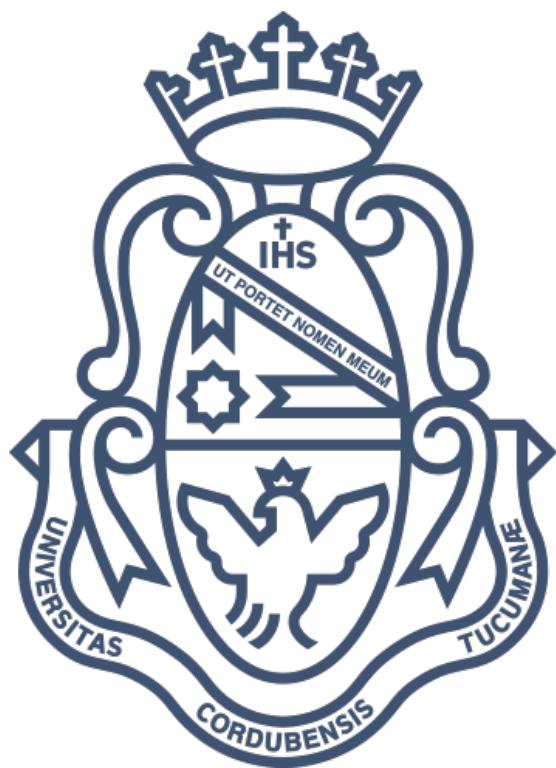


Universidad Nacional de Córdoba



April 18, 2016

Facultad de Ciencias Exactas, Físicas y Naturales

Infraestructura tecnológica virtual con automatización y orquestación.

Alumnos: Juan Arese, Werner Diers

Director de PI: Eschoyez, Maximiliano Andrés

Co-director de PI: Migliazzo, Oscar Andrés

Índice

I. Introducción	7
1. Resumen del Proyecto Integrador	7
1.1. Descripción	7
1.2. Objetivos	7
1.3. Intereses personales	8
1.4. Intereses Institucionales	8
1.5. Metodología	8
1.6. Requerimientos	9
1.7. Cronograma a seguir	9
1.8. Objetivo a alcanzar en cada etapa:	10
II. Marco teórico	10
2. Sistema operativo	11
2.1. FreeBSD	11
2.2. Debian	12
2.3. Debian GNU/kFreeBSD	12
2.4. PC-BSD	13
2.5. CentOS	13
2.6. Solaris	15
3. Virtualización	16
3.1. Tipos de virtualización	16
3.1.1. Virtualización completa	16
3.1.2. Para-virtualización	17
3.1.3. Para-virtualización de drivers	17
3.1.4. Virtualización a nivel del sistema operativo	17
3.1.5. Virtualización de aplicaciones	18
3.1.6. Emulación	18
3.2. Herramientas de virtualización	18
3.2.1. VirtualBox	19
3.2.2. KVM/Qemu	19
3.2.3. Xen	20
3.2.4. OpenVZ	20
3.2.5. Docker	21
4. Aprovisionamiento	21
4.1. Herramientas de aprovisionamiento	22
4.1.1. Cobbler	22
4.1.2. Fully Automatic Installation (FAI)	23
4.1.3. Vagrant	23
4.1.4. Foreman	24
4.2. Protocolo PXE	24

4.2.1. PXE APIs	25
4.2.2. Funcionamiento de PXE	25
5. Orquestación	27
5.1. Herramientas de orquestación	28
5.1.1. Puppet	28
5.1.2. Chef	29
5.1.3. Ansible	30
III. Desarrollo	31
6. Elección de la plataforma	31
6.0.4. Arquitectura de desarrollo	31
7. KVM/Qemu	33
7.1. Arquitectura	33
7.2. Requerimientos de KVM	34
7.3. Restricciones de KVM	35
7.4. Instalación de paquetes de virtualización	35
7.5. Instalación utilizando kickstart con virt-install	36
7.6. Booteo por red con libvirt	36
7.6.1. Configuración de la red	36
7.6.2. NAT con libvirt	36
7.6.3. Configuración del host	36
7.6.4. Configuración de las máquinas virtuales huéspedes	38
7.7. Bridged networking	38
7.7.1. Bridged networking con Virtual Machine Manager	38
7.7.2. Utilizar la interfaz por defecto virbr0	39
7.8. Pools de almacenamiento	39
7.8.1. Borrar un storage pool utilizando virsh	42
7.8.2. Crear sotrage pools basados en directorios con virsh	42
7.8.3. Borrar un storage pool utilizando virsh	44
8. Cobbler	45
8.1. Instalación.	45
8.2. Configuración.	46
8.3. Importar imágenes ISO al servidor Cobbler	49
8.4. Tópicos generales de Cobbler	49
8.4.1. Modelado	49
8.4.2. Distros	50
8.4.3. Profiles	50
8.4.4. Systems	50
8.4.5. Images	50
8.4.6. Repositorios	50
8.4.7. Repositorio local separado de Cobbler.	52
8.4.8. Buildiso	53
8.4.9. Import	53
8.4.10. Kickstarts	54

8.4.11. Snippets	56
8.4.12. Firewall	57
8.4.13. SELinux	58
8.4.14. PXE	58
8.4.15. Reinstalación	58
8.4.16. Virtualización	59
8.4.17. Integración con Puppet	59
8.4.18. Replicate	60
9. Puppet	61
9.1. Pre-instalación	61
9.1.1. Requerimientos de sistema y chequeo de versión de sistema operativo	61
9.1.2. Chequeo de la configuración de red	61
9.2. Instalación	62
9.2.1. Asignación de memoria	62
9.2.2. Instalar el paquete puppet-agent	62
9.2.3. Configuraciones para los agentes	63
9.2.4. Configuraciones para los servidores:	64
9.2.5. Ejecutar puppet	64
9.3. Tópicos generales de Puppet	65
9.3.1. Module	65
9.3.2. Node group	65
9.3.3. Resources	65
9.3.4. Manifiests	69
9.3.5. Catálogos	70
9.3.6. Classes	70
9.3.7. Funciones	70
9.3.8. Metaparámetros	73
9.3.9. Lenguaje	76
9.3.10. Orden y relaciones	77
9.3.11. Definición de nodos	79
10. Automatización de Windows 7	80
10.1. El lado de Windows	80
10.1.1. Instalar WAIK	81
10.1.2. WinPE	81
10.1.3. Instalar y personalizar Windows 7	82
10.1.4. Generalizar el equipo de referencia para preparar la imagen	84
10.1.5. Capturar el equipo de referencia	86
10.1.6. Autounattend.xml	88
10.1.7. Crear nuevos medios de instalación de Windows 7 para la imagen personalizada	100
10.2. El lado de Linux	100
10.2.1. SAMBA	101
10.2.2. Integración con Cobbler	103

Listas de tablas

1.	Pasos automatización Windows 7	81
----	--	----

Listas de figuras

1.	FreeBSD logo	11
2.	Debian logo	12
3.	Debian/kFreeBSD logo	12
4.	PC-BSD logo	13
5.	CentOS logo	13
6.	Solaris logo	15
7.	Diagrama de virtualización completa	17
8.	Diagrama de paravirtualización	17
9.	Diagrama de virtualización a nivel de SO	18
10.	VirtualBox logo	19
11.	KVM/Qemu logo	19
12.	Xen logo	20
13.	OpenVZ logo	20
14.	Docker logo	21
15.	Cobbler logo	22
16.	FAI logo	23
17.	Vagrant logo	23
18.	Foreman logo	24
19.	APIs de PXE	25
20.	Proceso de PXE	27
21.	Puppet logo	28
22.	Chef logo	29
23.	Ansible logo	30
24.	Arquitectura de desarrollo	32
25.	Arquitectura de desarrollo en un entorno mixto.	33
26.	Diagrama de arquitectura de KVM/Qemu	34
27.	Modelado de Cobbler	49
28.	Paso de archivos a Windows	83
29.	Paso de archivos a Windows	83
30.	Paso de archivos a Windows	84
31.	Modo auditoría	84
32.	Carga WinPE.iso para extraer imagen	85
33.	Prioridad de inicio en máquina virtual	85
34.	Ejecución sysprep	86
35.	Verificación de letras de unidad	87
36.	ImageX capturando imagen	88
37.	Windows System Image Manager	89
38.	Idioma y teclado.	89
39.	Idioma	90
40.	Firewall activado por defecto.	90
41.	Discos.	91

42. Configuración discos. Disco 0.	91
43. Discos. Particionamiento.	92
44. Discos. Particionamiento.	92
45. Discos. Particionamiento.	93
46. Discos. Particionamiento.	93
47. Instalación del SO.	94
48. Instalación del SO. Elección de la versión.	94
49. Elección de partición de instalación.	95
50. Aceptación de las condiciones de privacidad y uso.	95
51. Clave de producto.	96
52. Activación del producto.	96
53. Hostname, organización, propietario y zona horaria.	97
54. Idioma, teclado, regional.	97
55. Organización, propietario y zona horaria.	98
56. Configuración de red.	98
57. Contraseña administrador.	99
58. Cuentas de usuario, administrador.	99
59. Cuentas de usuario, invitado.	100

Capítulo I.

Introducción

1. Resumen del Proyecto Integrador

El Sistema desarrollado en este Proyecto Integrador, pretende facilitar algunas de las tareas que los administradores de los laboratorios realizan en las salas de informática. Las funcionalidades desarrolladas permiten a los administradores (o incluso a cualquier persona sin conocimiento técnico) crear máquinas virtuales con un solo click, dichas máquinas virtuales cuentan con todos los servicios y programas necesarios para los alumnos de las diferentes carreras o inclusive se pueden crear máquinas virtuales con otros perfiles, como un perfil para docentes. Además, permite orquestar las políticas a seguir de las diferentes máquinas de la red.

1.1. Descripción

El sistema de infraestructura virtual con automatización y orquestación, tiene como objetivo principal brindar una herramienta a los administradores de laboratorios que facilite la preparación y configuración de sus aulas de manera simple.

El sistema está dividido en dos partes. Una parte de la herramienta está destinada al despliegue en masa de máquinas virtuales. Permite al administrador, crear múltiples máquinas virtuales con escasa interacción humana, de forma automática. Pudiéndose especificar el sistema operativo deseado y componentes de hardware. Del mismo modo, en un laboratorio con máquinas físicas, es posible realizar el despliegue a través de la red.

La segunda parte está destinada a la administración de la configuración de las máquinas virtuales. El sistema permite aplicar cambios de configuración y políticas a un conjunto de máquinas como también a máquinas particulares. Esto evita que el administrador tenga que preparar cada estación de trabajo de a una por vez, disminuyendo la carga de trabajo y el tiempo requerido para llevar a cabo la tarea.

Teniendo en cuenta que las herramientas utilizadas en este sistema deben ser de uso libre se utilizará como sistema operativo base alguna distribución GNU/Linux.

El sistema incluye desarrollo en el lenguaje de programación Python, bash y el lenguaje propio de Puppet, la herramienta utilizada para la orquestación. El servidor de aprovisionamiento, haciendo uso del protocolo PXE, será el encargado de atender las peticiones de los dispositivos para su instalación.

Este Proyecto Integrador servirá como base para futuros Proyectos.

1.2. Objetivos

Principal:

- Desarrollar un sistema para manejar automatismos administrados de políticas de una institución o empresa, para que el administrador pueda configurar las máquinas

virtuales o físicas que se utilizan en la misma. Como caso particular se tomará un laboratorio informático de aprendizaje.

Secundarios:

- Estudiar Sistemas Operativos para servidor.
- Estudiar herramientas de virtualización.
- Estudiar herramientas de aprovisionamiento.
- Estudiar herramientas de administración de configuración.
- Analizar protocolos para inicio a través de la red como PXE.

Antecedentes de Proyectos similares

- No hay antecedentes en este tema.

1.3. Intereses personales

La principal motivación en este Proyecto Integrador es darle un final a la carrera de grado de Ingeniería en Computación intentando abarcar la mayor cantidad de temáticas posibles. Las asignaturas que abarca principalmente son Redes de Computadoras, Sistemas Operativos, Ingeniería de Software, Informática, Comunicaciones de Datos y Algoritmos y estructuras de Datos. Abordamos temas como sistemas operativos, sistemas de archivos, protocolos, metodologías de desarrollo, redes de datos, programación en Python, Shell scripting, virtualización, diagramas UML.

1.4. Intereses Institucionales

La Facultad de Ciencias Exactas, Físicas y Naturales actualmente cuenta con alrededor de cinco aulas de informática, en las cuales se dictan materias de todos los ciclos y especialidades de ingeniería. La idea del Proyecto Integrador es desarrollar un sistema de infraestructura con automatización y orquestación que permita disminuir la carga de trabajo de los administradores de estas aulas y facilitar las tareas de mantenimiento de las mismas, cumpliendo con las políticas del área que administra las aulas.

1.5. Metodología

Para afrontar el Proyecto Integrador de la carrera se utilizó una metodología de desarrollo ágil de software basado en el desarrollo iterativo e incremental. El trabajo desarrollado en una unidad de tiempo es llamado una iteración, las cuales constan de un corto lapso de tiempo de entre una y tres semanas. Cada iteración se compone de un ciclo de vida que integra diversas etapas como planificación, definición de los requerimientos, investigación, diseño, codificación, pruebas y documentación. En cada iteración se agrega una nueva “funcionalidad” al sistema y a medida que avanzan los ciclos el sistema aumenta de tamaño, por esto lo llamamos incremental. Otra característica de la metodología ágil que se utilizó, es una comunicación fluida con el “cliente” que en este caso son los Directores del Proyecto Integrador, de los que también se obtienen los requerimientos. Esto permite una buena retroalimentación, con la cual se devuelven correcciones . No ser estrictos con

la documentación es una característica que se tomó del desarrollo ágil, tener todo anotado luego facilita la generación del informe.

Lugar previsto para la realización:

- Laboratorio de Arquitectura de Computadoras, Facultad de Ciencias Exactas, Físicas y Naturales.

Requerimiento de Instrumental y Equipos:

- Computadora personal.

Inversión económica:

- Inversión provista por el alumno: ninguna
- Apoyo económico externo a la Facultad: ninguno.

1.6. Requerimientos

Los requerimientos del sistema se obtuvieron directamente desde el Director y el Codirector del Proyecto Integrador.

1. La herramienta debe poder aprovisionar distintos sistemas operativos:

CentOS

Ubuntu

Debian

Windows

2. La herramienta debe poder aprovisionar máquinas servidores .
3. La herramienta debe aprovisionar a través de la red.
4. La herramienta debe poder aprovisionar utilizando plantillas.
5. La herramienta debe poder aprovisionar utilizando repositorios locales.
6. La herramienta debe poder actualizar los repositorios locales.
7. La herramienta debe poder setear políticas a las máquinas virtuales, utilizando Puppet.
8. La herramienta debe poder monitorizar el estado de las máquinas virtuales.
9. La herramienta debe ser escalable, integrar nuevas máquinas virtuales fácilmente.
10. La herramienta debe utilizar licencias de código abierto.
11. La herramienta debe estar implementada en la versión más actual al momento de realizar el Proyecto Integrador.

1.7. Cronograma a seguir

El cronograma está dividido en seis etapas diferentes:

- Familiarización con el sistema operativo GNU/Linux elegido.

1.8 Objetivo a alcanzar en cada etapa:

- Investigación de diferentes herramientas de virtualización.
- Investigación de diferentes herramientas de aprovisionamiento.
- Pruebas de las herramientas seleccionadas.
- Pruebas de la herramienta de administración de configuración.
- Preparación y desarrollo del informe del trabajo final y cierre del mismo.

1.8. Objetivo a alcanzar en cada etapa:

- **Primera etapa:** conocer el sistema operativo GNU/Linux. La principal motivación, es la de informarse sobre la base donde se implementará el sistema desarrollado en este Proyecto Integrador.
- **Segunda etapa:** obtener los conocimientos suficientes para crear y administrar máquinas virtuales.
- **Tercera etapa:** conocer cuáles son las herramientas disponibles y utilizadas en ambientes de producción para el aprovisionamiento de máquinas, teniendo en cuenta que deben ser de código abierto, analizarlas y elegir la más apropiada para realizar el Proyecto Integrador.
- **Cuarta etapa:** implementar las herramientas seleccionadas en conjunto.
- **Quinta etapa:** realizar la implementación conjunta de todas las herramientas automatizando la instalación y realizar configuraciones pertinentes en los sistemas ya instalados, por medio de Puppet.
- **Sexta etapa:** desarrollo del informe del trabajo final

Capítulo II.

Marco teórico

Previamente al abordaje del desarrollo, requerimos conocimientos teóricos que nos posibiliten comprender el entorno donde se ejecutarán las aplicaciones que componen el Sistema, las herramientas que darán soporte o permitirán cumplir con las funcionalidades previstas y las que se utilizarán para desarrollar. Algunas de las mismas fueron explícitamente solicitadas en los requerimientos. El resto de las herramientas, que están implícitas en los requerimientos, precisaron de investigación y pruebas para conocer si permiten cumplir estos requerimientos y además, si hay varias opciones, elegir la más conveniente. Podemos dividir esta sección del informe en cuatro áreas diferentes:

- Sistema operativo
- Virtualización
- Aprovisionamiento
- Orquestación

La información contenida en esta sección se desprende de las investigaciones realizadas en cada una de las etapas del Proyecto Integrador.

2. Sistema operativo

2.1. FreeBSD



Figura 1: FreeBSD logo

FreeBSD es un sistema operativo basado en BSD para arquitecturas Intel (x86 e Itanium), AMD64, AlphaTM y UltraSPARC.

FreeBSD viene con una excelente colección de herramientas de sistema como parte del sistema base. A pesar de esto, existen otras que no vienen incluidas y se necesitan instalar para utilizarlas. FreeBSD ofrece dos tecnologías complementarias para instalar software de terceros en el sistema: la Colección de puertos o ports de FreeBSD y los paquetes binarios. Los paquetes binarios son archivos simples que descargamos desde repositorios. Contienen una copia de los programas binarios precompilados de la aplicación y se pueden manipular con las herramientas de gestión de paquetes de FreeBSD: `pkg_add`, `pkg_delete`, `pkg_info`, etc. Por otro lado, existen ciertos pasos que se deben llevar a cabo para compilar un programa (descargar, desempaquetar, parchear, compilar e instalar). Los ficheros que conforman un port permiten que el sistema se encargue de todo esto, mediante un conjunto simple de órdenes. La colección de puertos para instalar se encuentra en `/usr/ports`.

FreeBSD proporciona compatibilidad binaria con muchos otros sistemas operativos tipo UNIX, como Linux. Esto es necesario, ya que muchos desarrolladores y compañías sólo desarrollan para Linux. La compatibilidad binaria permite a los usuarios utilizar en FreeBSD cerca del 90% de las aplicaciones desarrolladas para Linux sin que sea necesario realizar alguna modificación sobre la aplicación.

Otras características:

- Servicios multiusuario que permiten a mucha gente usar el sistema FreeBSD simultáneamente.
- Conexión de redes TCP/IP muy robusta, con soporte para estándares industriales.
- La protección de memoria que garantiza que las aplicaciones (o los usuarios) no se estorben los unos a los otros.
- Compatibilidad binaria con muchos programas nativos de Linux, SCO, SVR4, BSDI y NetBSD.
- Soporte para multiprocesamiento simétrico con múltiples CPUs.

2.2. Debian



Figura 2: Debian logo

Debian es una distribución libre, por completo manejada por la comunidad, no está basada en ninguna otra distribución y por el contrario gran parte de las distribuciones actuales están basadas en ella. Debian es famoso por filosofía de estabilidad ante todo, por eso mismo, no tiene un cronograma de lanzamiento de nuevas versiones. Estas se liberán cuando estén realmente listas. Es una distribución ampliamente utilizada tanto como escritorio como en su versión para servidor.

En su última versión, Debian 8 Jessie, las siguientes arquitecturas tienen soporte: x32 (i386), x86-64 (amd64), Motorola/IBM PowerPC, MIPS, IBM S/390 y ARM.

Este release incluye el nuevo estándar sistema de inicio *systemd*.

Principales características de Jessie:

- Actualización de administrador de paquetes: apt 1.0.9.8.1
- Núcleo del sistema: Linux kernel 3.16
- Cambio a systemd.
- Entornos gráficos : Gnome 3.14, KDE 4.14,
- Los puertos para el kernel de FreeBSD (kfreebsd-amd64 y kfreebsd-i386), incluídos para versiones anteriores no son parte de esta versión.
- Soporte UEFI para amd64, i386 y arm64.

2.3. Debian GNU/kFreeBSD



Figura 3: Debian/kFreeBSD logo

Debian GNU/kFreeBSD es un sistema operativo de propósito general. Es una distribución oficial de GNU Debian que usa el Kernel de FreeBSD en vez del Kernel de Linux. Cerca del 90% del Software de Debian está disponible para Debian GNU/kFreeBSD.

Debian GNU/kFreeBSD soporta ZFS desde el kernel. Esto también puede significar un mejor rendimiento y mayor estabilidad en discos que utilicen el sistema de archivos ZFS. Debian GNU/kFreeBSD podría instalarse en una jaula. Esto permitiría aprovechar todas las ventajas de FreeBSD en el servidor, brindándole la opción de usar Debian a los usuarios del servidor y eliminando la necesidad de que se familiaricen con FreeBSD.

2.4. PC-BSD



Figura 4: PC-BSD logo

PC-BSD es un sistema operativo de escritorio basado en FreeBSD. La idea principal, es agregar las características conocidas de FreeBSD como sistema operativo para servidores, como lo son la estabilidad y la seguridad, a un sistema operativo de escritorio amigable.

Para la instalación, brinda una interfaz gráfica que permite la elección de opciones de instalación. Presenta dos perfiles de instalación, cada una con una serie de programas opcionales para seleccionar.

Un perfil de instalación es el Desktop o escritorio, orientado a computadoras de usuario. Básicamente esta opción es la idea de PC-BSD. Instala y configura el entorno de escritorio KDE por defecto. También se pueden seleccionar otros escritorios como GNOME o XFCE, herramientas de virtualización, herramientas de desarrollo, etc. El otro perfil de instalación está orientado a servidores. No instala paquetes de interfaz gráfica y permite instalar servidores web, servidores de base de datos, servidores de archivos, herramientas de virtualización, etc. PC-BSD además agrega lo que promete ser la nueva forma de administración de paquetes para FreeBSD: pkgng (pkg next generation). Esta herramienta no viene instalada por defecto en FreeBSD. Su funcionamiento se asemeja al apt-get de Linux.

Otra característica importante es que PC-BSD utiliza ZFS como sistema de archivos, instalándolo casi sin intervención del usuario (si así se desea).

2.5. CentOS



Figura 5: CentOS logo

CentOS es un distribución Linux empresarial, basada en Red Hat Enterprise Linux. CentOS concuerda con la política de distribución de Red Hat y apunta a ser binariamente compatible en su totalidad. Cada versión de esta distro tiene soporte por siete años en cuestiones de actualizaciones de mantenimiento y seguridad, lo que se traduce en un ambiente confiable, predecible, reproducible, de bajo mantenimiento y seguro.

La principal ventaja de esta distro es que se obtiene un conjunto estable de la mayoría de paquetes que por lo general sólo incluyen correcciones de errores. En su última versión, CentOS 7 solo está disponible para la arquitectura x86_64, y representa un gran cambio frente a versiones anteriores del sistema operativo, como la inclusión de systemd, Gnome 3, GRUB 2, y el sistema de archivos XFS. El entorno de escritorio KDE también forma parte de la oferta de CentOS 7.

Principales novedades de CentOS 7:

- Actualización del núcleo del sistema: Kernel 3.10.0.
- Soporte para Linux Containers.
- Inclusión de VMware Tools y controladores de gráficos 3D.
- OpenJDK-7 como JDK por defecto.
- Cambio a systemd.
- Cambio a firewalld y GRUB2 .
- XFS es el sistema de archivos por defecto y permite escalar la capacidad de almacenamiento del sistema hasta 500 terabytes. XFS es un sistema de archivos de 64 bits con journaling de alto rendimiento, y está especialmente indicado para discos grandes (superiores a 1 TB). No obstante y para necesidades menos exigentes se pueden emplear otros sistemas de archivos, como Ext4.
- iSCSI y FCoE (Fiber Channel over Ethernet) en el espacio del Kernel.
- Soporte para PPTv2 (Precision Time Protocol).
- Soporte para tarjetas Ethernet 40G.
- Soporte UEFI.

En cuanto a systemd, es el reemplazo de init como demonio para iniciar servicios, procesos y recursos del sistema. Systemd es la nueva forma predeterminada de iniciar los sistemas Linux, y ha sido adoptado por Red Hat, Debian y Ubuntu, entre otros. CentOS 7 es compatible con Microsoft Active Directory (y obviamente con Red Hat), por lo que puede trabajar con facilidad en entornos heterogéneos. CentOS 7 incluye PCP (Performance Co-Pilot), un conjunto de frameworks y servicios en tiempo real para supervisar y monitorizar el rendimiento del sistema.

2.6. Solaris



Figura 6: Solaris logo

Solaris es un sistema operativo de tipo Unix desarrollado por Sun Microsystems desde 1992 como sucesor de SunOS. Es un sistema certificado oficialmente como versión de Unix. Aunque Solaris fue desarrollado como software privado, la mayor parte de su código se ha liberado como proyecto de software libre denominado OpenSolaris. Solaris es famoso por su escalabilidad, especialmente en sistemas SPARC. Sun Solaris se ejecuta sobre la arquitectura SPARC en 32 y 64 bits, o sobre procesadores x86 (incluidos Intel y AMD). Sin embargo, en agosto de 2010, Oracle decidió interrumpir la publicación y distribución de OpenSolaris.

Solaris tiene una reputación de ser muy adecuado para el multiprocesamiento simétrico (SMP), soportando un gran número de CPUs. Históricamente Solaris ha estado firmemente integrado con la plataforma hardware de Sun, SPARC, con la cual fue diseñado y promocionado como un paquete combinado. Esto proporcionaba frecuentemente unos sistemas más fiables pero con un coste más elevado que el del hardware de PC.

A partir de su versión 10, Sun Microsystems ha promocionado Solaris con sus propias estaciones de trabajo y servidores de 64 bits basados en procesadores AMD Opteron e Intel Xeon, así como también en sistemas de 32 bits. Esta versión añadió soporte para paravirtualización cuando es utilizada como “sistema operativo invitado” en ambientes basados en Xen.

En su última versión, 11.3, sus principales características son:

- Incluye una nueva versión de OpenStack (Juno) con soporte para topologías de red adicionales y nuevos servicios.
- SNAT, soporte IPv6.
- Pools de almacenamiento.
- Aprovisionamiento de máquinas (bare metal provisioning) como servicio .
- Incluye soporte para desarrollo basado en la API REST utilizando el Demónio de Administración Remota (permite configuración remota de los sistemas Oracle usando Python, C y Java).
- Sistema de archivos ZFS.
- Solaris Containers

3. Virtualización

Virtualización es un término amplio para software ejecutándose, usualmente sistemas operativos, de manera concurrente y aislada de otros programas en el mismo sistema. Muchas de las implementaciones de virtualización utilizan un “hypervisor”, una capa de software que controla el hardware y provee sistemas operativos huéspedes con acceso a los dispositivos de hardware subyacentes. El hypervisor permite ejecutar múltiples sistemas operativos en el mismo sistema físico ofreciendo hardware virtualizado al sistema operativo huésped. Esta tecnología, provee un conjunto de herramientas para aumentar la flexibilidad y reducir los costos, los cuales son tópicos importantes en cualquier empresa o institución. En esencia, la virtualización incrementa la flexibilidad desacoplando un sistema operativo y los servicios y aplicaciones soportados por él, de una plataforma de hardware física específica, permitiendo el establecimiento de múltiples entornos virtuales sobre una plataforma de hardware compartida. Estos entornos pueden ser creados localmente o aprovisionados extremadamente. La virtualización se destaca también apoyando la innovación a través del uso de entornos virtuales para practicar y aprender. Un estudiante puede comenzar un curso o trabajo un entorno de sistema conocido, estándar y aislado del entorno de producción; si se produce algún tipo de daño solo afecta al sistema virtual. Además se puede establecer entornos únicos de software para el aprendizaje sin demandar el uso exclusivo de recursos de hardware. Aunque en comparación los costos de inversión para tener un número elevado de máquinas físicas son mucho mayores que el costo para invertir en un servidor con altos recursos para realizar la virtualización, se podría decir que la virtualización posee inconvenientes vinculados con sus exigentes requerimientos de hardware, en cuanto a capacidad de procesamiento y de memoria RAM y de almacenamiento. Otra desventaja es que del sistema de virtualización depende del sistema operativo anfitrión. Es decir, el anfitrión es el punto débil del sistema ya que se comparte por todos los sistemas virtualizados, si se rompe éste, se rompen todas las máquinas virtuales.

3.1. Tipos de virtualización

3.1.1. Virtualización completa

Consiste en la virtualización de paquetes y herramientas para correr de forma totalmente virtualizada, sin modificaciones, sistemas operativos huéspedes. Este modo cuenta con la ventaja de consolidar sistemas viejos en hardware nuevo, más eficiente y reducir el espacio físico y costos de operación relativos al consumo energético y refrigeración de estos sistemas menos eficientes. La virtualización completa ofrece, sin embargo, menor rendimiento de entrada/salida que instalaciones nativas (también llamadas “bare-metal” o “metal-pelado”) de sistemas operativos. Por ejemplo el software KVM, Xen, VMware Workstation o VirtualBox hacen uso de esta técnica. Cabe destacar que en el caso de KVM se requiere soporte de hardware para ejecutar la virtualización, ya sea con procesadores Intel o AMD.

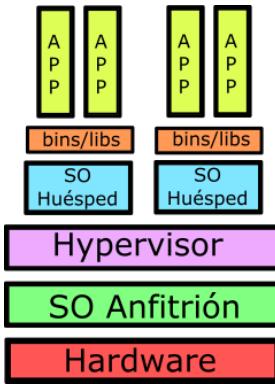


Figura 7: Diagrama de virtualización completa

3.1.2. Para-virtualización

Para-virtualización es un técnica de virtualización la cual implica ejecutar versiones modificadas de los sistemas operativos. El sistema operativo para-virtualizado es modificado para que se de cuenta que está siendo virtualizado, ofreciendo un habilidad aumentada para la optimización, ya que el huésped está al tanto de su entorno. El rendimiento está generalmente muy cerca de la ejecución nativa de sistemas operativos no virtualizados. Por ejemplo, utilizan esta técnica KVM, XEN y VMware Server ESX.

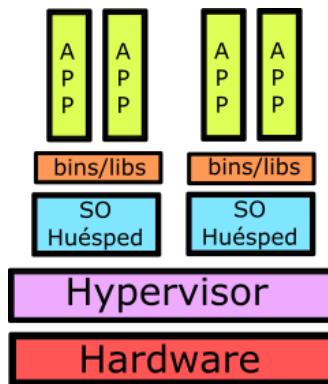


Figura 8: Diagrama de paravirtualización

3.1.3. Para-virtualización de drivers

La para-virtualización y la virtualización completa pueden ser combinadas para permitir a sistemas operativos no modificados recibir un rendimiento cercano de entrada/salida al de ejecución nativa, por medio de drivers para-virtualizados en sistemas operativos completamente virtualizados.

3.1.4. Virtualización a nivel del sistema operativo

También llamada virtualización basada en contenedores, esta técnica virtualiza un servidor físico a nivel del sistema operativo, permitiendo que múltiples servidores virtuales aislados y seguros se ejecuten sobre un solo servidor físico. Con la virtualización basada

en contenedores, no existe la sobrecarga asociada con tener a cada huésped ejecutando un sistema operativo completamente instalado. Este enfoque también puede mejorar el rendimiento porque hay un solo sistema operativo encargándose de los avisos de hardware. Una desventaja de la virtualización basada en contenedores, sin embargo, es que cada invitado debe utilizar el mismo sistema operativo que utiliza el host. Por ejemplo, Jaulas con Warden en FreeBSD, OpenVZ o Linux-Vserver usan esta técnica.

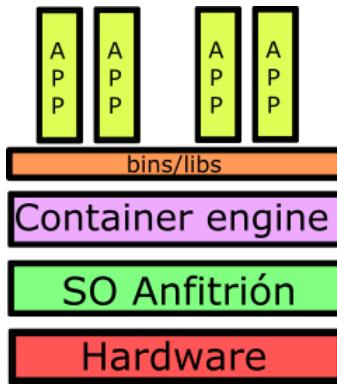


Figura 9: Diagrama de virtualización a nivel de SO

3.1.5. Virtualización de aplicaciones

La virtualización de aplicaciones consiste en correr una aplicación sobre una máquina virtual usando los recursos reales. Por ejemplo la máquina virtual JAVA.

3.1.6. Emulación

Un emulador es hardware o software que permite a un sistema de computación comportarse como otro sistema. Generalmente, un emulador permite a un sistema correr software o utilizar dispositivos periféricos diseñados para el otro sistema. Por ejemplo Qemu es un emulador muy usual en ingeniería.

3.2. Herramientas de virtualización

Algunas de las herramientas más utilizadas para virtualizar son las siguientes, sin embargo como uno de los requisitos es utilizar herramientas de código abierto, no se indagó acerca de VMWare:

- VirtualBox
- KVM/Qemu
- Xen
- OpenVZ
- Docker
- VMWare Workstation

3.2.1. VirtualBox



Figura 10: VirtualBox logo

VirtualBox, desarrollado por Oracle, es un virtualizador completo de propósito general para hardware x86, orientado al uso para servidor, escritorio y embebido. Como software de código abierto, se puede utilizar bajo la licencia GNU General Public License 2 (GPL2).

AL día de la fecha, VirtualBox corre en Windows, Linux, Macintosh y Solaris y soporta una amplia variedad de sistemas operativos, entre ellos RHEL(7,6,5,4), Windows (NT 4.0, 2000, XP, Server 2003, Vista, Windows 7, Windows 8, Windows 10), DOS/Windows 3.x, Linux (2.4, 2.6, 3.x and 4.x), Solaris y OpenSolaris, OS/2, y OpenBSD.

3.2.2. KVM/Qemu



Figura 11: KVM/Qemu logo

KVM (Kernel-based Virtual Machine), desarrollado por Red Hat Enterprise Linux, es una infraestructura de virtualización completa para el kernel de Linux que lo transforma en un hypervisor. Fue incorporado a la línea principal del kernel Linux en la versión 2.6.20. KVM requiere un procesador con hardware que permita extensión para virtualización (Intel VT o AMD-V). También está disponible para instalarlo desde los “ports” de FreeBSD en la forma de módulos de kernel.

La para-virtualización tiene soporte para ciertos dispositivos en Linux, OpenBSD, FreeBSD y Windows (entre otros) utilizando la API VirtIO. Se tiene placa Ethernet, un controlador de entrada/salida de disco paravirtual, gráficos VGA.

Qemu puede utilizarse como emulador y como virtualizador. Cuando se utiliza como virtualizador, Qemu toma un rendimiento cercano al nativo. Para ello, debe ejecutarse bajo el hypervisor Xen o KVM. En conjunto KVM/Qemu, KVM es quien hace las veces de árbitro del acceso al CPU y memoria, y Qemu emula los recursos de hardware.

3.2.3. Xen



Figura 12: Xen logo

Xen, desarrollado por University of Cambridge Computer Laboratory y mantenido por IBM, HP, Intel, AMD, RedHat, es una herramienta de virtualización que se ejecuta por debajo del sistema operativo y actúa como hypervisor.

Esta herramienta permite trabajar con virtualización completa y con paravirtualización. Como KVM y VirtualBox, también posee una larga lista de sistemas operativos soportados.

3.2.4. OpenVZ



Figura 13: OpenVZ logo

OpenVZ es una herramienta de virtualización para Linux basada en contenedores. OpenVZ crea múltiples contenedores aislados en un servidor físico y asegurando que las aplicaciones no entren en conflicto. Utiliza un kernel de Linux modificado y por consiguiente sólo puede correr Linux. Todos los contenedores comparten la misma arquitectura y versión del kernel. Cada contenedor se comporta como un servidor autónomo. Ya que OpenVZ emplea un modelo de kernel único, es tan escalable como kernel Linux 2.6, lo que significa que soporta hasta 64 CPUs y hasta 64 GiB de RAM. Un entorno virtual único se puede escalar hasta el equipo físico entero. También cuenta con migración en vivo que posibilita mover un contenedor de un servidor físico a otro sin apagar el contenedor. Un propietario (root) de un servidor físico OpenVZ (conocido como Nodo de Hardware) puede ver todos los procesos y archivos de los contenedores. Esto hace la administración masiva de escenarios posible: se puede ejecutar un simple script de intérprete de comandos que actualice todos (o sólo algunos seleccionados) los contenedores a la vez.

3.2.5. Docker

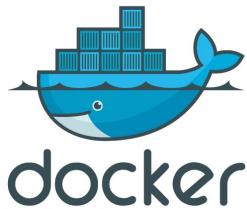


Figura 14: Docker logo

Docker es otra herramienta de virtualización para Linux basada en contenedores. La idea detrás de Docker es crear contenedores ligeros y portables para las aplicaciones de software que puedan ejecutarse en cualquier máquina con Docker instalado, independientemente del sistema operativo que la máquina tenga por debajo, facilitando así también los despliegues.

El contenedor Docker se puede desplegar en cualquier otro sistema (que soporte esta tecnología), con lo que se ahorra el tener que instalar en este nuevo entorno todas aquellas aplicaciones que normalmente se utilicen.

Un contenedor Docker no contiene todo un sistema completo, sino únicamente aquellas librerías, archivos y configuraciones necesarias para desplegar las funcionalidades que contenga. Asimismo Docker se encarga de la gestión del contenedor y de las aplicaciones que contenga.

Para obtener esta fluidez Docker extiende LXC (LinuX Containers), un sistema de virtualización ligero que permite crear múltiples sistemas totalmente aislados entre si sobre la misma máquina o sistema anfitrión. Y todo dado que no se emula un sistema operativo completo, sólo las librerías y sistemas de archivos necesarios para la utilización de las aplicaciones que se tengan instaladas en cada contenedor.

En las últimas versiones de Docker se ha introducido drivers de Docker y una librería llamada libcontainer, que ayuda a que Docker sea totalmente multiplataforma, teniendo compatibilidad con Windows y Mac OS X, además de Linux.

4. Aprovisionamiento

En general, aprovisionamiento, significa proveer o hacer que algo esté disponible. El término es utilizado en un gran variedad de contextos en el área de Tecnologías de Información. En este Proyecto Integrador, el término hace referencia a lo siguiente: Aprovisionamiento es el conjunto de acciones para preparar una máquina virtual, con el sistema apropiado, datos y software, y dejarla lista para su operación.

- Tareas típicas que se tienen que llevar a cabo para que esto suceda son:
- Seleccionar el conjunto de hardware virtualizado (memoria RAM, disco, cantidad de procesadores asignados, placa de red, etc) para crear una máquina virtual bare-metal.
- Cargar el sistema operativo adecuado.

- Configurar el sistema (dirección IP, gateway, DNS, hostname, MAC, etc).
- Actualizar el sistema y aplicar parches.
- Cargar el conjunto de aplicaciones necesarias.
- Configurar el sistema para adaptarlo a las políticas definidas de la institución.

En resumen, el aprovisionamiento de máquinas virtuales se realiza basado en los recursos disponibles y en los requisitos específicos de cada máquina virtual, según sea la funcionalidad que se le vaya a dar.

4.1. Herramientas de aprovisionamiento

Algunas de las herramientas de código abierto más utilizadas para aprovisionar son las siguientes:

- Cobbler
- FAI
- Vagrant
- Foreman

4.1.1. Cobbler



Figura 15: Cobbler logo

Cobbler es un servidor Linux de aprovisionamiento que centraliza y simplifica el control de los servicios incluyendo DHCP, TFTP y DNS con el propósito de realizar instalaciones de sistemas operativos basadas en la red. Puede ser configurado para PXE, reinstalaciones y huéspedes virtualizados utilizando Xen, KVM o VMWare como también dispositivos físicos. Está dirigido especialmente a Red Hat Linux y sus derivados, pero es posible configurarlo para que inicie con PXE otras distribuciones de Linux como Debian, Ubuntu o Knoppix. Es una herramienta que se encuentra en creciente desarrollo y cada vez añade más soporte a distintas distros e incluso a FreeBSD y a futuro Windows. Actualmente cuenta con poco soporte para el sistema operativo de Microsoft.

Cuenta con un sistema integrado para administración de configuración pero también cuenta con soporte para la integrar la herramienta de administración de configuración Puppet. Cobbler se basa en el mecanismo de Kickstart y ofrece perfiles de instalación que pueden ser aplicados a una o muchas máquinas. La información de contenida en un plantilla kickstart puede ser modificada dinámicamente pasando variables (llamadas ksmeta) o utilizando snippets, donde se puede mantener el código común simplificando la lectura y minimizando el tamaño del archivo kickstart.

4.1.2. Fully Automatic Installation (FAI)



Figura 16: FAI logo

FAI es un sistema no interactivo para instalar, personalizar y administrar sistemas Linux y configuraciones de software en computadoras como también en máquinas virtuales y entornos chroot, desde pequeñas redes hasta una infraestructura grande escalable y clusters. Es una herramienta para la instalación totalmente automática de Debian y otras distros de Linux como Suse, Red Hat, Solaris, vía red, DVDs personalizados de instalación o en entornos chroot.

Algunas de las características más importantes:

- Instalar y actualizar Debian, Ubuntu, SUSE, Red Hat, etc.
- Despliegue centralizado y administración de configuración.
- Recuperación de desastre integrado.
- Fácil configuración de software RAID y LVM.
- Instalar máquinas virtuales usando KVM, Xen y VirtualBox.
- Control remoto vía SSH durante la instalación.

4.1.3. Vagrant



Figura 17: Vagrant logo

Vagrant provee entornos fáciles de reproducir, configurar construidos sobre tecnología industrial estándar. Vagrant es un software que crea y configura entornos de desarrollo virtuales aprovisionando sobre VirtualBox, VMware, KVM y contenedores Linux. Es una software que se encuentra una capa encima de estas herramientas. Luego herramientas de administración de configuración como Ansible, Chef, Salt, y Puppet pueden ser utilizadas para instalar y configurar automáticamente el software en la máquina.

4.1.4. Foreman



Figura 18: Foreman logo

Foreman es una herramienta para el aprovisionamiento, configuración y monitorización de servidores físicos y virtuales. Puede aprovisionar máquinas bare-metal, virtualizadas y en la nube a través de instalaciones desatendidas por medio de DHCP, DNS, TFTP y PXE . Tiene una gran integración con software de administración de configuración como Puppet, Chef, Salt y otros por medio de plugins.

Algunas de sus características son:

- Descubrir, aprovisionar y actualizar toda la infraestructura bare-metal.
- Crear y gestionar instancias entre nubes privadas y públicas.
- Agrupar hosts y dirigirlos en conjunto, sin importar la ubicación.
- Revertir cambios históricos para auditoría y resolución de problemas.

4.2. Protocolo PXE

El protocolo PXE (Preboot Execution Environment) es un estándar que permite a las computadoras, dentro de una red, que todavía no fueron cargadas con un sistema operativo, ser configuradas e iniciadas remotamente por un administrador. Utiliza una extensión de opciones del protocolo DHCP.

En resumen, PXE funciona del siguiente modo:

El cliente inicializa el protocolo realizando un broadcast de DHCPDISCOVER, conteniendo una extensión que identifica la solicitud como la de un cliente que utiliza PXE. Asumiendo que un servidor DHCP que implementa este protocolo extendido, luego de algunos pasos intermedios, el servidor envía al cliente una lista de Boot Servers. El cliente descubre un Boot Server del tipo seleccionado y recibe el nombre de un archivo ejecutable en dicho servidor. El cliente utiliza TFTP (Trivial File Transfer Protocol) para descargar el ejecutable. Finalmente, el cliente inicia la ejecución de la imagen seleccionada.

Si un cliente utiliza PXE y el servidor no, el servidor ignora el código PXE previniendo la discontinuidad en las operaciones de DHCP y del Bootstrap Protocol.

PXE especifica los protocolos por los cuales un cliente solicita y descarga una imagen ejecutable desde un Boot Server y los mínimos requisitos para el entorno de ejecución cuando la imagen es corrida.

Las ventajas de utilizar PXE incluyen:

- La máquina cliente no necesariamente necesita un sistema operativo o un disco rígido.
- Como este protocolo es independiente del vendedor nuevos tipos de computadoras pueden ser añadidos a la red.

4.2.1. PXE APIs

Preboot Services API: Contiene muchas funciones de control e información.

Trivial File Transport Protocol (TFTP) API: Habilita la apertura y cierre de conexiones TFTP, la lectura desde una conexión TFTP y la escritura en otra.

User Datagram Protocol (UDP) API: Habilita la apertura y cierre de conexiones TFTP, la lectura desde una conexión UDP y la escritura en otra.

Universal Network Driver Interface (UNDI) API: Habilita el control básico de entrada/salida a través de la interfaz de red del cliente. Esto permite la utilización de protocolos universales de controladores para que el mismo controlador pueda ser usado en cualquier interfaz que soporte esta API.

El siguiente diagrama ilustra la relación entre los NBP y las APIs de PXE:

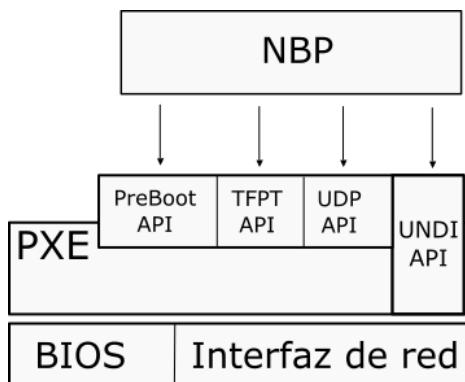


Figura 19: APIs de PXE

4.2.2. Funcionamiento de PXE

1. El cliente realiza un broadcast de un mensaje DHCPDISCOVER al puerto estándar DHCP (UDP 67). Un campo de opciones en este mensaje contiene:
 - a) Etiqueta para el identificador del cliente UUID.
 - b) Etiqueta para la versión de UNDI (Universal Network Device Interface) del cliente.
 - c) Etiqueta para la arquitectura del cliente.
 - d) La opción 60, Class ID, puesta a to “PXEClient:Arch:xxxxx:UNDI:yyyyzzz”.
2. El servidor DHCP responde enviando un mensaje DHCPOFFER al cliente en el puerto estándar DHCP (UDP 68). El mensaje contiene parámetros estándar DHCP:
 - a) Una dirección IP para el cliente.

- b) Parámetros configurados por el administrador.
3. Del DHCPOFFER que es recibido, el cliente guarda:
 - a) La dirección IP.
 - b) Parámetros configurados por el administrador.
 - c) Una lista de Boot Servers del campo “Boot Server” en las etiquetas PXE del DHCPOFFER.
4. El cliente debe enviar una solicitud por la dirección del Boot Server al servidor y esperar por el acuse de recibo (acknowledgment – ACK).
5. El cliente selecciona y descubre un Boot Server. Este paquete puede ser enviado por broadcast al puerto 67. Este paquete es el mismo que el DHCPDISCOVER inicial en el paso uno, excepto que es codificado como DHCPREQUEST y ahora contiene lo siguiente:
 - a) La IP asignada al cliente por el servidor DHCP.
 - b) Una etiqueta con el identificador del cliente (UUID).
 - c) Una etiqueta con la versión de UNDI del cliente.
 - d) Una etiqueta con la arquitectura del cliente.
 - e) La opción 60, Class ID, puesta a to “PXEClient:Arch:xxxxx:UNDI:yyzzz”.
 - f) El tipo de Boot Server en el campo de opciones de PXE.
6. El Boot Server envía un paquete DHCPACK unicast al cliente. Este ACK contiene:
 - a) El nombre del archivo ejecutable.
 - b) Parámetros de configuración MTFTP.
 - c) Otras opciones necesarias para que el NBP pueda ser ejecutado.
7. El cliente descarga el archivo ejecutable utilizando TFTP (puerto 69). El archivo descargado y la ubicación del código descargado en memoria depende de la arquitectura de la CPU del cliente.
8. El cliente PXE determina si es necesaria la verificación de autenticidad del archivo descargado. Si se requiere se envía otro DHCPREQUEST preguntando por credenciales.
9. El cliente inicia la ejecución del código descargado.

El cliente PXE esperará por la información necesaria unos 60 segundos. La etapa DHCPDISCOVER puede repetirse hasta cuatro veces, con tiempos de espera de 4,8,16 y 32 segundos respectivamente. Si el cliente recibe la DHCPOFFER dentro de ese tiempo, se procederá con DHCPREQUEST. Si no, se detendrá con un error de PXE.

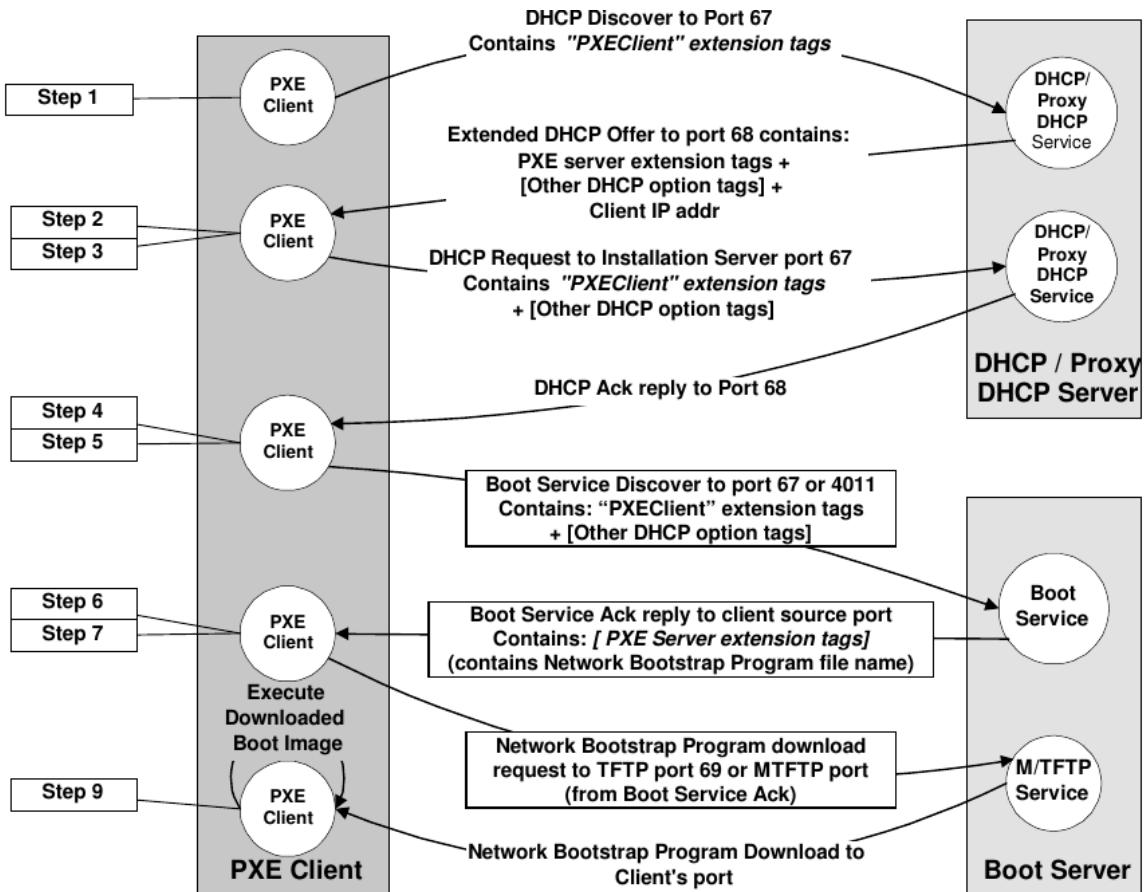


Figura 20: Proceso de PXE

5. Orquestación

La orquestación describe el alineamiento automatizado, la coordinación y la administración de complejos sistemas de computadoras, middleware y servicios. En este sentido, la orquestación se trata de alinear los requisitos de negocio con las aplicaciones, datos e infraestructura. Define las políticas y niveles de servicio a través de flujos de trabajo automatizados, aprovisionamiento y gestión de cambio. Esto crea una infraestructura alineada con la aplicación que puede ser escalada hacia arriba o abajo basándose en las necesidades de cada aplicación.

La orquestación también provee la gestión centralizada de los recursos. Por ejemplo, reduce el tiempo y esfuerzo para desplegar múltiples instancias de una sola aplicación. Cuando es necesario que se creen más instancias de diferentes aplicaciones, herramientas automatizadas pueden realizar tareas que, previamente, podían ser llevadas a cabo sólo por múltiples administradores.

Un escenario que se puede encontrar, por ejemplo, un administrador necesita desplegar una aplicación web, pero para hacerlo, primero debe crear el servidor de base de datos. Luego debe incluir en la base de datos todas las direcciones IP que pueden conectarse al servidor, y también agregar este nuevo servidor a la herramienta que lo monitorea, o abrir un puerto particular antes de proceder. Cada tarea expuesta puede ser automatizada, pero el conjunto de estas automatizaciones, junto con la coordinación secuencial de las mismas,

realizada sin tener en cuenta el tipo de sistema operativo en el que corren, describen un proceso, en el cual actúa la orquestación.

No hay que confundir los términos automatización y orquestación. Éstos se podrían comparar con tarea y proceso.

La optimización de un proceso, por ejemplo, no se puede conseguir simplemente por la automatización. A la automatización le concierne una tarea: ejecutar un servidor web, configurar un servidor web, detener un servicio. A la orquestación, sin embargo, le concierne la ejecución de un flujo de trabajo (si se quiere automatizado) de un proceso. Un proceso de aprovisionamiento lleva a cabo múltiples tareas e involucrar múltiples sistemas. El objetivo de la orquestación no es sólo ejecutar automáticamente un servidor, lo cual aumenta la velocidad en el proceso de despliegue y lleva las aplicaciones a producción más rápido. También permite una oportunidad para optimizar aquellos procesos para mejorar aún más la velocidad de despliegue.

Una de las maneras más simples de optimizar un proceso es eliminar los pasos repetitivos.

Entonces, automatización trata acerca de codificar tareas y orquestación acerca de codificar procesos. Esta última toma ventaja de la automatización para reutilizar bloques básicos.

5.1. Herramientas de orquestación

Algunas de las herramientas de código abierto más utilizadas para orquestar son las siguientes:

- Puppet
- Chef
- Ansible

5.1.1. Puppet



Figura 21: Puppet logo

Es un sistema de orquestación que permite definir el estado de la infraestructura, forzando automáticamente que se llegue al estado correcto definido.

Puppet es una herramienta diseñada para administrar la configuración de sistemas similares a Unix y a Microsoft Windows de forma declarativa, es decir, se establece el estado requerido en vez como llegar al mismo.. El usuario describe los recursos del sistema y sus estados utilizando el lenguaje declarativo que proporciona Puppet. Esta 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. 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 específicos del sistema operativo (como rpm, yum, apt).

Puppet funciona bajo la arquitectura cliente servidor donde un puppet master indica a sus agentes las configuraciones que deben aplicar. Además, los masters pueden aplicar manifiestos a sí mismos. Notar que hay dos etapas:

1. Compilar los catálogos
2. Aplicar los catálogos

Un catálogo es un archivo que describe los deseos de un estado de sistema para un nodo en particular. Enumera todos los recursos que necesitan ser administrados, así como las dependencias entre esos recursos.

En esta arquitectura, los nodos administrados corren la aplicación puppet-agent, usualmente en segundo plano y uno o más servidores corren la aplicación puppetmaster administrada por un servidor web (como Apache.) Periódicamente, los agentes piden al master el catálogo. El master, compila y corre el catálogo del nodo usando varias fuentes de información a las que tiene acceso. Una vez que recibe el catálogo, el agente chequea cada recurso descrito en él. Si encuentra algún recurso que no está en el estado deseado, se realizan los cambios necesarios para corregirlos. Luego de aplicar el catálogo, el agente envía un reporte al master.

Los sistemas soportados son Linux (Red Hat Enterprise y derivados, Debian, Ubuntu, Fedora), Unix (BSD, Mac OS X, Oracle Solaris, AIX) y Windows.

5.1.2. Chef



Figura 22: Chef logo

Es una herramienta de automatización de infraestructura de sistemas o administración de configuraciones. Se enfoca en seguir un conjunto de pasos (llamados recetas) con el propósito de presentar un producto final ya listo para trabajar y/o probar.

Existen 2 tipos de versiones:

Chef Server está enfocado a ser el servidor central que permite suministrar a los diferentes nodos clientes con las diversas configuraciones necesarias, las cuales se mantienen alojadas en el servidor. El cliente sondea periódicamente al Chef Server para corroborar

las últimas políticas y estado de la red, en caso que haya algún parámetro desactualizado, el cliente lo actualiza. Además ofrece balanceo de carga, escalabilidad, búsquedas rápidas entre otros.

Chef Solo es la versión de código abierto y reside localmente en el nodo, esto quiere decir que toda la información y recetas necesarias para configurar el nodo deben estar presentes en su disco duro. Esta herramienta utiliza Ruby-DLS para escribir las “recetas” (configuraciones de sistemas que describen como son manejadas las aplicaciones). Estas recetas, que pueden ser agrupadas para facilitar la administración, describen de forma secuencial una serie de recursos que deben estar en un estado particular.

Chef Server es soportado sobre RHEL/CentOS/Oracle Linux, y Ubuntu. Mientras que el soporte para los clients es AIX, RHEL/CentOS, FreeBSD, Mac OS X, Solaris (OS), Microsoft Windows, Ubuntu, ArchLinux, Debian, Fedora, y otros.

5.1.3. Ansible



Figura 23: Ansible logo

Es una plataforma para configurar y administrar computadoras. Combina instalación multi-nodo, ejecuciones de tareas ad hoc y administración de configuraciones.

Ansible distingue dos tipos: controladores y nodos. Primero, existe una única máquina de control donde la orquestación comienza. Los nodos son manejados desde esa máquina por OpenSSH. La máquina de control conoce a los nodos a través de un inventario. Esta herramienta usa una arquitectura sin agentes, es decir, los nodos no necesitan instalar ni ejecutar en segundo plano ningún proceso que se comunique con la máquina de control. Sin embargo, los nodos deben contar con Python ≥ 2.4 y las máquinas de control con Python 2.6.

Dispone de módulos que trabajan sobre JSON y la salida estándar puede ser escrita en cualquier lenguaje. Nativamente utiliza YAML para describir configuraciones de los sistemas.

Los sistemas operativos soportados en las máquinas de control son la mayoría de las distribuciones Linux y Unix (Red Hat, Debian, CentOS, OSX, y BSD) entre otros excepto Windows.

Ansible puede instalarse en ambientes virtualizados, nubes públicas y privadas, incluyendo VMWare, OpenStack, AWS, Eucalyptus, KVM y CloudStack.

Capítulo III.

Desarrollo

6. Elección de la plataforma

Una vez realizada la interiorización de las diferentes herramientas utilizadas en el mercado, se decidió unificar la plataforma para el desarrollo del Proyecto Integrador.

Al principio se había comenzado a trabajar con FreeBSD, de manera remota con un servidor que dispone el Laboratorio de Arquitectura de Computadoras. Sin embargo, no se encontraron herramientas de aprovisionamiento y orquestación con la suficiente documentación o soporte para este sistema operativo. Por lo tanto, como sistema operativo base fue elegido CentOS 7, su última versión al día de la fecha. Esto es debido a que junto con su versión empresarial, es un sistema operativo muy robusto y confiable orientado a servidores y tal vez sea el más utilizado en el mercado. Además, se tuvo en cuenta la nueva estandarización del iniciador de sistemas, systemd, para esta versión del sistema operativo y para gran parte de las futuras entregas en diferentes distribuciones Linux, orientadas a servidor y también de escritorio.

Habiendo elegido CentOS 7, se optó por virtualizar haciendo uso de KVM, la cual fue desarrollada nativamente para esta distribución de Linux y que cuenta con una extensa documentación. A su vez, en el Laboratorio de Computación, existe un servidor que utiliza esta herramienta para servir a las terminales de ciertas aulas de informática. Consultando con los Directores del Proyecto, a profesionales en el tema se llegó a la conclusión que KVM es una herramienta estable y apta para un entorno de producción.

Para el aprovisionamiento de máquinas virtuales también por motivos de desarrollo nativo, recomendación de profesionales que la han utilizado y en base a lecturas en foros que tratan este tema, se eligió a Cobbler, ya que es la más estable y fiable para entornos de producción. También cuenta con soporte integrado para orquestación con Puppet, herramienta que fue elegida por ser parte de los requerimientos del Proyecto.

6.0.4. Arquitectura de desarrollo

El siguiente esque representa la arquitectura de desarrollo utilizada para todas las pruebas.

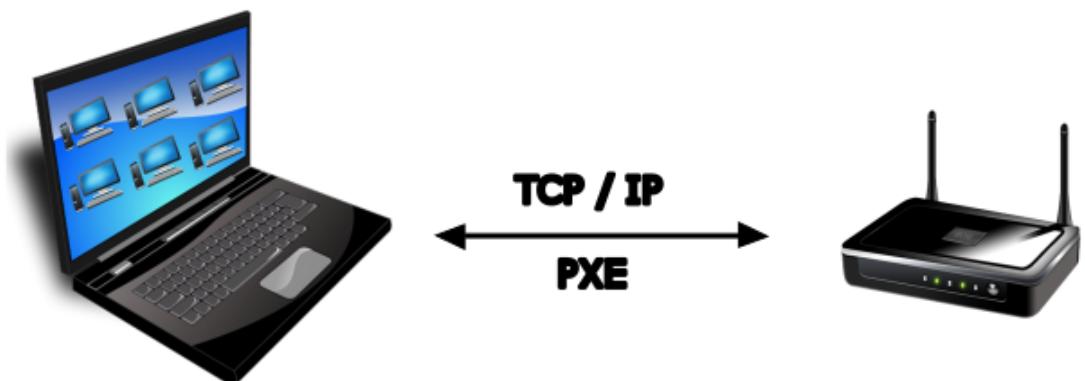


Figura 24: Arquitectura de desarrollo

Sin embargo, como también se puede aplicar para máquinas de escritorio o un entorno mixto, con equipos virtualizados y reales, se realizaron pruebas como se muestra en el siguiente esquema.

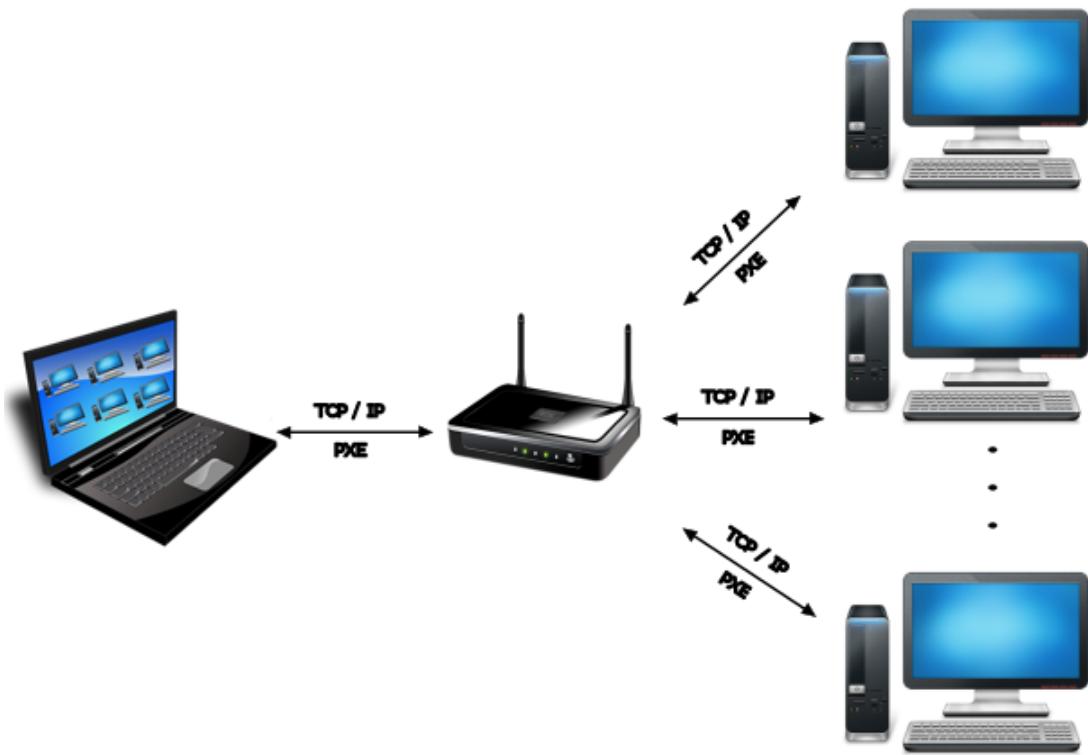


Figura 25: Arquitectura de desarrollo en un entorno mixto.

7. KVM/Qemu

7.1. Arquitectura

En este Proyecto se utiliza la técnica de virtualización completa. En particular, la arquitectura que utiliza KVM es la siguiente:

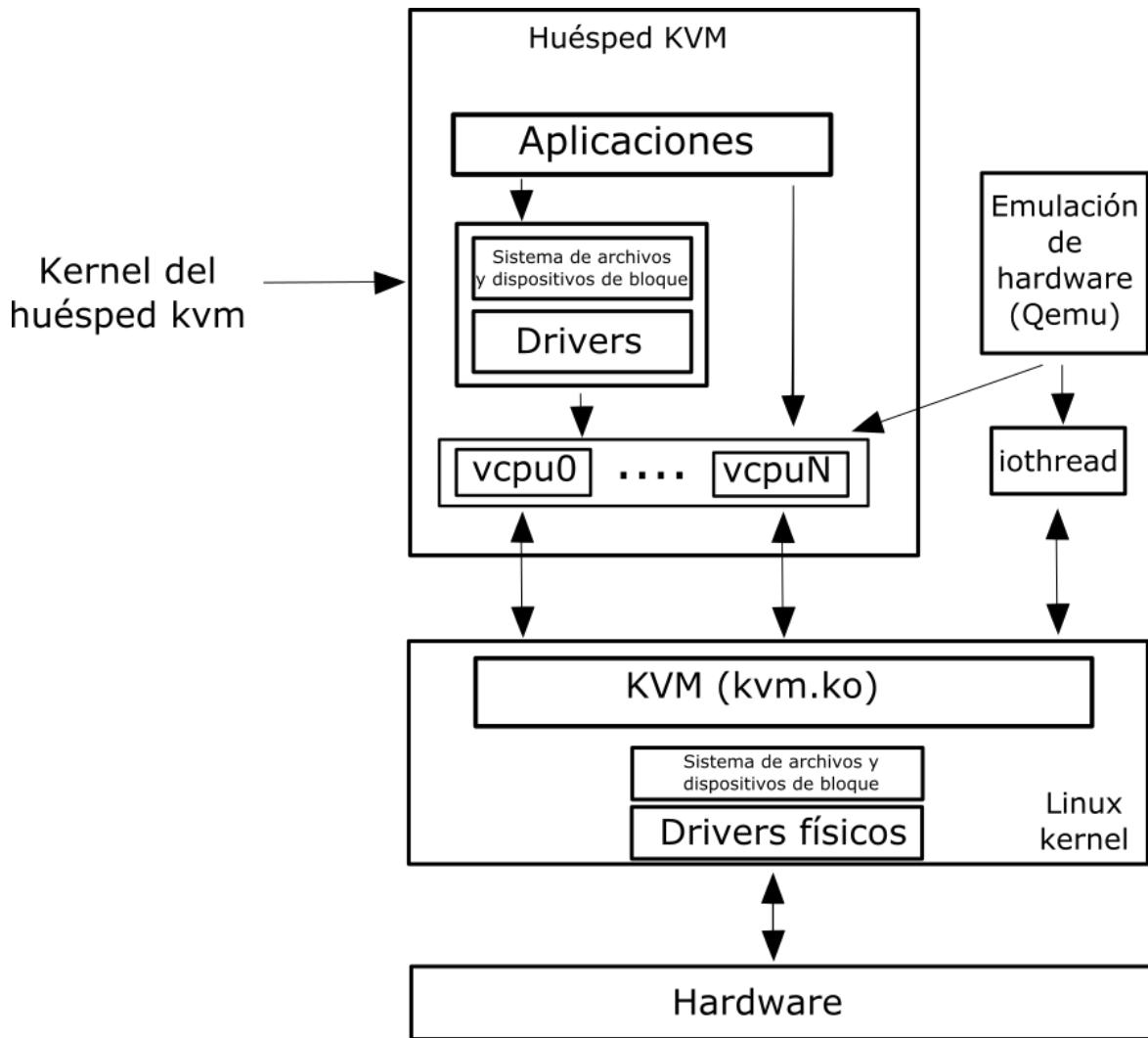


Figura 26: Diagrama de arquitectura de KVM/Qemu

7.2. Requerimientos de KVM

El hipervisor KVM requiere que el microprocesador cuente con VT-x para procesadores de Intel o con AMD -V para los propios de AMD. Para poder confirmar que un procesador cuenta con esto, en los sistemas basados en Linux, se debe ejecutar el siguiente comando:

```
grep -E 'svm|vmx' /proc/cpuinfo
```

La salida de este comando es una porción del archivo `/proc/cpuinfo` en el cual se detallan las diferentes flags que contiene el procesador, entre ellas, la `svm` (AMD) o `vmx` (Intel). En caso de no poseer esas flags, el procesador no soporta hiper-virtualización y la salida será vacía.

La siguiente, es la salida obtenida con un AMD Athlon(tm) II P360 Dual-Core Processor de 1,7GHz:

```
flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36
clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm 3dnowext
3dnow constant_tsc rep_good nopl nonstop_tsc extd_apicid pni monitor cx16 popcnt
lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a 3dnowprefetch osvw ibs
skinit wdt nodeid_msrs hw_pstate npt lbrv svm_lock nrip_save
```

Se debe asegurar que el módulo de KVM este cargado, para esto ejecutamos :

```
lsmod | grep kvm
```

La salida obtenida, nuevamente en la misma máquina que en el caso anterior es:

```
kvm_amd 60554 0 kvm 448375 1 kvm_amd
```

En caso de no estar cargados los módulos, se deben cargar manualmente.

7.3. Restricciones de KVM

- El número máximo de CPUs por huésped es elevado (240 para RHE 7.1) por lo que no aplica en este trabajo.
- La virtualización anidada no está soportada.
- Sobre utilización de memoria es soportada por KVM utilizando el disco de swap.
- Sobre utilización de CPUs es soportada por KVM, se recomienda no utilizar más de diez CPUs virtuales por cada CPU física.
- Virtualización de dispositivos SCSI no está soportada. Virtualización de dispositivos IDE en KVM es limitada a cuatro por huésped.
- Soporta 32 slots para dispositivos PCI (paravirtualizados) y 8 de estos por cada slot (datos RHE7)
- La asignación de dispositivos referenciados a dispositivos físicos son de uso exclusivo a la VM .
- La migración y salvado, o restauración de la VM no está soportada mientras el dispositivo esté en uso.
- KVM no soporta kernels de real time.

7.4. Instalación de paquetes de virtualización

Se deben ejecutar los siguientes comandos:

```
yum update
```

```
yum install qemu-kvm qemu-img
```

Y se recomienda la instalación de los siguientes paquetes, con el siguiente comando:

```
yum install kvm libvirt python-virtinst qemu-kvm virt-manager libvirt qemu-system-x86  
qemu-img libvirt-python libvirt-client virt-install virt-viewer python-virtinst
```

7.5. Instalación utilizando kickstart con virt-install

Para conocer como utilizar virt-install revisar el man del mismo con:

```
man virt-install
```

Se puede automatizar la instalación de un sistema operativo utilizando un kickstart, es decir, un archivo que le indica al sistema operativo como debe instalarse, éste, además, permite ejecutar configuraciones pre y pos instalación. Se indica el archivo kickstart deseado en la creación de la maquina virtual, añadiendo la siguiente sección al comando virt-install:

```
--extra-args="ks=http://192.168.100.1/ks.cfg"
```

Como se ve, el archivo se llama (en este caso) `ks.cfg` y est alojado en un host con la dirección IP 192.18.100.1

7.6. Booteo por red con libvirt

Se necesita un servidor para PXE con DHCP y TFTP, dnsmasq y un servidor configurado por cobbler.

7.6.1. Configuración de la red

CentOS 7 soporta las siguientes configuraciones de red para la virtualización:

- Redes virtuales usando NAT (Network Address Translation)
- Dispositivos físicos distribuidos usando la asignación de dispositivos PCI
- Redes puenteadas (bridge)

Se debe habilitar NAT, bridge o asignar directamente un dispositivo PCI para permitir a host externos acceder a los servicios de red en las máquinas virtuales huéspedes.

7.6.2. NAT con libvirt

Uno de los métodos más comunes para compartir las conexiones de red es usar NAT forwarding (también conocido como redes virtuales).

7.6.3. Configuración del host

Cada instalación estándar de libvirt provee una conectividad basada en NAT a las máquinas virtuales como red virtual por defecto. Verificar que está disponible con el comando '`virsh net-list --all`'.

```
# virsh net-list --all
```

Nombre	Estado	Inicio automático	Persistente
<hr/>			
default	activo	si	si

Si no se encuentra, lo siguiente puede ser usado en el archivo de configuración XML (`/etc/libvirtd/qemu/myguest.xml`) para el huésped:

```
ll /etc/libvirt/qemu
total 24
drwxr-xr-x 3 root root 4096 sep 7 11:42 .
drwxr-xr-x 6 root root 4096 sep 1 11:57 ../
-rw----- 1 root root 3435 sep 3 10:58 centosLImpio.xml
-rw----- 1 root root 3458 sep 7 11:42 master.xml
drwxr-xr-x 3 root root 4096 sep 1 11:57 networks/
```

La red por defecto está definida desde `/etc/libvirt/qemu/networks/default.xml`

Para marcar la red por defecto para iniciar automáticamente:

```
# virsh net-autostart default
Network default marked as autostarted
```

Para iniciar la red por defecto:

```
# virsh net-start default
Network default started
```

Una vez que la red por defecto de libvirt está corriendo, se verá un dispositivo bridge aislado. Este dispositivo no tiene ninguna interfaz física añadida. El nuevo dispositivo utiliza NAT e IP forwarding para conectarse a la red física. No añadir nuevas interfaces.

```
# brctl show
bridge name      bridge id      STP enabled      interfaces
virbr0          8000.000000000000      yes
```

libvirt añade reglas de iptables las cuales permiten el tráfico desde y hacia las máquinas virtuales huéspedes unidas al dispositivo virbr0 en las cadenas (chains) INPUT, FORWARD, OUTPUT y POSTROUTING. libvirt luego intenta habilitar el parámetro ip_forward. Algunas otras aplicaciones tal vez deshabiliten ip_forward por lo tanto lo mejor es dejar esta configuración fija añadiendo lo siguiente a `/etc/sysctl.conf`.

```
net.ipv4.ip_forward = 1
```

7.6.4. Configuración de las máquinas virtuales huéspedes

Una vez que la configuración del host está completa, una máquina virtual huésped puede ser conectada a la red virtual basada en su nombre. Para conectar a un huésped a la red virtual por defecto, lo siguiente puede ser utilizado en el archivo de configuración XML para el huésped (`/etc/libvirtd/qemu/myguest.xml`):

```
<interface type='network'>
    <source network='default' />
</interface>
```

Definir la dirección MAC es opcional. Si no se define una, una dirección MAC es automáticamente generada y usada como la dirección MAC del dispositivo bridge utilizado por la red. Definirla manualmente es útil para mantener la consistencia o la facilidad de referencia a través del ambiente, o para evitar la posibilidad de conflicto (muy escasa).

```
<interface type='network'>
    <source network='default' />
    <mac address='00:16:3e:1a:b3:4a' />
</interface>
```

7.7. Bridged networking

Bridged networking (también conocido como virtual network switching) es usado para poner las interfaces de red de las máquinas virtuales en la misma red que la interfaz física.

7.7.1. Bridged networking con Virtual Machine Manager

Procedimiento para crear un bridge con virt-manager:

1. Desde el menú principal de virt-manager, ir a Editar > Detalles de la Conexión > Interfaces de Red (Edit > Connection Details > Network Interfaces) .
2. Pulsar el ícono + en la parte inferior.
3. En el Tipo de Interfaz (Interface type) del menú que se despliega, seleccionar Bridge y luego continuar.
4. En el campo Nombre (Name) ingresar un nombre como por ejemplo virbr0 o br0.
5. Seleccionar un Modo de inicio (Start mode) del menú desplegable. Seleccionar onboot (activa la interfaz bridge en el próximo reinicio de la máquina virtual).
6. Seleccionar la casilla Activar ahora (Activate now) para activarlo inmediatamente.
7. Para configurar Configuraciones IP (IP settings) o Configuraciones Bridge (Bridge settings) realizar los cambios necesarios y pulsar OK al finalizar.
8. Seleccionar la interfaz física para conectar a las máquinas virtuales.

9. Pulsar Finalizar (Finish).
10. Seleccionar el bridge a utilizar y pulsar Aplicar (Apply).

Para detener la interfaz, pulsar Detener Interfaz (Stop Interface). Luego para eliminarla pulsar Delete Interface (Borrar Interfaz).

7.7.2. Utilizar la interfaz por defecto virbr0

Para utrillizar la interfaz bridge por defecto, en el caso de arranque a través de la red por medio de PXE, en el paso 5/5 de la creación de una nueva máquina virtual con virt-manager, seleccionar bajo Opciones Avanzadas “Especificiar el nombre del dispositivo compartido” y en “Nombre del bridge” ingresar `virbr0`.

7.8. Pools de almacenamiento

Un pool de almacenamiento es un conjunto de almacenamiento guardado por un administrador. Los pools de almacenamiento son divididos en volúmenes de almacenamiento por los administradores, y los volúmenes son asignados a las VM's como dispositivos de bloques.

Por ejemplo, el administrador de almacenamiento responsable por un servidor NFS crea un disco compartido que almacena toda la información de las VM's. El administrador definiría un pool de almacenamiento en el host de virtualización usando el detalle de disco compartido. En este ejemplo, el administrador quiere que `nfs.example.com:/path/to/share` sea montado en `/vm_data`. Cuando el pool es iniciado, libvirt monta el compartido en el directorio específico, tal como lo haría el administrador del sistema logueándose y ejecutando `mount`. Si el pool está configurado con autostart, libvirt asegura que el disco compartido NFS es montado en el directorio especificado cuando libvirt es iniciado. Una vez que el pool esté iniciado, el directorio en el disco compartido NFS es reportado como un volumen de almacenamiento y el path de los SV (storage volumes) pueden ser consultados por las APIs de libvirt. El path del SV puede entonces ser copiado en la sección que describe la fuente de almacenamiento en el archivo XML de las VM's para dispositivos de bloques. En el caso de NFS, una aplicación que usa las APIs de libvirt puede crear y eliminar SV en el SP(storage pool). No todos los tipos de SP soportan creación y destrucción de volúmenes.

Los SP y SV no son requeridos por la mayoría de las operaciones de las VM's. Note que uno de las características de libvirt es el protocolo remoto, entonces es posible administrar todos los aspectos de los ciclos de vida de las VM's así como las configuraciones de los recursos requeridos por las VM's. Esas operaciones deben representar a un host remoto con las API de libvirt. En otras palabras, un administrador usando aplicaciones de libvirt puede asegurar un usuario para desempeñar todas las tareas para configurar la maquina física para las VM's. Aunque el SP es un contenedor virtual, está limitado por dos factores:

- El tamaño máximo permitido por qemu-kvm y
- el tamaño del disco de la máquina física.

Los siguientes son los tamaños máximos permitidos:

- `virtio-blk -> 8 Exabytes`

- Ext4 -> 16 Terabytes
- XFS -> 8 Exabytes

Libvirt usa un directorio basado en un SP, el `/var/lib/libvirt/images`, como el SP por defecto. Este, puede ser cambiado por otro.

- **Local storage pools:** Los LSP están directamente unidos a la máquina física servidor. Los LSP incluyen : Directorios locales, discos directamente conectados, particiones físicas y LVM. Esos SP almacenan las imágenes de las VM's o son unidas a la VM's como almacenamiento adicional. Los LSP no son apropiados para muchos entornos de producción dado que no soportan migración en vivo
- **Networked storage pools:** Los NSP incluyen almacenamiento de dispositivos compartidos sobre una red usando protocolos estándar. NSP es requerido cuando las VM's migran entre dos máquinas físicas con virt-manager, pero es opcional cuando migran con virsh. Los protocolos soportados por los NSP incluyen:
 - Fibre Channel-based LUNs
 - iSCSI
 - NFS
 - GFS2
 - SCSI RDMA protocols (SCSI RCP)

Crear un SP basado en un disco usando virsh.

1. Crear una etiqueta GTP (GUID partition table) en el disco:

```
#  
parted /dev/sdb  
GNU Parted 2.1  
  
Using /dev/sdb Welcome to GNU Parted! Type 'help' to view a list of commands.  
  
(parted) mklabel  
New disk label type? gpt  
(parted) quit  
Information: You may need to update /etc/fstab.  
#
```

2. Crear archivo de configuración del SP

Crear un archivo XML temporal conteniendo la información del SP requerida por el nuevo dispositivo. El archivo debe contener el formato mostrado abajo y contener los siguientes campos:

`<name>guest_images_disk</name>` El parámetro determina el nombre del SP.
`<device path = '/dev/sdb' />` Especifica el path donde se almacena el dispositivo.
`<target> <path>/dev</path></target>` Determina la localización en el host físico donde es unido el volúmen creado con el SP.

<formattype= 'gpt' /> Especifica el tipo de tabla de la partición.

A modo de ejemplo se tiene:

```
<pool type='disk'>
  <name>guest_images_disk</name>
  <source>
    <device path='/dev/sdb' />
    <format type='gpt' />
  </source>
  <target>
    <path>/dev</path>
  </target>
</pool>
```

3. Asignar el dispositivo

Añadir la definición del storage pool usando el comando virsh pool-define con la configuración XML creada en el paso anterior.

```
# virsh pool-define ~/guest_images_disk.xml
Pool guest_images_disk defined from /root/guest_images_disk.xml
# virsh pool-list --all
Name      State      Autostart
-----
default   active     yes
guest_images_disk   inactive   no
```

4. Iniciar el storage pool

Iniciar el storage pool con el comando virsh pool-start. Verificar que el pool es iniciado con el comando virsh pool-list -all.

```
# virsh pool-start guest_images_disk
Pool guest_images_disk started
# virsh pool-list --all
Name      State      Autostart
-----
default   active     yes
guest_images_disk   active     no
```

5. Habilitar inicio automático

El inicio automático configura el servicio libvirtd para iniciar el storage pool cuando el servicio inicia.

```
# virsh pool-autostart guest_images_disk
Pool guest_images_disk marked as autostarted
# virsh pool-list --all
Name      State      Autostart
-----
default    active    yes
guest_images_disk    active    yes
```

6. Verificar la configuración del storage pool

Para verificar que el storage pool fue creado correctamente, los tamaños reportados correctamente y el estado sea 'running', ejecutar `virsh pool-info`.

```
# virsh pool-info guest_images_disk
Name:          guest_images_disk
UUID:         551a67c8-5f2a-012c-3844-df29b167431c
State:        running
Capacity:     465.76 GB
Allocation:   0.00
Available:    465.76 GB
# ls -la /dev/sdb
brw-rw----. 1 root disk 8, 16 May 30 14:08 /dev/sdb
# virsh vol-list guest_images_disk
Name      Path
-----
```

7.8.1. Borrar un storage pool utilizando virsh

1. Para evitar inconvenientes con otras máquinas virtuales huéspedes utilizando el mismo pool, lo mejor es pararlo y liberar los recursos usados por el mismo:

```
# virsh pool-destroy guest_images_disk
```

2. Eliminar las definiciones del storage pool:

```
# virsh pool-undefine guest_images_disk
```

7.8.2. Crear sotrage pools basados en directorios con virsh

1. Crear la definición del storage pool

Usar el comando `virsh pool-define-as` para definir un nuevo storage pool. Hay dos opciones requeridas para la creación de un storage pool basado en directorio:

- El nombre del storage pool

- El path a un sistema de directorios de archivos para guardar los archivos imágenes del huésped. Si el directorio no existe, virsh lo creará.

Este ejemplo utilizat el directorio /guest_images.

```
# virsh pool-define-as guest_images dir - - - "/guest_images" Pool guest_images defined
```

2. Verificar que el storage pool aparece en la lista

Verificar que el storage pool es creado correctamente y el estado lo muestra como inactivo.

```
# virsh pool-list --all
```

Name	State	Autostart
default	active	yes
guest_images	inactive	no

3. Crear el directorio local

Usar el comando virsh pool-build para construir el storage pool basado en directorio para el directorio guest_images (por ejemplo) como se muestra:

```
# virsh pool-build guest_images
Pool guest_images built
# ls -la /guest_images
total 8
drwx----- . 2 root root 4096 May 30 02:44 .
dr-xr-xr-x. 26 root root 4096 May 30 02:44 ..
# virsh pool-list --all
```

Name	State	Autostart
default	active	yes
guest_images	inactive	no

4. Iniciar el storage pool

Usar el comando pool-start para habilitar un directorio de storage pool, permitiendo que los volúmenes permitidos del pool sean usados como imágenes de disco de los huéspedes.

```
# virsh pool-start guest_images
Pool guest_images started
# virsh pool-list --all
Name      State      Autostart
-----
default   active    yes
guest_images   active    no
```

5. Habilitar inicio automático

El inicio automático configura el servicio libvirtd para iniciar el storage pool cuando el servicio inicia.

```
# virsh pool-autostart guest_images
Pool guest_images marked as autostarted
# virsh pool-list --all
Name      State      Autostart
-----
default    active    yes
guest_images    active    yes
```

6. Verificar la configuración del storage pool

```
# virsh pool-info guest_images
Name: guest_images
UUID: 779081bf-7a82-107b-2874-a19a9c51d24c
State: running
Persistent: yes
Autostart: yes
Capacity: 49.22 GB
Allocation: 12.80 GB
Available: 36.41 GB
```

```
# ls -la /guest_images
total 8
drwx----- 2 root root 4096 May 30 02:44 .
dr-xr-xr-x. 26 root root 4096 May 30 02:44 ..
#
```

Para verificar que el storage pool fue creado correctamente, los tamaños reportados correctamente y el estado sea '**running**'. Si se quiere que el pool sea accesible incluso si la máquina virtual huésped no está corriendo, asegurarse que **Persistent** sea indicado como **yes**. Si se quiere que el pool inicie automáticamente cuando comience el servicio, asegurarse que **Autostart** esté indicado como **yes**.

7.8.3. Borrar un storage pool utilizando virsh

1. Para evitar inconvenientes con otras máquinas virtuales huéspedes utilizando el mismo pool, lo mejor es pararlo y liberar los recursos usados por el mismo:

```
# virsh pool-destroy guest_images_disk
```

2. Opcionalmente, si se quiere eliminar el directorio donde el storage pool reside, utilizar el siguiente comando:

```
# virsh pool-delete guest_images_disk
```

3. Eliminar la definición del storage pool:

```
# virsh pool-undefine guest_images_disk
```

8. Cobbler

Para poder alojar las imágenes de los sistemas operativos, el servidor Cobbler debe contar con al menos 25 GB de disco. Para evitar inconvenientes, deshabilitar SELinux y el firewall, de la siguiente forma. Editar el archivo `/etc/sysconfig/selinux` y setear:

```
SELINUX=disabled
```

En el caso del firewall ejecutar:

```
systemctl stop firewalld.service  
systemctl mask firewalld.service  
systemctl status firewalld.service
```

O bien, si no se desea desactivarlo, permitir el acceso al los siguientes puertos de http 80/443, cobbler 69 y 25151.

8.1. Instalación.

Primero y principal, Cobbler necesita Python, alguna versión superior a la 2.6. Además, requiere la instalación de los siguientes paquetes:

- createrepo
- httpd (apache2 for Debian/Ubuntu)
- mkisofs mod_wsgi (libapache2-mod-wsgi for Debian/Ubuntu)
- mod_ssl (libapache2-mod-ssl)
- python-cheetah
- python-netaddr
- python-simplejson
- python-urlgrabber
- PyYAML (python-yaml for Debian/Ubuntu)
- rsync
- syslinux
- tftp-server (atftpd for Debian/Ubuntu)
- yum-utils

Mientras que cobbler web solo requiere Django (python-django para Debian/Ubuntu).

Entonces:

```
yum update
```

```
yum install *
```

Se debe añadir los repositorios necesarios para la instalación de Cobbler:

```
wget https://dl.fedoraproject.org/pub/epel/epel-release-latest-7.noarch.rpm
rpm -Uvh epel-release-*
```

Instalar Cobbler junto con:

```
yum install cobbler cobbler-web dhcp pykickstart system-config-kickstart dhcp
tftp httpd xinetd fence-agents-all -y
```

8.2. Configuración.

Las siguientes configuraciones deberían ser realizadas antes de comenzar a usar Cobbler.

Editar /etc/xinetd.d/tftp modificando disable = yes por no. Luego ejecutar:

```
systemctl start rsyncd
```

```
systemctl enable rsyncd
```

Configurar DHCP Copiar el archivo de configuración de ejemplo:

```
cp /usr/share/doc/dhcp-4.2.5/dhcpd.conf.example /etc/dhcp/dhcpd.conf
```

Verificar en cada caso la versión de dhcp.

Luego editar /etc/dhcp/dhcpd.conf y modificarlo como sea necesario, por ejemplo se tiene:

```
# A slightly different configuration for an internal subnet.

subnet 192.168.100.0 netmask 255.255.255.0 {
    range 192.168.100.100 192.168.100.254;
    option domain-name-servers puppet;
    option domain-name "localdomain";
    option routers 192.168.100.1;
    option broadcast-address 192.168.100.255;
    default-lease-time 600;
    max-lease-time 7200; }
```

Añadir a /etc/hosts la dirección del servidor y su hostname (puppet). Luego configurar el parámetro ServerName en /etc/httpd/conf/httpd.conf con el nombre del host, en este caso será puppet.

ServerName puppet

Inicio y activación de los servicios para inicio automático:

```
systemctl start httpd.service
systemctl start dhcpcd.service
systemctl start xinetd.service
systemctl start cobblerd.service
```

```
systemctl enable httpd.service
systemctl enable dhcpcd.service
systemctl enable xinetd.service
systemctl enable cobblerd.service
```

Al momento de ingresar al servidor por medio de la página web, se pedirá usuario y contraseña. Por defecto, la contraseña y usuario del servidor es “cobbler”. Entonces, se genera una nueva contraseña encriptada con:

```
openssl passwd -1
```

Lo que dará algo similar a:

```
Password: Verifying - Password: $1$U.Svb2gw$MNHrAmG.axVHYQaQRySR5/
```

Es necesario editar el archivo `/etc/cobbler/settings` para cambiar la línea “`default_password_crypt`” con la nueva contraseña generada.

```
default_password_crypted: "$1$U.Svb2gw$MNHrAmG.axVHYQaQRySR5/"
```

Para habilitar la interfaz web de Cobbler y configurar usuario y contraseña, modificar las siguientes líneas del archivo `/etc/cobbler/modules.conf` para que queden de este modo:

```
[authentication] module = authn_configfile
[authorization] module = authz_allowall
```

Ahora, para cambiar el usuario y la contraseña para la interfaz web, correr el siguiente comando e ingresar la contraseña preferida dos veces:

```
htdigest /etc/cobbler/users.digest "Cobbler" admin
```

En este caso el usuario es admin y la contraseña se ingresa luego de ejecutar el comando. Por defecto se tiene usuario cobbler y contraseña cobbler.

Luego modificar “`manage_dhcp: 0`” para habilitar que cobbler administre DHCP:

```
manage_dhcp: 1
```

Configurar ahora la dirección IP del servidor Cobbler en las variables “`server`” y “`next_server`”, por ejemplo:

```
next_server: 192.168.100.200
```

```
server: 192.168.100.200
```

El siguiente paso es modificar el archivo `/etc/cobbler/dhcp.template` y realizar los cambios necesarios:

```
subnet 192.168.100.0 netmask 255.255.255.0 {
    option routers 192.168.100.1;
    option domain-name-servers 192.168.100.1;
    option subnet-mask 255.255.255.0;
    range dynamic-bootp 192.168.100.100 192.168.100.254;
    default-lease-time 21600;
    max-lease-time 43200;
    next-server $next-server;
    class "pxeclients" { match if substring (option vendor-class-identifier, 0, 9) = "PXEClient";
        if option pxe-system-type = 00:02 {
            filename "ia64/elilo.efi";
        } else if option pxe-system-type = 00:06 {
            filename "grub/grub-x86.efi"; }
        else if option pxe-system-type = 00:07 {
            filename "grub/grub-x86_64.efi"; }
        else { filename "pxelinux.0";
        }
    }
}
```

Editar el archivo `/etc/debmirror.conf` comentando lo siguiente:

```
#@dists="sid";
#@arches="i386";
```

El próximo paso es descargar los “network boot loaders” con el comando `cobbler get-loaders`.

Por último reiniciar los servicios:

```
systemctl restart httpd.service
systemctl restart dhcpcd.service
systemctl restart xinetd.service
systemctl restart cobblerd.service
```

Sincronizar `cobbler sync`.

8.3. Importar imágenes ISO al servidor Cobbler

Para hacer esto se utiliza el comando mount. Primero crear un directorio y luego montar el archivo ISO:

```
mkdir /mnt/centos
```

```
mount -t iso9660 -o loop,ro /path/to/isos/CentOS-7-x86_64-DVD-1503-01.iso /mnt/centos
```

Luego ejecutar:

```
cobbler import --name=centos7 --arch=x86_64 --path=/mnt/centos
```

Ésto creará una copia local en el servidor, dando lugar a un nuevo objeto “distro” y “profile”. Los cuales se pueden verificar con:

```
cobbler distro list
```

```
cobbler profile list
```

8.4. Tópicos generales de Cobbler

8.4.1. Modelado

Cobbler utiliza objetos para definir la configuración de aprovisionamiento. A medida que se desciende por el árbol de objetos, las variables se sobre escriben y se añaden a la información definida en los objetos superiores.

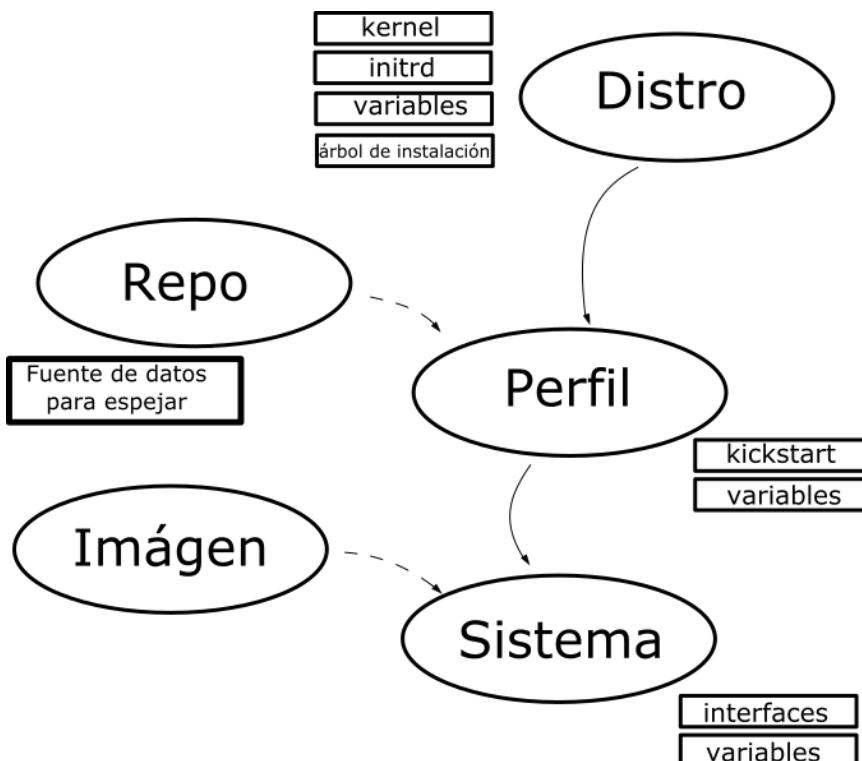


Figura 27: Modelado de Cobbler

8.4.2. Distros

Distribución que se desea instalar. Importar el contenido de la distro ayuda a disminuir el tiempo de instalación ya que no se utilizan fuentes de instalación externas. Generalmente es más fácil utilizar el comando *import* en vez de añadir la distribución manualmente.

8.4.3. Profiles

Un perfil o profile asocia una distribución a opciones especializadas adicionales, como puede ser un kickstart. Los perfiles son el núcleo del aprovisionamiento y debe existir al menos uno por cada distribución. Un perfil puede representar, por ejemplo, una configuración de web server o de escritorio.

8.4.4. Systems

Los grabaciones de sistemas mapean una pieza de hardware (o una máquina virtual) con el profile asignado a correr en ella. Esto puede verse como una forma de asignarle un rol a un sistema específico. Cuando se aprovisiona vía koan y PXE, no es necesario crearlos ya que son útiles cuando una personalización de un sistema específico es necesaria. Por ejemplo, personalizar la MAC, si hay un rol específico para una máquina dada, se debería crear una grabación del sistema para ésta.

8.4.5. Images

Cobbler puede booteear imágenes físicamente o virtualmente. Los despliegues de máquinas no basadas en imágenes son generalmente más fáciles para trabajar y llevan a una infraestructura más sustentable. La mayoría de las instalaciones de cobbler están directamente basadas en la distribución (kernel + initrd). La siguiente página documenta algunas cosas que no están basadas en kernel + initrd y muestra como instalarlas con cobbler y koan. Por ejemplo, trata la instalación de sistemas operativos Windows usando qemu/KVM:

<https://fedorahosted.org/cobbler/wiki/AllAboutImages>

<https://fedorahosted.org/cobbler/wiki/KoanWithIsos>

8.4.6. Repositorios

Espejar repositorios le permite a Cobbler espejar el árbol de instalación (`cobbler import`) y también paquetesopcionales. Si se espeja todo esto localmente en la red, las instalaciones y actualizaciones serán más rápidas (usualmente es válido realizar esto para largos setups en datacenters, laboratorios, etc). Si un profile tiene un repositorio dado, este repositorio puede ser automáticamente configurado durante el aprovisionamiento y los sistemas instalados podrán usarlo como espejo (`yum_post_install_mirror` debe estar habilitado). Si se especifica una lista de paquetes para `-rpm-list`, se puede espejar solo esa parte del repositorio, más sus dependencias. Por ejemplo, si se espeja FC6 Extras, para descargar Cobbler y Koan, ponemos `-rpm-list='cobbler koan'` y se saltea la parte de los paquetes de juegos. Esta función sólo funciona para repositorios http o ftp.

Los repositorios pueden ser creados del siguiente modo:

```
cobbler repo add --mirror=url --name=string [--rpmlist=list]
[--createrepo-flags=string] [--keep-updated=Y/N] [--priority=number] [--arch=string]
[--mirror-locally=Y/N] [--breed=yum|rsync|rhn]
```

Donde

mirror: Es la dirección del espejo yum. Puede ser una URL rsync:// , una ubicación ssh o una ubicación http:// o ftp:// de un espejo. Direcciones del filesystem también funcionan. Esta dirección debe especificar un repositorio exacto a espejar, solo una arquitectura y una distribución.

name: Este nombre es el usado para guardar la ubicación del espejo.

rpm-list: Con esta opción se puede decidir espejar solo una parte de un repositorio (la lista de paquetes dados más dependencias). Por ejemplo : --rpm-list="paquete_1 paquete_2". Esta opción sólo funciona con repositorios http:// y ftp:// para espejos de otros tipos esta opción será ignorada.

createrepo-flags: Especifica banderas opcionales para añadir a la herramienta createrepo la cual es llamada cuando se ejecuta cobbler reposync para el repositorio dado. Por defecto se tiene '-c cache'.

keep-updated: Especifica si el repositorio debería ser o no actualizado durante una ejecución normal de cobbler reposync. El repositorio puede seguir siendo actualizado por el nombre.

mirror-locally: Cuando se configura a N, especifica que este repositorio yum se utiliza para ser referenciado directamente por kickstarts y no para ser espejado localmente en el servidor cobbler. Solo espejos con URLs http:// y ftp:// son soportados cuando se utiliza --mirror-locally=N, no se puede usar URLs del filesystem.

priority: Especifica la prioridad del repositorio (menor número, mayor prioridad) que se aplica a máquinas instaladas usando los repositorios que tienen el plugin yum priorities instalado. Por defecto se tiene 99.

arch: Especifica la arquitectura que el repositorio debería utilizar. Por defecto se utiliza la arquitectura del servidor cobbler.

breed: Usualmente cobbler comprenderá este parámetro si no se entrega.

Para crear un repositorio local, por ejemplo para instalar Puppet en una instalación desde cero, y sin una conexión a Internet, primero es necesario tener los paquetes necesarios y sus dependencias, para ello se ejecuta:

```
sudo yum install --downloadonly --downloaddir=<directory> <package>
```

Donde se debe reemplazar <directory> por el directorio donde se descargará el paquete con sus dependencias y <package> por el puppet.

Una vez obtenidos, crear una carpeta con el nombre del repositorio en /var/www/cobbler/repo_mirror por ejemplo:

```
sudo mkdir /var/www/cobbler/repo_mirror/puppet
```

Luego es necesario añadirlo al servidor:

```
cobbler repo add --name=puppet --keep-updated=N --arch=x86_64 --mirror-locally=Y
--breed=yum
```

Donde `--name` debe ser el mismo que el de la carpeta creada anteriormente. Acto seguido ejecutar:

```
createrepo /var/www/cobbler/repo_mirror/puppet
cobbler reposync
```

Para añadir este nuevo repositorio a un profile de instalación existente:

```
cobbler profile edit --name=centos7 --repos=puppet
```

Se muestra la información acerca del mismo con:

```
cobbler repo report --name=puppet
```

8.4.7. Repositorio local separado de Cobbler.

En el caso que se desee crear un repositorio local que no dependa del servidor Cobbler, se debe primero instalar los servicios necesarios para el funcionamiento del servicio de repositorios:

```
sudo yum install -y createrepo vsftpd lftp
```

Una vez hecho ésto, se tiene que crear el árbol de directorios adecuado de acuerdo a la aplicación. Éste puede estar dividido como se desee, por ejemplo, por sistema operativo, por distribución, por arquitectura, etc.

Se recomienda utilizar como base el directorio `/var/ftp` dado que se utilizará este sistema de transmisión de archivos.

Una vez creado el sistema de archivos, se guardan los correspondientes paquetes rpm en los lugares adecuados, acordes a cómo se haya creado el árbol de archivos, y se ejecuta el siguiente comando:

```
createrepo -v /var/ftp/nombre_repo.repo
```

Es necesario entonces configurar el demonio `vsftpd` editando el archivo `/etc/vsftpd/vsftpd.conf`. Hay diversas configuraciones posibles, pero hay dos puntos importantes que deben existir:

1. `anonymous_enable=YES`: Indica que se puede acceder vía ftp de manera anónima.
2. `anon_root=/var/ftp`: Indica la raíz del directorio al cual se puede acceder de manera anónima.

Además, añadir la regla adecuada al firewall para permitir el acceso al puerto 21 o en su defecto, desactivar el firewall.

Por último, iniciar el servicio:

```
systemctl start vsftpd
```

En el lado del cliente es necesario informar del nuevo repositorio. Para ésto, crear un archivo en `/etc/yum.repos.d/nombredelrepo.repo` con el siguiente contenido:

```
[nombredelrepo]
name=nombredelrepo
comment ="Repositorio local para proyecto integrador"
baseurl=ftp://IP_servidor/nombre_del_sistema_de_archivos
gpgcheck=0
enabled=1
```

Para utilizar este repositorio primero hay que habilitarlo:

```
yum --enablerepo="nombredelrepo.repo"
```

Luego es necesario actualizar base de datos de repositorios:

```
yum makecache
```

Al momento de instalar un paquete, yum pedirá la clave pública GPG, si no se tiene instalada, se puede importar con el comando:

```
rpm --import public.gpg.key
```

En caso de no se tenga clave pública alguna, es posible desactivar la verificación de la misma en `/etc/yum.conf` poniendo a cero el valor de `gpgcheck`.

Es posible utilizar sólo el repositorio recién creado y excluir los demás, para ello:

```
yum --disablerepo=* --enablerepo=nombredelrepo install paquete
```

8.4.8. Buildiso

Frecuentemente un entorno no puede soportar PXE porque otro grupo posee el control sobre las configuraciones DHCP y no entregará una entrada de `next-server` o sólo se están usando IPs estáticas. Esto se soluciona fácilmente:

```
# cobbler buildiso
```

Este comando copia todos el kernel/initrd de la distro a una 'imagen de CD' booteable y genera un menú para la ISO que es esencialmente equivalente al menú PXE provisto para la instalación de máquinas por red vía Cobbler. Por defecto el menú del CD booteable va a incluir todos los profiles y systems.

Si se necesita instalar en un laboratorio u otro ambiente que no tenga acceso por red al servidor cobbler, se puede copiar completamente el árbol de la distribución más el profile y los systems records a una imagen.

```
# cobbler buildiso --standalone -distro="distro1"
```

8.4.9. Import

El propósito de “`cobbler import`” es configurar un servidor de instalación por red para una o más distribuciones. Éste espeja contenido basado en un imagen DVD, un archivo ISO, un árbol en un filesystem, un espejo externo rsync o una ubicación SSH.

```
$ cobbler import --path=/path/to/distro -name=F12
```

Este ejemplo muestra los dos argumentos requeridos para import: `--path` y `--name`.

Luego de que import es ejecutado, cobbler tratará de detectar el tipo de distribución y automáticamente asignar kickstarts. Por defecto, proveerá el sistema borrando el disco duro, configurando eth0 para DHCP y utilizando la contraseña por defecto “cobbler”. Si esto no es deseado, editar los archivos kickstart en `/var/lib/cobbler/kickstarts` para hacer algo distinto o cambiar la configuración del kickstart después que cobbler cree el profile. El contenido espejado es guardado automáticamente en `/var/www/cobbler/ks_mirror`.

Ejemplos:

1. `cobbler import --path=rsync://mirrorserver.example.com/path/ --name=fedora --arch=x86`
2. `cobbler import --path=root@192.168.1.10:/stuff --name=bar`
3. `cobbler import --path=/mnt/dvd --name=baz --arch=x86_64`
4. `cobbler import --path=/path/to/stuff -name=glorp`
5. `cobbler import --path=/path/where/filer/is/mounted --name=anyname \ --available-`

Una vez importado, ejecutar “`cobbler list`” o “`cobbler report`” para ver que se ha añadido. Si se quiere forzar la utilización de una plantilla kickstart de cobbler para todos los profiles creados por un import, se puede pasar la opción `-kickstart` a import para saltar la auto detección del kickstart.

8.4.10. Kickstarts

Los kickstarts son archivos que indican cómo debe ser configurado el sistema operativo, el archivo contiene palabras claves, valores y en otros casos solo contienen la palabra clave que en sí misma es una configuración específica.

Algunas palabras clave (keywords) son opcionales, mientras que otras son necesarias para la instalación.

Keywords

- **autopart (optional)** : Creación automática de particiones, 1 GB o más para el directorio raíz (/), una partición de intercambio y una partición de arranque apropiada para la arquitectura . Uno o más de los tamaños de las particiones por defecto puede ser redefinido con la zona de directivas.
- **ignoredisk (optional)** : Hace que el instalador ignore los discos especificados.

La sintaxis es:

```
ignoredisk -drives=drive1,drive2,...
```

- **auth or authconfig (required)** :Establece las opciones de autenticación para el sistema. Es similar al comando authconfig , que se puede ejecutar después de la instalación . Por defecto, las contraseñas son encriptadas y no utilizan shadow .
- **bootloader (required)**: Especifica cómo se debe instalar el gestor de arranque.
- **clearpart (optional)** : Elimina las particiones del sistema, antes de la creación de nuevas particiones . Por defecto no se eliminan las particiones .

- **cmdline (optional)** : Realiza la instalación en un modo de línea de comandos completamente no interactivo. Cualquier solicitud por interacciones detendrá la instalación.

- **device (optional)** : El comando de dispositivo , indica al programa de instalación para instalar módulos adicionales , es en este formato :

```
device <type><moduleName> -opts=<options>
```

- **driverdisk (optional)** : Disquetes de controladores se pueden usar durante instalaciones kickstart.

- **firewall (optional)** : Esta opción corresponde a la pantalla de configuración de firewall en el programa de instalación.

- **firstboot (optional)** : Determinar si el agente de configuración se inicia la primera vez que se arranca el sistema . Si se activa, el paquete firstboot debe estar instalado. Si no se especifica, esta opción está desactivada por defecto.

- **halt (optional)** : Detiene el sistema después de que la instalación se ha completado con éxito . Esto es similar a una instalación manual, en donde Aanaconda muestra un mensaje y espera a que el usuario presione una tecla antes de reiniciar. Durante una instalación Kickstart, si no se especifica el método de terminación, la opción reboot se utiliza como predeterminado.

- **graphical (optional)** : Realice la instalación kickstart en modo gráfico . Este es el valor predeterminado .

- **install (optional)** : Le dice al sistema para instalar un sistema nuevo en lugar de actualizar un sistema existente. Este es el modo por defecto.

- **ignore disk (optional)** : Se utiliza para especificar los discos que Aanaconda no debe tocar durante la partición , el formato, y la limpieza . Este comando tiene un único argumento necesario , que toma una lista separada por comas de nombres de unidad de ignorar.

```
ignoredisk -drives=[disk1,disk2,...]
```

- **interactive (optional)** : Utiliza la información proporcionada en el archivo kickstart durante la instalación, pero permite la inspección y modificación de los valores dados . Se le presentará con cada pantalla del programa de instalación con los valores del archivo kickstart . Puede aceptar los valores haciendo clic en Siguiente o cambiar los valores y haga clic en Siguiente para continuar.

- **key (optional)**: Especifique una clave de instalación, que es necesaria para ayudar en la selección de paquetes e identificar su sistema con fines de apoyo. Este comando es Red Hat Enterprise Linux específico.

- **keyboard (required)** : Establece el tipo de teclado.

- **lang (required)** : Establece el idioma que desea utilizar durante la instalación y el idioma predeterminado para utilizar en el sistema instalado.

- **logvol (optional)** : Crea un Logical Voume con la sintaxis:

```
logvol <mntpoint> -vgname=<name> -size=<size> -name=<name><options>
```

- **logging (optional)** : Este comando controla el registro de errores de Aanaconda durante la instalación. No tiene ningún efecto en el sistema instalado.

- **monitor (optional)** : Si no se da el comando monitor, Aanaconda utilizará X para detectar automáticamente la configuración del monitor.
- **network (optional)** : Configura la información de red para el sistema. Si la instalación no requiere redes y la información de la red no se proporciona en el archivo kickstart, el programa de instalación asume que la instalación debe hacerse sobre eth0 a través de una dirección IP dinámica (BOOTP / DHCP), y configura el sistema final, instalado para determinar su dirección IP de forma dinámica.
- **part or partition (required for installs, ignored for upgrades)** : Crea una partición en el sistema.
- **poweroff (optional)** : Apaga el sistema luego de que la instalación se complete exitosamente.
- **raid (optional)** : Monta un sistema RAID.
- **reboot (optional)** : Reinicia el sistema después de una instalación exitosa.
- **repo (optional)** : Configura un repositorio adicional YUM que puede ser utilizado como fuente para la instalación de paquetes.
- **rootpw (required)** : Establece la contraseña de root.
- **selinux (optional)** : Establece el estado del SELinux en el sistema instalado.
- **services (optional)** : Modifica el conjunto predeterminado de servicios que se ejecutarán bajo el nivel de ejecución predeterminado.
- **shutdown (optional)** : Apaga el sistema después de una instalación exitosa.
- **text (optional)** : Realiza la instalación kickstart en modo texto. Las instalaciones Kickstart se ejecutan en modo gráfico por defecto.
- **timezone (required)** : Selecciona la zona horaria del sistema.
- **upgrade (optional)** : Indica que se realiza una actualización del sistema instalado.
- **user (optional)** : Crea usuario en el sistema.
- **vnc (optional)** : Permite que la instalación gráfica pueda ser vista de forma remota a través de VNC.
- **volgroup (optional)** : Crea logical volume group con la sintaxis:
`volgroup <name><partition><options>`
- **zerombr (optional)** : Si se especifica zerombr, y si es su único argumento, cualquier tabla de partición no válidas que se encuentran en los discos son inicializadas. Esto destruye todos los contenidos de discos con tablas de partición inválidas.

8.4.11. Snippets

Los snippets son una forma de reutilizar bloques de código entre kickstarts (también funcionan en otros tipos de archivos). Esto quiere decir que cada vez que el texto SNIPPET aparezca en un archivo kickstart será reemplazado por los contenidos en el archivo correspondiente dentro de `/var/lib/cobbler/snippets/`. Esto permite la reutilización de código en cada plantilla, aliviando también la lectura de las mismas.

Para utilizar un snippet, es necesario crear un archivo en el directorio `/var/lib/cobbler/snippets/nuevo_snippet` y, en un archivo kickstart, al momento de llamar a esta porción de código se utiliza:

```
$SNIPPET('nuevo_snippet')
```

Los snippets pueden ser guardados en subdirectorios para una mejor organización. El orden de precedencia será como sigue:

```
/var/lib/cobbler/snippets/$subdirectorio/$nombre_del_snippet
```

Para referenciarlo desde el archivo kickstart, ahora se tiene:

```
$SNIPPET('direcorio/nuevo_snippet')
```

Cobbler no reconoce caracteres que no estén en el alfabeto inglés, por este motivo se recomienda no utilizar caracteres especiales como ñ u otros que lleven tilde.

8.4.12. Firewall

Dependiendo del uso, será necesario asegurar que está configurado para permitir el acceso a los servicios correctos. Un ejemplo de configuración es el siguiente:

```
# Firewall configuration written by system-config-securitylevel
# Manual customization of this file is not recommended.

*filter :INPUT ACCEPT [0:0]
:FORWARD ACCEPT [0:0]
:OUTPUT ACCEPT [0:0]
-A INPUT -p icmp -icmp-type any -j ACCEPT
-A INPUT -m state --state ESTABLISHED,RELATED -j ACCEPT
# LOCALHOST
-A INPUT -i lo -j ACCEPT
# SSH
-A INPUT -m state --state NEW -m tcp -p tcp --dport 22 -j ACCEPT
# DNS
- TCP/UDP -A INPUT -m state --state NEW -m udp -p udp --dport 53 -j ACCEPT -A
INPUT -m state --state NEW -m tcp -p tcp --dport 53 -j ACCEPT
# DHCP
-A INPUT -m state --state NEW -m udp -p udp --dport 68 -j ACCEPT
# TFTP
- TCP/UDP -A INPUT -m state --state NEW -m tcp -p tcp --dport 69 -j ACCEPT -A
INPUT -m state --state NEW -m udp -p udp --dport 69 -j ACCEPT
# NTP
-A INPUT -m state --state NEW -m udp -p udp --dport 123 -j ACCEPT
# HTTP/HTTPS
```

```

-A INPUT -m state --state NEW -m tcp -p tcp --dport 80 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 443 -j ACCEPT
# Syslog for cobbler
-A INPUT -m state --state NEW -m udp -p udp --dport 25150 -j ACCEPT
# Koan XMLRPC ports
-A INPUT -m state --state NEW -m tcp -p tcp --dport 25151 -j ACCEPT
-A INPUT -m state --state NEW -m tcp -p tcp --dport 25152 -j ACCEPT
#-A INPUT -j LOG
-A INPUT -j REJECT --reject-with icmp-host-prohibited
COMMIT

```

8.4.13. SELinux

Si no se desea desactivarlo, para permitir el acceso del servidor web Apache por SELinux se puede hacer:

```
setsebool -P httpd_can_network_connect true
```

8.4.14. PXE

La instalación en máquinas “bare metal” desde la red utilizando PXE es directa. Se necesita configurar DHCP:

- Si el servidor DHCP está en otro lado y no en el servidor Cobbler, se debe setear “next-server” para especificar el servidor Cobbler.
- Si se está corriendo DHCP localmente y se quiere que Cobbler lo administre, se debe configurar la variable manage_dhcp a 1 en `/etc/cobbler/settings`, editar `/etc/cobbler/dhcp.template` para cambiar configuración por defecto y ejecutar “`cobbler sync`”.

Una vez que se tenga el PXE configurado, todos los profiles compatibles aparecerá por nombre en el menú de booteo PXE. Se puede seleccionar uno de la lista o por defecto la máquina booteará localmente. Si se quiere adjuntar un sistema particular a un profile particular la próxima vez que reinicie, se debe ejecutar:

```
cobbler system add --name=example --mac=$mac-address --profile=$profile-name
```

Luego la máquina booteará directamente con el profile seleccionado sin mostrar el menú.

8.4.15. Reinstalación

Si se necesita reinstalar un sistema operativo a una máquina que tiene corriendo uno distinto, se puede utilizar:

```
yum install koan
koan --server=bootserver.example.com --list=profiles koan --replace-self --server=boot
--profile=F12-i386 /sbin/reboot
```

El sistema instalará el nuevo sistema operativo luego del reinicio, sin interacción requerida.

8.4.16. Virtualización

Si se quiere instalar un huésped virtual (KVM o Xen) se puede hacer:

```
yum install koan
```

```
koan --server=bootserver.example.com --virt --virt-type=xenpv -profile=F12-i386-xen
```

Se puede utilizar KVM u otro método de virtualización.

8.4.17. Integración con Puppet

Este ejemplo es relativamente avanzado, involucrando “`mgmt-classes`” de Cobbler para controlar diferentes tipos de configuración inicial. Pero si en cambio se opta por poner la mayor parte de la configuración inicial en Puppet en vez de aquí, entonces podría ser más simple.

Manter class mappings en cobbler Primero se debe asignar “`management classes`” a la distro, profile o system.

```
cobbler distro edit --name=distro1 --mgmt-classes="distro1"
cobbler profile add --name=webserver --distro=distro1 --mgmt-classes="webserver
likes_llamas" --kickstart=/etc/cobbler/my.ks
cobbler system edit --name=system --profile=webserver --mgmt-classes="orange"
--dns-name=system.example.org
```

Para Puppet el `--dns-name` (mostrado arriba) debe estar configurado porque esto es lo que Puppet estará enviando a Cobbler y es como encontrará el sistema. Puppet no tiene conocimiento sobre el nombre del sistema objeto en Cobbler. Para hacerlo de forma segura, probablemente se utilice FQDN aquí (lo cual es lo que se quiere si se utiliza cobbler para administrar DNS).

External Nodes Cobbler provee uno, así configura Puppet para usar `/usr/bin/cobbler-ext-nodes`:

```
[main]
```

```
external_nodes = /usr/bin/cobbler-ext-nodes
```

y también añadir lo siguiente al archivo de configuración:

```
node_terminus = exec
```

Ésto es un script simple que toma el información en la siguiente URL, la cual es una URL que siempre retorna un documento YAML en la forma que Puppet espera que sea retornado. Este archivo contiene todos los parámetros y clases que están para ser asignadas en el nodo en cuestión. Esta URL de Cobbler es: <http://cobbler/cblr/svc/op/puppet/hostname/foo>

y esto retornará datos como:

```
--- classes:
  - distro1
  - webserver
```

```

    - likes_llamas
    - orange
parameters:
  tree: 'http://.../x86_64/tree'

```

Estos parámetros vienen de todo lo que Cobbler monitorea en “`--ks-meta`” (también es un parámetro). De este modo se puede fácilmente añadir parámetros como añadir clases y mantener todo organizado en un lugar. En caso de tener parámetros o clases globales para añadir, esto se puede hacer editando los siguientes campos en `/etc/cobbler/settings`:

```

mgmt_classes: []
mgmt_parameters:
  from_cobbler: 1

```

8.4.18. Replicate

Este comando descarga la configuración de un servidor Cobbler a otro. Sirve para tener implementaciones de High Availability, recuperación de desastres o para balanceo de carga.

```
cobbler replicate --master=master.example.org
```

Con los argumentos por defecto, solo la metadata de la distribución y del perfil es sincronizada. A continuación se muestra los argumentos que se le pueden pasar a Cobbler para que replique:

```
# cobbler replicate --help
Usage: cobbler [options]
Options: -h, --help show this help message and exit
--master=MASTER Cobbler server to replicate from.
--distros=PATTERN pattern of distros to replicate
--profiles=PATTERN pattern of profiles to replicate
--systems=PATTERN pattern of systems to replicate
--repos=PATTERN pattern of repos to replicate
--image=PATTERN pattern of images to replicate
--omit-data do not rsync data
--prune remove objects (of all types) not found on the master
```

Setup En cada servidor que será la réplica del master, instalar Cobbler normalmente y asegurarse que `/etc/cobbler/settings` y `/etc/cobbler/modules.conf` están configurados apropiadamente. Utilizar `cobbler check` para ver si existe algún error. El comando no modificará estos archivos.

Los archivos son transferidos por `rsync` (sobre `ssh`) o por `scp`, por lo que es necesario tener un agente `ssh` antes de utilizar el comando de réplica o si no, utilizar `authorized_keys` en el host remoto.

9. Puppet

El agente y el servidor se comunican vía HTTPS con verificación de cliente. El nodo maestro (servidor) provee una interfaz HTTPS con varios extremos disponibles. Cuando se pide o envía cualquier cosa al servidor, el agente hace un pedido HTTPS o a uno de esos extremos.

Client-verified HTTPS quiere decir que cada maestro o agente tiene un identificador por certificado SSL y examinan los certificados de sus contrapartes para decidir si permite un intercambio de información. Puppet incluye un constructor de certificado de autorización para administrar los certificados. Los agentes pueden pedir automáticamente los certificados vía la API HTTP del maestro. El administrador del nodo maestro puede usar el comando `puppet cert` para inspeccionar los pedidos y firmar nuevos certificados; los agentes pueden entonces descargar los certificados firmados.

9.1. Pre-instalación

Puppet usualmente corre bajo la arquitectura cliente-servidor, pero además, puede correr en una arquitectura autocontenido. La decisión determina que paquetes serán instalados y que configuraciones extra necesarias se harán.

Se toma la opción de utilizar la arquitectura cliente-servidor. Se debe completar la instalación y configuración de todos los puppet servers antes de instalar cualquier agente. El servidor necesariamente debe correr en un sistema basado en Unix.

9.1.1. Requerimientos de sistema y chequeo de versión de sistema operativo

- **Hardware:** El agente Puppet no tiene requerimientos particulares de hardware y corre prácticamente en cualquier computadora, sin embargo, el servidor es un recurso intensivo y debe ser instalado en un servidor robusto y dedicado. Como mínimo, el servidor debe tener dos procesadores y al menos 2GB de RAM, para administrar eficientemente mil nodos, debe poseer entre 2 y 4 procesadores y 4GB de RAM.
- **Sistemas operativos soportados:** Hay una gran variedad de distribuciones Linux que soportan puppet, entre ellas destaca la utilizada para la realización del Proyecto, CentOS 7.
- **Ruby:** Se soportan varias versiones de Ruby, pero se recomienda el uso de las versiones 2.1.x.
- **Librerías obligatorias:** Facter 2.4.3 o posterior, Hiera 2.0.0 o posterior, json gem cualquier versión moderna, rgen gem 0.6.6 o posterior.
- **Librerías opcionales:** msgpack gem es requerido si se utiliza msgpack racionalización.

9.1.2. Chequeo de la configuración de red

En un agent/master deployment se debe preparar la red para el tráfico de puppet.

- **Firewall:** Si no se deshabilita, el servidor debe permitir conexiones entrantes al puerto 8140 y los agentes deben ser capaces de conectarse a ese puerto.
- **Resolución de nombres:** Cada nodo debe tener un nombre único.

9.2. Instalación

Primero se debe instalar puppetserver. Para ello habilitar los paquetes de los repositorios de Puppet Labs:

```
sudo rpm -ivh https://yum.puppetlabs.com/puppetlabs-release-pc1-el-7.noarch.rpm
```

Instalar el puppet master con el siguiente comando:

```
yum install puppetserver
```

No iniciar el servicio aún.

9.2.1. Asignación de memoria

Por defecto, puppet server está configurado para usar 2GB de RAM pero si se quiere experimentar con puppet server en una VM se puede asignar tan poco como 512MB de memoria. Para cambiar la asignación de memoria se edita el archivo de configuración, que se encuentra en `/etc/sysconfig/puppetserver` y modificar la siguiente línea:

```
# Modify this if you'd like to change the memory allocation, enable JMX, etc
```

```
JAVA_ARGS="-Xms2g -Xmx2g"
```

Si se desea por ejemplo, utilizar 1536MB se debe reemplazar 2g por 1536m:

```
JAVA_ARGS="-Xms1536m -Xmx1536m"
```

Luego, para aplicar los cambios, se debe reiniciar el servicio. No obstante, no se recomienda utilizar puppetmaster con menos de 2GB.

9.2.2. Instalar el paquete puppet-agent

Según sea el sistema operativo sobre el que se instalará, se tiene:

Centos y derivados: Del mismo modo que para el servidor, es necesario añadir el repositorio y luego instalar:

```
sudo rpm -Uvh https://yum.puppetlabs.com/puppetlabs-release-pc1-el-7.noarch.rpm
```

```
sudo yum install puppet-agent
```

Ubuntu y derivados: Habilitar el repositorio según sea el caso como el siguiente ejemplo:

```
wget https://apt.puppetlabs.com/puppetlabs-release-pc1-wheezy.deb
```

```
sudo dpkg -i puppetlabs-release-pc1-wheezy.deb
```

```
sudo apt-get update
sudo apt-get install puppet-agent
```

Windows: Descargar el paquete instalador desde https://downloads.puppetlabs.com/windows/?_ga=1

- Arquitectura x64: Se recomienda el uso de `puppet-agent-<VERSION>-x64.msi` o `puppet-agent-<VERSION>-x86.msi`.
- Arquitectura x86: Debe utilizar `puppet-agent-<VERSION>-x86.msi`.

La instalación puede ser gráfica o automática. Para la última, desde el cuadro “Ejecutar”:

```
msiexec /qn /norestart /i puppet-agent-<VERSION>-x64.msi PUPPET_MASTER_SERVER=puppet
```

No iniciar el servicio aún.

Por defecto, el valor del hostname de servidor es `puppet`, si se nombró de otra forma a la máquina servidora del `puppet server`, se debe editar esto.

9.2.3. Configuraciones para los agentes

Básicas:

- `server`: El nombre del nodo maestro al cual se le pedirán los manifiestos. Por defecto es `puppet`
- `certname`: Nombre con el cual el nodo agente pide el certificado y se presenta al servidor.
- `environment`: Indica el entorno solicitado cuando se contacta al maestro. De cualquier forma, el maestro puede configurarse para ignorar esta configuración.

Comportamiento de la ejecución:

- `noop`: Si está habilitado, el agente no reializará ningún trabajo, en cambio mirará qué cambios tendría que realizar y lo reporta al servidor.
- `priority`: Permite asignar el valor “`nice`” para evitar que otras aplicaciones de la CPU no mueran por inanición mientras se aplican los catálogos.
- `report`: Indica si se deben enviar reportes, por defecto es “`true`” .
- `tags`: Limita a los agentes a correr recursos con ciertas etiquetas.
- `usecacheonfailure`: Se utiliza para tomar el último buen catálogo si el master no posee uno bueno.
- `prerun_command` y `postrun_command`: Comandos que se desean correr de cada lado de `puppet`

Comportamiento del servicio:

- `runinterval`: Indica cada cuanto tiempo el agente se contacta con el servidor para pedirle los manifiestos. Por defecto es 30 minutos.
- `waitforcert`: Indica al agente que persista si no puede obtener su certificado. Por defecto está habilitado.

Útiles cuando se ejecutan los agentes desde Cron:

- splay y splaylimit: Se utiliza para sincronizar el agente y el servidor si el primero utiliza un cron en lugar del demonio.
- daemonize: Se debe colocar esta opción en falso si se utiliza un cron.
- onetime: Sale luego de terminar el puppet actual. Debe ser “true” si se utiliza un cron.

9.2.4. Configuraciones para los servidores:

Básicas:

- dns_alt_names: Una lista de los hostnames de los servidores permitidos para usar cuando actúan como “masters environment”.
- path: Indica la ubicación del entorno.
- basemodulepath: Una lista de las ubicaciones que contienen módulos que pueden ser usados en todos los entornos.
- manifest: El principal punto de entrada para compilar los catálogos. Por defecto es “site.pp”.
- reports: Controlador de reportes que se usa.

Configuraciones de CA:

- ca: Si actúa como un autoridad de certificación.
- ca_ttl: Indica por cuánto tiempo son válidos los certificados.
- autosign: Indica si los certificados deben ser firmados de forma automática o manual.

Como ejemplo de configuración se tiene, en el nodo que corre puppetserver, a la configuración por defecto, añadir:

```
certname = puppet
server = puppet
runinterval = 2m
```

9.2.5. Ejecutar puppet

Para el puppet server, si es el único maestro en el deployment, o si actuará como el servidor de CA para un sitio con múltiples maestros, se debe ejecutar puppet como sigue:

```
sudo puppet master --verbose --no-daemonize
```

Esto creará el certificado de CA y el certificado del puppet master, con los nombres DNS apropiados incluidos. Una vez que se muestre en pantalla **Notice: Starting Puppet master version <VERSION>**, tipar Ctrl-C para matar el proceso.

Para corroborar la creación y firmado del servicio:

```
sudo puppet cert list --all
```

En caso que el maestro no haga las veces de CA, para solicitar un certificado, ejecutar:

```
sudo puppet agent --test --ca_server=<SERVER>
```

Luego en el nodo maestro que sí oficia de CA:

```
sudo puppet cert list
```

```
sudo puppet cert --allow-dns-alt-names sign <NAME>
```

Este último es para firmar el certificado. Luego en el nuevo maestro, ejecutar nuevamente

```
sudo puppet agent --test --ca_server=<SERVER>
```

 para recibir el certificado.

9.3. Tópicos generales de Puppet

9.3.1. Module

Un módulo o module, es un conjunto de código de Puppet enpaquetado junto con los otros archivos y datos, que se necesita administrar sobre algún aspecto del sistema. Consiste en una estructura predefinida de directorios que ayudan a Puppet a encontrar los contenidos del módulo. Para ver los módulos instalados se puede ejecutar:

```
puppet module list
```

Existe un repositorio público (The Puppet Forge) donde se pueden encontrar módulos hechos por la comunidad y también mantenidos por Puppet Labs.

Los módulos son auto-contenidos y separados. Su estructura de archivo le da a Puppet una forma consistente de localizar cualquier clase, plantillas, plugins y binarios requeridos para satisfacer la funcionalidad del módulo.

Todos los módulos accesibles por el puppet master están localizados en los directorios especificados por la variable 'modulepath' en el archivo de configuración de Puppet. Para encontrar esta variable en cualquier sistema con Puppet, se puede ejecutar:

```
puppet agent --configprint modulepath
```

9.3.2. Node group

Los grupos de nodos o node groups permiten segmentar todos los nodos de la infraestructura en grupos separados configurables basados en la información colectada por 'facter tool'.

9.3.3. Resources

Cada recurso o resource, describe algún aspecto de un sistema y su estado, como por ejemplo, un servicio que debería estar ejecutándose o un paquete que se quiere instalado. El bloque de código que describe un recurso se llama declaración de recurso (resource declaration). Estas declaraciones de recurso están escritas en código Puppet, un DLS (Domain Specific Language) construido en Ruby. El DLS de Puppet es un lenguaje declarativo en

vez de imperativo. Esto quiere decir que en vez de definir un proceso o un conjunto de comandos, el código de Puppet describe (o declara) solo el estado final deseado, y depende de proveedores integrados para lidiar con la implementación.

```
puppet resource tool -> puppet resource <type> <name>
```

Puppet incluye una variedad de tipos de recursos integrados, que permiten administrar varios aspectos de un sistema. Algunos de los tipos de recursos claves que generalmente se encuentran en un sistema son los siguientes:

user	Un usuario
group	Un grupo de usuario
file	Un archivo específico
package	Un paquete de software
service	Un servicio corriendo
cron	Un trabajo programado de cron
exec	Un comando externo
host	Un host

Una declaración de recurso seguirá un patrón como el de abajo:

```
tipo {'título':  
    atributo => 'valor',  
}
```

- **Título:** Es un string que identifica un recurso para el compilador de Puppet.

El título no tiene que coincidir con lo que va a administrar en el sistema, pero a menudo se desea eso.

Los títulos deben ser únicos por tipos de recursos, se puede tener un paquete y un servicios ambos con el mismo título, pero no dos servicios con ese título.

- **Atributos:** Los atributos describen el estado deseado para un recurso; cada atributo maneja algún aspecto del recurso.

Cada tipo de recurso tiene su propio juego de atributos. Muchos tipos de recursos tienen atributos claves y una gran cantidad de opcionales.

Todos los atributos declarados deben tener un valor; el tipo de dato del valor depende de los que acepte el atributo.

- **Comportamiento:** Una declaración de recurso agrega un recurso al catálogo y le dice a Puppet que administre el estado del recurso. Cuando Puppet aplica el catálogo compilado, lo que hará es:

- Leer el estado actual del recurso en el sistema objetivo.
- Comprar el estado actual con el deseado
- Si es necesario, realizar cambios para llevar el estado actual al deseado.

- **Recursos no administrados:** Si el catálogo no contiene un recurso, implica que Puppet ya no lo administra, pero no que lo “elimina”, si se desea eliminarlo, se debe aclararlo en su estado deseado.

- `ensure => absent`
- **Singularidad:** Puppet no permite que se declare un mismo recurso dos veces. Esto prevee conflictos de valores. Si múltiples clases requieren el mismo recurso se puede usar una clase o un recurso virtual para añadirlo al catálogo en múltiples lugares sin duplicar.
- **Relaciones y orden:** Por defecto, Puppet aplica los recursos sin seguir el orden en que fueron escritos. Esto, se puede desactivar con la opción de ordenado. Sin embargo, si un recurso debe ser aplicado antes o después de otro, se puede indicar una relación entre ellos. Incluso se puede indicar que cambios en un recurso causen que otro se refresque.
- **Cambios, eventos y reportes:** Si Puppet realiza cambios, en un recurso, registra esos cambios como eventos. Esos eventos aparecerán en el *log* y en el reporte de ejecución de puppet.
- **Independencia de alcance:** Los recursos no están sujetos a los alcances. Un recurso, en cualquier ámbito, se puede referenciar desde cualquier otro ámbito.
- **Atributos especiales de los recursos.**
 - **Name/Namevar:** Define un recurso en el sistema objetivo. Por ejemplo, el *name* de un servicio o paquete es el nombre por el cual las herramientas de paquetes o servicios lo reconocen o en el caso de un archivo, su namevar es el path. Esto es diferente al título, el cual identifica un recurso para el compilador de Puppet. Sin embargo, ellos a veces tienen el mismo valor. La separación de nombre y título permite administrar un recurso que mantiene su título, pero que tiene diferente nombre en diferentes plataformas. Por ejemplo, un servicio *ntp* en sistemas Red Hat tiene por nombre *ntpd* y en sistemas debian *ntp* .
 - **Ensure:** Esto generalmente maneja el aspecto más importante de un recurso en el sistema objetivo. Indica si el archivo existe, si el servicio está corriendo o parado, si el paquete está instalado, etc.

Tipos de recursos Todos los tipos tienen un atributo especial llamado namevar. Este es el atributo usado para identificar unívocamente un recurso en el sistema de destino. Si no se especifica un valor para el namevar, este valor es tomado por defecto según el título del recurso.

Ejemplo:

```
file { '/etc/passwd':
  owner => root,
  group => root,
  mode  => 644
}
```

En este código, */etc/passwd* es el título del recurso *file*, otros códigos de Puppet pueden hacer referencia al recurso como *File['/etc/passwd']* para declarar relación. Porque el path

es el *namevar* para el tipo file y si no se le provee un valor, toma uno por defecto que es */etc/passwd*.

Atributos: A veces llamados parámetros, determinan el estado deseado para un recurso. Cualquiera de ellos modifica directamente el sistema (internamente, las llamadas “propiedades”) o afectan cómo el recurso se comporta.

Proveedores (providers): Implementan el mismo tipo de recursos en diferentes tipos de sistemas, ellos suelen hacer esto llamando a comandos externos. Aunque Puppet seleccionará automáticamente un proveedor apropiado por defecto, se lo puede sobreescibir con el atributo *provider*. Por ejemplo, el recurso *package* de sistemas Red Hat tiene por defecto YUM como *provider*, pero se puede especificar *provider => gem* para instalar librerías de Ruby con *gem*.

Características (features): Son habilidades que algunos proveedores pueden no soportar. Generalmente una característica corresponderá con algunos valores permitidos por un recurso de un atributo, por ejemplo, si un paquete soporta la característica *purgeable*, se puede especificar *ensure => purged* para borrar los archivos de configuración instalados por el paquete.

Algunas de las referencias de tipo más importantes son las explicadas a continuación:

- **Computer:** Manejo de objetos de la computadora usando DirectoyService en un Mac OS X. Sirve para adjuntar políticas MCX a los objetos localhost creados.
- **Cron:** Instalar y manejar trabajos Cron. Todo Cron creado por Puppet requiere un comando y al menos un atributo de un periodo (horas, minutos, meses, etc). Mientras el nombre del Cron no es parte del trabajo actual, el nombre es almacenado en un comentario comenzando con *#Puppet Name:*. Ese comentario es usado para coincidir entadas crontab creadas por Puppet con un recurso Cron.
- **Exec:** Ejecuta comandos externos. Cualquier comando en un recurso Exec debe poder correr múltiples veces sin causar daños.
- **File:** El manejo de archivos incluye contenido, dueño, y permisos. El tipo archivo puede manejar archivos, directorios y enlaces simbólicos.
- **Filebucket:** Un repositorio para almacenar y recuperar archivos conformados por MD5 checksum. Puede ser local a cada nodo agente o centralizado en un servidor Puppet. Todos los Puppet Master proveen un servicio filebucket que los agentes pueden acceder vía HTTP.
- **Group:** Manejo de grupos. En muchas plataformas esto sólo puede crear grupos. La membresía de los grupos debe ser administrada por cada usuario individual.
- **Host:** Instalar y manejar entradas de hosts. Para muchos sistemas, esas entradas deben estar solo en */etc/hosts*, pero algunos SO tienen diferentes soluciones.
- **Interface:** Esto representa una interfaz de router o switch. Es posible gestionar el modo interfaz y las características switchport.
- **K5login:** Controla el archivo *.k5login* para un usuario.
- **Mailalias:** Crea un alias email en la base de datos de alias local.
- **Maillist:** Controla la lista de emails. Este tipo de recurso puede solo crear y remover listas, no puede reconfigurar listas actuales.

- **Mcx:** Manejo de objetos MCX usando DirectoryService en MAC OS X.
- **Mount:** Maneja de filesystems montados, incluyendo agregar la información de montaje a la tabla de montaje. El comportamiento actual depende del valor del parámetro *ensure*.
- **Notify:** Envío de un mensaje arbitrario al log del agente en tiempo de ejecución.
- **Package:** Manejo de paquetes. Hay una bifurcación básica en los paquetes soportados correctamente: Algunos tipos de paquetes como yum y apt pueden recuperar sus propios archivos de paquetes, mientras que otros no pueden. Para esos paquetes, se puede usar el parámetro *source* para poner el archivo adecuado.
- **Resources:** Este es un metatipo que puede controlar otro tipo de recursos. Cualquier metaparámetro especificado aquí será pasado a los recursos generados, por lo que puede purgar recursos no administrados.
- **Schedule:** Define el programa para Puppet. Los recursos pueden ser limitados por un programa usando el metaparámetro *schedule*.
- **SelBoolean:** Manejo de SELinux Boolean en sistemas que soportan SELinux.
- **SelModule:** Manejo de carga y descarga de módulos de política de SELinux en el sistema.
- **Service:** Controla servicios en ejecución. El soporte de este recurso varía ampliamente según el concepto de servicio de la plataforma.
- **Ssh_authorized_key:** Manejo de llaves autorizadas SSH. Actualmente solo dos tipos de llaves son soportadas.
- **Sshkey:** Instala y administra las llaves de host SSH.
- **Stage:** Un tipo de recurso para crear nuevos escenarios.
- **Tidy:** Remueve archivos base no deseados con criterios específicos.
- **User:** Administración de usuarios.
- **Vlan:** Administra VLANs en un router o switch
- **Yumrepo:** La descripción del lado del cliente de un repositorio YUM. La configuración de un repositorio se encuentra en */etc/yum.conf* y el archivo indicado por la opción *reposdir* en ese archivo.
- **Zfs:** Administración de ZFS. Crea, destruye y configura las propiedades en instancias ZFS.
- **Zpool:** Administración de zpools. Crea y elimina zpools. El proveedor no sincroniza, solo reporta diferencias. Soporta vdevs con espejos, raidz, logs y spares

9.3.4. Manifests

Un manifiesto o manifest, es un archivo de texto que contiene código Puppet y posee la extensión .pp. Para comprobar la sintaxis de un manifiesto se puede utilizar:

```
puppet parser validate <manifiesto.pp>
```

El parseador no retornará nada si no hay errores, en caso de que se detecte un error debe ser corregido antes de continuar. Si se trata de aplicar un manifiesto que no ha sido

declarado, no cambiará nada en el sistema. Para ésto se debe crear un .pp que contenga un sentencia:

```
include módulo::clase
```

Antes de aplicar cambios en el sistema, se puede utilizar la bandera `-noop` para compilar el catálogo (catálogo) y notificar los cambios que Puppet habría realizado si hubiera sido ejecutado sin `-noop`.

```
puppet apply --noop
```

9.3.5. Catálogos

Los manifiestos de Puppet pueden usar lógica condicional para describir muchas configuraciones de nodos como una. Antes de configurar un nodo, Puppet compila los manifiestos en un catálogo, el cual solo es válido para un único nodo y no contiene lógica ambigua.

Los catálogos son documentos estáticos los cuales contienen recursos y relaciones.

En la arquitectura estándar maestro/agente, los nodos solicitan los catálogos al Puppet Server, el cual los compila cuando son solicitados. Los agentes mantienen en caché sus más recientes catálogos, si al pedir el catálogo, el master falla al compilarlo, ellos reusaran su catálogo cacheado.

9.3.6. Classes

Una clase es un bloque de código Puppet con nombre. Una clase administrará generalmente un conjunto de recursos relacionados a una función simple o un componente del sistema. Las clases usualmente contienen otras clases; este anidamiento provee una forma estructurada de juntar funciones de clases diferentes como componentes de soluciones más grandes. Para utilizar una clase, se necesita definirla escribiendo una definición de clase y guardándola en un archivo manifiesto. Cuando Puppet se ejecuta, parseará este manifiesto y guardará la definición de clase; luego ésta puede ser declarada para aplicarla en los nodos de la infraestructura. En Puppet las clases son singleton, lo que quiere decir que una clase puede ser declarada sólo una vez en un nodo dado. Cuando se declara una clase:

```
include módulo::clase
```

`módulo` le indica a Puppet donde encontrar esa `clase`. Sin embargo, para la clase principal de un módulo, además de llevar el mismo nombre que el módulo mismo, Puppet reconoce el nombre especial del archivo `'init.pp'` como el manifiesto que contendrá la clase principal de un módulo.

9.3.7. Funciones

Hay dos tipos de funciones en Puppet, statements (declaraciones) y rvalues. Las statements no retornan argumentos, son utilizadas para hacer trabajos independientes como importar. Rvalues retornan valores y pueden ser usadas solo en un statement requiriendo un valor, como una asignación o una declaración case.

Las funciones se ejecutan en el Puppet master, no se ejecutan en el agente. Por lo tanto sólo tienen acceso a los comandos y datos disponibles en el nodo maestro.

Algunas de las funciones disponibles son:

- **alert:** (statement) Deja un mensaje en el log del servidor en el nivel de alerta.
- **assert_type:** (rvalue) Retorna el valor dado si este es una instancia del tipo dado, y levanta un error en caso contrario.
- **contain:** (statement) Contiene una o más clases dentro de la clase actual. Si alguna de estas clases están sin declarar, serán declaradas como si fueran llamadas con la función *include*.
- **create_resources:** (statement) Convierte un hash en un conjunto de recursos y los añade al catálogo.
- **crit:** (statement) Deja un mensaje en el log del servidor en el nivel crítico.
- **debug:** (statement) Deja un mensaje en el log del servidor en el nivel debug.
- **defined:** (rvalue) Determina si una clase dada o un tipo de recurso está definido. También puede determinar si un recurso específico está definido o si una variable ha sido asignada con un valor.
- **digest:** (rvalue) Retorna el valor de hash de un string dado usando la configuración digest_algorithm del archivo de configuración de Puppet.
- **each:** (rvalue) Aplica un bloque parametrizado a cada elemento en una secuencia de entradas seleccionadas del primer argumento y retorna el primero argumento.
- **emerg:** (statement) Deja un mensaje en el log del servidor en el nivel emergencia.
- **epp:** (rvalue) Evalúa una plantilla Embedded Puppet y retorna el texto renderizado resultante como un string.
- **err:** (statement) Deja un mensaje en el log del servidor en el nivel error.
- **fail:** (statement) Fallo con un error del parser.
- **file:** (rvalue) Carga un archivo desde un módulo y retorna sus contenidos como un string.
- **filter:** (statement) Aplica un bloque parametrizado a cada elemento en una secuencia de entradas del primer argumento y retorna un array o un hash con las entradas para cada bloque evalúa a *true*.
- **fqdn_rand:** (rvalue) genera un número entero aleatorio mayor o igual a cero y menor a MAX, combinando \$fqdn y el valor de SEED para aleatoriedad repetible. Ésto quiere decir que cada nodo obtendrá un número aleatorio diferente de esta función, pero, el resultado de un nodo dado, será el mismo cada vez a menos que su hostname cambie (uso: fqdn_rand(MAX, [SEED])).
- **generate:** (rvalue) Llama a un comando externo en el Puppet master y retorna los resultados del comando.
- **hiera:** (rvalue) Realiza una búsqueda de prioridad estándar y retorna el valor más específico para una clave dada.
- **hiera_array:** (rvalue) Retorna todas las coincidencias a través de la jerarquía como un array plano de valores únicos.

- **hiera _hash:** (rvalue) Retorna un hash mezclado de coincidencias a través de la jerarquía.
- **hiera _include:** (rvalue) Asigna clases a un nodo usando un array de búsqueda (array merge lookup) que retorna el valor para una clave de usuario-específico de la fuente de datos de Hiera.
- **include:** (statement) declara una o más clases, causando que los recursos en ellas sean evaluados y añadidos al catálogo.
- **info:** (statement) Deja un mensaje en el log del servidor en el nivel info.
- **inline _epp:** (rvalue) Evalúa una plantilla Embedded Puppet y retorna el texto renderizado resultante como un string.
- **inline _template:** (rvalue) Evalúa un string plantilla y retorna su valor.
- **lookup:** (rvalue) Busca datos definidos usando Data Binding y Data Providers utilizando diferentes estrategias.
- **match:** (statement) Retorna el resultado de coincidir un string o un array[string] con regexp, string (transformado a regexp, tipo pattern o tipo regexp).
- **md5:** (rvalue) Retorna un valor de hash MD5 de un string dado.
- **notice:** (statement) Deja un mensaje en el log del servidor en el nivel noticia.
- **reduce:** (rvalue) Aplica un bloque parametrizado a cada elemento en una secuencia de entradas del primer argumento (*el enumerable*) y retorna el último resultado de la invocación del bloque parametrizado.
- **regsubst:** (rvalue) Realiza un reemplazo regexp en un string o array de strings.
- **requiere:** (statement) Evalúa una o más clases, añadiendo la clase requerida como dependencia.
- **scanf:** (rvalue) Escanea un string y retorna un array de uno o más valores convertidos dirigidos por un formato dado string.args.
- **sha1:** (rvalue) Retorna un valor hash SHA1 de un string dado.
- **shellquote:** (rvalue) Cita y concatena argumentos para usar en Bourne Shell.
- **split:** (rvalue) Divide una variable string en un array usando el divisor regexp especificado.
- **sprintf:** (rvalue) Realiza un formateo de texto con estilo printf.
- **tag:** (statement) Añade las etiquetas especificadas a la clase o definición que la contiene. Luego todos los objetos también adquirirán esa etiqueta.
- **tagged:** (rvalue) Una función booleana que dice si el contenedor actual está etiquetado con las etiquetas especificadas. Las etiquetas son operadas con AND, así que todas deben ser incluidas para que la función retorne true.
- **versioncmp:** (rvalue) Compara dos números de versión.
- **warning:** (statement) Deja un mensaje en el log del servidor en el nivel advertencia.
- **with:** (rvalue) Llama a un bloque de código lambda con los argumentos dados. Como los parámetros de lambda son locales para el alcance de ella, esto puede ser utilizado para crear secciones privadas de lógica en una clase para que las variables no sean visibles fuera de la clase.

9.3.8. Metaparámetros

Los metaparámetros son atributos que trabajan con cualquier tipo de recurso, incluido los tipos personalizados y los tipos definidos.

En general, ellos afectan el comportamiento de Puppet en preferencia a el deseo del estado del recurso.

Los metaparámetros hacen cosas como agregar metadata a un recurso (alias, tag), poner límites cuando el recurso debe ser sincronizado (require, schedule, etc.), evita que Puppet realice cambios (noop), y cambia la verborrea del log (loglevel).

Metaparámetros disponibles:

- **Alias:** Crea un alias para el recurso. Puppet usa esto internamente cuando se provee un título simbólico y un valor de *namevar* explícito.

```
file { 'sshdconfig':
  path => $operatingsystem ? {
    solaris => '/usr/local/etc/ssh/sshd_config',
    default => '/etc/ssh/sshd_config',
  },
  source => '...'
}

service { 'sshd':
  subscribe => File['sshdconfig'],
}
```

Cuando se usa esta característica, el parseador pone *sshdconfig* como el título, y la librería pone que es un alias para el archivo, entonces funciona la dependencia de búsqueda en *Service['sshd']*. Se puede usar este metaparámetro, pero note que el alias generalmente solo trabaja para crear relaciones; cualquier cosa que se refiera a un recurso existente debe usar el título exacto del recurso. Por ejemplo, el siguiente código no funciona:

```
file { '/etc/ssh/sshd_config':
  owner => root,
  group => root,
  alias => 'sshdconfig',
}
```

```
File['sshdconfig'] {
  mode => '0644',
}
```

Puppet no tiene forma de saber que ésto debe afectar el mismo archivo.

- **before:** Cuando este atributo está presente, este recurso se aplicará *luego* del recurso o los recursos de los cuales depende.
- **loglevel:** Coloca el nivel de información que será registrada. Los niveles de log tienen el mayor impacto cuando los eventos a registrar son enviados al syslog (así es por defecto).

El orden de los niveles, en prioridad decreciente, es:

- crit
- emerg
- alert
- err
- warning
- notice
- info / verbose
- debug

- **noop:** Es para aplicar este recurso en modo noop.

Cuando aplica un recurso en modo noop, Puppet verificará si está sincronizado, como cuando corre normalmente. Sin embargo, si un atributo de un recurso no está en el estado deseado (como lo declara el catálogo), Puppet no realizará ninguna acción, y en su lugar, reportará los cambios que habría hecho. Esos cambios simulados aparecen en el reporte enviado al Puppet Master, o serán mostrados en consola si `puppet agent` o `puppet apply` corren en primer plano.

- **notify:** Cuando este atributo está presente, este recurso será aplicado antes que el recurso notificado.
 - Si Puppet hace cambios en este recurso, causará que todos los recursos notificados se refresquen. Este comportamiento varía según el tipo de recurso, por ejemplo, los servicios se reiniciarán, objetos montados se desmontarán y montarán nuevamente, etc.
- **require:** Cuando este atributo está presente, el recurso o los recursos requeridos serán aplicados *antes* que este recurso.
- **schedule:** Un programa para gestionar cuándo Puppet tiene permitido administrar este recurso. El valor de este metaparámetro debe ser el *nombre* de un recurso *schedule*. Esto significa que se debe declarar un recurso *schedule* y luego referirse a él por su nombre.

```
schedule { 'everyday':
```

```

    period => daily,
    range  => "2-4"
}

exec { '/usr/bin/apt-get update':
  schedule => 'everyday'
}

```

- **stage:** En cuál estado de ejecución esta clase debería residir. Este metaparámetro sólo puede ser usado con clases, y solo declarándolo con la sintaxis de recursos. No se puede usar con recursos normales o en clases declaradas con *include*.

Por defecto, todas las clases son declaradas en el estado principal. Para asignar una clase a un estado diferente, se debe:

- Declarar un nuevo estado como un recurso *stage*.
- Declarar un orden de relación entre el nuevo el nuevo estado y el estado *main*.
- Usar la sintaxis de recurso para declarar la clase y poner el *stage* deseado.

Por ejemplo:

```

stage { 'pre':
  before => Stage['main'],
}

```

```

class { 'apt-updates':
  stage => 'pre',
}

```

- **subscribe:** Cuando este atributo está presente, el recurso suscripto aplicara los cambios *antes* que este recurso.

- Si Puppet realiza cambios a cualquier recurso subscripto, esto causará que este recurso se refresque.

- **tag:** Agrega la etiqueta especificada al recurso asociado. si bien todos los recursos son automáticamente etiquetados con la mayor cantidad de información posible, puede ser útil agregar su propia etiquetas a los recursos.

Ejemplo:

```

file {'/etc/hosts':
  ensure => file,
  source => 'puppet:///modules/site/hosts',
  mode   => '0644',
  tag    => ['bootstrap', 'minimumrun', 'mediumrun'],
}

```

9.3.9. Lenguaje

Variables Las variables guardan valores para que sean accedidos más tarde. En Puppet, las variables son en realidad constantes ya que no pueden ser reasignadas. Los nombres de variables comienzan con el signo \$ y son sensibles a mayúsculas. La mayoría de los nombres deben comenzar con una letra en minúscula o un guión bajo. Los nombres pueden incluir:

- Mayúsculas y minúsculas
- Números
- Guión bajo

Si el primer carácter es un guión bajo, esa variable debería ser accedida solo desde su propio alcance.

Sintaxis `$contenido = "algún contenido\n"`

Los nombres de variables tienen como prefijo un signo \$. Los valores son asignados a ellas con el símbolo = y se pueden asignar valores de cualquier tipo de dato. La variable contendrá el valor que la declaración resuelve, en vez de una referencia a la declaración.

Las variables solo pueden ser asignadas utilizando su nombre corto. Esto es, un alcance dado no puede asignar a variables en un ámbito exterior.

Asignar múltiples variables Se pueden asignar múltiples variables de una vez desde un array o hash.

Arrays Cuando se asignan múltiples variables desde un array, debe haber un número igual de variables y valores. Si no coinciden, la operación fallará. Arrays anidados también pueden ser usados.

```
[$a, $b, $c] = [1,2,3] # $a = 1, $b = 2, $c = 3
[$a, [$b, $c]] = [1, [2,3]] # $a = 1, $b = 2, $c = 3
[$a, $b] = [1, [2]] # $a = 1, $b = [2]
[$a, [$b]] = [1, [2]] # $a = 1, $b = 2
```

Hashes Cuando se asignan múltiples variables con un hash, las variables son listadas en un array en el lado izquierdo de la asignación, y el hash está del lado derecho. Las claves de hash deben coincidir su nombre correspondiente de variable.

```
[$a, $b] = {a => 10, b => 20} # $a = 10, $b = 20
```

```
Resolución file {'/tmp/testing'}:
ensure => file,
content => $content,
}
```

```
$address_array = [$address1, $address2, $address3]
```

El nombre de la variable puede ser usado en cualquier lugar donde un valor de su tipo de dato fuera aceptado, incluyendo expresiones, funciones y atributos de recursos. Puppet reemplazará el nombre de la variable por su valor. Por defecto, variables sin asignar tienen el valor *undef*.

```
Interpolación $rule = "Allow * from $ipaddress"
file { "${homedir}/.vim":
  ensure => directory,
}
```

Puppet puede resolver variables en strings entre comillas dobles, esto es llamado interpolación. Dentro de las comillas dobles, se puede opcionalmente envolver el nombre de la variable (la porción luego de \$) con llaves (\${nombre_variable}). Esta sintaxis ayuda a evitar ambigüedades y permite ubicar las variables directamente siguiendo a caracteres que no son espacios en blanco. Estas llaves opcionales, están permitidas solo dentro de strings.

Comportamiento

Alcance El área de código donde una variable dada es visible está dictada por su alcance. Las variables en un alcance dado solo están disponibles dentro de ese alcance/ámbito su alcance hijo, y cualquier alcance local puede sobre escribir localmente las variables que recibe de sus padres.

Acceso a variables fuera de alcance Se puede acceder a variables fuera de alcance utilizando su nombre completo (qualified name).

```
$vhostdir = $apache::params::vhostdir
```

9.3.10. Orden y relaciones

Por defecto, Puppet aplica los recursos en el orden en que son declarados en el manifiesto. Sin embargo, si un grupo de recursos deben siempre ser administrados en un orden específico se debe explicitar declarando tales relaciones con metaparametros de relación, flechas y la función *require*.

Utilizando metaparametros El valor de todas las relaciones debe ser una referencia a un recurso, apuntando a uno o más recursos objetivos.

before: Aplica un recurso *antes* que el recurso objetivo.

requiere: Aplica un recurso *después* que el recurso objetivo.

notify: Aplica un recurso *antes* que el recurso objetivo. El recurso objetivo se refresca si el recurso que notifica cambia.

`subscribe`: Aplica un recurso *después* que el recurso objetivo. El recurso suscripto se refresca si el recurso cambia.

Estos dos ejemplos crean la misma relación de orden:

- Ejemplo uno:

```
package { 'openssh-server':
  ensure => present,
  before => File['/etc/ssh/sshd_config'],
}
```

- Ejemplo dos:

```
file { '/etc/ssh/sshd_config':
  ensure => file,
  mode => '0600',
  source => 'puppet:///modules/sshd/sshd_config',
  require => Package['openssh-server'],
}
```

Utilizando flechas Se pueden crear relaciones entre dos recursos o grupos de recursos usando los operadores `->` y `~>`.

`->` : Flecha de ordenamiento. Aplica el recurso izquierdo *antes* que el recurso derecho.

`~>` : Flecha de notificación. Aplica el recurso izquierdo primero. Si este cambia, el recurso derecho se refrescará.

- Ejemplo uno:

```
Package['ntp'] -> File['/etc/ntp.conf'] ~> Service['ntpd']
```

- Ejemplo dos:

```
# Primero tener:
package{ 'openssh-server':
  ensure => present,
} -> # Luego tener:
file { '/etc/ssh/sshd_config':
  ensure => file,
  mode => '0600',
  source => 'puppet:///modules/sshd/sshd_config',
} ~> # Y por último:
service { 'sshd':
  ensure => running,
  enable => true,
}
```

9.3.11. Definición de nodos

Una definición o declaración de nodo es un bloque de código Puppet que sólo será incluido en catálogos de nodos que coincidan. Esta característica permite asignar configuraciones específicas a nodos específicos.

Las declaraciones de nodos sólo coinciden con los nombres de los nodos. Por defecto, el nombre de un nodo es su *certname*.

```
# node 'www1.example.com' { include common include apache include squid }
node 'db1.example.com' { include common include mysql }
```

Ubicación Las definiciones de los nodos deben estar en el manifiesto principal. Éste puede ser un archivo o un directorio conteniendo muchos archivos.

```
# /etc/puppetlabs/puppet/manifests/site.pp

node 'www1.example.com' {
  include common
  include apache
  include squid
}

node 'db1.example.com' {
  include common
  include mysql
}
```

En este ejemplo, solo el primero nodo obtendrá las clases *apache* y *squid* mientras que el segundo tendrá *mysql*. Ambos recibirán la clase *common*.

Nombramiento Una declaración de nodo debe realizarse según:

- Un string entre comillas conteniendo sólo letras, números, guiones bajos, guiones medios y puntos.
- Expresiones regulares.
- La palabra *default*, sin comillas.

Se pueden utilizar listas de nombres separados por comas para crear grupos de nodos son una sola declaración de nodo:

```
node 'www1.example.com', 'www2.example.com', 'www3.example.com' {
  include common
  include apache, squid
}
```

El nodo default Si ninguna declaración de nodo es dada o no puede ser encontrada, el nodo *default* será utilizado.

Expresiones regulares Pueden ser utilizadas como nombres de nodo. Este es otro método para escribir una sola definición de nodo que coincide con múltiples nodos.

Coincidencias Un nodo dado sólo obtendrá los contenidos de una definición de nodo, incluso si dos declaraciones de nodo pueden coincidir con el nombre del mismo. Puppet realizará las verificaciones para decidir cual definición utilizar:

1. Si hay una definición de nodo que contenga el nombre exacto del nodo, entonces utilizará esta.
2. Si hay una expresión regular que coincide con el nombre del nodo, entonces utilizará esta.
3. Si el nombre de nodo es del tipo FQDN (Fully Qualified Domain Name), Puppet cortará el grupo final y comenzará nuevamente desde el punto uno.
4. Puppet utilizará el nodo default.

10. Automatización de Windows 7

La instalación automatizada de Windows 7 requiere su preparación sobre un sistema Windows 7 pero su despliegue se realiza desde el servidor Cobbler, bajo Linux.

10.1. El lado de Windows

Para automatizar la instalación de Windows 7, utilizando el método de imagen estándar personalizada, se realiza en siete pasos principales. Para ello es necesario contar con:

- **Máquina técnica:** Con este término, Microsoft se refiere a un sistema Windows utilizado para correr las herramientas y otras operaciones que forman parte de la automatización automatizada. En esta máquina es necesario contar con cualquier versión de Windows 7 que cuente con las herramientas de WAIK (Windows Automated Installation Kit).
- **Máquina de referencia:** Con este término, Microsoft se refiere al equipo sobre el cual se instalará el sistema operativo a partir del cual se extraerá la imagen personalizada con las configuraciones y programas deseados.
- **Windows Automated Installation Kit:** El kit de instalación automatizada de Windows es un conjunto de herramientas y documentación compatible con la configuración y la implementación de los sistemas operativos Windows. Mediante el WAIK, puede automatizar las instalaciones de Windows, capturar imágenes de Windows con ImageX, configurar y modificar imágenes usando Administración y mantenimiento de imágenes de implementación (DISM), crear imágenes de Windows PE, entre otras características.
- **Imagen de instalación de Windows 7:** Ya sea un DVD o un archivo ISO que provea la fuente de instalación del sistema operativo.

La siguiente tabla contiene una columna para cada equipo. Los pasos de la columna en la máquina técnica se realizan en su ese equipo. Los pasos de la columna del equipo de referencia se realizan en el equipo en el que se cree la imagen personalizada.

Paso	Máquina técnica	Máquina referencia
1	Instalar WAIK	
2	Crear un disco de inicio del Entorno de Preinstalación de Windows (WinPE)	
3		Instalar y personalizar Windows 7
4		Generalizar el equipo de referencia para preparar la imagen para la duplicación
5		Capturar el equipo de referencia a un archivo de imagen mediante ImageX
6	Crear archivo Autounattend.xml	
7	Crear nuevos medios de instalación de Windows 7 para la imagen personalizada	

Tabla 1: Pasos automatización Windows 7

10.1.1. Instalar WAIK

Desde la página oficial de Microsoft descargar el archivo KB3AIK_EN.iso, grabarlo en un DVD o montarlo en una unidad virtual e instalarlo.

10.1.2. WinPE

WinPE o Windows Preinstallation Environment es un Sistema operativo mínimo diseñado para preparar una computadora para la instalación de Windows. Puede ser utilizado para iniciar una computadora sin sistema operativo, para particionar y formatear discos rígidos, para copiar imágenes de discos y para iniciar la instalación de Windows desde una ubicación compartida en la red. WinPE se carga directamente en memoria y permite utilizar herramientas como ImageX para capturar, modificar y crear imágenes de instalación basadas en archivos.

Para crear una imagen ISO de WinPE es necesario ejecutar Deployment Tools Command Prompt como administrador.

Se debe copiar a una ubicación elegida los archivos necesarios. Como se ha utilizado una versión de 64 bits se especifica amd64, si no i386.

```
copye amd64 d:\path-del-directorio\winpe
```

Montar el archivo .WIM con:

```
imagex /mountrw d:\path-del-directorio\winpe\winpe.wim 1 d:\path-del-directorio\winpe
```

Cuando se carga en memoria el entorno de preinstalación, es ejecutado un script llamado Startnet.cmd el cual de forma predeterminada inicia Wpeinit.exe, que precisamente sirve para instalar dispositivos Plug and Play, procesar configuraciones de Autounattend.xml

y cargar recursos de red. Se debe asegurar que se incluye una llamada a wpeinit en el script Startnet.cmd personalizado:

```
d:\ path-del-directorio\winpe\mount\Windows\System32\startnet.cmd
```

Utilizar un editor de texto y dejarlo como se muestra a continuación, reemplazando los valores correspondientes:

```
wpeinit
echo Conectando con servidor...
net use y: \\IP_del_servidor_Cobbler\entrada_SAMBA
y:
echo Preparando instalacion...
setup.exe /unattend:Autounattend.
```

Una vez modificado los valores del script, desmontar winpe.wim. Para ello es necesario cerrar el explorador de Windows y cualquier programa que esté haciendo uso de directorio anterior, caso contrario dará error. Luego:

```
imagex /unmount d:\path-del-directorio\winpe\mount /commit
```

Se copia entonces winpe.wim al directorio renombrándolo como boot.wim:

```
copy d:\path-del-directorio\winpe\winpe.wim d:\path-del-directorio\winpe\ISO\sources\boot.wim
```

Por último es necesario crear una imagen ISO de WinPE.

```
oscdimg -nt -m -h -bd:\path-del-directorio\winpe\etfsboot.com d:\path-del-directorio\winpe\winpe_cobbler_amd64.iso
```

Las opciones especificadas son las siguientes:

-nt: Permite nombres largos de archivo que sean compatibles con Windows NT 3.51.

-m: Pasa por alto el límite de tamaño máximo de una imagen.

-h: Incluye todos los directorios y archivos ocultos bajo el d:\path-del-directorio\ para esta imagen.

-b: Esta opción se utiliza para especificar el archivo que se escribirá en el sector de arranque del disco. No debe existir espacio alguno entre esta opción y el valor pasado.

-u2: Genera una imagen que sólo incluye el sistema de archivos UDF. Los sistemas que no sean capaces de leer UDF, sólo verán un archivo de texto predeterminado donde se alerta al usuario de que esa imagen sólo está disponible en equipos compatibles con UDF. Esta opción no puede combinarse con las opciones -nt.

10.1.3. Instalar y personalizar Windows 7

Una máquina de referencia tiene una instalación personalizada de Windows que se planea replicar en uno o más computadoras. Se puede crear utilizando un DVD o archivo ISO de Windows.

Este punto conviene realizarlo manualmente y de la forma usual. Sin embargo, cuando en el proceso de instalación, se llega a la pantalla de bienvenida de Windows (diálogo de creación de usuario y hostname) se debe presionar *Ctrl + Shift + F3* para ingresar en modo auditoría. Ahora la máquina debería reiniciarse e ingresar automáticamente en una cuenta temporal en modo administrador. Esto permanecerá así por más que se reinicie el equipo hasta que el comando Sysprep sea ejecutado luego de realizar todas las actualizaciones, los controladores de dispositivo e instalación de las aplicaciones.

Cada vez que sea reiniciado el equipo estando en este modo, aparecerá un cuadro de diálogo del modo auditoría. Simplemente ignorarlo hasta terminar de preparar la imagen.

Para pasarle los programas a instalar, alojados en el servidor en */windows/PROGRAMAS/*, desde la máquina de referencia ir a *Equipo → Conectar a unidad de red* e ingresar la dirección IP del servidor con el formato:

`\IP_del_servidor_Cobbler\entrada_SAMBA`

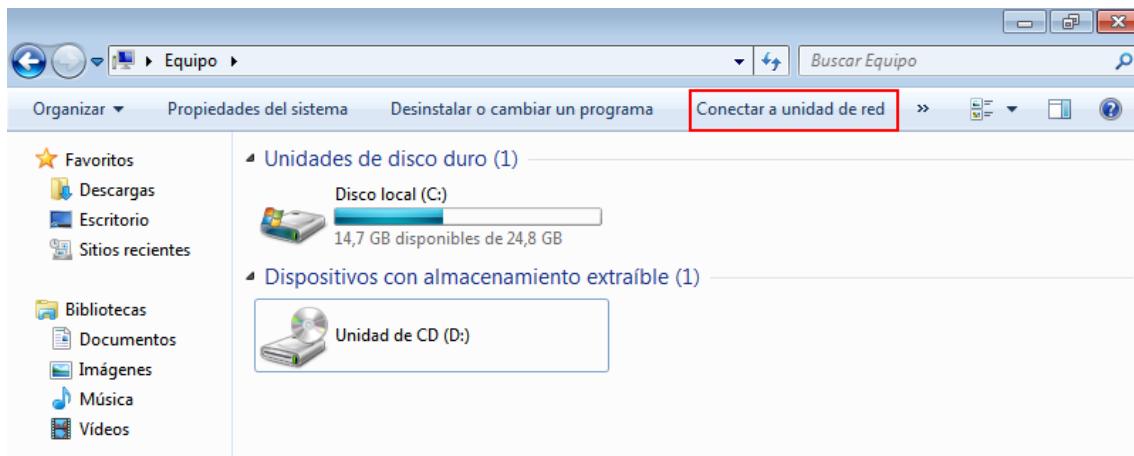


Figura 28: Paso de archivos a Windows

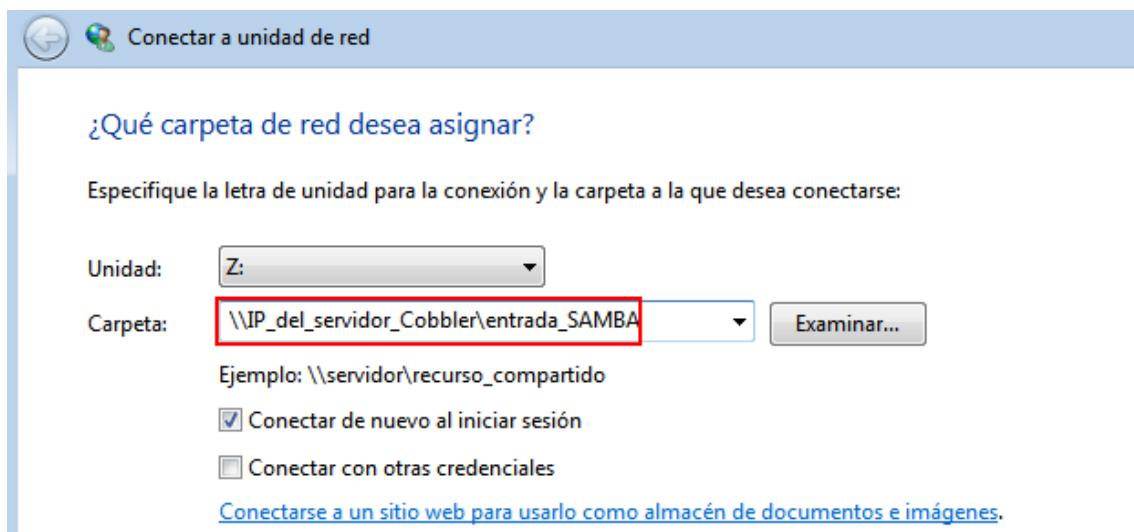


Figura 29: Paso de archivos a Windows

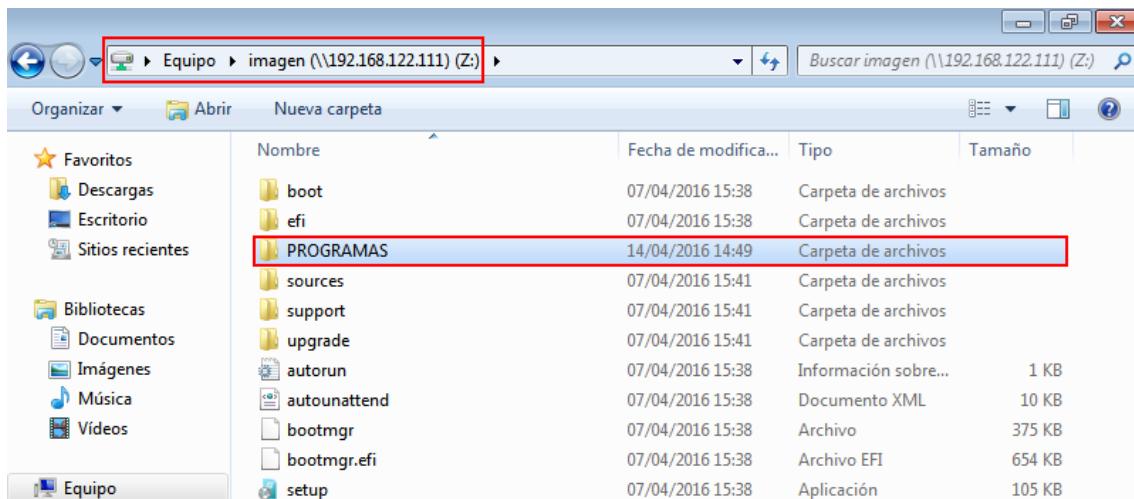


Figura 30: Paso de archivos a Windows

10.1.4. Generalizar el equipo de referencia para preparar la imagen

En este paso, se generaliza la imagen y se prepara para su inicio en Bienvenida de Windows después de haberla instalado en cada equipo. Al generalizar la imagen, se elimina de ella la información que depende de hardware, se restablece el temporizador de activación y se limpia Windows 7 para que se pueda duplicar la imagen en otros equipos.

Para esto se hace uso de la herramienta Sysprep, la cual limpia las configuraciones específicas de usuario y del hardware.

De forma gráfica, cuando inicia el sistema operativo en modo auditoría, Windows 7 ejecuta Sysprep de forma automática.

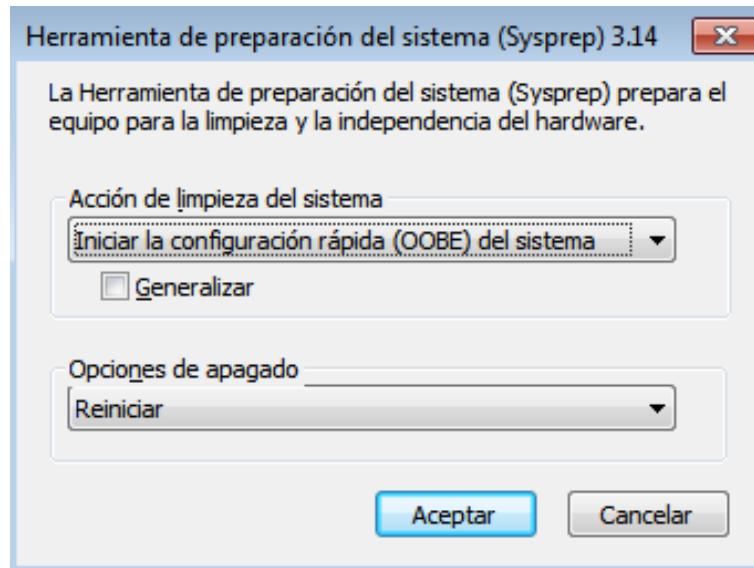


Figura 31: Modo auditoría

- En la lista Acción de limpieza del sistema, seleccionar Iniciar la configuración rápida (OOBE) del sistema.
- Activar la casilla Generalizar.
- En la lista Opciones de apagado, seleccionar Apagar.
- Aceptar para ejecutar Sysprep y apagar el equipo.

En este caso particular se decidió realizar la generalización del sistema por consola. Para ello es necesario cargar un WinPE.iso sin modificaciones.

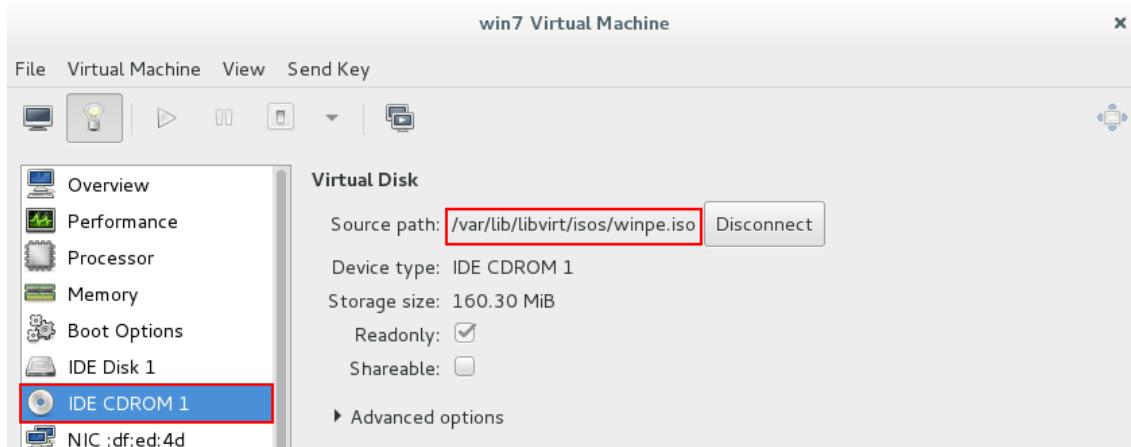


Figura 32: Carga WinPE.iso para extraer imagen

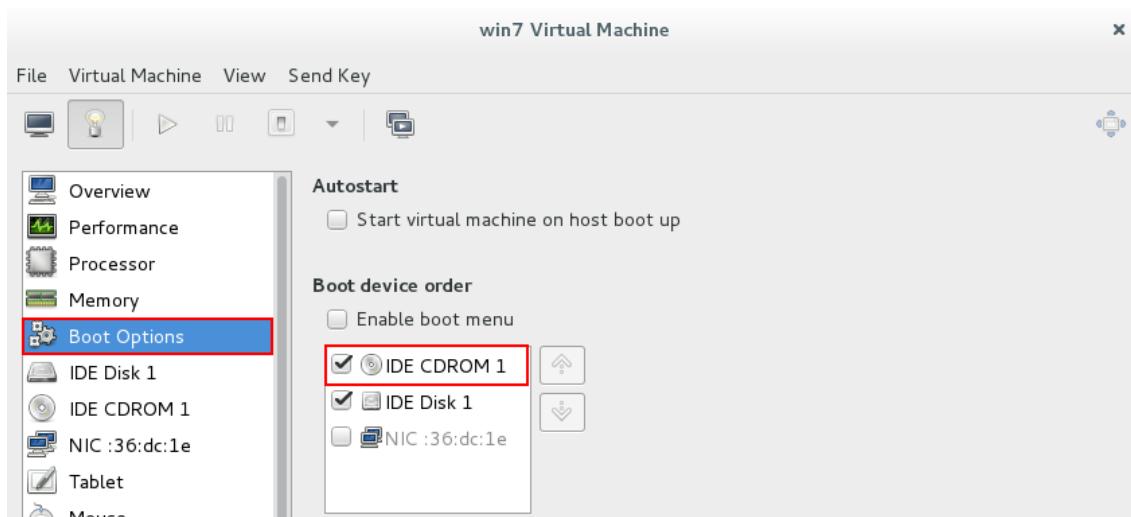


Figura 33: Prioridad de inicio en máquina virtual

Desde la consola, ejecutar por línea de comando desde C:\Windows\System32\Sysprep\:
`sysprep /generalize /oobe /shutdown`

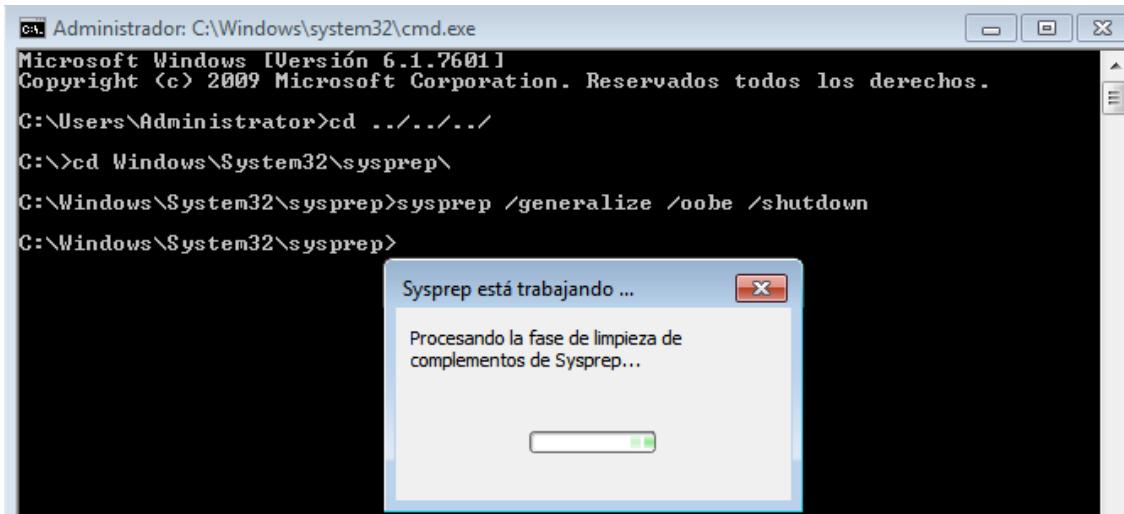


Figura 34: Ejecución sysprep

Donde las opciones más importantes son:

/generalize: Prepara a la instalación de Windows para crear una imagen. Si esta opción es especificada, toda la información única del sistema es removida. El SID (ID de Seguridad) se resetea, cualquier punto de restauración que exista es quitado y todos los logs son eliminados. La próxima vez que la computadora inicia, la fase de configuración especial es ejecutada y se asigna un nuevo SID y el reloj de activación de Windows se resetea si no ha sido reseteado ya tres veces, esto es, este proceso no puede repetirse más de tres veces por instalación de Windows. En ese caso, se debe comenzar con una nueva instalación desde cero.

/oobe: El término OOBE significa Out Of Box Experience y es la experiencia que un usuario tiene cuando se prepara para usar por primera vez un producto. Cuando esta opción es seleccionada, reinicia el equipo en modo Bienvenida de Windows. Este modo permite a los usuarios personalizar el sistema operativo, crear cuentas de usuario, cambiar el hostname y demás. Cualquier configuración pasada al oobeSystem por medio de un archivo de respuestas de instalación es procesada inmediatamente antes que la Bienvenida de Windows inicie.

/shutdown: Apaga la máquina luego que el comando sysprep finaliza.

/unattend:answerfile: Aplica las configuraciones dentro de un archivo de respuestas de instalación sobre el sistema operativo durante una instalación desatendida; answerfile especifica el path y el nombre del archivo que contiene las respuestas. Esta opción se puede ignorar, ya que como se verá más adelante basta con agregar este archivo en el directorio principal de instalación para que sea tomado por el instalador de Windows.

A esta altura del proceso, ha instalado Windows 7 en el equipo de referencia y está listo para capturar una imagen del mismo.

10.1.5. Capturar el equipo de referencia

En este paso se capture una imagen de la máquina de referencia utilizando la herramienta ImageX, que está en el WinPE anteriormente creado. La imagen será guardada en el

directorio compartido SAMBA. Se carga nuevamente el disco WinPE y se obtiene una consola.

Verificar que la máquina ve al servidor que tiene SAMBA configurado y corroborar que se tiene acceso a la ubicación compartida:

```
net use y: \\IP_del_servidor_Cobbler\entrada_SAMBA
y:
dir
```

Verificar la letra asignada al disco WinPE, usualmente es *e*: y verificar la letra de la partición donde está instalado Windows ya personalizado, usualmente *c*:

```
X:\windows\system32>net use y: \\192.168.122.111\imagen
The command completed successfully.

X:\windows\system32>e:
E:>dir
Volume in drive E is CD_ROM
Volume Serial Number is 6BF9-3D25

Directory of E:\

02/09/2016  12:55 PM    <DIR>          BOOT
07/13/2009  01:39 PM        383,562  BOOTMGR
07/13/2009  01:45 PM        667,712  BOOTMGR.EFI
02/09/2016  12:55 PM    <DIR>          EFI
07/14/2009  02:10 AM        581,008  IMAGEX.EXE
02/09/2016  12:55 PM    <DIR>          SOURCES
                           3 File(s)     1,632,282 bytes
                           3 Dir(s)           0 bytes free

E:>d:
D:>dir
Volume in drive D has no label.
Volume Serial Number is D4B6-74A5

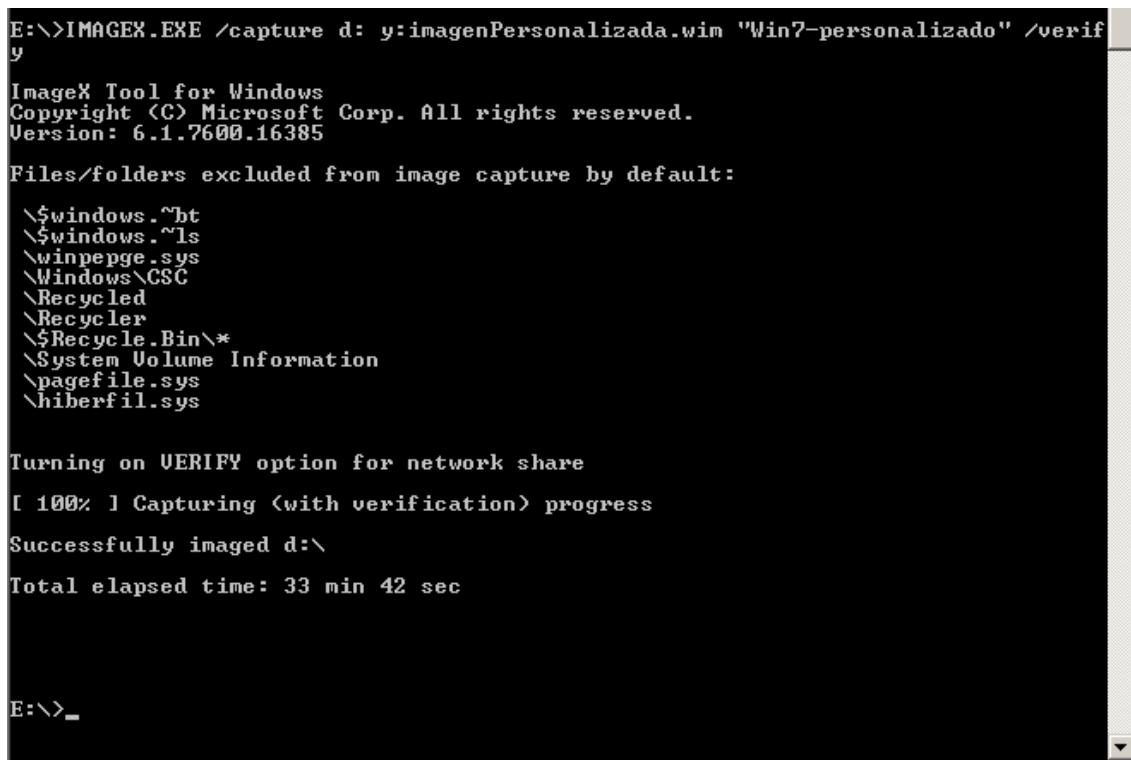
Directory of D:\

07/13/2009  07:20 PM    <DIR>          PerfLogs
04/14/2016  12:37 AM    <DIR>          Program Files
04/14/2016  12:39 AM    <DIR>          Program Files (x86)
11/20/2010  06:50 PM    <DIR>          Users
04/14/2016  01:45 AM    <DIR>          Windows
                           0 File(s)           0 bytes
                           5 Dir(s)   11,730,722,816 bytes free
```

Figura 35: Verificación de letras de unidad

Desde la unidad que contiene a WinPE:

```
e:\imageX.exe /capture d y:\windows\imagenPersonalizada.wim "Win7-Personalizado"
/verify
```



```

E:>IMAGEX.EXE /capture d: y:imagenPersonalizada.wim "Win7-personalizado" /verify
y

ImageX Tool for Windows
Copyright (C) Microsoft Corp. All rights reserved.
Version: 6.1.7600.16385

Files/folders excluded from image capture by default:
\$windows.\$bt
\$windows.\$ls
\winpege.sys
\Windows\CSC
\Recycled
\Recycler
\$Recycle.Bin\*
\System Volume Information
\pagefile.sys
\hiberfil.sys

Turning on VERIFY option for network share
[ 100% ] Capturing (with verification) progress
Successfully imaged d:\
Total elapsed time: 33 min 42 sec

E:>_

```

Figura 36: ImageX capturando imagen

Donde se tiene:

/capture c: Indica al programa que se quiere capturar la imagen ubicada en esa unidad de partición.

y:\windows\imagenPersonalizada.wim Es el path completo donde se guardará la imagen en SAMBA.

"Win7-Personalizado" Es una etiqueta para el archivo imagen creado.

/verify Indica al programa que se desea verificar la consistencia de la imagen creada.

10.1.6. Autounattend.xml

El archivo *Autounattend.xml* es donde se guardan los valores correspondientes a los diálogos de instalación del sistema operativo elegido. Ítems como creación de cuentas de usuario, idioma, configuraciones regionales, configuraciones de red, particionado de discos, etc. son los que se automatizarán. Para generararlo, una vez instalado el WAIK, se debe abrir el modo Administrador el programa Windows System Image Manager desde el menú Inicio.

Bajo la sección *Windows Image* seleccionar *Select a Windows image or catalog file*. En este punto se debe buscar en el directorio de la imagen del sistema operativo ya descomprimida, un archivo *.clg* que en este caso particular es *install_Windows 7 ULTIMATE.clg*. Acto seguido crear el archivo de respuestas de instalación *Create a New Answer File*.

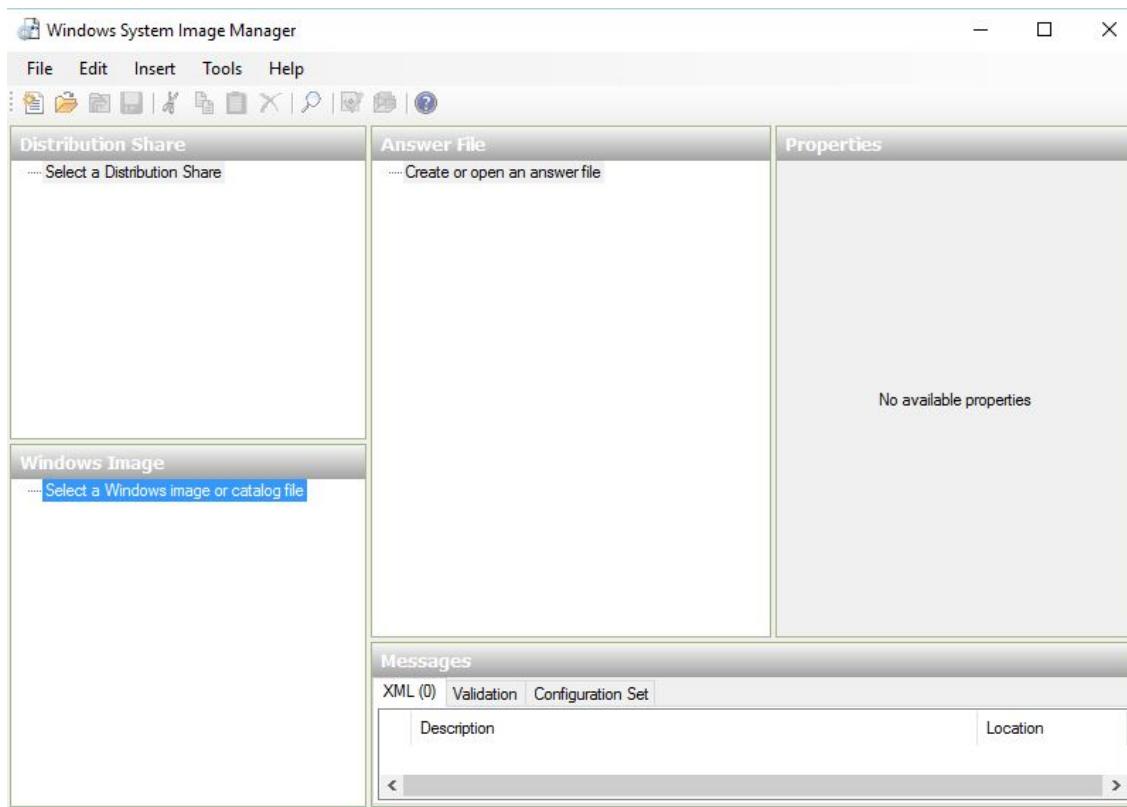


Figura 37: Windows System Image Manager

Se muestra a continuación una configuración personalizada completa utilizando Windows AIK.

Se ha configurado la entrada de texto, interfaz gráfica y ayuda del sistema en idioma español.

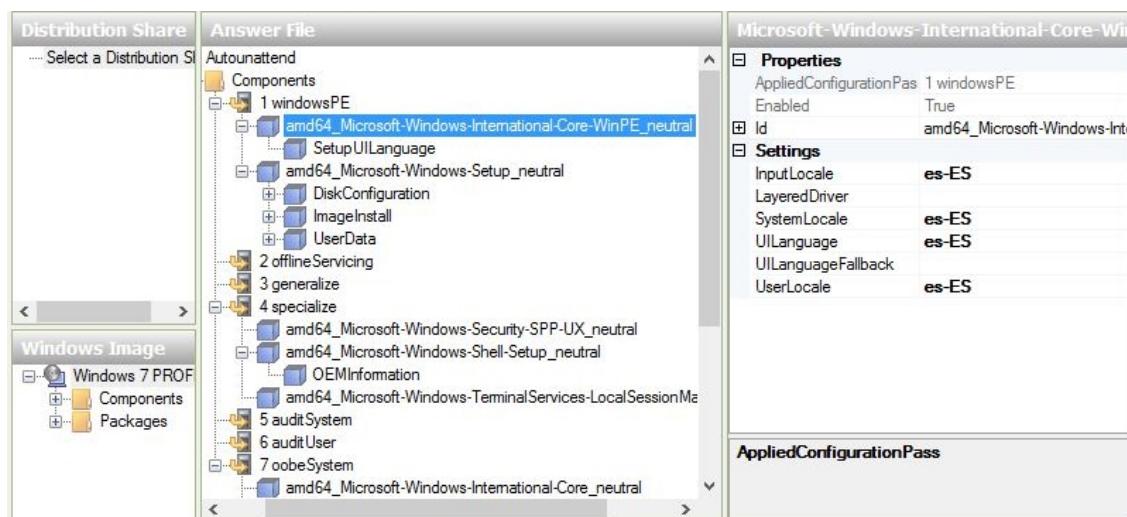


Figura 38: Idioma y teclado.

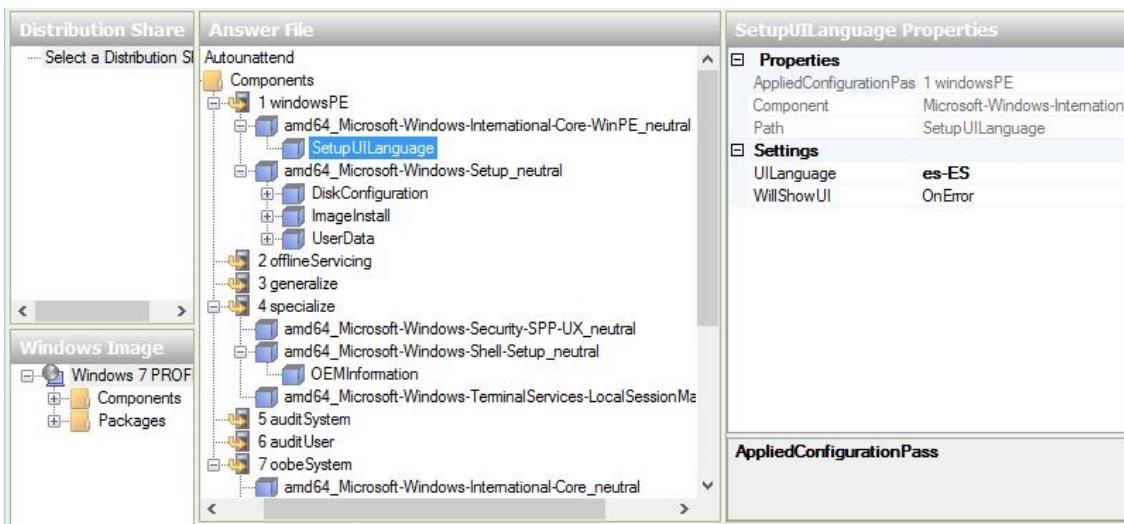


Figura 39: Idioma .

Se ha configurado el firewall para que esté habilitado, valor por defecto.

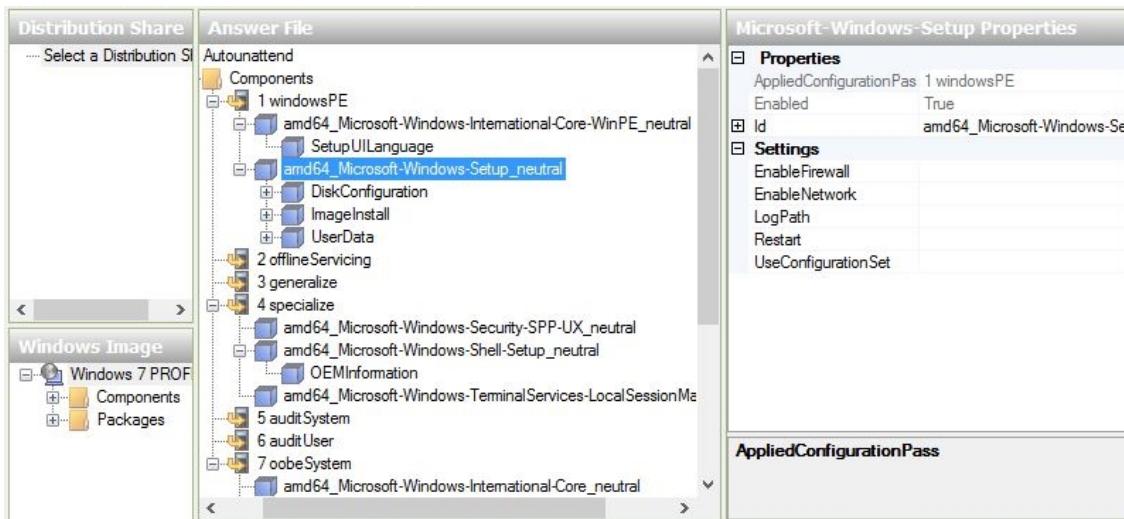


Figura 40: Firewall activado por defecto.

Se han configurado los discos para que existan dos particiones. Una donde se instalará el MBR y otra donde se copiarán los archivos del sistema operativo.

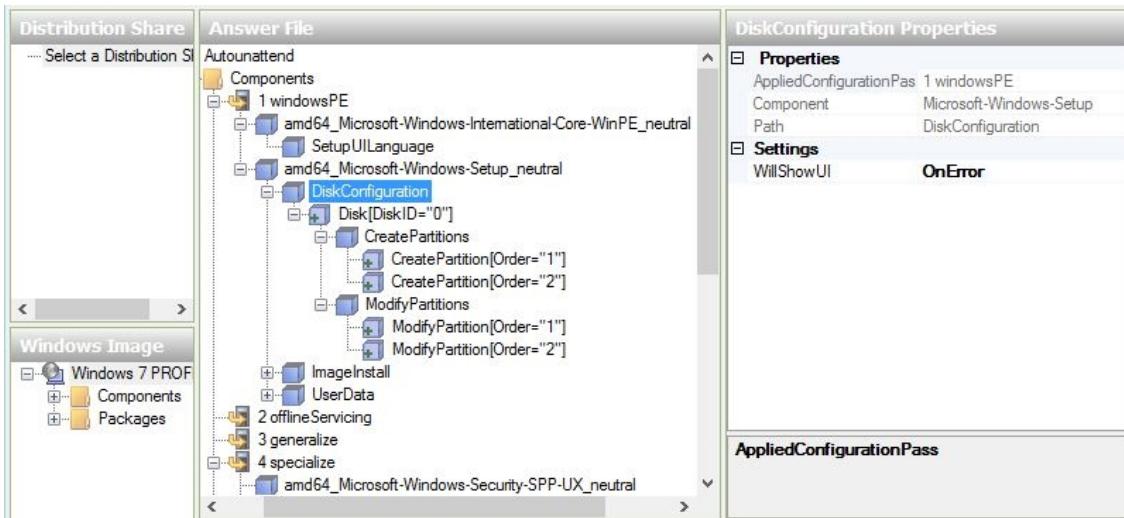


Figura 41: Discos.

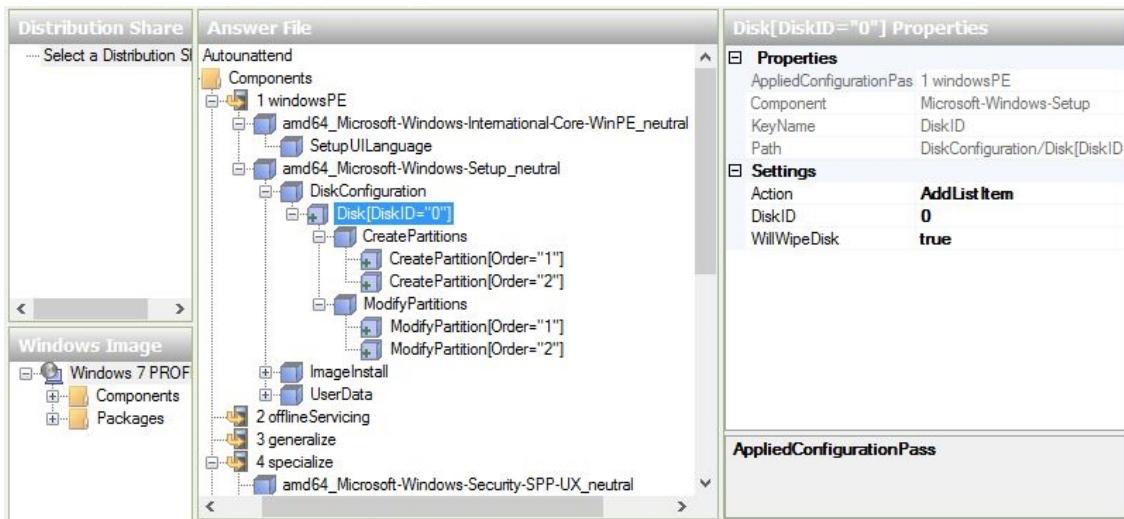


Figura 42: Configuración discos. Disco 0.

Partición de 300 MB, donde será instalado el MBR.

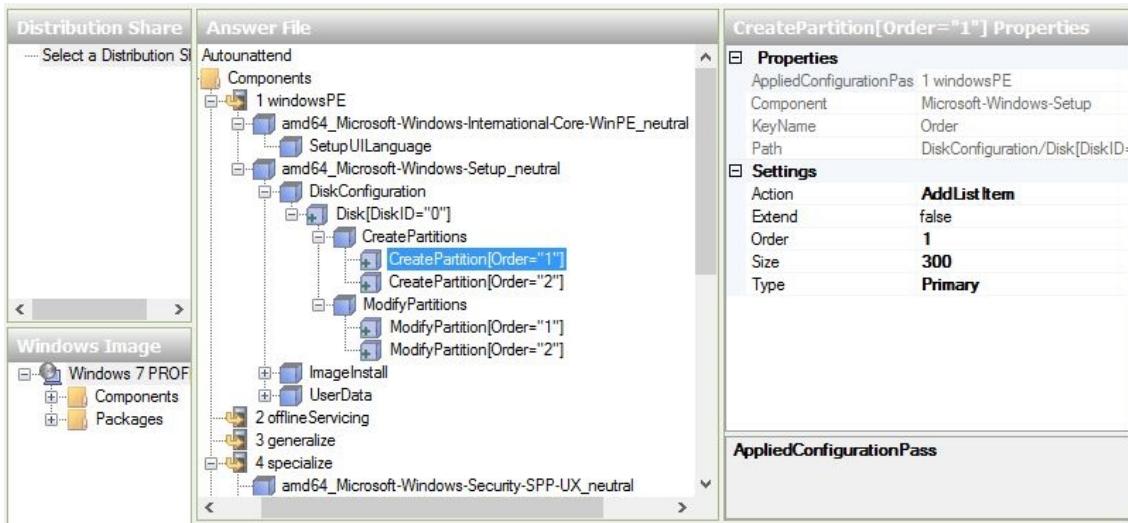


Figura 43: Discos. Particionamiento.

Partición que utilizará el espacio restante del disco, donde se copiarán los archivos de Windows.

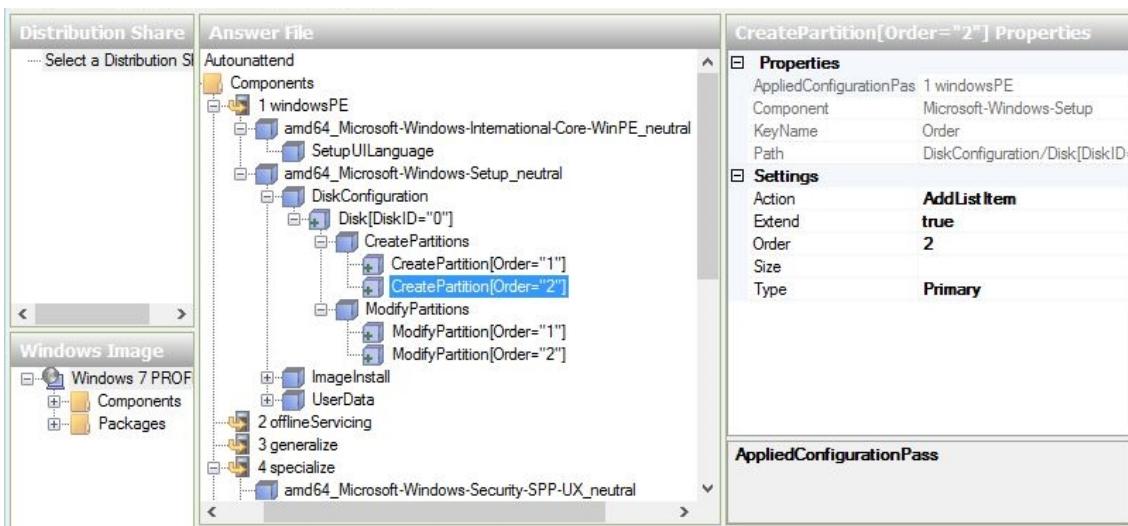


Figura 44: Discos. Particionamiento.

Formato de la partición.

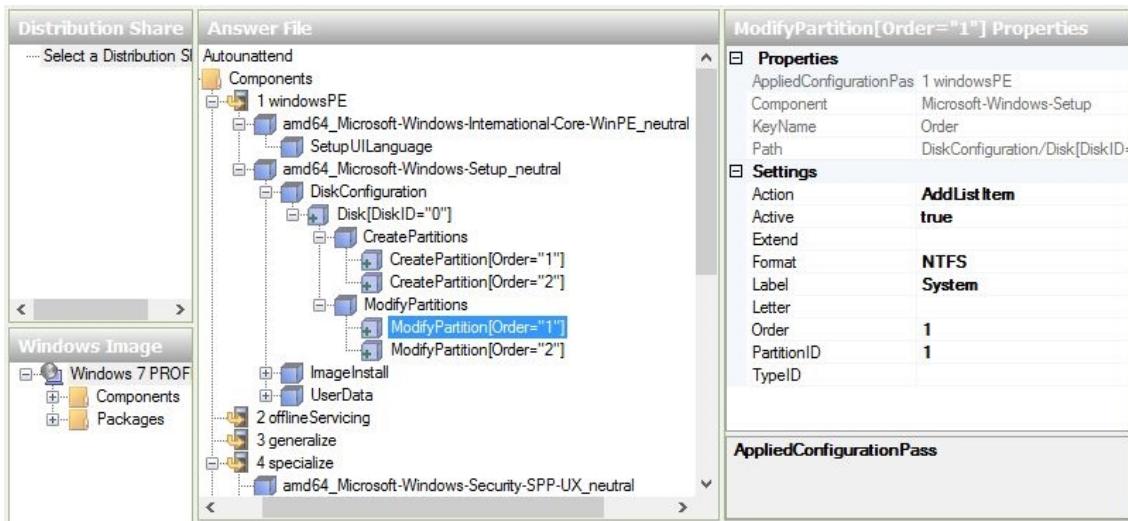


Figura 45: Discos. Particionamiento.

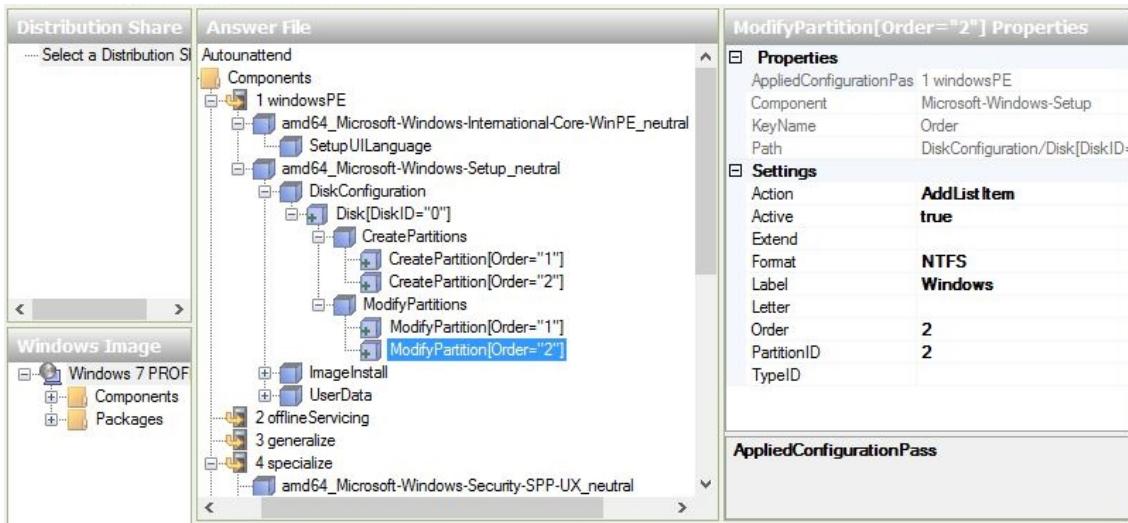


Figura 46: Discos. Particionamiento.

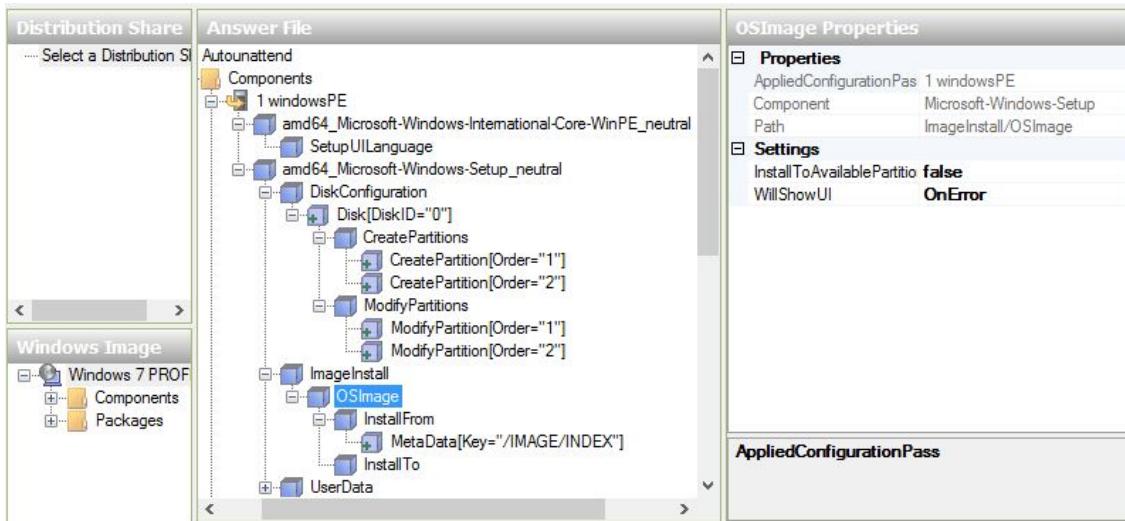


Figura 47: Instalación del SO.

En el caso de contar con una imagen de instalación multiversión, Windows Home, Basic, Professional y Ultimate y/o con ambas arquitecturas, i386 y amd64, es necesario seleccionar cuál de ellas se desea instalar. La opción más fácil y efectiva es contar la posición de la versión que se quiere instalar y seleccionarla como se muestra a continuación, donde se ve que en este caso es la primera opción.

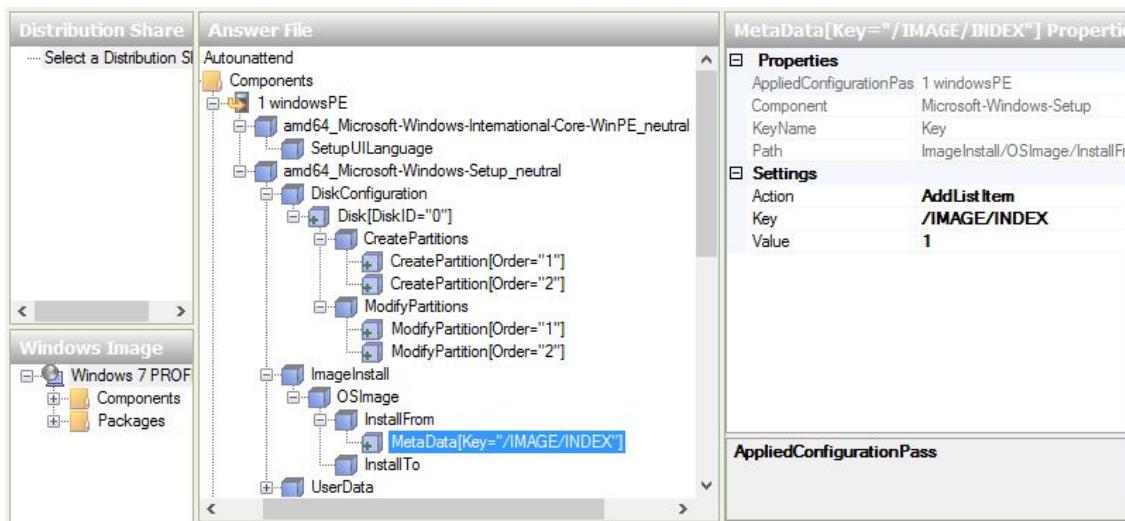


Figura 48: Instalación del SO. Elección de la versión.

Selección de la partición de instalación de Windows.

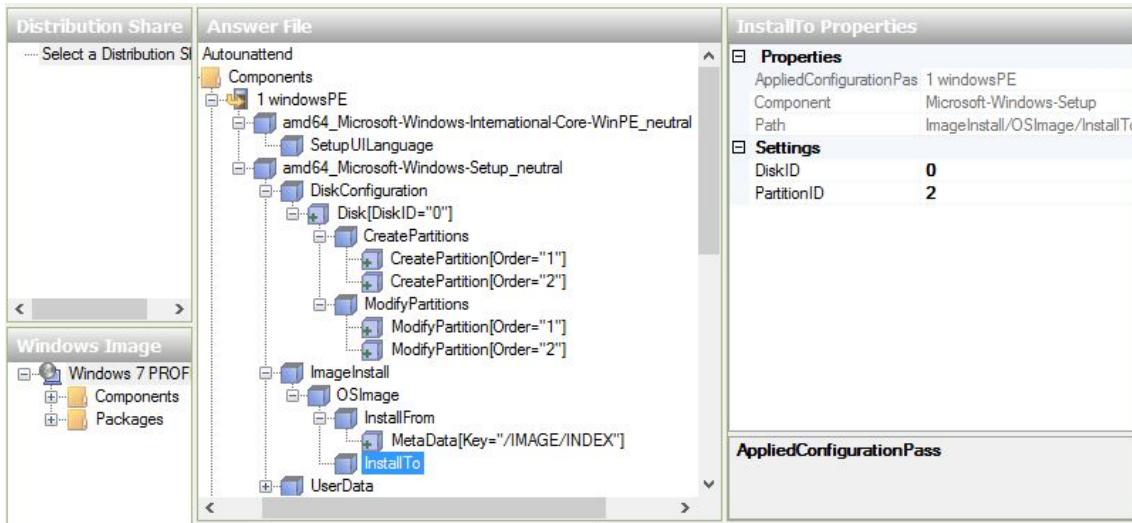


Figura 49: Elección de partición de instalación.

Aceptación de las condiciones de privacidad y uso del sistema operativo.

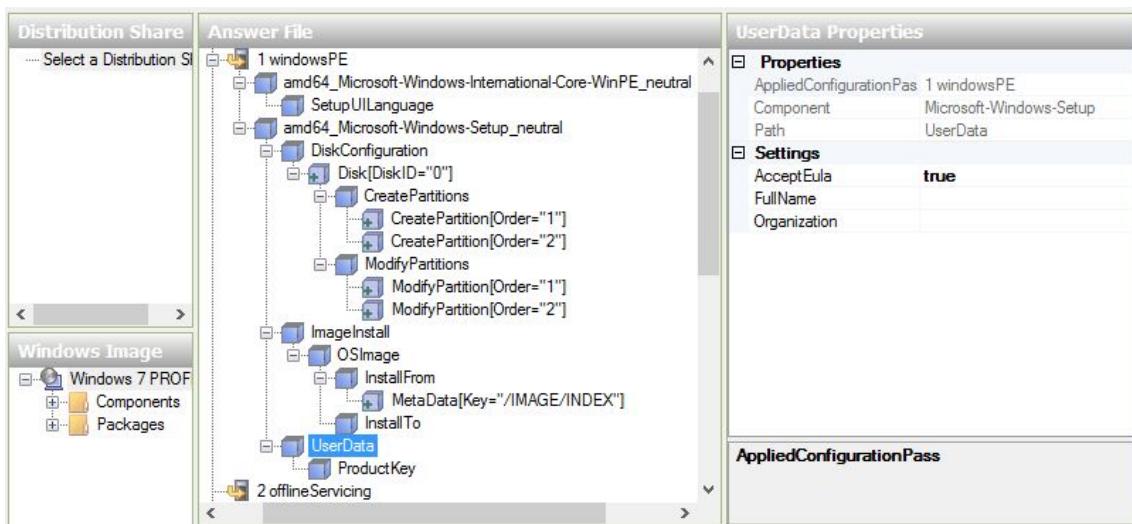


Figura 50: Aceptación de las condiciones de privacidad y uso.

La clave de producto, para poder realizar instalaciones desatendidas debe ser del tipo de Volúmen, esto es, una clave para realizar instalación en masa. A modo de ejemplo y para pruebas, Microsoft entrega una serie de claves para ésto: <https://technet.microsoft.com/en-us/library/jj612867.aspx>

Entonces, la clave de producto utilizada en este caso es: **FJ82H-XT6CR-J8D7P-XQJJ2-GPDD4**.

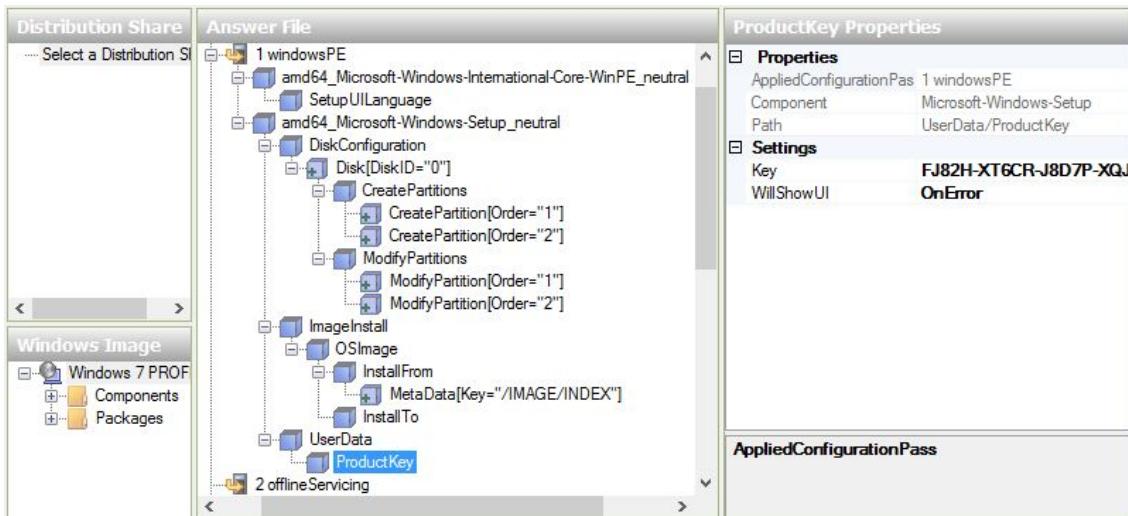


Figura 51: Clave de producto.

Saltar activación del producto.

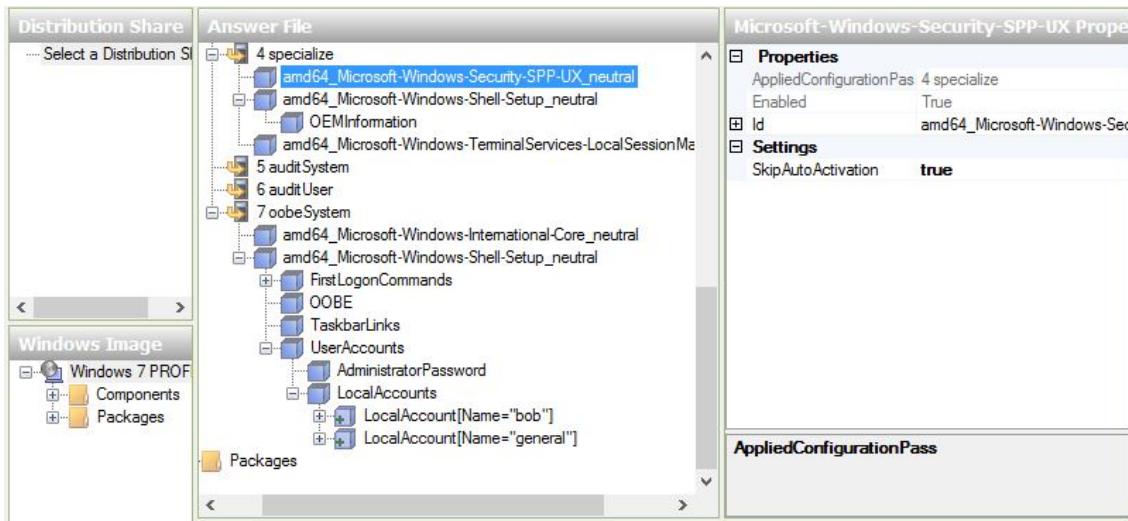


Figura 52: Activación del producto.

Para solucionar el problema de asignar un hostname único a cada máquina que sea instalada con este *Autounattend.xml* se dió con la siguiente solución. Utilizando un string representativo en el campo RegisteredOwner y un asterisco en el campo ComputerName, se obtiene un hostname pseudo-aleatorio, ya que estará compuesto por máximo ocho caracteres de RegisteredOwner y/o RegisteredOrganization concatenado con caracteres aleatorios. El campo ComputerName es un string con un tamaño máximo de quince caracteres.

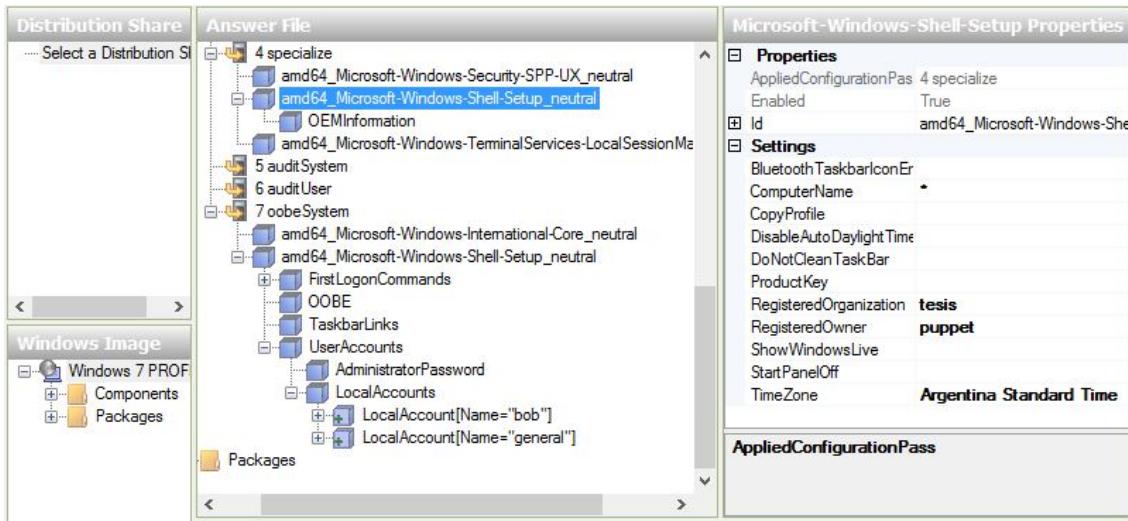


Figura 53: Hostname, organización, propietario y zona horaria.

Se debe ingresar nuevamente la configuración deseada.

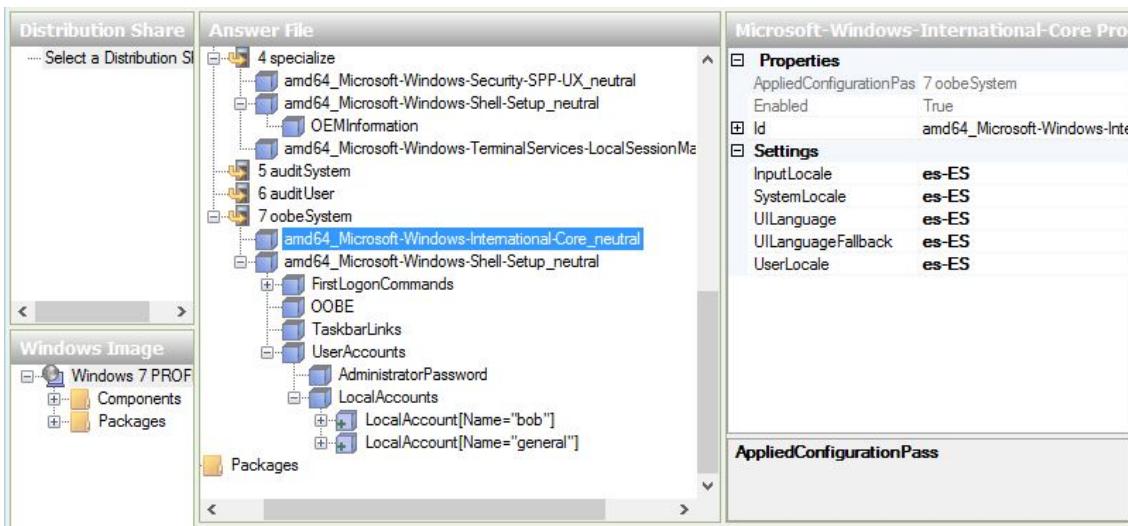


Figura 54: Idioma, teclado, regional.

Se debe ingresar nuevamente la configuración deseada.

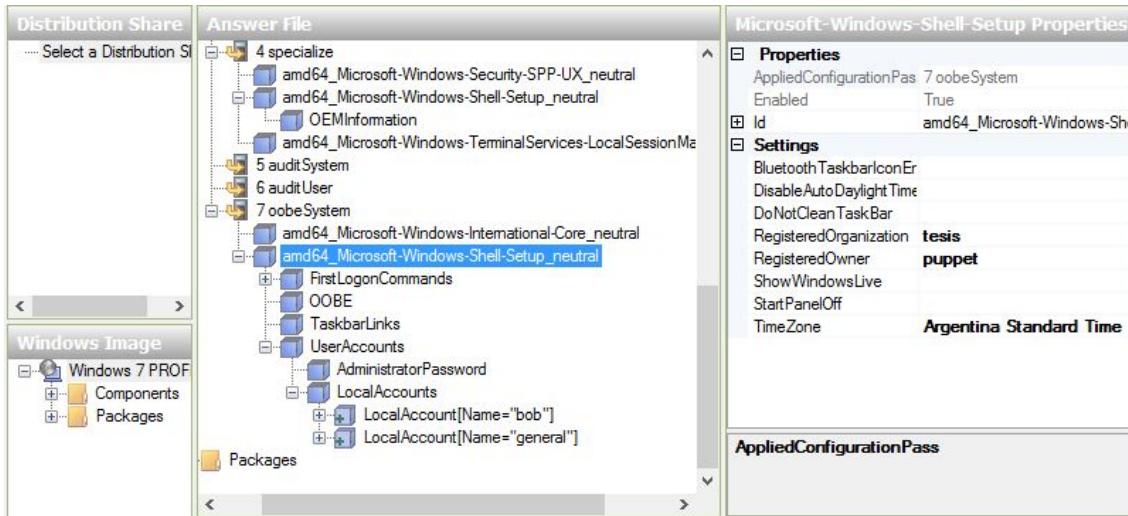


Figura 55: Organización, propietario y zona horaria.

Selección de la configuración rápida de red.

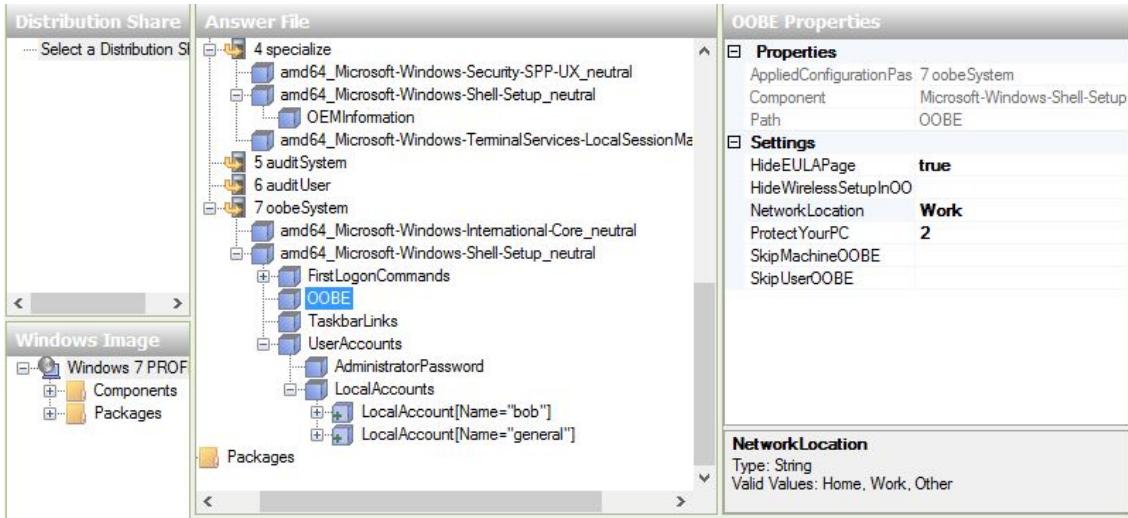


Figura 56: Configuración de red.

Configuración de contraseñas de administrador.

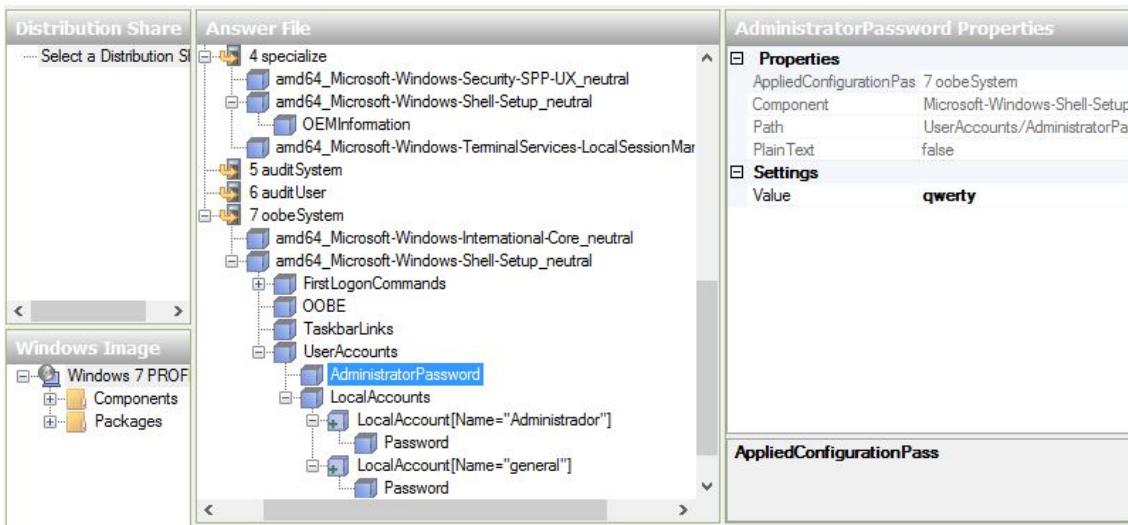


Figura 57: Contraseña administrador.

Creación de cuentas de usuario.

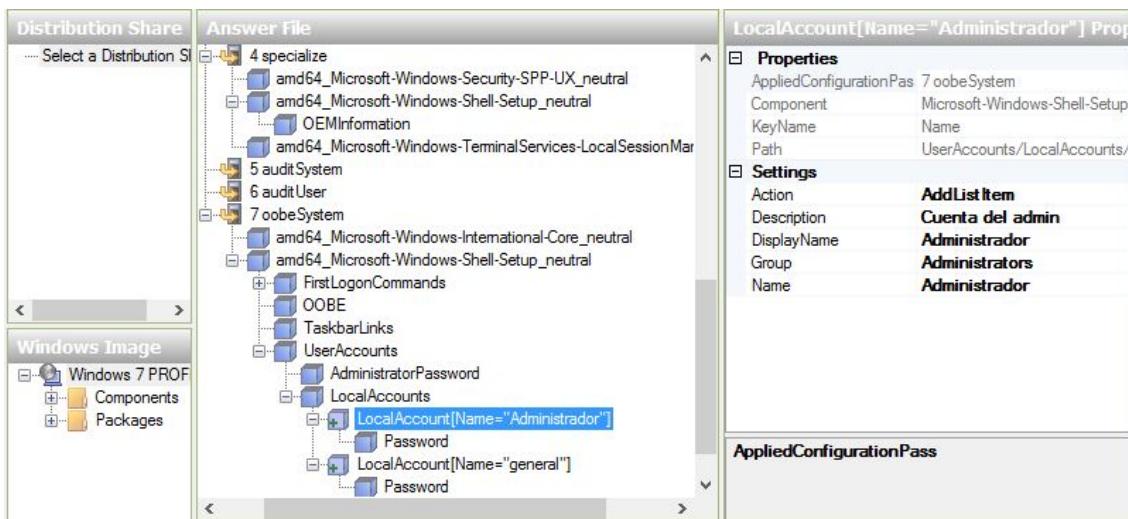


Figura 58: Cuentas de usuario, administrador.

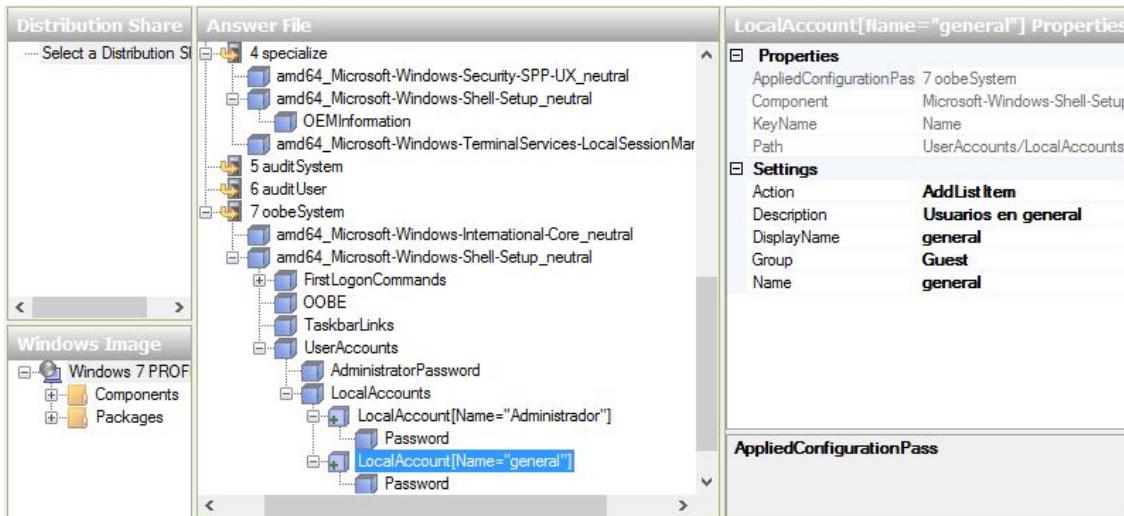


Figura 59: Cuentas de usuario, invitado.

10.1.7. Crear nuevos medios de instalación de Windows 7 para la imagen personalizada

En la máquina técnica realizar las siguientes operaciones:

- Descomprimir todo el contenido de la imagen de instalación sin modificar de Windows 7 en \path-diretorio-instalacion\.
- En el sub-directorio Sources (\path-diretorio-instalacion\Sources\), substituir la imagen install.wim, por la recientemente imagen capturada imagenPersonalizada.wim, renombrándola como install.wim.
- En el directorio principal de donde ha sido extraída la imagen, copiar el archivo de respuestas de instalación Autounattend.xml.
- Abrir Deployment Tools Command Prompt como administrador, y ejecutar:
 - oscdimg -m -h -u2 -bd:\path-diretorio-instalacion\boot\etfsboot.com d:\path-diretorio-instalacion\ d:\path-diretorio-destino\Win7-personalizado.iso

En este punto se tiene una imagen de instalación personalizada y automatizada lista para ser instalada en cualquier equipo. A continuación, se verá la preparación necesaria del servidor Linux para poder instalar y configurar SAMBA y la integración de Windows 7 con Cobbler para el posterior despliegue a través de la red.

10.2. El lado de Linux

Lo primero que se debe hacer es copiar todos los archivos necesarios, anteriormente generados en Windows, al servidor de despliegue, Cobbler.

En el servidor crear el siguiente directorio, donde se colocarán los archivos winpe_cobbler_amd64.iso y Win7-personalizado.iso:

```
sudo mkdir /var/lib/cobbler/isos
```

Darle los permisos necesarios:

```
sudo chmod -R 777 /var/lib/cobbler/isos/*.iso
```

También es necesario crear un directorio que se utilizará como zona de intercambio entre Linux y Windows:

```
sudo mkdir /windows/
```

En este directorio se debe copiar el contenido de la imagen de instalación de Windows personalizada, para ello:

```
sudo mkdir /mnt/windows
```

```
sudo mount -o loop /root/isos/WIN7_X64.iso /mnt/windows
```

```
sudo cp -rf /mnt/windows /windows
```

```
sudo umount /mnt/windows
```

La zona de intercambio entre ambos sistemas operativos se logra utilizando SAMBA.

10.2.1. SAMBA

SAMBA es una implementación libre del protocolo SMB (Server Message Block) de archivos compartidos de Microsoft para sistemas de tipo UNIX. De esta forma, es posible que computadoras con GNU/Linux, o Unix en general se vean como servidores o actúen como clientes en redes de Windows. También permite validar usuarios haciendo de Controlador Principal de Dominio (PDC), como miembro de dominio e incluso como un dominio Active Directory para redes basadas en Windows; aparte de ser capaz de servir colas de impresión, directorios compartidos y autenticar con su propio archivo de usuarios.

Como alternativa se podría utilizar el protocolo FTP que es utilizado tanto sobre Linux como sobre Windows.

Para instalar SAMBA en CentOS 7 ejecutar:

```
sudo yum install -y samba samba-client samba-common samba-winbind
```

Una vez finalizada la instalación agregar una ubicación compartida en el archivo de configuración */etc/samba/smb.conf* como se muestra a continuación:

```
[global]
workgroup = PXESERVER
server string = Samba Server Version %v
log file = /var/log/samba/log.%m
max log size = 50
idmap config * : backend = tdb
cups options = raw
netbios name = pxe
map to guest = bad user
```

```

dns proxy = no
public = yes
## Para instalaciones multiples no bloquear el kernel
kernel oplocks = no
nt acl support = no
security = user
guest account = nobody

```

[imagen]

```

comment = Windows 7
path = /windows
read only = no
browseable = yes
public = yes
printable = no
guest ok = yes
oplocks = no
level2 oplocks = no
locking = no

```

Habiendo modificado la configuración reiniciar el servicio:

```
sudo systemctl restart smb
```

Corroborar que se pueden compartir archivos, si se hace desde otra pc Linux, es necesario tener los siguientes paquetes:

```
sudo yum install samba samba-client samba-common cifs-utils
```

Luego ejecutar:

```

sudo smbclient -L IP_del_servidor_Cobbler
sudo mkdir /media/samba
sudo mount // IP_del_servidor_Cobbler /imagen/media/samba/
sudo ls /media/samba/

```

Deberían aparecer ya los archivos compartidos en esa ubicación.

10.2.2. Integración con Cobbler

Para integrar la imagen de instalación Windows al servidor Cobbler se debe añadir una entrada distro y una entrada de perfil a Cobbler.

```
sudo cobbler distro add --name=windows7-x86_64 --kernel=/usr/share/syslinux/memdisk  
--initrd=/var/lib/cobbler/isos/winpe_cobbler_amd64.iso --kopts="raw iso" --arch=x86_64  
--breed=windows
```

```
sudo cobbler profile add --name=windows7-x86_64 --distro=windows7-x86_64
```

```
sudo cobbler sync
```

```
sudo systemctl restart cobblerd
```