en términos de escalabilidad;
- opcionalmente, visualizar una representación tridimensional de los resultados obtenidos a partir de los cálculos efectuados.

Un clúster computacional (o de cálculo) es una infraestructura que posee las capacidades listadas anteriormente. De manera simplificada, es posible representar un clúster computacional como una red de ordenadores que trabajan conjuntamente y en paralelo con el objetivo de resolver problemas complejos. Los ordenadores que forman parte de esta red, se denominan nodos de cómputo y pueden comunicar entre sí a gran velocidad como así también acceder a un espacio de almacenamiento de datos compartido.

Los avances tecnológicos y el desarrollo continuo de casi todas las áreas de la ciencia y la ingeniería, requieren el diseño de nuevas soluciones para el procesamiento de grandes cantidades de datos con el objetivo de poder resolver un número elevado de problemas complejos en un plazo razonable. La computación científica es la disciplina que se ocupa

del diseño de modelos matemáticos y métodos numéricos para resolver de manera eficiente estos problemas complejos mediante el uso de computadoras. Debido a los requisitos en tér minos de capacidad de cálculo y de memor ia, los problemas abordados por la computación científica serían inmanejables sin recurrir a la utilización de computadoras paralelas.

La HPC representa una disciplina estratégica para que empresas, centros de investigación y entes gubernamentales puedan generar nuevos conocimientos, susceptibles de convertirse en productos y servicios innovadores. La HPC está cambiando las reglas existentes en materia de competitividad económica y liderazgo científico.

Las supercomputadoras son imprescindibles para modelar fenómenos complejos a través de la implementación de modelos matemáticos cuyos resultados se obtienen gracias a la utilización paralela de varios miles de procesadores. Por ejemplo, un modelo cuya resolución implique 24 horas de cálculo en una computadora hogareña, tomaría sólo unos pocos segundos en una supercomputadora. O bien, se podría resolver en menos de 24 horas el mismo modelo, con un caso de base 12.000 veces mayor evaluando la sensibilidad sobre más de 5.000 parámetros.

Estas ganancias de tiempo (o de fiabilidad) pueden reducir los costos en cada una de las etapas de desarrollo de un producto (diseño, optimización, validación) y potencialmente afectan muchas áreas de aplicación industrial como los sectores automotriz y aeroespacial, energético, químico, financiero y sectores de punta como el estudio de los nano-materiales, la bio-medicina o la bio-genética aplicada a la producción de alimentos.

Un país que no posee la capacidad de cálculo para resolver sus propios problemas no tendrá mas alternativa que adquirir las soluciones que otros le provean. Uno de los aspectos mas importantes de la soberanía en el siglo XXI lo constituye la posibilidad de producir modelos matemáticos y contar con los medios materiales para resolverlos. En este momento la disponibilidad de potencia de cálculo acorde a las propias riquezas aumenta de manera exponencial la capacidad de un país para producir conocimiento. (Russo, 2016)

1.2 Software para el funcionamiento del HPC

Para explicar el software necesario para el funcionamiento del clúster HPC, se describirá su estructura. La figura 1-1 muestra una suerte de plano lógico del clúster, estableciendo sus principales componentes y exponiendo la conexión vía SSH de los usuarios.

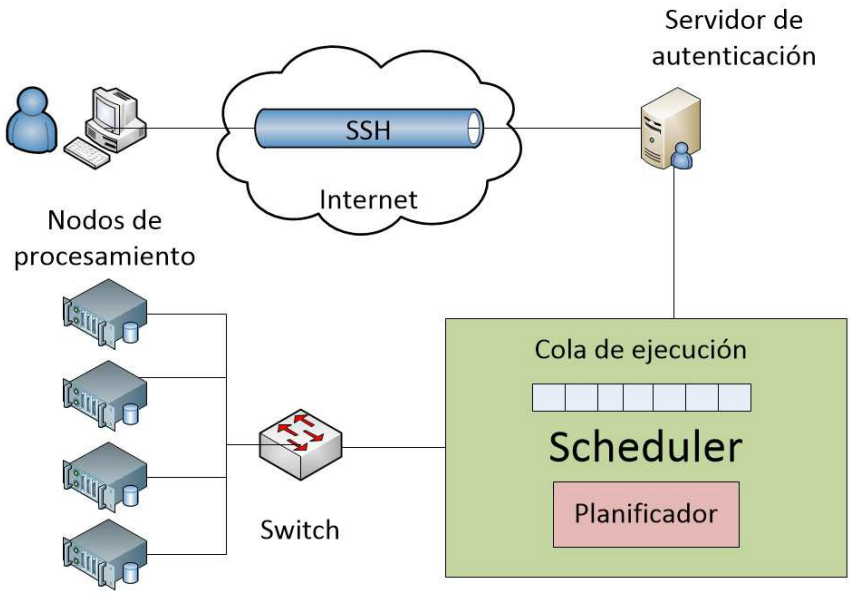

Figure 1–1: Estructura de un clúster. *Fuente. Autor.*

Sin embargo, es necesario ver la manera en que se relacionan e interactúan los componentes del clúster HPC. Para ello, la figura 1-2 muestra el esquema de un clúster de altas prestaciones, con todos los componentes esenciales para el correcto funcionamiento del mismo, de forma que se observe con claridad la interrelación entre ellos, lo que supondrá que a la hora de explicar dicho componente el lector pueda situarlo adecuadamente y vea sus dependencias con otros componentes.

1.2.1 Gestores de cola

Un gestor de colas es un sistema de administración y distribución de los trabajos que se envían al clúster. En este apartado se exponen los gestores de colas más extendidos.

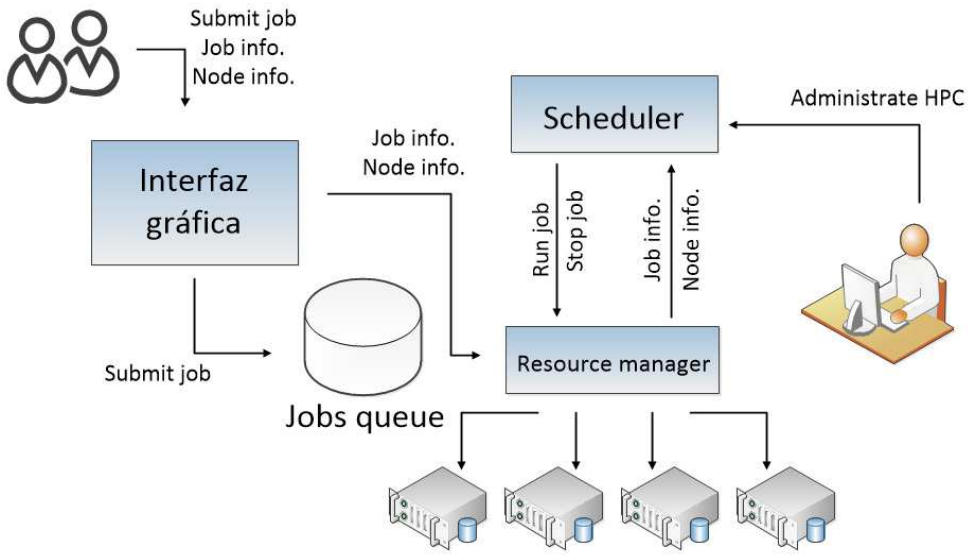

Figure 1–2: Interacción entre los componentes de un clúster. *Fuente. Autor.*

PBS

Gestor de colas originalmente diseñado por la NASA que ha llegado a ser un estándar de facto en clústeres Linux. Proporciona una serie de herramientas para la gestión de trabajos batch, utilizando una unidad de programación de tareas. Además, permite el enrutamiento de estos trabajos a través de diferentes computadores.

Cuenta con capacidades para definir e implementar políticas sobre la utilización de los recursos disponibles. El PBS llegó a ser usado por el 90% de los clústeres de producción Linux por ser el más extendido y fiable.

PBS está compuesto básicamente por dos componentes, los comandos de usuario y los demonios del sistema.

PBS se compone de tres demonios, dos de ellos son necesarios para su correcto funcionamiento mientras que el scheduler puede ser reemplazado por un planificador externo.

- *Server*. Demonio encargado de recibir los trabajos a ejecutar y estar a la espera de comandos de usuario.
- *MOM*. Demonio que se ejecuta en cada nodo de cómputo y que envía y recibe los trabajos a ejecutar.
- *Scheduler*. Planificador que decide qué trabajos se van a ejecutar en el clúster dependiendo de la política seleccionada.

(Ferreira, 2001; Vrenios, 2002)

Slurm

Gestor open-source diseñado para clústeres GNU/Linux de todos los tamaños. Provee algunas funcionalidades clave que hacen muy interesante su uso.

- Ubica acceso exclusivo y/o no exclusivo de los recursos (nodos de cómputo) a los usuarios por alguna duración de tiempo de manera de que ellos puedan realizar algún trabajo.
- Provee un ambiente de trabajo para comenzar, ejecutar y monitorear trabajos (típicamente paralelos) en un conjunto de nodos específicos.
- Se encarga de arbitrar peticiones a recursos conflictivos, mediante el manejo de una cola de trabajos pendientes.
- Provee un API para la integración con planificadores externos tales como el MAUI Scheduler. Mientras en otros gestores de cola no existe.
- Es lo suficientemente simple para el usuario final, entender su código fuente y agregar funcionalidades.
- Permite manejar aspectos clave de las arquitecturas multicore como puede ser la afinidad.

SLURM está compuesto básicamente por dos tipos de demonios y 5 comandos de usuario. En la figura 1-3 podemos ver los diferentes componentes de SLURM y la interrelación entre cada uno de ellos.

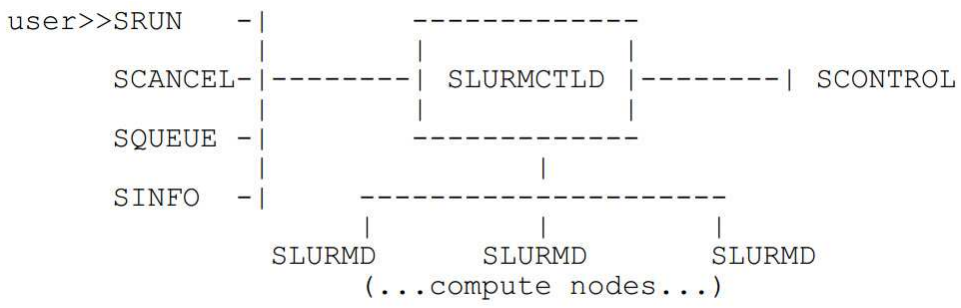

Figure 1–3: Interrelación entre los componentes de SLURM. *Fuente. (SLURM, 2007).*

Como podemos apreciar en la figura 1-3, SLURM se compone de dos demonios, SLURMCTLD y SLURMD.

- SLURMCTLD: Demonio encargado del control principal de SLURM. A diferencia del demonio servidor de PBS, SLURMCTLD tiene la propiedad de ser Multi-Threaded.
- SLURMD: Demonio que se ejecuta en cada nodo de cómputo y que envía y recibe los trabajos a ejecutar.

(SLURM, 2007)

LoadLeveler

Gestor de colas diseñado por IBM, que se caracteriza por su facilidad para construir, subministrar y procesar los trabajos en el clúster.

Permite la ejecución tanto de trabajos serie como paralelos (MPI) y es fácilmente escalable a miles de procesadores. Loadleveler fue uno de los primeros sistemas en incorporar el algoritmo de planificación backfill que ha continuado mejorando la escalabilidad, velocidad y rendimiento de dicho algoritmo para su gestor de colas. Fue el primer sistema en ofrecer una completa API de scheduling que ofrecía una amplia flexibilidad para la programación del scheduler. Actualmente ofrece un completo conjunto de API's que hace que se sea una atractiva posibilidad a tener en cuenta a la hora de escoger un gestor de colas. Posee una mejor gestión de tolerancia a fallos y auto-reparación, mediante el uso de checkpoints.

Proporciona una interfaz gráfica de usuario, que a pesar de ser muy rudimentaria y ejecutarse en la consola, fue muy atractiva cuando se implementó, ya que permitía subministrar trabajos y hacer tareas de monitorización fácilmente.

Como desventajas de este gestor colas, hay que hacer notar que los comandos son bastante diferentes a los utilizados por PBS, Torque o Slurm en cuanto a su sintaxis. Cabe decir también que es software privado y para poder utilizar todo el pack es necesario comprar el producto. (Ferreira, 2001)

Torque

Basado en PBS y creado por la empresa clúster Resources Inc. Se caracteriza porque se puede integrar con un atractivo entorno gráfico para enviar trabajos, consultar estadísticas; desplazando para ello la forma de enviar trabajos via SSH por consola a un segundo plano.

Sus principales características son las siguientes.

- Tolerancia a fallos.
  - Posee chequeos y manejos de condiciones de fallo.
  - Permite la reparación de muchos errores.
  - Tiene un script de soporte para chequear el estado o salud de los nodos.
- Interfaz de planificación.
  - Permite integrarse con el scheduler MOAB, lo que le da una inmensa potencia y flexibilidad.
- Escalabilidad.
  - Servidor significativamente mejorado para el modelo de comunicación MOM.
  - Habilidad para manipular grandes clústeres (sobre 15 TF/2,500 procesadores).

Una vez más, su parte negativa es la de ser un programa privado; para conseguir poder obtener toda su potencia, es necesario adquirir la versión comercial. (adaptaive computing, 2014; Panadero Martínez, 2008)

1.2.2 Planificadores externos

Podemos definir planificadores externos, como el sistema que se encarga de distribuir los trabajos en el clúster dependiendo de los parámetros de configuración, prioridades y la política de este.

Es el encargado de organizar los trabajos en las colas y lanzarlos al clúster o incluso, si lo considera necesario, interrumpir un trabajo y devolverlo a la cola para lanzar a otro más prioritario en su lugar.

Debido a la complejidad que presenta este componente y al gran número de parámetros de configuración que posee, como puede ser: la selección de la política de planificación, configuración de los parámetros del simulador, configuración del checkpoint, selección del modo de trabajo a utilizar(normal, monitor, simulador), reserva de nodos, número máximo de recursos a utilizar por un determinado tipo de trabajo, etcétera; hace que resulte uno de los componentes más complejos del clúster y que más dedicación ha tenido en este estudio. (Panadero Martínez, 2008)

MAUI

MAUI es un planificador externo integrado en el sistema, que periódicamente itera sobre los trabajos pendientes en el gestor de colas. Cada iteración (cuyo intervalo de tiempo varía en función del parámetro RMPPOLL), consulta al gestor de colas el estado de los trabajos en el sistema. A partir de esa información, MAUI ordena los trabajos pendientes de ejecución en función de las políticas de prioridad del sistema, y los ejecuta en el orden de mayor a menor prioridad.

Presenta la ventaja de ser de código abierto, lo que ha facilitado su rápido crecimiento, y llegó a ser uno de los planificadores más extendidos. Permite ser integrado en la gran mayoría de gestores de colas, tales como PBS, LoadLeveler, SGE y BPROC.

Es totalmente configurable a las necesidades del usuario. Como características destacaremos las siguientes.

- Permite definir diferentes políticas con diferentes prioridades.
- Posee una interface Metascheduling.
- Permite configurar múltiples políticas backfill.
- Diferentes modos de uso como son el normal, test o monitor.
- Posee un simulador de scheduler que permite analizar workloads para diferentes políticas.

(Eadline, 2009; Panadero Martínez, 2008)

MOAB

Planificador de la empresa clúster Resource Inc. Es la versión comercial de MAUI, su antecesor.

Se ha impuesto rápidamente gracias a su gran potencia y versatilidad de configuración.

Al igual que MAUI, MOAB incorpora un simulador, el cual permite poder probar en un tiempo razonable un extenso abanico de configuraciones posibles del clúster.

Las características son básicamente las mismas que en MAUI, sólo se expondrá algunas mejoras y optimizaciones que se han incorporado.

- Maximiza las prestaciones del clúster gracias a la gestión inteligente de recursos.
- Permite reservas recursos por un periodo de tiempo determinando para algún usuario en concreto.
- Permite acciones arbitrarias asociadas a los eventos del clúster. Estas acciones se conocen como triggers.

(Eadline, 2009; Panadero Martínez, 2008)

LoadLeveler Scheduling

Planificador que viene incorporado en el paquete comercial de LoadLeveler. Es el menos avanzado de los mostrados en esta sección, pero su facilidad de configuración y su simplicidad hace que sea atractivo cuando no se requieren actividades demasiado complejas del clúster. (Ferreira, 2001)

1.2.3 Librerías de paso de mensajes

Permiten la comunicación entre los diferentes nodos de cómputo. Es necesario tener instaladas las librerías en todos los nodos de cómputo para su correcto funcionamiento. Seguidamente se presentan los 2 estándares más utilizados.

Standard MPI

Librería de paso de mensajes propuesta como estándar por un comité de vendedores, implementadores y usuarios. Implementa las siguientes características y funcionalidades.

- Colección de funciones que ocultan detalles de bajo nivel, tanto software como hardware.
- Diseñado para obtener un alto rendimiento tanto en máquinas paralelas como en clústeres.
- La unidad básica de paralelismo son los procesos independientes.
- Tienen espacios de memoria independientes.
- Intercambio de datos y sincronización mediante paso de mensajes.
- A cada proceso se le asigna un identificador interno propio.
- Proporciona una funcionalidad flexible, incluyendo diferentes formas de comunicación, rutinas especiales para comunicaciones "colectivas" y la habilidad de usar tipos de datos y topologías definidas por el usuario dentro de las comunicaciones.
- Programación en Fortran y C. En la revisión del estándar (MPI-2) se soporta C++ y Fortran 90.

(Sloan, 2004)

PVM (Parallel Virtual Machine)

Software de libre distribución creado por Oak Ridgre en 1989 que permite ejecutar aplicaciones paralelas sobre máquinas distribuidas y heterogenias. Sus principales características son los siguientes.

- Existe un conjunto de nodos (hosts) susceptibles de poder ser usados en tareas de computación, definido por el usuario.
- El usuario selecciona el conjunto de máquinas donde se ejecutarán la tarea de computación paralela. Dicho conjunto puede alterarse en tiempo de ejecución, añadiendo o quitando máquinas, esto es importante para la tolerancia a fallos.

- Acceso semitransparente al hardware. Podemos aprovechar las características de cada nodo para ejecutar determinados cálculos.
- Computación basada en el proceso.
- Modelo de paso de mensajes explícito.
- Soporte para arquitecturas heterogéneas, en términos de máquinas, redes y aplicaciones.
- Soporta los lenguajes C, C++ y Fortran.

(Sloan, 2004; Vrenios, 2002)

1.2.4 Software para la instalación de aplicaciones de tipo científico y la gestión de dependencias

Se expondrán ahora los principales softwares usados para la instalación de aplicaciones y gestión de dependencias en ambiente de HPC que son los llamados gestores de paquetes.

Nix

Nix requiere de un almacén (store) donde contiene todos los softwares y librerías que instala y utiliza expresiones Nix (expressions) para realizar todos los pasos necesarios para instalar un paquete necesario en el almacén. Expresiones Nix son especificaciones declaradas que describen todos los aspectos para la compilación de un componente (software o librería), por ejemplo, obtener las fuentes, compilarlas, determinar los componentes del cuales depende y las restricciones impuestas a esas dependencias. Estas expresiones no son hechas en un lenguaje de programación específico si no que usan un simple lenguaje funcional con un conjunto de atributos, esto ofrece la ventaja de la flexibilidad. Los usuarios avanzados o administradores del sistema pueden adaptar las expresiones Nix para compilar una variante de un componente adaptado específicamente a sus necesidades. Por ejemplo, alguna funcionalidad requerida desactivada por defecto puede ser habilitada y las funcionalidades innecesarias se puede desactivar, también los componentes puede ser compilados con los parámetros de optimización específicos para el entorno de destino.

Tanto el gestor de paquetes Nix como el sistema operativo (NixOS) soportan la instalación de forma arbitraria de muchas configuraciones de software. Como en la mayoría de los sistemas HPC, este instala cada paquete en un prefix (ubicación) único, que Nix lo determina por el hash del archivo del paquete y sus dependencias por lo que el resultado no tiene una convención de nombre legible por humanos. (Dolstra, 2004, 2008; Gamblin, 2016)

Maali

Para apoyar a una amplia y diversa comunidad de usuarios el centro de Pawsey Supercomputing (previamente conocido como iVEC) ha desarrollado la herramienta Maali, un sistema automatizado y ligero para la gestión de un conjunto diverso de aplicaciones y ibliotecas científicas en el HPC de su centro. Inicialmente, Maali era conocido como iVEC Build System y fue desarrollado en el 2012 para apoyar los sistemas de clúster Epic y Fornax.

Maali en su núcleo es un conjunto de secuencias de comandos (scripts) BASH que están diseñados para permitir la automatización del proceso de Autoconf (configure, make y make install) comúnmente utilizado por muchas aplicaciones HPC. Maali trabaja a partir de un conjunto de archivos de configuración a nivel de sistema que define un conjunto de variables de entorno por defecto. Esto permite controlar varios aspectos de la instalación; tales como la ubicación del directorio de compilación predeterminado, el directorio de instalación, la ubicación del código fuente (paquete original para ser descargado y almacenado), el lugar de los archivos de registro, y los compiladores utilizados para configurar y compilar una aplicación.

La herramienta Maali automatiza los siguientes pasos:

- Descargar el software.
- Descomprimir el software.
- Compilar e instalar el software (configure, make y make instalar).
- Crear un archivo de módulo.

- Documentar el procedimiento.

(Bording, 2016)

SWTools

SWTools también es una "infraestructura para el manejo de software" que fue desarrollado por el Centro Nacional de Ciencias de la Computación (NCCS) y Oak Ridge National Lab (ORNL).

En el diseño de la estructura de directorios para el sistema de gestión de software SWTools, presenta una estructura jerárquica que consiste en tener instalaciones de documentación y software individuales para cada máquina. En la raíz del sistema, se crea una única carpeta para cada la máquina. Un compromiso es hecho en relación con el tema de las carpetas para las máquinas que son casi idénticas en cuanto al uso y el hardware, los principales ejemplos de esto son los sistemas Cray XT4. En el momento de desarrollo de la herramienta NCCS tiene consta de varios sistemas Cray XT4, y debido a esto se elige crear un directorio único para todo el software y la documentación XT4 (este directorio se encuentra en lo que se llamaría nivel de máquina). Esto es, en un intento de encontrar la solución que equilibre la duplicación del trabajo y el riesgo de incompatibilidad entre todas las máquinas. Por lo tanto, todos los sistemas XT4 comparten ejecutables y software. La mayoría de las otras máquinas dentro del NCCS no comparten ningún software y usan únicamente los softwares que son compilados específicamente para ese sistema.

El gestor de software SWTools de NCCS ha sido una gran desafío a pesar de estar estructurado únicamente para los requerimientos presentes en el NCCS, aun así este puede servir de información para ser usado en otros centros u organizaciones con requerimientos similares. (Jones and Fahey, 2008; SWTools, 2011)

Smithy

Smithy es un seguimiento de SWTools, y también se ha desarrollado en el NCCS/ORNL. Es compatible con la infraestructura SWTools y por lo tanto puede ser utilizado como un reemplazo directo de este, pero también es compatible con un enfoque

alternativo utilizando fórmulas basadas en el sistema de gestión de paquetes Homebrew para Mac OS X. Como tal, proporciona soporte funcional disponible para ser utilizado en estas fórmulas, y permite la reutilización de código a través de fórmulas. Smithy está disponible al público junto con la documentación detallada. La actividad de desarrollo se ha ralentizado significativamente desde septiembre del 2013, con cambios ocasionales que en su mayoría se centra en correcciones de errores.

Smithy está diseñado para gestionar programas dentro de un entorno HPC Linux o Mac utilizando modulefiles para cargar software para el usuario en una Shell de comandos. (Geimer *et al.*, 2016; Girolamo, 2016)

Spack

Spack es otro de los gestores de paquetes modernos que es considerado como una buena alternativa. Está escrito en Python, gracias a su flexibilidad ha tenido un creciente uso en HPC. Al igual que los sistemas anteriores, Spack es compatible con un número arbitrario de las instalaciones de software, y como Nix puede identificarlos con los hashes. A diferencia de cualquier sistema anterior, Spack proporciona un lenguaje conciso para especificar y administrar el espacio combinatorio de configuraciones de software de HPC. Spack ofrece las siguientes características únicas:

- Paquetes componibles, explícitamente parametrizados por la versión, plataforma, compilador, las opciones y dependencias.
- Una sintaxis novedosa de especificación recursiva para el grafo de dependencia y limitaciones, que ayuda en la gestión de la construcción del espacio de parámetros.
- Dependencias versionadas virtuales para manejar versionadas, incompatible ABI interfaces como MPI.
- Un nuevo proceso de concretización que traduce una especificación de compilación abstracta en una especificación de compilación completa y concreta.
- Un entorno de compilación que utiliza contenedores de compilador para hacer cumplir la consistencia de compilación y simplificar la escritura de paquetes.

En Spack, los packages son scripts de Python, estos son los que compilan los softwares. Cada packages es una clase que se hereda de la clase base Package, esta implementa la mayor parte del proceso de compilación, pero las subclases proporcionan sus propios métodos de instalación para manejar los detalles de los paquetes particulares. (Gamblin, 2016)

EasyBuild

EasyBuild es una plataforma modular para compilar e instalar software, escrito en Python. Inicialmente dio soporte a pocos paquetes de software pero rápidamente se convirtió en una parte muy importante de las herramientas de soporte de usuario utilizadas por el equipo del HPC UGent. Hoy en día, EasyBuild 2.8.0 tiene soporte para 931 paquetes de software científico y está siendo desarrollado y mejorado continuamente. EasyBuild permite limitar la cantidad de tiempo y esfuerzo necesarios para instalar y actualizar paquetes de software para el usuario final, solamente invirtiendo el tiempo necesario una sola vez para poner en práctica el procedimiento de instalación en un easyblock o quizás tan solo crear un archivo easyconfig. Compilaciones subsecuentes de nuevas versiones de un paquete de software o compilaciones que usan diferentes parámetros por lo general se puede obtener con muy poco o ningún esfuerzo, ahorrando así mucho de tiempo y mano de obra. (Hoste, 2016)

En abril de 2012, después de más de tres años de desarrollo interno en UGent, se lanzó EasyBuild en GitHub2 como software libre bajo la licencia GPLv2, siendo considerada la versión 1.0 la primera versión pública, estable y robusta de esta plataforma, también EasyBuild en su versión más reciente está disponible en (PyPi)3. Actualmente está publicada la versión 3.5.3 de EasyBuild. EasyBuild solamente tiene dos dependencias directas: Python (Python 2.6 o superior) y módulos de entornos (Tcl(/C) o Lmod). (Geimer *et al.*, 2016)

EasyBuild confía fuertemente en los módulos de entorno, por lo que el paquete de software para gestionar módulos de entorno es un requisito previo importante. EasyBuild

genera automáticamente los módulos de entorno para cada paquete de software que se instala, aliviando de este modo que el usuario tenga que crear manualmente archivos de módulos apropiados. Por otra parte, se basa en el conjunto de módulos de entornos disponibles para la obtención de información acerca de los paquetes de software instalados y para resolver las dependencias.

Mediante la generación de un módulo de entorno para cada instalación completa, EasyBuild permite mantener diferentes versiones sin que se afecten entre sí. Además de proporcionar acceso al paquete de instalación para los usuarios finales, estos módulos también ayudan a EasyBuild a localizar el software para instalaciones posteriores de otros paquetes de software dependientes de este, es decir, para resolver las dependencias. (Easy-Build, 2016; Ruiz Bosch, 2016)

1.3 Servicios desatendidos

El proceso de instalar un sistema operativo, instalar el software necesario y configurarlo puede ser sencillo para un ordenador personal o un solo dispositivo sea el que sea. Sin embargo, este proceso resulta tedioso y propenso a errores si lo vemos a la escala de un clúster HPC. Para que el clúster funcione como tal se precisa instalar un sistema operativo en cada nodo, instalar el software que se expuso en la sección anterior y configurarlo todo para el correcto funcionamiento del clúster.

Se hace pues necesario el uso de herramientas que permitan llevar a cabo la instalación de manera desatendida, así como una configuración inicial que permita luego manejar también de forma desatendida las instalaciones y configuraciones del software pertinente.

1.3.1 Herramientas de instalación desatendida

Para la instalación desatendida existen varias herramientas. Para la instalaciń desatendida de las distribuciones de Linux existen dos herramientas fundamentales. *Preseed* usada por el *debian-installer* de *Debian* y *Kickstart*, ideada por *Red Hat* para el uso principal (pero no exclusivo) de su sistema operativo *Red Hat Enterprise Linux*. La filosofía de ambos es la misma, responder a las preguntas que hace el asistente de instalación.

Preseed

Es un archivo que contiene pre configurada la respuesta a cada una de las preguntas realizadas durante el proceso de instalación, se le puede agregar un conjunto de script pre-install y post install facilitando el trabajo de los usuarios, esta opción se puede aprovechar a la hora de conectarse con otros sistemas para el manejo de configuraciones. Las respuestas están encabezadas por debian-installer (d-i) proporcionando una guía desatendida al proceso. Al igual que kickstart puede ser exportado y usado en varias variantes de instalación. (debian.org, 2018)

Kickstart

Muchos administradores de sistemas prefieren usar un método de instalación automatizado para instalar Red Hat enterprise linux en sus máquinas. Sin embargo, algunas personas quieren desplegar redes incluyendo otras distribuciones de GNU/Linux. El objetivo del proyecto de compatibilidad es permitir utilizarlo como una capa encima de preseed de debian-installer.

Con kickstart, un administrador del sistema puede crear un solo archivo que contenga las respuestas a todas las preguntas que normalmente se harían durante una instalación típica de Red Hat enterprise linux. Los archivos de kickstart pueden mantenerse en un sistema de servidor único y ser leídos por computadoras individuales durante la instalación. Este método de instalación puede admitir el uso de un solo archivo kickstart para instalar Red Hat enterprise linux o cualquier distribución basada en Red Hat como CentOS en múltiples máquinas, lo que lo hace ideal para administradores de red y sistemas. (Dolezelova, 2017)

Kickstart permite en la mayor parte de una instalación de las distribuciones de Red Hat, automatizar la selección del idioma, configuración de mouse, selección del teclado, instalación del cargador de arranque, partición del disco, configuración de la red, autenticación NIS, LDAP, Kerberos, Hesiod y Samba, configuración del firewall, selección de

paquetes personalizados y la ejecución de script antes y después del proceso de instalación. (Bokok, 2017)

1.3.2 Herramientas de configuración desatendida

Con la evolución de la computación en la nube, la virtualización, los medios de cómputo físicos, la agilización de la metodología de los desarrolladores y la explosión de los datos en la actualidad, aparece la necesidad de manejo de la infraestructura a gran escala. Las herramientas y prácticas para el manejo de configuraciones en la actualidad tienen como objetivo la automatización de todos los estados a gran escala, dinamismo y complejidad de la infraestructura. El manejo de las herramientas de configuración tienen el núcleo en tecnologías como Puppet, Ansible y Chef. (Casad, 2016)

Una vez instalado el sistema operativo en una máquina y configurado con las herramientas anteriormente planteadas, es necesario completar la instalación y configuración según el ambiente de trabajo y las especificaciones deseadas.

El empleo de estas herramientas permite un ahorro considerable de tiempo; sistemas en los que se necesitarían horas para configurarse quedarían listos en minutos. El crecimiento de los servidores de grandes compañías en la actualidad ha sido posible solamente gracias al empleo de estas tecnologías, un solo administrador podría administrar hasta 800 servidores. (Puppet, 2016)

Puppet

Puppet es una herramienta de gestión de la configuración de código abierto. Está escrito en Ruby y fue liberado bajo GPL de GNU hasta la versión 2.7.0 y después bajo la licencia Apache 2.0. Puppet Labs y Puppet fueron fundados por Luke Kanies en el 2005.

Es una herramienta diseñada para administrar la configuración de sistemas similares a Unix y a Microsoft Windows de forma declarativa. El usuario describe los recursos del sistema y sus estados utilizando el lenguaje declarativo que proporciona Puppet. La información es almacenada en archivos denominados manifiestos. Puppet descubre la información del sistema a través de una utilidad llamada Facter, y compila los manifiestos

en un catálogo específico del sistema que contiene los recursos y la dependencia de dichos recursos. Estos catálogos son ejecutados en los sistemas de destino, el esquema de funcionamiento se muestra en la figura 1-4. (Casad, 2016)

Puppet ha sido desarrollado para ayudar a la comunidad de administradores de sistemas a compilar y compartir herramientas maduras que eviten el duplicado de soluciones para resolver el mismo problema. Esto lo logra en dos maneras.

- Provee un poderoso framework para simplificar la mayoría de las tareas técnicas que los administradores de sistemas necesitan realizar.
- El trabajo del administrador de sistemas es escrito en el lenguaje personalizado de Puppet el cual es fácil de compartir como cualquier otro código.

Esto significa que el trabajo de un administrador de sistemas puede ser hecho mucho más rápido, porque puede dejar a Puppet manejar la mayoría o todos los detalles, o puede descargar código de otro administrador de sistemas.

Puppet es típicamente usado en una manera cliente/servidor con todos los clientes hablando a uno o varios servidores centrales. Cada cliente contacta al servidor periódicamente (cada media hora por defecto), descarga la configuración más reciente y se asegura de que está sincronizado con esta configuración. Una vez hecho esto el cliente envía un reporte al servidor indicando si algo necesitó ser cambiado. La figura 1-4 muestra el flujo de datos en una implementación normal de Puppet.

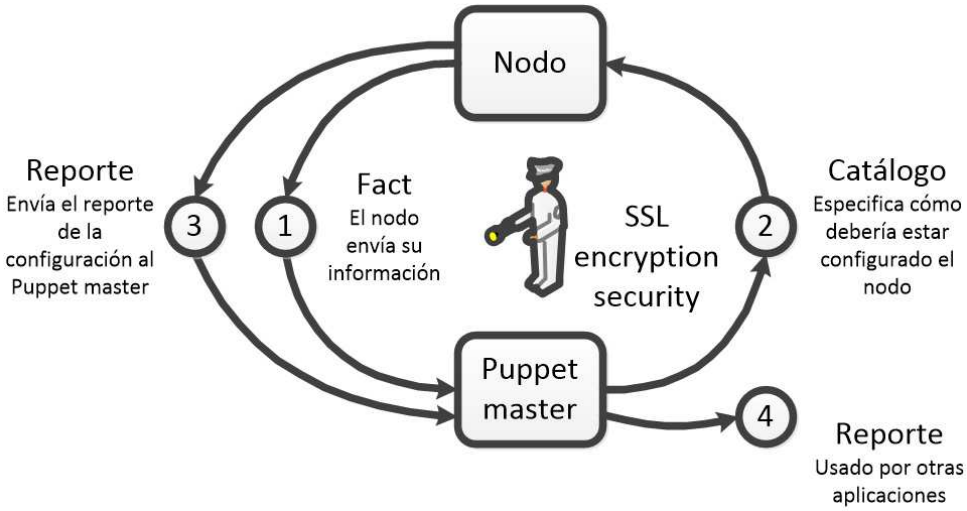

Figure 1–4: Flujo de datos de Puppet. *Fuente. Autor.*

Puppet se compone de un lenguaje declarativo para describir la configuración del sistema, que puede ser aplicado directamente en el sistema o compilado en un catálogo y distribuido al sistema de destino a través del paradigma cliente-servidor (usando una Interfaz de programación de aplicaciones de transferencia de estado representacional REST API) y el agente utiliza proveedores específicos del sistema para aplicar el recurso especificado en los "manifiests". La capa de abstracción de recursos permite a los administradores describir la configuración en términos de alto nivel, tales como usuarios, servicios y paquetes sin necesidad de especificar los comandos del sistema operativo, como son: rpm, yum o apt. La figura 1-5 muestra las capas en el sitema de Puppet.

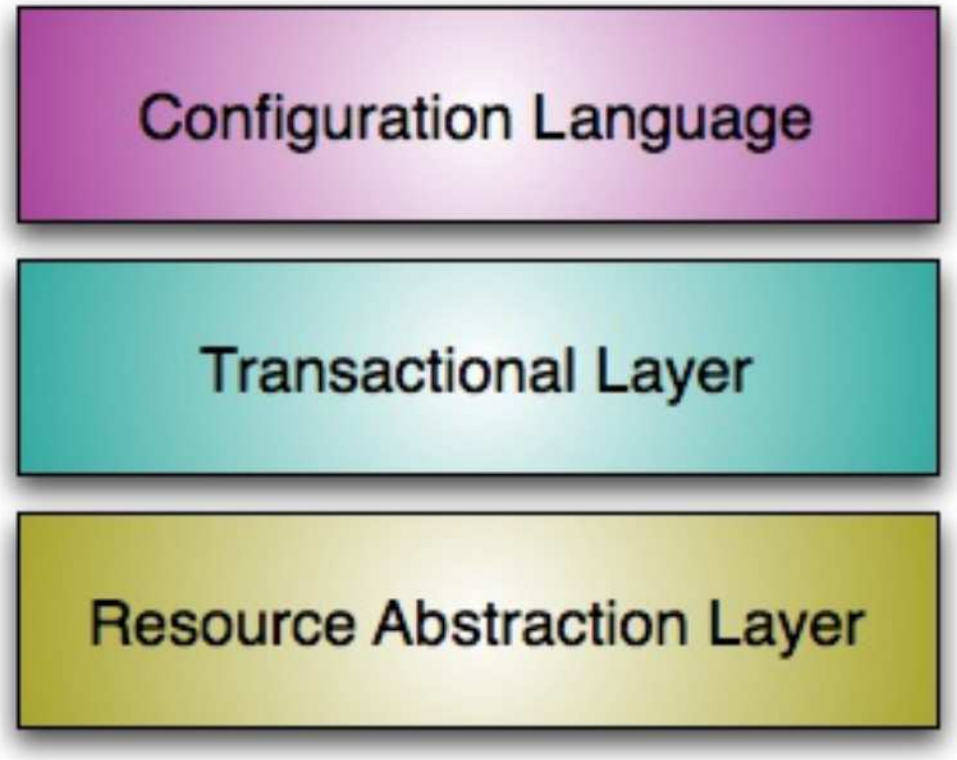

Figure 1–5: Sistema de Puppet. *Fuente. (Puppet, 2018).*

Con Puppet se pueden controlar servicios, trabajo con los archivos, gestión usuarios y grupos, tareas programadas con cron y la ejecución de comandos, Puppet es construido para multiplataforma, funciona en las distribuciones de Linux, incluyendo Red Hat Enterprise Linux (y sus clones como CentOS y Oracle Linux), Fedora, Debian, Mandriva, Ubuntu, y SUSE, así como en múltiples sistemas Unix (Solaris, BSD, Mac OS X, AIX, HP-UX), y cuenta con apoyo para Microsoft Windows. Es utilizado por la Fundación

Wikimedia, Dell, Rackspace, Zynga, Twitter, la Bolsa de Nueva York, Disney, Citrix Systems, Oracle, la Universidad del Norte de Texas, el Laboratorio Nacional de Los álamos, la Universidad Stanford, y Google, entre otros. Existen dos versiones, la Open Source y la versión Enterprise.

Puppet open source es libre para uso, es mayormente usada en pequeñas infraestructuras, no presenta una interfaz web que permite la administración de una forma más fácil, permite interactuar de una forma más fácil los reportes y la respuesta de Facter.

Puppet Enterprise por su parte es usado por una gran comunidad, su uso no es libre, necesita ser pagada. Provee actualizaciones de seguridad para instalación de paquetes de software. Permite el uso de una interfaz gráfica, capaz de gestionar el trabajo de una manera más fácil. (Puppet, 2018)

Ansible

Ansible es una moderna herramienta de servicios automatizados que maneja configuraciones de manera desatendida. Solo precisa definir la configuración en que se está interesado. Ansible se encarga de desde la instalación de un paquete hasta la configuración de una aplicación en el servidor o el reinicio de un servicio. (Praveen and Patawari, 2018)

Ansible es una herramienta sencilla, flexible y extremadamente potente que permite automatizar tareas de infraestructuras comunes, ejecutar comandos ad hoc e implementar aplicaciones que abarcan múltiples máquinas. Se puede usar Ansible para correr varios comandos en varios servidores en paralelo. (Schneller, 2016)

Las tareas son llamadas playbooks y se ejecutan como roles. Ansible está escrito en Python, no necesita agentes para ejecutar las tareas en los clientes, los clientes necesitan tener instalados un interpretador de Python y la comunicación con el cliente se realiza mediante SSH. (Shah, 2016)

Chef

Chef es una plataforma de automatización de sistemas e infraestructuras que facilita el despliegue de servidores y aplicaciones a cualquier localización física, virtual o en la nube, sin importar el tamaño de la infraestructura. Se usa para acelerar el despliegue de aplicaciones. (Taylor and Vargo, 2014)

Salt

Salt es un nuevo sistema de gestión de configuración que ha tenido algún éxito durante los últimos años y ahora es utilizado por conocidas empresas e instituciones. Salt tiene un enfoque más ligero que por ejemplo Chef y Puppet, pero aún utiliza la arquitectura máster-agent. Esto hace de Salt una alternativa interesante en algún lugar entre los mayores sistemas de gestión de configuración. Salt proporciona soporte para las plataformas más comunes, ofrece apoyo comercial y tiene una comunidad activa.

En conclusión Salt es un moderno sistema de gestión de configuración que está bien adaptado para un ambiente cloud. Salt es fácil para aprender y para el uso y su principal ventaja en comparación con los otros es la escalabilidad del sistema.

Cada organización está compuesta por una o más estaciones de trabajo (workstation), y un servidor principal. Cada nodo es configurado y mantenido por el cliente Chef. Los cookbooks y recetas se usan para indicar al cliente Chef cómo se debe configurar cada nodo en la organización.

Las definiciones y recetas reusables contenida en los cookbooks están escritas en el lenguaje de programación Ruby. El despliegue de las infraestructuras descrito en estos cookbooksse encuentra detallado en las diferentes acciones a realizar indicando cómo debe desplegarse, configurarse y manejarse cada parte. Aplicando estas definiciones en el servidor, Chef produce y automatiza la infraestructura. Los cookbooks y recetas están compuestos de bloques denominados recursos.

Muchos de estos recursos están incluidos en Chef y pueden encontrarse en la comunidad Chef. El servidor de Chef almacena los datos de la configuración de red, tanto

la actual como la deseada, y las recetas, y gestiona los nodos que forman la infraestructura. Los datos describen los estados que componen la infraestructura. Las recetas son las instrucciones paso a paso para juntar estos ingredientes y formar un sistema en funcionamiento completo.

El cliente Chef es un programa que ejecuta los cookbooks en nodos de la red, tanto físicos como virtuales o en la nube. Además, se puede usar una estación de trabajo para actualizar el estado del servidor Chef y tener un control de revisiones, allí se crean y modifican los cookbooks almacenadas en un repositorio base. (Casad, 2016)

1.3.3 Orquestación

Foreman

Foreman es un proyecto de código abierto que representa una herramienta de gestión del ciclo de vida para servidores físicos y virtuales. Proporciona a los administradores de sistemas la capacidad de automatizar fácilmente las tareas repetitivas, implementar rápidamente las aplicaciones y administrar proactivamente los servidores, tanto en servidores locales como en la nube. (Portal Díaz, 2017)

Las herramientas de gestión de ciclo de vida son las encargada de manejar el estado en que se tienen que encontrar sus clientes y los servidores que son administrados, pasando por las etapas definidas.

Funcionalidades de Foreman.

- Descubrir, proporcionar y actualizar toda la infraestructura.
- Crear y administrar instancias a través de nubes privadas y públicas.
- Agrupar a los servidores y administrarlos.
- Revisar cambios históricos para auditoría o solución de problemas.
- Ampliar según sea necesario a través de una robusta arquitectura de complemento.

El soporte a los servidores y máquinas se realiza usando herramientas anteriormente mencionadas, como kickstart y preseed, logrando instalar el sistema operativo deseado

mediante un interfaz web, siendo más fácil su empleo, con las tecnologías anteriormente mencionadas. La configuración de los clientes se realiza mediante Puppet, aunque mediante plugins es capaz de emplear Ansible y Chef. El uso Foreman facilita el trabajo con estas herramientas ya que mediante la interfaz es posible adicionar los módulos para cada servidor y eliminar el uso de scripts para declarar. Es capaz de generar reportes y tener la información de servidores y estaciones de trabajo mediante Facter.

Para exponer la arquitectura de Foreman y su *smart proxy* se presenta la figura 1-6. Se presenta un proxy inteligente mediante el cual se controlan los servicios de DHCP, TFTP, DNS, HTTP, Puppet, Puppet CA y al adicionarle funcionalidades mediante plugins controla por SSH la ejecución de comandos en tiempo real. (Benjamin, 2016)

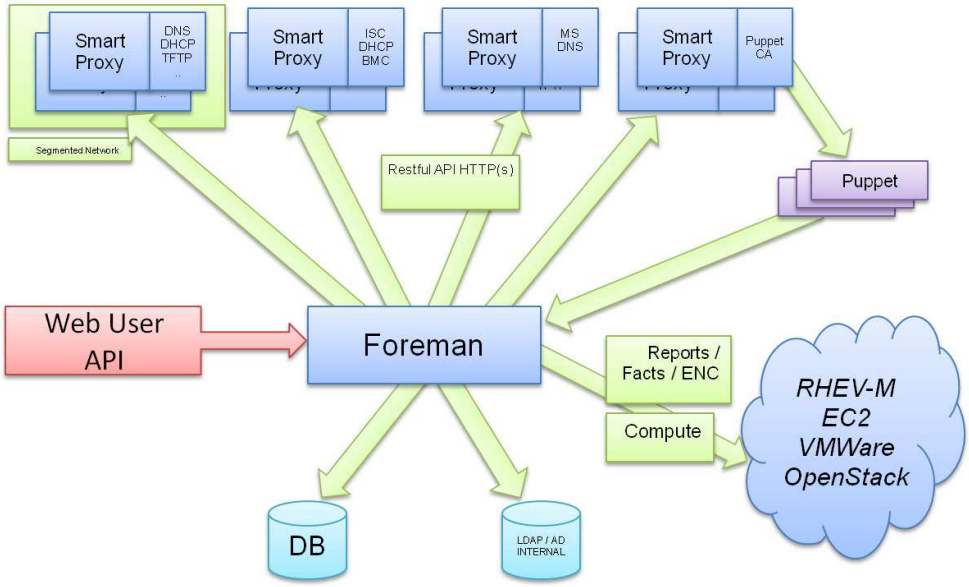

Figure 1–6: Arquitectura de Foreman. *Fuente. (foreman.org, 2018).*

Puppet dashboard

Dentro de las herramientas proporcionadas por el grupo Puppet Labs encontramos Puppet Enterprise, la cual es una plataforma completa de gestión de configuración con un conjunto de componentes optimizados y probados. Combina Puppet, una consola web para analizar los reportes y controlar la infraestructura, potentes características de orquestación, herramientas de aprovisionamiento para la nube y soporte profesional.

Con Puppet Enterprise los administradores de sistemas pueden automatizar tareas repetitivas con facilidad, desplegar rápidamente aplicaciones críticas y controlar proactivamente infraestructuras. (Martín, 2018)

Puppet Dashboard es una aplicación web Rails escrita en Ruby que permite interactuar con Puppet Master y los clientes en este caso los agentes. Puede mostrar y visualizar los reportes de Puppet, asignar las clases y parámetros a los nodos y mostrar el inventario de datos. Está presente en la versión Enterprise de Puppet.

1.4 Conclusiones del capítulo

De este capítulo se puede concluir que existen distintos tipos de clústeres de computadoras que aunque difieren en sus propósitos son similares en cuanto a, por ejemplo, sus componentes. A esto no son ajenos los clústeres relevantes a esta investigación. Además, se pudo observar que se necesita software instalado y configurado en los nodos (maestro y esclavos) para el funcionamiento del HPC. También puede arribarse a la conclusión de que existen herramientas que permiten la automatización del despliegue inicial de un clúster de computadoras, de las que varias son herramientas de software libre de uso viable en instituciones educacionales de nuestro país. Descritos estos elementos, queda explicado el estado del arte y de la práctica respecto al tema de la investigación permitiendo establecer eficazmente el hilo conductor de la misma y proseguir pues, con la selección de las herramientas adecuadas y su modo de uso.

CAPÍTULO 2

2
SELECCIÓN DE LAS HERRAMIENTAS Y DISEÑO DEL SISTEMA

Introducción

Este capítulo abordará los aspectos relacionados (tal como lo indica su título) con la elección de las herramientas adecuadas. Se explicará el por qué de las selecciones hechas brindando los datos que las justifiquen. También se pretende exponer la manera en que estas herramientas se usarían integradas en un sistema que permita automatizar el despliegue del HPC. En este sentido se hará alusión a las funcionalidades que presentan que permiten la integración de unas a otras. Una vez visto esto se expondrá el diseño de la estructura del sistema que se desea implementar y el papel y manera de empleo de las herramientas para estos propósitos. Se expresará así lo que vendría a ser los materiales y métodos a emplear en la solución de la tarea de la investigación.

2.1 Selección de las herramientas

Habiendo hecho una exposición de las herramientas existentes para el despliegue automatizado, se está apto para la selección de las más adecuadas a las precisiones del sistema que se desea desarrollar e implementar para automatizar el despliegue del clúster HPC.

Se había expuesto que *kickstart* había sido pensado para trabajar como una capa encima de *preseed* de debian-installer y permitir la automatización de la instalación de otras distribuciones de Linux, principalmente (aunque no únicamente) Red Hat y derivados. Es innecesario pues establecer comparación alguna e inviable el seleccionar una de ellas.

Expuesto esto, se realizará la selección de las herramientas que sí ameritan un escrutinio al detalle.

2.1.1 Puppet como gestor de configuraciones

Puppet es la herramienta que se seleccionó para la gestión de configuraciones en el sistema que se desea desarrollar e implementar. La selección de Puppet está basada en una sólida revisión de las herramientas expuestas en el capítulo anterior y que se describirá ahora.

Descríbase pues herramienta por herramienta los pros y contras que surgen de su revisión basado en:

1. Idempotencia.
2. Seguridad.
3. Orquestación.
4. Simplicidad.
5. Lenguaje de programación.
6. Reusabilidad del código.
7. Protocolo de mensajes.

Se expondrá así qué tiene para ofrecer cada una de las herramientas.

Salt

Capacidades de Salt.

- Orquestación y automatización para CloudOps.
- Automatización para ITOps.
- Integración continua y despliegue continuo.
- Automatización del flujo de trabajo DevOps soportando Puppet, Docker, Jenkins, Git.

Pros de Salt.

- Eficiente para ambientes de alta escalabilidad.

- Simple uso pasada la parte de la instalación y la configuración.
- Fuerte instrospección.
- Sintaxis YAML consistente y rica en características.

Contras de Salt.

- El proceso de instalación puede no ser fácil para nuevos usuarios.
- La interfaz web provee capacidades y características limitadas.
- No es la mejor opción para sistemas operativos que no sean Linux.
- La plataforma es nueva y no enteramente madura comparada a Puppet.

Ansible

Capacidades de Ansible.

- Manejo de configuraciones.
- Despliegue de aplicaciones.
- Entrega continua.
- Seguridad en los procesos automatizados.
- Orquestación simplificada.

Pros de Ansible.

- Ejecución remota fácil.
- Comparte datos entre múltiples servidores.
- Motor de orquestación poderoso.
- Sintaxis fácil de aprender para nuevos usuarios.
- Orden de ejecución secuencial.
- Seguridad por SSH.

Contras de Ansible.

- Los *playbooks* y las plantillas pueden variar.
- Interfaz no desarrollada.
- La plataforma es nueva y no enteramente madura comparada a Puppet.

Chef

Capacidades de Chef.

- Automatización de la infraestructura.
- Automatización para el flujo de trabajo DevOps.
- Manejo de la seguridad.
- Automatización del flujo de trabajo para entrega continua.

Pros de Chef.

- Diseñado para programadores.
- Fuerte documentación, soporte y contribuciones de una comunidad activa.
- Maduro, estable y fiable en ambientes públicos y privados.
- Orden de ejecución secuencial.

Contras de Chef.

- Curva de aprendizaje difícil.
- Complicada configuración inicial.

Puppet

Capacidades de Puppet.

- Orquestación.
- Provisionamiento automático.
- Automatización de la configuración.
- Visualización y reporte.
- Manejo del código.
- Manejo de los nodos.

Pros de Puppet.

- Fuertes herramientas de automatización y reporte.
- Soporte para herramientas de desarrollo y *cookbbooks*.
- Interfaz intuitiva para tareas diversas.

- Trabajo a nivel del shell del sistema operativo robusto.
- Simple configuración inicial y soporte para varios sistemas operativos.
- Particularmente fiable para empresas.

Contras de Puppet.

- Usar múltiples masters puede complicar la gestión.
- El soporte está enfocado al Puppet DSL.
- Mejor para la automatización del manejo del sistema.

(Bagul, 2016)

Estas son todas buenas alternativas que proporcionan la funcionalidad que se está buscando, pero sus enfoques diferencian un poco uno de otro. Salt y Ansible son sistemas más ligeros escritos en Python. Ambos son fáciles para aprender y usar mientras Chef y Puppet son sistemas más complejos. Chef y Puppet están escritos en Ruby. Si un sistema complejo con muchas funcionalidades debe ser implementado Puppet y Chef son los candidatos más interesantes.

Si el usuario está familiarizado con programación Ruby, Chef o Puppet podrían ser una mejor opción. Chef puede proporcionar un poco más de flexibilidad que Puppet, pero por otra parte Chef es puro Ruby lo que puede ser duro para la gente sin experiencia en Ruby. Puppet proporciona un DSL que es lógico y fácil para aprender y por lo tanto podría ser más interesante para algunos usuarios.

Ambos Ansible y Salt son más fáciles de usar y deben ser elegidos si lo más importante es poder administrar un sistema en una forma sencilla. En contraste con Salt, Ansible no utiliza ningún tipo de agentes, lo que lo hace más rápido y más fácil para instalar. Otra diferencia es que Ansible GUI es más desarrollado. Sin embargo, Salt es más escalable y soporta Windows.

Entre los más complejos y profesionales se puede decir que Puppet generalmente es la mejor opción. Esto se debe a que proporciona una solución muy capaz con una GUI madura y un lenguaje que es más fácil de aprender y entender que el de Chef. Entre

los más ligeros se recomienda Salt, ya que es más escalable que Ansible. También tiene soporte para Windows, lo que lo hace útil en más situaciones. Si sólo se puede elegir un sistema se recomendaría Puppet, ya que es el sistema más completo y puede ser utilizado en la más amplia gama de situaciones, sin ser demasiado duro para aprender y mejorar. (Torberntsson and Rydin, 2014)

2.1.2 Orquestación con Foreman

Desde el momento en punto en que Puppet Dashboard (la interfaz gráfica para Puppet) se distribuye junto a Puppet Enterprise que es la versión de pago de Puppet, se tiene que no es la variante más adecuada si lo que se desea es un sistema basado en herramientas de software libre para el uso en las instituciones académicas de nuestro país. Sin embargo, la elección de Foreman no se basa solo en un trasfondo económico, sino en sus virtudes que ahora se tratarán de expresar.

Para ello conviene hacerse una idea de qúe hace Foreman. Ello se reduciría a:

1. Aprovisionamiento.
2. Configuración.
3. Monitoreo.

Para el aprovisionamiento via red, señala el servidor por el que ha de iniciar y crea la configuración apropiada para el PXE mediante el Smart Proxy (que se exponía en el capítulo anterior) para instalar el sistema operativo. Los contenidos del inicio via red se preparan via plantillas que se personalizan antes de la instalación. Se apoya en scripts de terceros para los medios de instalación teniendo plantillas disponibles para Jumpstart, Kickstart, Preseed y AutoYast.

Cuando se crea un sistema operativo via Foreman, ello requiere detalles que son entonces utilizados en las plantillas. Ello incluye tamaño de disco, procesadores, memorias, red, clases de Puppet, grupos de hosts, variables y arquitectura.

En cuanto a la configuración Foreman provee una solución completa de gestión de la configuración incluyendo un clasificador de nodos externo (*ENC* por sus siglas en

inglés) para Puppet, soporte para clases parametrizadas y almacenamiento de parámetros jerárquicos. Foreman almacena variables, parámetros, metadatos, clases y datos. Las clases parametrizadas controlan cómo se comportan las clases para uno o varios sistemas. Los grupos de hosts agrupan sistemas y aplican clases y parámetros al grupo.

El monitoreo de Foreman se basa en recolectar reportes y datos de Puppet para así monitorear la configuración de los hosts. Reporta estados, distribuciones y tendencias. Si los hosts son manejados por Puppet se puede saber, por ejemplo si Puppet ha corrido recientemente. Además posee una interfaz que integra una *dashboard* que nada tiene que envidiar a la de Puppet Dashboard. Ello ayuda a gestionar las estadísticas.

Podemos resumir que las características de Foreman que suponen fortalezas son:

- Integración con Puppet y sus herramientas de reportes.
- Interfaz.
- API y CLI robustas.
- Manejo de DHCP, DNS, TFTP y PXE.
- PuppetCA.
- Roles, usuarios y LDAP.
- Datos de Foreman accesibles desde los maifiestos de Puppet eliminando la necesidad de Puppetdb.

(Oglive, 2016)

2.1.3 Sistema de control de versiones

El objetivo del sistema de control de versiones es facilitar la distribución y mantener el código de manera que sea de fácil mantenimiento además de hacer al equipo de trabajo más eficiente. El problema es que los sistemas de control de versiones como Git son geniales para ello pero nada intuitivos y agobiantes para el usuario inexperto.

Gitlab y R10k

Gitlab es un sistema de manejo de repositorios Git. Está escrito en Ruby y permite desplegar fácilmente un significativo control de versiones.

Gitlab tiene varias características. Las más relevantes seguramente son su intuitiva interfaz y su posibilidad de manejar los permisos. Las características de documentación y de seguimiento de cambios necesarios son puntos fuertes de Gitlab.

La interfaz semeja la de Github que ha probado ser funcional para varios desarrolladores. Es consistente y plana a pesar de las funcionalidades que tiene. Se pueden filtrar solicitudes de subida, ver diferencias e incluso editar archivos utilizando la propia interfaz.

Gitlab posee roles muy bien diseñados para manejar quién tiene permiso para hacer qué. Van desde invitado hasta master y permiten limitar u otorgar permisos.

Gitlab es especialmente relevante porque provee una vía para trabajar sobre Git flexible para los administradores y de usabilidad para los usuarios. (Hethey, 2013)

R10k es una utilidad de manejo del código que permite hacer ambientes Puppet a partir de ramas de Gitlab. Si se hace una rama *testing* se hará el ambiente Puppet *testing*.

La versión Enterprise de Puppet presenta una herramienta que maneja R10k. Puppet code manager. Para manejar R10k se puede usar, sin embargo, scripts de gitlab-ci.

Gitlab-ci es probablemente la característica de Gitlab de la que se obtendrán mayores beneficios para la investigación. Es un sistema de *integración continua / despliegue continuo* (*ci / cd* por sus siglas en inglés) que viene con Gitlab para probar, compilar y distribuir código fuente. Integración continua es una práctica de desarrollo que permite integrar código a un proyecto frecuentemente. Las entradas se prueban mediante un proceso automático de compilación. Despliegue continuo es lo mismo pero enfocado a la entrega de software al usuario. (Nolin, 2016)

2.2 Uso integrado de las herramientas

2.2.1 Integración Foreman-Puppet

Foreman coopera con Puppet que proporciona la configuración deseada a los nodos. Los entornos Puppet se asignan directamente en Foreman. Pueden ser utilizados en los distintos niveles a través de la interfaz Foreman. Mediante Puppet se separan las clases para diferentes tipos de host, lo que se puede desarrollar o probar en un ambiente (desarrollo) antes de ser subido a producción, donde puede utilizar para diferentes configuraciones.

Puppet puede crear clases en Foreman por dos métodos. Una forma es que Foreman sí ofrece una posibilidad de crear un nuevo ambiente y también las clases Puppet. La otra forma es que Foreman puede detectar todos los entornos y clases Puppet contenidas en el Puppet master e importarlos automáticamente al sistema. Las clases y entornos importados o exportados pueden ser entonces asignados a hosts o un grupo de hosts (hostgroup) provisionados por Foreman.

Puppet también proporciona una opción de variables inteligentes. Las variables inteligentes son una herramienta que puede proporcionar lógica adicional a las clases Puppet que un usuario puede desear aplicar. Pueden tener varios valores, dependiendo de la jerarquía. Un ejemplo de usar variables inteligentes puede ser un simple cambio a un solo host. (Hadvig, 2013)

El manejo de configuraciones desde Foreman se puede realizar de una forma más fácil y amena para los administradores de sistemas, usando Puppet por defecto.

Los roles y los catálogos escritos según la necesidad y las especificaciones de la infraestructura deben ser ubicados en el directorio indicado en las propiedades de Foreman de cada una de las herramientas que se van a usar. En la interfaz de Foreman se puede observar el inventario de los scripts existentes en el servidor y la asignación de las clases que deben correr los servidores o en otras palabras, el estado en que se quiere que se encuentren los nodos. Siempre queda la opción de variar los parámetros de algunas variables en caso de no ser los adecuados para adaptarlas según el ambiente de trabajo o el servidor.

El trabajo con Puppet se facilita ya que el Smart Proxy de Foreman es capaz de manejar los certificados con PuppetCA, facilitando el trabajo desde la interfaz a la hora de tener el inventario de los certificados existentes y el control de estos a la hora de firmarlos, revocarlos o eliminarlos, librando al administrador del trabajo con los comandos.

2.2.2 El Smart Proxy de Foreman

Smart proxy es un componente independiente que proporciona varios servicios a varios subsistemas. Su principal propósito es proporcionar API para herramientas como Foreman, con un nivel más alto de orquestación, así que el nuevo subsistema que se desea añadir al sistema, podría ser agregado, o el subsistema existente podría ser extendido.

Debido a los servicios de Smart Proxy, es necesario al menos uno de estos Smart Proxy por una subred porque cada subred contiene su propia dirección de red, que debe ser manejada por DHCP, DNS, tftp y Puppet.

Cada uno de estos servicios pueden existir en una máquina separada o varios de ellos pueden ser alojados en la misma máquina. Foreman automáticamente provee de los servicios que detecte. (Hadvig, 2013)

Para un mejor entendimiento del flujo de trabajo de Foreman y una vez explicado el Smart Proxy para entender los servicios que dependen de él se presenta la figura 2-1. Se muestra además el orden que le da Foreman a los servicios que realiza para el aprovisionamiento, la configuración y el monitoreo.

2.3 Diseño del sistema

Se puede plantear la estructura del sistema a desarrollar e implementar habiendo visto las herramientas que lo conformarán. Dicha estructura debe permitir la escalabilidad del sistema así como la integración continua de los cambios sobre los *scripts* de configuración.

La figura 2-2 permite apreciar la estructura general que se plantea para el sistema a desarrollar e implementar que luego se desglosará para exponer los elementos que lo

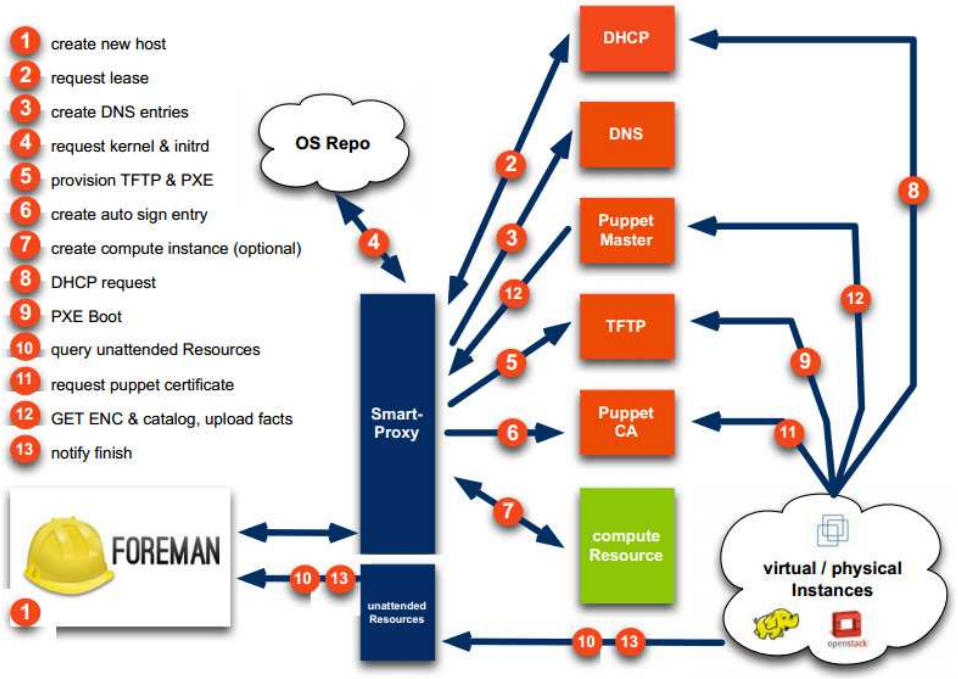

Figure 2–1: Flujo de trabajo de Foreman. *Fuente.* (Domrose, 2014).

integran y dentro de ellos los detalles relevantes a la presente investigación y que posibilitan el cumplimiento de los objetivos de la misma.

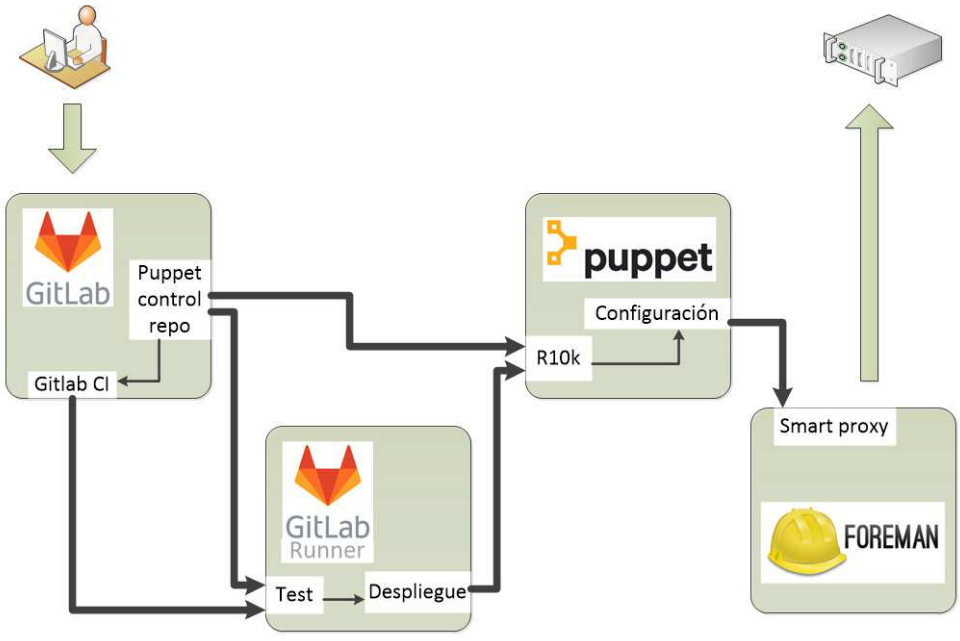

Figure 2–2: Estructura general del sistema. *Fuente. Autor.*

Puppet no está pensado para la instalación desatendida de sistema operativo sino para el despliegue de nuevos paquetes, configuración de los mismos, gestión de cuentas de

usuario y en general para el manejo de configuraciones de forma automatizada una vez instalado el sistema operativo.

Sin embargo, es necesario instalar un sistema operativo de manera desatendida antes de configurar paquete alguno o usuario alguno. De hecho se había hecho mención de los ficheros kickstart y preseed que se usarían para ello. La vía para orquestarlo todo la proporcionará Foreman. Se expuso que una de las funcionalidades de Foreman era el aprovisionamiento. Para ello Foreman permite la instalación desatendida (sobre estaciones que puede descubrir automáticamente) de sistemas operativos basado en ficheros kickstart y preseed, entre otros.

El Puppet Master es el encargado del manejo de configuraciones de todos los clientes dependiendo del servidor o del grupo de servidores al que pertenezcan. La comunicación con los clientes se realiza instalando puppet-agent en los nodos que se desean administrar. La configuración es definida en el Puppet Master, compilada, y luego enviada a los clientes de Puppet cuando estos se conectan.

La comunicación es encriptada y autenticada mediante el uso de certificados SSL autofirmado, que son validados y autorizados por el mismo Puppet Master. Se debe tener en cuenta que existen casos en los que el firmando de certificados se puede declarar como un proceso automático, aceptando a todos los clientes dentro de reglas establecidas, como una subred.

Cada nodo (mediante el puppet agent) le envía la información correspondiente de su estado haciendo uso de la herramienta *facter* al servidor master, este analiza la información y compara cuál sería el estado en que tendrían que estar cada nodo, si no está como se desea, el master manda el catálogo correspondiente a ese nodo para que el puppet agent lo ejecute y luego de esto es enviado el reporte al master para que sea analizado. Estos reportes y las estadísticas que derivan de ellos se podrán observar a través de Foreman que permite la visualización de datos provenientes de *facter*.

De forma general se puede decir que con Puppet se logra definir una serie de módulos encargados de desplegar las tareas en los nodos. Estos módulos pueden instalar rápidamente gestor de colas, gestor de paquetes y sistema de archivos, y lo más importante, los puppet agent garantizarán que esto no cambie a través del tiempo manteniendo continuamente la configuración en los clientes que los administradores planificaron desde el servidor.

Estos scripts de configuración que serán usados por Puppet se almacenan a su vez en un repositorio de GitLab. Esto permite tener un control de versiones sobre los scripts de configuración y probar automáticamente los cambios que se realicen sobre los mismos para ponerlos en producción una vez estén bien también de manera automática.

Para la gestión del HPC este repositorio de control de Puppet tendría una estructura en su rama en producción una estructura similar a la de la figura 2-3. El fuente algunos de estos scripts será expuesto y explicado en el Capítulo 3.

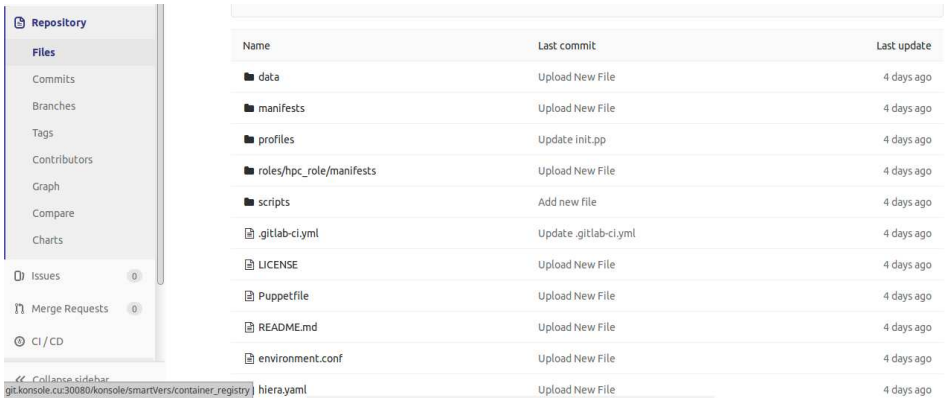

Figure 2–3: Estructura del repositorio de control. *Fuente. Autor.*

La figura 2-4 muestra el algoritmo para las pruebas que se ejecutarán cuando un administrador hace un push al repositorio de control antes de que los cambios sean puestos en producción, o sea, antes de que los cambios sean puestos a disposición de Puppet para la configuración.

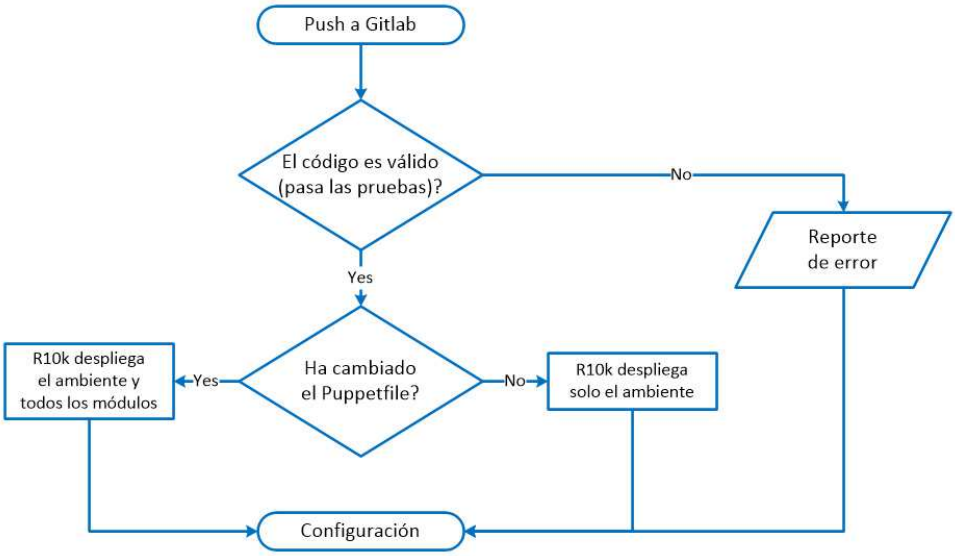

Figure 2–4: Algoritmo de pruebas. *Fuente. Autor.*

2.4 Conclusiones del capítulo

En este capítulo se expusieron los datos que resultaron del estudio de las herramientas existentes para la instalación y configuración desatendida planteando de manera sintética las características, contras y pros de cada una. También se expusieron las bondades que supondría el uso de Gitlab y sus herramientas de integración continua para la automatización de las pruebas y el despliegue de cada cambio que se ejecute en el repositorio de control de Puppet. Se vio que Foreman posee una integración con Puppet que permite que sea el orquestador del sistema. Se presentó una estructura para el sistema en general definiendo el papel de cada herramienta. Es posible pues desarrollar e implementar el sistema diseñado.

CAPÍTULO 3

3
DESARROLLO E IMPLEMENTACIÓN DEL SISTEMA

Introducción

Este capítulo abordará el desarrollo del sistema que se diseñó en el capítulo anterior. Para ello transitará por las funciones que tendrá Foreman dentro del sistema y la manera en que se usará. Se analizará el repositorio de control de Puppet y sus *scripts* más relevantes. Se plantearán las pruebas de integración a ejecutar en el repositorio de control y la manera de realizarlas. También se analizará la implementación del sistema y los resultados derivados de la misma.

3.1 Desarrollo del sistema

3.1.1 Foreman

Foreman es la principal herramienta para proveer un front-end para Puppet pero además es el orquestador que organizará la instalación y configuración desatendida.

Foreman es el principal competidor de Puppet Enterprise (PE). Pero Foreman ha conseguido últimamente más atención en comparación con PE debido a su costo de propiedad y lanzamientos más rápidos. Debe decirse que Foreman pretende hacer algo más que la consola PE. Foreman tiene la disposición de configurar nuevas máquinas virtuales en OpenStack, Google Compute Engine, Rackspace, AWS, VMware, entre otros.

La gestión con el front-end hace que el trabajo se facilite en gran medida, teniendo en un solo ambiente de trabajo varias tecnologías capaces de administrar grandes cantidades de servidores. Con Foreman se puede gestionar la instalación desatendida por la red,

es decir, proveer nuevos servidores, el manejo de configuraciones usando Puppet, el descubrimiento de nuevos servidores mediante una herramienta propia del Foreman, reportes y estadísticas, y la integración con sistemas de autentificación LDAP.

El aprovisionamiento del servidor Foreman se puede realizar partiendo de la misma filosofía de la instalación desatendida, siendo capaz de configurar de forma automática todos los servicios de red necesarios TFTP, DNS, DHCP, NFS y HTTP mediante el Smart Proxy de Foreman.

Una vez instalado Foreman en el servidor y existir conexión con la red, es necesario acceder a pestaña de Configuración y en la opción de Smart Proxy chequear el funcionamiento de los servicios de red instalados por defectos, necesarios para proveer los nuevos servidores, si no existe problema se puede continuar con la configuración del servidor, y en caso contrario chequear los logs de fallos y realizar los arreglos pertinentes.

Primeramente, es necesario realizar una configuración básica de los servicios necesarios para poder instalar los sistemas operativos por la red, como son principalmente DHCP, TFTP y los de PXEe iPXE. Esta opción se encuentra en la sección Infraestructura/Configuración de Provisión. Esta opción es capaz de determinar la presencia de algún error en las conexiones del servidor, en caso de no existir problemas, se continúa con el llenado de campos de configuración de los servicios necesarios y al concluir Foreman a partir de los elementos completados genera un script que debe ser copiado y ejecutado en la terminal del servidor. Una vez terminado el proceso se tiene instalado un servidor capaz de instalar sistemas operativos por la red, para hacer uso en la subred donde se encuentre instalado. Seguidamente es necesario la inclusión de algún sistema operativo al sistema por lo que deben ser llenado los campos necesarios para su configuración, así como la asignación de las plantillas PXE, de configuración, las de arranque local, preseed, kickstart y los medios de instalación.

A groso modo el aprovisionamiento de la red prepara los servicios destinados a la instalación de los sistemas operativos completamente y en un primer momento es capaz

de configurar hasta una imagen correspondiente con el sistema operativo del servidor Foreman con sus respectivas plantillas.

Se debe precisar que existen las plantillas para generar automáticamente el archivo específico para el servidor que se desee instalar, tanto por arranque por BIOS o por UEFI, nombrado con la MAC del mismo. Estas plantillas usan terminologías kickstart o preseed dependiendo de la distribución que se desea instalar (recordar que la primera corresponde a la distribución Red Hat y a Ubuntu, y la segunda a Debian). Existen las plantillas pxelinux, que se generan automáticamente cuando se crea un nuevo servidor, la cual carga la imagen indicada; las plantillas de aprovisionamiento para responder al proceso de instalación y lograr el proceso de forma desatendida y la plantilla de finalizar que se genera una vez que se instala completamente el servidor, permitiendo iniciar normalmente el servidor haciendo uso del disco local. Además se puede ajustar la configuración de la partición del disco. Al crear las plantillas es necesario asociarlas a los sistemas operativos creados, para su uso posterior.

Descubrimiento

El descubrimiento es un complemento que le permite a Foreman descubrir automáticamente servidores o máquinas desconocidos en la red de aprovisionamiento. Los nuevos nodos se auto-registran en Foreman y cargan los datos recogidos por *Facter* (identificador de serie, interfaces de red, memoria, discos). Los nodos registrados aparecen en la página Hosts Descubiertos y el aprovisionamiento puede iniciarse manualmente o automáticamente a través de reglas de descubrimiento predefinidas.

Foreman Discovery se basa en interceptar el proceso de arranque normal para máquinas no registradas en Foreman. Para lograr esto, el archivo default.cfg de PXE necesita ser modificado para instruir a las nuevas máquinas a iniciar la imagen de descubrimiento. (foreman.org, 2018)

De esta forma cada vez que se inicie un servidor que se desee proveer y no se encuentre dentro de los clientes creados en Foreman, se incorpora al sistema mediante la imagen de

descubrimiento de Foreman. Desde la interfaz gráfica se puede configurar el sistema operativo y las plantillas necesarias para la instalación desatendida, determinar los módulos con Puppet, para el manejo de configuraciones, logrando un sistema completamente desatendido.

Se selecciona un servidor descubierto y se elige Provisión. Esto redirecciona a la página de edición para crear un nuevo servidor, con los datos descubiertos rellenados donde sea posible. Se llenan los campos y se crean los ficheros necesarios para proveer al nuevo servidor. Al guardar la configuración, Foreman modifica el archivo pxelinux correspondiente al servidor en la raíz del servicio TFTP y reinicia el servidor descubierto automáticamente, por lo que al arrancar puede cargar la imagen del sistema operativo deseada de forma desatendida, luego el mismo Foreman cambia el fichero de arranque para que inicie por el disco y no haga uso de PXE.

Aprovisionamiento

Una vez Foreman descubre nodos en la subred para ser instalados con él los lista como nodos descubiertos. Pueden pues gestionarse dichos nodos descubiertos desde la interfaz de Foreman.

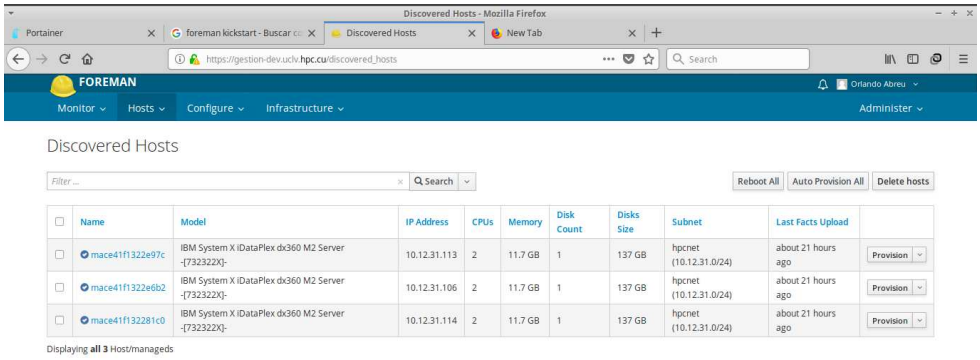

Figure 3–1: Nodos descubiertos en Foreman. *Fuente. Autor.*

Puede pues hacerse que el nodo herede de alguna plantilla de configuración declarada previamente y luego personalizar la opciones pertinentes para el aprovisionamiento del

nodo. Para esta investigación es pertinente, por ejemplo modificar *nombre del nodo* y *dirección ip*.

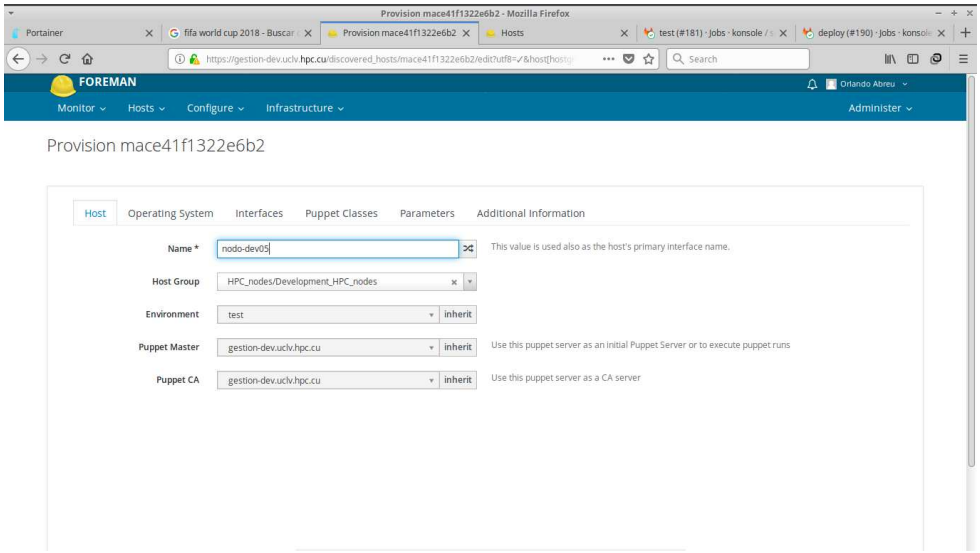

Figure 3–2: Interfaz de configuración de aprovisionamiento. *Fuente. Autor.*

Foreman se encarga de la comunicación con el repositorio de control de Puppet para aplicar los manifiestos y esperar los reportes de configuración. Una guía completa de la configuración del aprovisionamiento para agregar nuevos nodos al clúster se encontrará en el Anexo 1.

3.1.2 Repositorio de control de Puppet

Se ha mencionado que Puppet ha de poseer un repositorio de control que contenga los manifiestos pertinentes para la gestión de las configuraciones en el sistema. Este repositorio presentaría una estructura como la mostrada en la figura 2-3. Puede pues relizarse un análisis sobre las partes relevantes de este repositorio de control y sus scripts.

En el directorio *data* se ubican los datos que *Hiera* se encargará de pasar a los manifiestos. de los nodos tanto el *master* como los *esclavos*. El siguiente fragmento muestra el código que define dos variables para el nodo master.

```
1 slurm_profile::master: true
2 nfs::server: true
```

El directorio *manifests* contiene los manifiestos que deciden los roles a aplicar a los distintos nodos atendidos por Puppet. Mientras, los *profiles*, que están en el directorio

de igual nombre, definen las características de configuración básicas para la instalación y configuración de los *softwares*. Una parte del fuente del manifiesto del *slurm profile* (perfil que maneja la configuración de *slurm*) se muestra ahora.

```
class slurm_profile (
  Boolean $master = false,
  Boolean $node = false,
  String $cluster_name = 'mambi',
  String $master_name = 'master',
){

  class { '::nfs':;}

  class {'::slurm_profile::ldap_sync':;}

  class {'slurm':
    with_slurmctld     => $master,
    with_slurmdbd      => $master,
    with_slurmd        => $node,
    version            => '17.02.10',
    clustername        => $cluster_name,
    controlmachine     => $master_name,
    selecttype_params  => ['CR_Core_Memory'],
    nodes              => {'nodo001' => 'NodeAddr=10.12.31.101 Sockets=2
        CoresPerSocket=4 ThreadsPerCore=1'},
    partitions         => {'public' => {nodes => 'nodo00[1-2] Default=YES
        MaxTime=INFINITE', state => 'UP'}},
  }
}
```

Los *roles* dicen qué instalar en un nodo en que están definidos. Se puede encontrar un directorio con un manifiesto para cada rol en el directorio *roles*.

3.1.3 Integración con Gitlab

Para probar el fuente que se agrega o modifica en el repositorio de control se utiliza la herramienta *gitlab-ci* de Gitlab. Solo se necesita agregar al repositorio de control un archivo *.gitlab-ci.yml* para definir las pruebas a ejecutar. Es necesario además configurar un *runner* que no es más que una aplicación para correr pruebas que para los propósitos de esta investigación utilizará el ejecutor de *docker*. Esto plantea una ventaja dado que el uso del ejecutor de *docker* garantiza estar utilizando en cada prueba un ambiente nuevo y no afectado por los cambios que pudieran relizar pruebas anteriores.

Esto plantea la necesidad pues de hacer una imagen de *docker* para que el *runner* la descargue y cree a partir de ella un contenedor donde se ejecutarán las pruebas. Esta imagen es un ubuntu base donde se instalará una herramienta de chequeo de sintaxis de Puppet nombrada *puppet-lint* y una herramienta de gestión de código Puppet nombrada *r10k* para el despliegue de los ambientes de Puppet en el servidor de Foreman. El fichero Dockerfile para generar dicha imagen se puede ver en el Anexo 3.

Puesto que es necesario especificar las pruebas a ejecutar, estas se declaran en el fichero *.gitlab-ci.yml*

```
1  image: git.uclv.edu.cu:5005/hpcuclv/puppet-controlrepo/puppet-r10k-poli:
       v2
2
3  stages:
4      - test
5      - deploy
6
7  test:
8      stage: test
9      script:
10         - puppet-lint .
11     only:
12         - branches
13
```

```
14 deploy:
15     stage: deploy
16     script:
17         - cp ./scripts/r10k.yaml /etc/puppetlabs/r10k/r10k.yaml
18         - cat /etc/puppetlabs/r10k/r10k.yaml
19         - echo " "
20         - r10k deploy environment -v
21     only:
22         - branches
```

3.2 Implementación del sistema

3.2.1 Integración continua ante modificaciones en el repositorio

Puesto que se encuentra definido el *script* de *gitlab-ci* al modificar el repositorio se ejecutan automáticamente las pruebas previstas. Estas figuras (3-3 y 3-4) muestran los resultados de las pruebas *test* y *deploy* respectivamente.

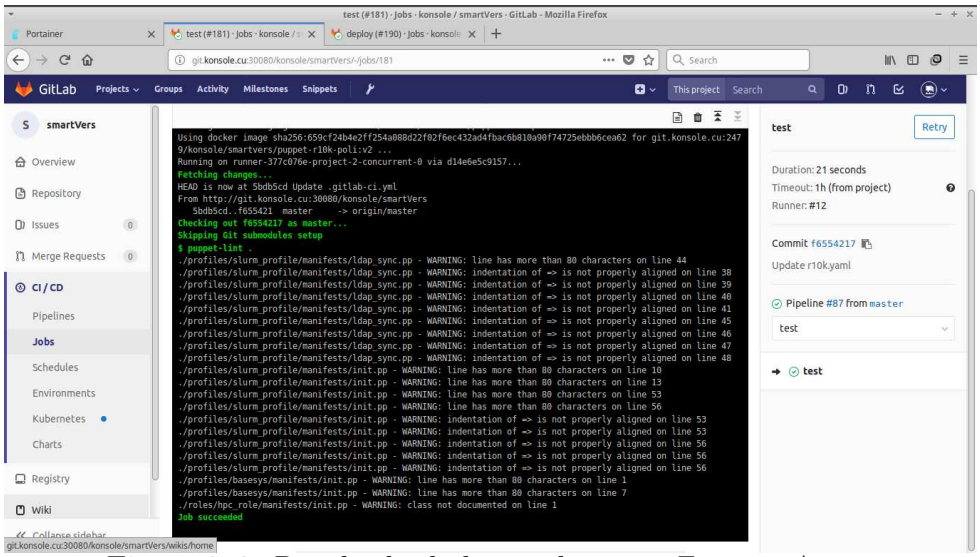

Figure 3–3: Resultado de la prueba *test*. *Fuente*. *Autor*.

3.2.2 Despliegue en nodos de prueba

Este sistema se implementó en nodos de prueba para verfiicar la eficiencia del mismo. Esta implementación se llevó a cabo de forma exitosa. Los primeros ejemplos de su implementación en sentido general son precisamente las pruebas de integración anteriormente referidas que son el primer resultado tangible de la presente investigación.

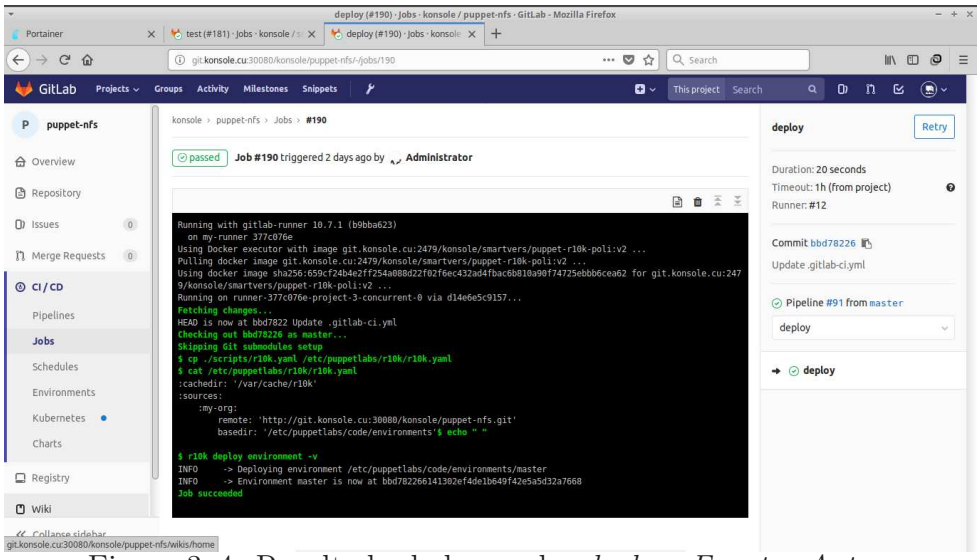

Figure 3–4: Resultado de la prueba *deploy*. *Fuente. Autor.*

Después de desplegados los ambientes de Puppet de manera automática mediante *gitlab-ci* se procedió al uso de dichos ambientes para la puesta en funcionamiento de nodos de prueba a través de Foreman de manera exitosa. Así se puede apreciar en la figura 3-5.

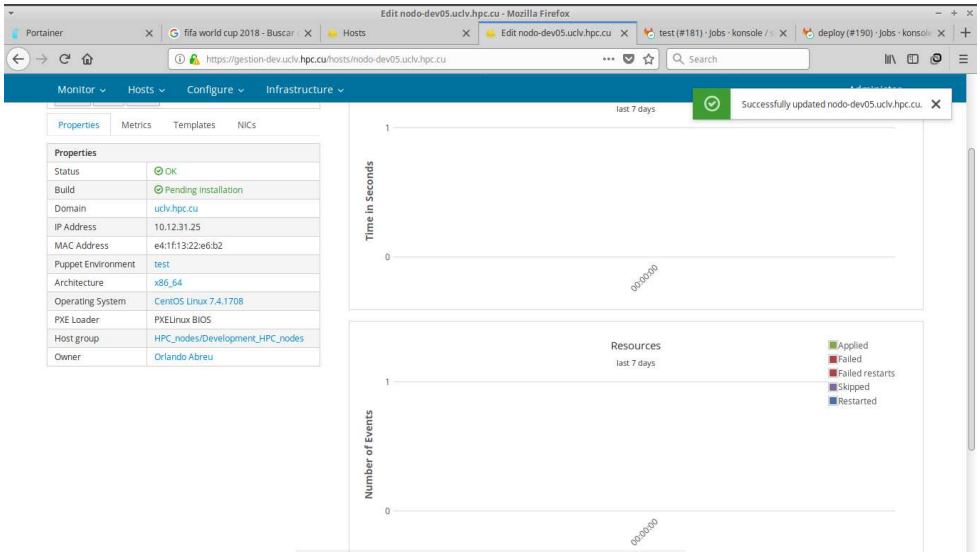

Figure 3–5: Uso exitoso de los ambientes desplegados. *Fuente. Autor.*

La figura 3-6 muestra el antes y el después de la pestaña de *Hosts* de la interfaz gráfica de Foreman. Se puede evidenciar que se agregaron nodos que recibieron su configuración haciendo uso de Puppet.

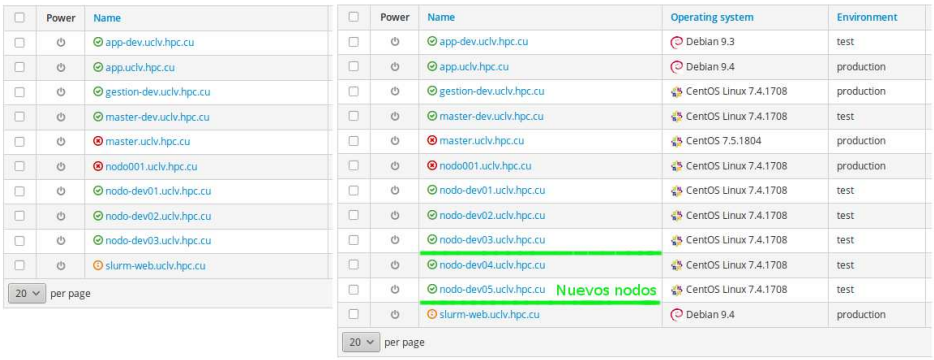

Figure 3–6: Nuevos nodos. *Fuente. Autor.*

3.2.3 Resultados de la implementación

Este sistema tiene varias implicaciones tras su uso. Ejemplo de ello es que es fácilmente utilizable por una sola persona. También es posible que aún una sola persona ejecute la puesta apunto de 100 nodos en 21 minutos visto que 2 nodos son desplegados en 26 segundos. Es posible agregar nuevos nodos aprovechando las bondades de descubrimiento de Foreman garantizando un sistema escalable. Es posible modificar el repositorio de control ejecutando pruebas sobre los cambios y desplegando los ambientes si las pasan las pruebas. Momento a momento se puede tener información de la configuración de los nodos gracias a la integración de Foreman con Puppet y las herramientas de monitoreo de este.

3.3 Conclusiones del capítulo

Este capítulo, al traste con el objetivo general de la investigación, abordó el desarrollo de un sistema que permitiese la puesta a punto de un clúster de manera automatizada y garantizando la fiabilidad y la escalabilidad. Se analizó el papel de Foreman como orquestador. Se analizó el repositorio de control de Puppet y se plantearon las pruebas de integración a ejecutar en él. También se analizó la implementación del sistema y los resultados positivos derivados de la misma.

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

Mientras que Puppet se muestra como la opción de mejores prestaciones para la gestión de la configuración, Foreman se propone como la mejor opción para el aprovisionamiento y la orquestación, así como para visualizar los reportes de Puppet.

Las posibilidades de descubrimiento de Foreman y la gestión del código Puppet mediante Gitlab y r10k, permiten la escalabilidad del sistema. Mientras, las pruebas de integración de Gitlab hacen que las actualizaciones al repositorio de control sean chequeadas garantizando así la fiabilidad.

Del desarrollo y la implementación del sistema se desprende una visible mejora en la escalabilidad, la fiabilidad y la rapidez, dado que pueden ser fácilmente instalados y configurados por una sola persona, 100 nodos libres de errores en apenas 26 minutos.

RECOMENDACIONES

Desarrollar algoritmos más eficientes y abarcadores para pruebas de integración con Gitlab.

Extender el sistema para la instalación mediante EasyBuildd de las aplicaciones más usadas una vez instalado lo demás.

Extender el sistema a empresas de la producción que lo precisen, así como a instituciones educacionales.

REFERENCIAS BIBLIOGRÁFICAS

Abarca, M. (2008). Sistemas distribuidos. estructuración de un cluster beowulf. Technical report. Universidad Católica de Temuco.

adaptaive computing (2014). *Torque resource manager. administrator guide 4.2.8*. adaptative computing.

Bagul, Arun (2016). *DevOps tools. Chef, Ansible, Saltstack and Puppet comparison*. Indian gnu.

Benjamin, S. (2016). Foreman in your data center. *Berlin*.

Bokok, Petr et al (2017). *Red Hat enterprise linux 7 installation guide*. Red Hat.

Bording, B. (2016). *Using Maali to efficiently recompile software post-cle update on a cray xc system*. cug. Online. https://cug.org/proceedings.

Casad, J. (2016). Configuration management. *ADMIN network & security*.

debian.org (2018). *Debian installer preseed*. Debian. Online. https://wiki.debian.org/DebianInstaller/Preseed.

Dolezelova, Marie et al (2017). *Red Hat enterprise linux 7 administrator's guide*. Red Hat.

Dolstra, E. et al (2004). Nix. a safe and policy-free system for software deployment. *LISA* pp. 79–92.

Dolstra, E. et al (2008). Nixos. a purely functional linux distribution. *acm sigplan notices* pp. 367–379.

Domrose, Mils (2014). *Deploying Foreman at enterprise environments 2.0*. inovex.

Eadline, D. (2009). *High performance computing for dummies*. Wiley publishing.

EasyBuild (2016). Easybuild documentation. release 20160518.01. Technical report. Ghent university.

Ferreira, Luis et al (2001). *Linux HPC cluster installation*. IBM corporation. International technical support organization.

foreman.org (2018). *Foreman manual*. Foreman. Online. https://foreman.org.

Gamblin, T. et al (2016). The spack package manager. bringing order to hpc software chaos. *Proceedings of the international conference for high performance computing, networking, storage and analysis*.

Geimer, Hoste and McLay (2016). Modern scientific software management using easybuild and lmod. Technical report. Julich Supercomputer Center and HPC.UGent and Texas Advanced Computing Center.

Girolamo, D. (2016). *Smithy*. Author. Online. https://anthonydigirolamo.github.io/smithy.

Hadvig, Jakub (2013). Virtualization and repository management in academic environment. Engineering degree thesis. Universitas masarpkiana.

Hethey, Jonathan M. (2013). *Gitlab repository management*. Packt publishing.

Hoste, Kenneth et al (2016). Easybuild. building software with ease. Technical report. Ghent university.

Jones, M. and M. Fahey (2008). Design, implementation and experiences of third-party software administration at the ornl nccs. *Proceedings of the 50th. cray user group*.

Martín, S. M. (2018). Implementación de una herramienta de productividad para el desarrollo de aplicaciones. Engineering degree thesis. Universidad Politécnica de Madrid.

Mrozek, D. (2014). *High performance computational solutions in protein bioinformatics*. Springer.

Nolin, Scott (2016). *Puppet, Gitlab and R10k*. University of Wisconsin-Madison.

Oglive, Glen (2016). *How to keep track of Puppet with Foreman*. oss.

Panadero Martínez, Javier (2008). Entorno de desarrollo para cluster. Engineering degree thesis. Universitat Autonoma de Barcelona.

Portal Díaz, Jorge Armando (2017). Arquitectura para instalación y configuración de sistemas operativos y servicios en una red de computadoras de manera automatizada. Engineering degree thesis. Universidad Central “Marta Abreu” de Las Villas.

Praveen, K. and A. Patawari (2018). *ansible-workshop documentation*. Ansible.

Puppet (2016). *NYSE and ICE. compliance, devops and efficient growth with puppet enterprise*. Puppet labs.

Puppet (2018). *Puppet documentation*. Puppet labs.

Rocha Quesada, J. J. et al (2011). Diseño e implmentación de un cluster de cómputo de alto rendimiento. *Acta Universitaria* **21**, 24 – 33.

Ruiz Bosch, Javier Antonio (2016). Implementación en Python del soporte para el software Yade en la plataforma EasyBuild. Engineering degree thesis. Universidad Central "Marta Abreu" de Las Villas.

Russo, J. (2016). Computación de alto desempeño. estado del arte en argentina y en los países del g20. Technical report. Universidad nacional de Córdoba.

Schneller, D. (2016). Automation with ansible. *ADMIN network & security*.

Shah, G. (2016). *Ansible playbook essential*. Packt publishing.

Sloan, J. D. (2004). *High performance Linux cluster with OSCAR, Rocks, OpenMosix and MPI*. O'Reilly.

SLURM (2007). Slurm reference manual. Technical report. University of California.

Sterling, Thomas (2001). *Beowulf cluster computing with Linux*. MIT Press.

SWTools (2011). *SWTools*. SWTools. Online. https://www.olcf.ornl.gov/center-projects/swtools.

Taylor, M. and S. Vargo (2014). *Learning chef*. Publisher. author.

Torberntsson, Kim and Ylva Rydin (2014). A study of configuration management systems. Technical report. Uppsala universitet.

Vrenios, Alex (2002). *Linux cluster architecture*. Sams Publishing.

ANEXOS

Anexo 1

Aprovisionamiento de un nodo descubierto en Foreman paso a paso

Interfaz de Foreman mostrando nodos recién descubiertos.

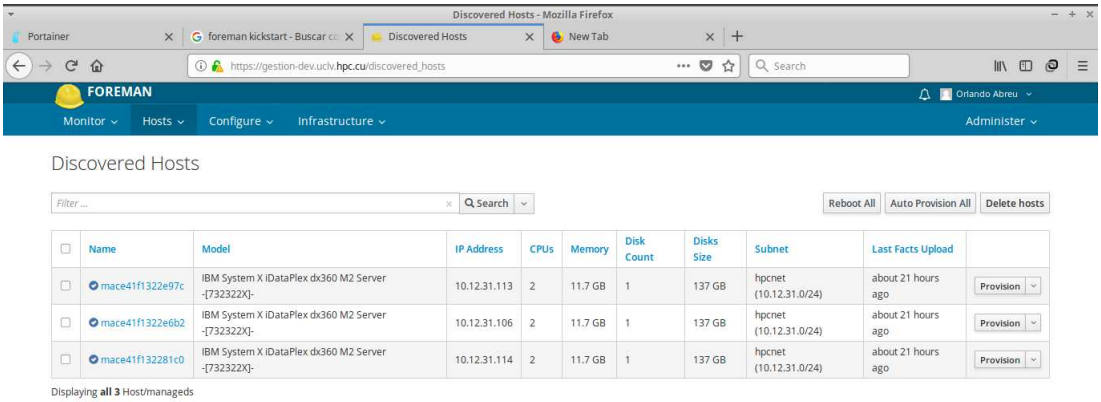

Figure 7: Nodos recién descubiertos.

Seleccionar *Provision*. Vista para hacer heredar un nodo de una configuración previamente definida.

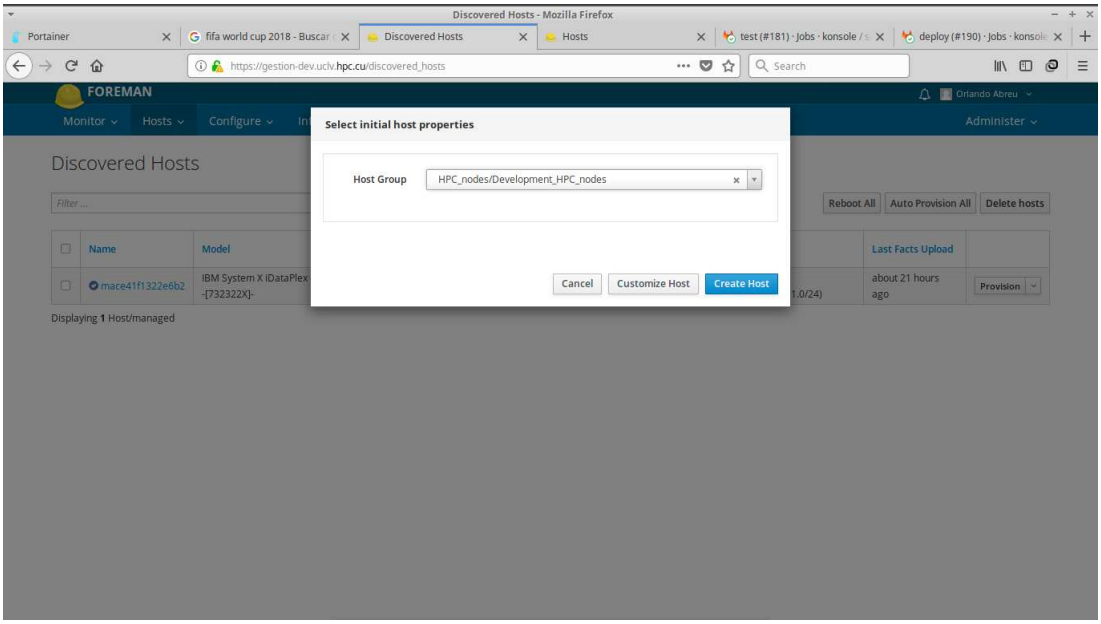

Figure 8: Vista para hacer heredar un nodo de una configuración previamente definida.

Seleccionar un tipo de nodo para heredar. Seleccionar *Customize host*. Vista para la personalización de la configuración del nodo.

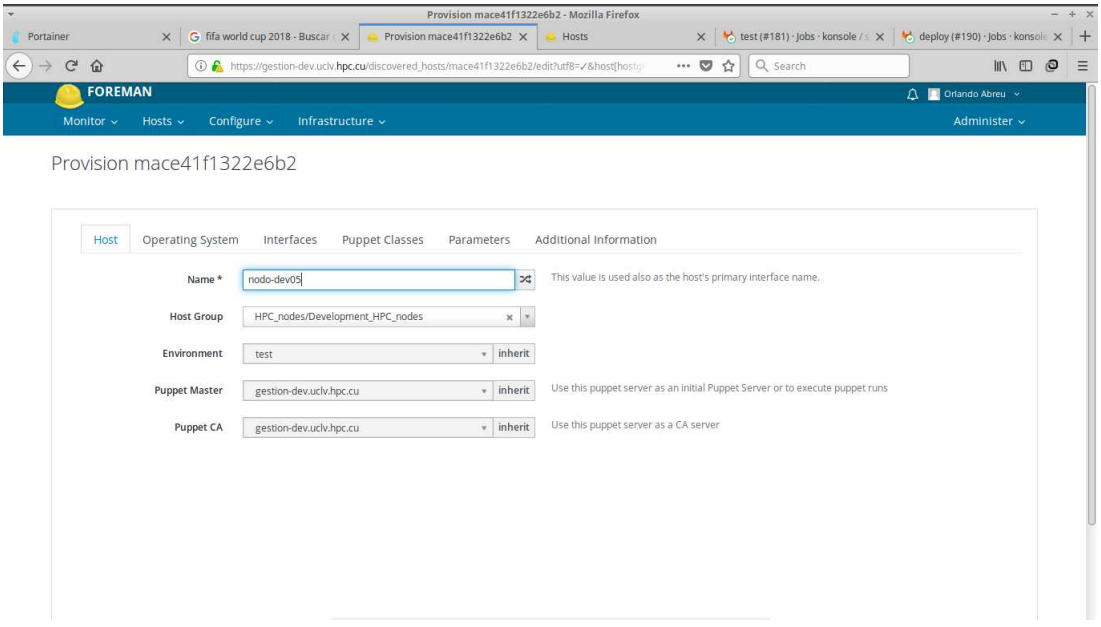

Figure 9: Vista para la personalización de la configuración del nodo.

Transitar por las opciones de personalización. Vista para la personalización de las opciones de sistema operativo.

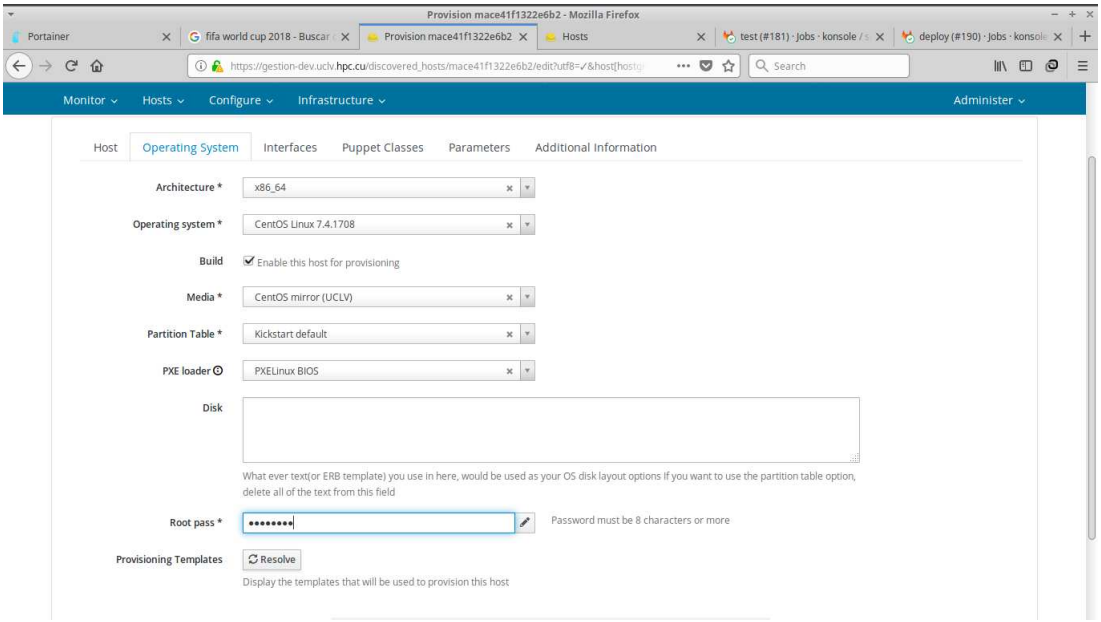

Figure 10: Vista para la personalización de las opciones de sistema operativo.

Vista para personalización de opciones de red. Personalizar *dirección ip*.

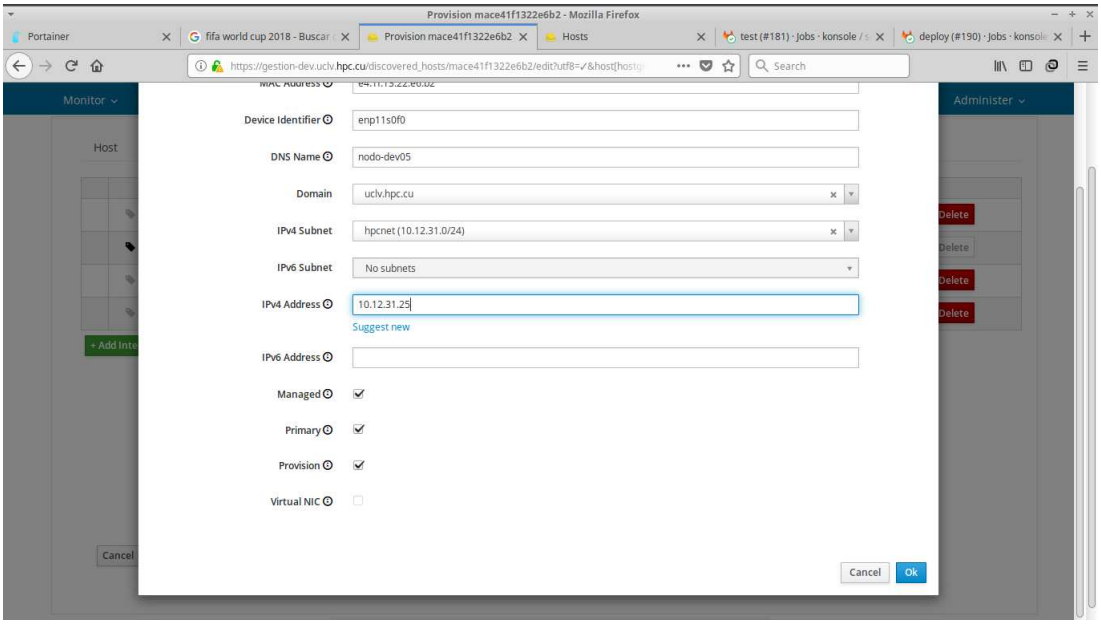

Figure 11: Personalizar *dirección ip*.

Uso exitoso de los ambientes desplegados.

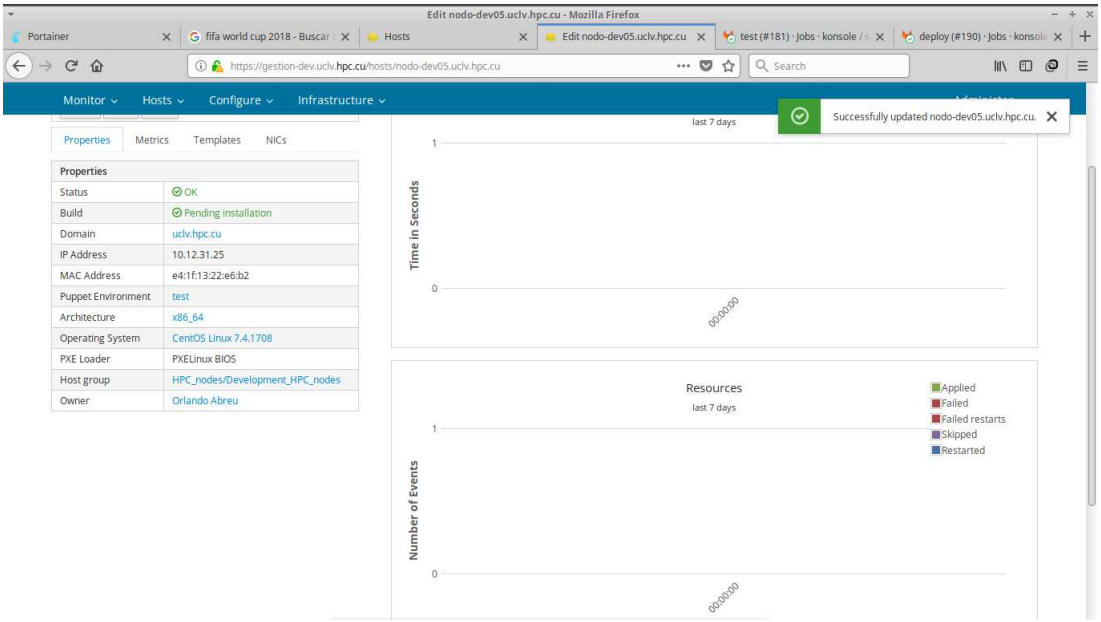

Figure 12: Uso exitoso de los ambientes desplegados.

Anexo 2

Estructura del repositorio de control de Puppet

data/nodes/master.uclv.hpc.cu.yaml

```
slurm_profile::master: true
nfs::server: true
```

data/nodes/nodo002.uclv.hpc.cu.yaml

```
slurm_profile::node: true
nfs::client: true
```

manifests/site.pp

```
## site.pp ##

# This file (/etc/puppetlabs/puppet/manifests/site.pp) is the main entry point
# used when an agent connects to a master and asks for an updated configuration.
#
# Global objects like filebuckets and resource defaults should go in this file,
# as should the default node definition. (The default node can be omitted
# if you use the console and don't define any other nodes in site.pp. See
# http://docs.puppetlabs.com/guides/language_guide.html#nodes for more on
# node definitions.)

## Active Configurations ##

```

```
14 # Disable filebucket by default for all File resources:
15 #https://docs.puppet.com/pe/2015.3/release_notes.html#filebucket-
       resource-no-longer-created-by-default
16 File { backup => false }
17
18 # DEFAULT NODE
19 # Node definitions in this file are merged with node data from the
       console. See
20 # http://docs.puppetlabs.com/guides/language_guide.html#nodes for more
       on
21 # node definitions.
22
23 # The default node definition matches any node lacking a more specific
       node
24 # definition. If there are no other nodes in this file, classes declared
        here
25 # will be included in every node's catalog, *in addition* to any classes
26 # specified in the console for that node.
27
28 node 'master.uclv.hpc.cu' {
29   include ::hpc_role
30 }
31
32 node 'nodo001.uclv.hpc.cu', 'nodo002.uclv.hpc.cu'{
33   include ::hpc_role
34 }
35
36 node default {
37   # This is where you can declare classes for all nodes.
38   # Example:
39   #   class { 'my_class': }
40   class {'::basesys':}
41 }
```

profiles/basesys/manifests/init.pp

```
# This class set the base and common packages and configurations for all
    nodes and master

class basesys {

  #the base profile should include component modules that will be on all
      nodes

  #El orden es importante. Primero debemos configurar nuestros
      repositorios correctamente
   class {'::locales':
     default_locale => 'en_US.UTF-8',
     locales        => [
       'en_US.UTF-8 UTF-8',
       'es_US.UTF-8 UTF-8',
     ],
   }
   class {'::nano':;}
   class {'::vim':;}
   class {'::midnight_commander':;}
}
```

profiles/slurmprofile/manifests/init.pp

```
# Class: slurm_profile
# ====================

class slurm_profile (
  Boolean $master = false,
  Boolean $node = false,
  String $cluster_name = 'mambi',
  String $master_name = 'master',
){
```

```
10 #Install NFS-server on master and NFS-client on nodes, to share /home/
       CLUSTER dir
11   class { '::nfs':;}
12
13 #Install and config openldap-clients, nss-pam-ldapd, nslcd, authconfig,
       on all nodes and master. To user sync
14 #with LDAP server 10.12.31.4
15   class {'::slurm_profile::ldap_sync':;}
16
17 #Insltall and config slurm packages on master an nodes. This class use
       slurm's module at https://github.com/ULHPC/puppet-slurm
18   class {'slurm':
19     #Demons to install
20     with_slurmctld     => $master,
21     with_slurmdbd      => $master,
22     with_slurmd        => $node,
23
24     #manage_firewall
25 #    manage_firewall    => true,
26
27     #using a cread key to munge at puppet:///modules/slurm_profile/munge
           .key
28     munge_create_key   => false,
29     munge_key_source   => 'puppet:///modules/slurm_profile/munge.key',
30
31     #slurm.conf file
32     version            => '17.02.10',
33     src_checksum       => '89f0258430417028c9fe30c7a3a3fe34',
34     clustername        => $cluster_name,
35     controlmachine     => $master_name,
36     controladdr        => '10.12.31.100',
37 #     srunportrange      => '8192-60000',
38     use_pam            => false,
39     manage_pam         => false,
```

```
40      selecttype_params => ['CR_Core_Memory'],
41  #     returntoservice    => 2,
42  #     jobacctgathertype => 'linux',
43  #     proctracktype      => 'linuxproc',
44  #     taskplugin         => 'affinity',
45
46  #     disablerootjobs    => false,
47  #     manage_pam         => false,
48  #     use_pam            => false,
49  #     enforcepartlimits => 'NO',
50  #     common_rpms_basename => $paquetes,
51
52      #nodes.conf file (not yet in separate file)
53      nodes              => {'nodo001' => 'NodeAddr=10.12.31.101 Sockets=2
            CoresPerSocket=4 ThreadsPerCore=1 RealMemory=11841 MemSpecLimit
            =1024 State=UNKNOWN', 'nodo002' => 'NodeAddr=10.12.31.102 Sockets
            =2 CoresPerSocket=4 ThreadsPerCore=1 RealMemory=11841
            MemSpecLimit=1024 State=UNKNOWN'},
54
55      #partitions.conf file (not yet in separate file)
56      partitions         => {'public' => {nodes => 'nodo00[1-2] Default=YES
             MaxTime=INFINITE', state => 'UP'}},
57    }
58  }
```

profiles/slurmprofile/manifests/ldapsync.pp

```
1  # class ldap_sync
2  # Clase para sincronizar los usuarions del hpc con el servidor de ldap
3
4  class slurm_profile::ldap_sync {
5    package {'authconfig':
6      ensure => installed,
7      name   => authconfig,
8    }
```

```
9    package {'openldap-clients':
10     ensure => installed,
11     name   => openldap-clients,
12   }
13   package {'nss-pam-ldapd':
14     ensure => installed,
15     name   => nss-pam-ldapd,
16   }
17   service { 'nslcd':
18     ensure     => true,
19     enable     => true,
20     name       => nslcd,
21     hasrestart => true,
22   }
23 #  file {'/etc/nslcd.conf':
24 #    ensure => 'present',
25 #    owner  => 'root',
26 #    group  => 'root',
27 #    mode   => '0644',
28 #    source => 'puppet:///modules/slurm_profile/nslcd.conf',
29 #  }
30 #  file {'/etc/openldap/ldap.conf':
31 #    ensure => 'present',
32 #    owner  => 'root',
33 #    group  => 'root',
34 #    mode   => '0644',
35 #    source => 'puppet:///modules/slurm_profile/ldap.conf',
36 #  }
37   exec { 'authconfig --enableforcelegacy --update':
38     cwd  => '/var/tmp',
39     path => ['/usr/bin', '/usr/sbin',],
40     user => 'root',
41     subscribe => Package['openldap-clients'],
42     refreshonly => true,
```

```
43    }
44    exec { 'authconfig --enableldap --enableldapauth --ldapserver
          ="10.12.31.5" --ldapbasedn="ou=People,dc=hpc,dc=cu" --
          enablemkhomedir --update':
45      cwd  => '/var/tmp',
46      path => ['/usr/bin', '/usr/sbin',],
47      user => 'root',
48      subscribe => Package['openldap-clients'],
49      refreshonly => true,
50    }
51  }
```

roles/hpcrole/manifests/init.pp

```
1  class hpc_role {
2    class {'::slurm_profile':;}
3    class {'::basesys':;}
4  }
```

Puppetfile

```
1  forge "https://forge.puppet.com"
2
3  # Modules from the Puppet Forge
4  # Versions should be updated to be the latest at the time you start
5  #mod "puppetlabs/inifile",     '1.5.0'
6  #mod "puppetlabs/stdlib",     '4.11.0'
7  #mod "puppetlabs/concat",     '2.1.0'
8
9  # Modules from Git
10 # Examples: https://github.com/puppetlabs/r10k/blob/master/doc/
       puppetfile.mkd#examples
11 #mod 'apache',
12 #   :git    => 'https://github.com/puppetlabs/puppetlabs-apache',
13 #   :commit => '83401079053dca11d61945bd9beef9ecf7576cbf'
14
```

```
#mod 'apache',
#  :git      => 'https://github.com/puppetlabs/puppetlabs-apache',
#  :branch => 'docs_experiment'

#test git acconunt
mod 'saz-locales', '2.5.0'
mod 'cornfeedhobo-nano', '0.1.0'
mod 'ULHPC-vim', '0.2.1'
mod 'kb-midnight_commander', '0.0.8'
#slurm dependencies
#mod 'ULHPC-slurm', '1.1.3'
mod 'puppetlabs/stdlib', '4.24.0'
mod 'puppetlabs-mysql', '3.11.0'
mod 'puppetlabs-vcsrepo', '2.3.0'
mod 'puppetlabs-firewall', '1.12.0'
mod 'puppet-archive', '1.3.0'
mod 'puppet-yum', '2.2.0'
mod 'puppet-selinux', '1.5.2'
mod 'stahnma-epel', '1.3.0'
mod 'svarrette-ulimit', '1.0.6'
mod 'crayfishx-firewalld', '3.4.0'
mod 'bodgit-rngd', '2.0.1'
mod 'puppetlabs-inifile', '2.2.0'
mod 'ghoneycutt-pam', '2.33.0'
#mysql dependencies
mod 'puppetlabs-translate', '1.1.0'
mod 'puppet-staging', '3.1.0'
#yum dependencies
mod 'puppetlabs-concat', '4.1.1'
#nfs module
#mod 'derdanne-nfs', '2.0.7'


#mod 'nano'
```

```
49 #   :git     => 'https://github.com/Puppet-Finland/nano'
50
51 mod 'nfs',
52   :git     => 'http://gitlab.uclv.hpc.cu/hpcuclv/puppet-nfs.git',
53   :branch => 'master'
54 mod 'slurm',
55   :git     => 'https://github.com/ULHPC/puppet-slurm.git',
56   :tag     => 'v1.0.4'
```

Anexo 3

Dockerfile para imagen para pruebas

```
1 FROM ubuntu:xenial
2 RUN echo "deb http://repos.uclv.edu.cu/ubuntu xenial  main restricted
    universe multiverse">/etc/apt/sources.list && echo "deb http://repos.
    uclv.edu.cu/ubuntu xenial-updates  main restricted universe
    multiverse">>/etc/apt/sources.list && echo "deb http://repos.uclv.edu
    .cu/ubuntu xenial-security  main restricted universe multiverse">>/
    etc/apt/sources.list && echo "deb http://repos.uclv.edu.cu/ubuntu
    xenial-backports  main restricted universe multiverse">>/etc/apt/
    sources.list && echo "deb http://repos.uclv.edu.cu/ubuntu xenial-
    proposed  main restricted universe multiverse">>/etc/apt/sources.list
3 RUN apt-get update -y && apt-get upgrade -y && apt-get install python -y
     && apt-get install nano -y && apt-get install vim -y && apt-get
    install puppet-lint -y && apt-get install r10k -y
4 RUN mkdir /etc/puppetlabs && mkdir /etc/puppetlabs/r10k && mkdir /etc/
    puppetlabs/code && mkdir /etc/puppetlabs/code/environments
5 RUN echo ":cachedir: '/var/cache/r10k'">/etc/puppetlabs/r10k/r10k.yaml
    && echo ":sources:">>/etc/puppetlabs/r10k/r10k.yaml && echo "    :my-
    org:">>/etc/puppetlabs/r10k/r10k.yaml && echo "      remote: 'http://
    git.uclv.edu.cu/hpcuclv/puppet-controlrepo.git'">>/etc/puppetlabs/
    r10k/r10k.yaml && echo "      basedir: '/etc/puppetlabs/code/
    environments'">>/etc/puppetlabs/r10k/r10k.yaml
```