Escuela Técnica Superior de Ingenieros de
Telecomunicación
Universidad de Valladolid

SISTEMA DE INTEGRACIÓN CONTINUA PARA EL DESARROLLO DEL SOFTWARE Y HARDWARE EN SERVIDORES Y ESTACIONES DE TRABAJO DE EMPRESA

CONTINUOUS INTEGRATION SYSTEM FOR SOFTWARE AND HARDWARE DEVELOPMENT INTO COMPANY SERVERS AND WORKSTATIONS

Autor: Daniel Fradejas Pascual
Tutor: Federico Simmross Wattenberg
Tutor de empresa: Jorge Gonzalez Villalonga

Valladolid a 28 de junio de 2019

Para mis padres y hermana,
que siempre creyeron en mí,
para mis amigos, tutores, compañeros de trabajo,
profesores y gente que me han visto y hecho
crecer física e intelectualmente.
Os lo debo todo a vosotros.

Resumen

El tiempo empleado para desplegar un determinado proyecto es una de las características mayores a tener en cuenta. Cuanto menor sea este, la capacidad de respuesta frente a un fallo es mayor, y por tanto aumenta la calidad del propio proyecto. Además, la frecuencia en la que se despliega el código también es un factor a tener en cuenta, pues conforme éste se modifica, es buena práctica comprobar que dichos cambios no han estropeado el despliegue del sistema, y que se siguen cumpliendo una serie de factores. A esta práctica se le llama comúnmente como integración continua.

En este trabajo fin de grado se va a proponer una solución de integración continua a la empresa *Elecnor Deimos Space* en su proyecto *MSF*. Para ello se propondrá una herramienta capaz de implementar este sistema en el proyecto, teniendo en cuenta las tecnologías ya existentes en el mismo, y las prácticas habituales de los empleados. Esta herramienta se desarrolla en el lenguaje de programación *Perl* y consta de cuatro partes bien diferenciadas: la encargada de la instalación (despliegue del código desarrollado) de las máquinas de forma desatendida y con un control centralizado de las mismas, una interfaz web dónde se recoja información necesaria para el usuario que realice el despliegue del código y que sirva como centro de operaciones, una encargada de generar y compactar una imagen de disco que contenga el código actualizado del proyecto y, por último, una aplicación encargada de gestionar estas tres para implementar una integración continua en el proyecto.

Esta ultima aplicación será llevada a estudio, se valorará si es más eficiente y adecuada la utilización de una herramienta ya existente en el mercado, o desarrollar una aplicación centralizada propia de manera análoga a las anteriores.

Por último se evaluará la solución propuesta y se validará en un entorno de producción pertinente a la empresa para la que se ha desarrollado este sistema.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y conocimientos previos

1.1. Motivación

El mundo del desarrollo de *software* se encuentra en un punto álgido hoy en día. Esto es debido a la gran cantidad de avances tecnológicos de estos últimos años, pero también debido a la mejora de los sistemas sobre los que se desarrolla y las prácticas habituales en el equipo desarrollador. Respecto a los sistemas, estos pueden ser repositorios compartidos en red, herramientas de desarrollo, sistemas operativos, métodos de despliegue, etc. Por otro lado, las prácticas habituales son la organización y estructuración de tareas, conocimiento general sobre las nuevas tecnologías, etc. Pero, sin duda alguna, una de las prácticas utilizadas más importantes ha sido la integración continua.

"La **integración continua** ayuda a que los desarrolladores fusionen los cambios que introducen en el código para incorporarlos a una división compartida (o rama) con más frecuencia, incluso diariamente. Una vez que se fusionan los cambios implementados por un desarrollador en una aplicación, se validan con el desarrollo automático de la aplicación y la ejecución de distintos niveles de pruebas automatizadas (generalmente, pruebas de unidad e integración) para verificar que los cambios no hayan dañado la aplicación."[Red].

Esta práctica ha permitido a los equipos de desarrolladores el ahorro de una valiosa cantidad de tiempo y recursos. Normalmente, con una frecuencia relativamente grande, se ponía en común el código desarrollado por cada miembro del equipo. Esto suponía la aparición de numerosos errores en el código introducido por cada uno de los miembros, además de suponer una tarea ardua y difícil el corregir y encontrar estos fallos al desplegar el código.

Con el uso de la integración continua, el código añadido por cada uno de los miembros se pone en común con una frecuencia relativamente alta, y sin suponer ningún trabajo extra, pues se realiza automáticamente y de forma desatendida. Además de poner en común el código, lo realmente eficiente de esta práctica, (y lo que ha marcado la diferencia), es la capacidad de poder probar, desplegar y encontrar errores de manera rápida y en un periodo de tiempo relativamente corto (y, por supuesto, totalmente

automatizado).

Pero, ¿Qué nos permite la integración continua? Con la integración continua, los desarrolladores envían los cambios de forma periódica a un repositorio compartido con un sistema de control de versiones... Un servicio de integración continua crea y ejecuta automáticamente pruebas unitarias en los nuevos cambios realizados en el código, para así identificar inmediatamente cualquier error. Es decir, para una buena integración continua se necesita un sistema de control de versiones, donde todos los desarrolladores del proyecto en cuestión pongan en común sus cambios. De esta manera, todo el código se encuentra centralizado y estructurado en versiones para su posterior despliegue y depuración de errores.

En concreto, en el proyecto *MSF* (proyecto en el cual se ha desarrollado el sistema que trata este trabajo de fin de grado), esta practica no se lleva a cabo. Al contrario, se realiza un despliegue manual en el que se emplea una gran cantidad de recursos por parte de la empresa. Esto es debido a que su despliegue consta de operaciones delicadas y que se necesitan realizar con suma rigidez, debido a que el proyecto *MSF* no se basa únicamente en un *software*, sino que se trata del conjunto de *software* y sistema, junto con unas instrucciones de despliegue. Es decir, no solo se desarrolla el código, sino que también hay que tener en cuenta el desarrollo de las funcionalidades del sistema operativo, de su instalación, la topología de la red, etc. Esto conlleva a que el despliegue del proyecto no se realice únicamente instalando el *software* una única vez (como viene siendo práctica habitual en la mayoría de los sistemas desarrollados hoy día) y posteriormente lanzando actualizaciones periódicas, sino que es necesaria una reinstalación casi constante del sistema, suponiendo esto una gran carga de trabajo para los integradores de sistemas del proyecto.

El despliegue de este proyecto *MSF* se ha visto mejorado en varias ocasiones. En primer lugar, el despliegue consiste en una instalación manual del sistema operativo en los servidores y estaciones de trabajo, configurando posteriormente el *software* desarrollado y modificando las configuraciones del sistema operativo manualmente. Cada paso queda registrado en un manual extenso y complejo[1], que explicaba paso por paso qué hacer en cada momento y los resultados esperados para realizar un despliegue satisfactorio. Esto conlleva una gran cantidad de tiempo y recursos por parte de la empresa, por lo que en determinado momento se valora una mejora de dicho sistema.

Valorando el problema, se proponen y se llevan a cabo varias soluciones. Se mejora la instalación del sistema operativo creando unas imágenes ISO[2] (que comúnmente se llamará *build* en este trabajo de fin de grado) en las cuales se incluye el sistema operativo utilizado junto a el código del proyecto y algunas directivas de instalación que se ven interpretadas automáticamente gracias a una tecnología *Kickstart* [Red19a] que se explicará más adelante en este mismo trabajo. Esta imagen ISO se graba en

[1]Se trata de un manual de 400 páginas, en los que se detallaban paso a paso cada operación y sus resultados esperados para desplegar el proyecto.

[2]Almacena una representación perfecta de un DVD o CD. En este proyecto, se generan y posteriormente se graban en un DVD para su posterior ejecución en servidores y estaciones de trabajo. También pueden ser montadas estas imágenes .iso virtualmente o servidas por red.

un DVD, y se inserta en los sistemas a instalar, pidiendo varios parámetros necesarios para la instalación al integrador del sistema. Esta simplificación permite, mediante 20 o 30 pasos, un despliegue del proyecto *MSF* de manera más eficiente que en el caso del manual. Sin embargo, este modo de despliegue requiere una presencia física del integrador del sistema, tanto para insertar el DVD como para introducir las opciones del sistema operativo, arrancar desde el DVD, crear la imagen ISO, etc.

La solución a esta interacción humana necesaria para el despliegue del proyecto, es lo que se recoge en este trabajo de fin de grado. Con este sistema se podrá automatizar la creación de estas imágenes ISO, y su posterior instalación **simultánea** en todos los servidores y estaciones de trabajo del proyecto, sin ninguna interacción humana, y con un posterior control de errores. Esta herramienta (o sistema) nos permite implementar una integración continua en el proyecto *MSF* que ahorrará gran cantidad de recursos a la empresa a la que se destina este proyecto (*Elecnor Deimos Space*).

1.2. Objetivos

El objetivo de este trabajo fin de grado es la **implementación de un sistema que nos permita una integración continua** en el proyecto *MSF*, lo que lleva consigo una automatización del despliegue del proyecto. El desarrollo de este sistema busca un gran cambio en el proyecto *MSF*, liberando el tiempo empleado en el despliegue de los sistemas a los integradores, permitiendo así una liberación del tiempo disponible, para poder emplearlo a las demás tareas. Hoy día, el tiempo útil de los empelados en una empresa se traduce en beneficios para la misma, por lo que se busca la mayor automatización posible en las tareas repetitivas y en las cuales se invierte la mayor parte del tiempo.

En concreto, el proyecto *MSF*, cómo se ha comentado anteriormente, dispone de un despliegue del código muy peculiar y, a la vez, complejo de realizar. La práctica común de la mayoría de los sistemas se basa en la utilización de repositorios centralizados, donde se almacena el código del proyecto y en los cuales se almacenan a la vez ficheros encargados de compilar este código y generar un binario ejecutable de la aplicación. Sin embargo, el proyecto *MSF* se basa en entregar dos imágenes de disco (una para el servidor y otra dedicada para la estación de trabajo), con las cuales se pueda desplegar tanto un sistema operativo como el proyecto *MSF* con los menores pasos posibles (para así evitar errores en producción). Por tanto el despliegue de este proyecto no se puede realizar con una herramienta ya existente en el mercado (cómo puede ser *Atlassian Bamboo* [Atl19], *Jenkins* [Jen19], *RunDeck* [Run19]...), sino que es necesaria su simplificación, para luego usar (o no) dichos sistemas. Por tanto otro de los objetivos de este trabajo de fin de grado es **simplificar y automatizar el despliegue**, de manera que se pueda realizar de manera más sencilla y desatendida.

En concreto, en este trabajo de fin de grado se discutirá sobre si es adecuado utilizar un sistema ya desarrollado y comercializado encargado de proporcionar una integración continua; pero, para ello, lo primero consistirá en abstraer las tareas necesarias del despliegue, de forma que su ejecución sea tan simple como la ejecución de un *script* o

una petición *HTTP*.

1.3. Estructura de la memoria

En el capitulo dos de este trabajo de fin de grado se detallarán las tecnologías base existentes hoy en día en el mercado, que nos proporcionan una solución de integración continua en el proyecto, y se valorará si son de utilidad para el proyecto *MSF* o no. Además, se comentará la estructura de la red corporativa (la cual es necesaria conocer para una mejor comprensión de las tareas realizadas), y de las tecnologías existentes en las cuales se desarrollará el sistema.

En el tercer capitulo se realizará un análisis exhaustivo de las aplicaciones a desarrollar, las cuales abstraerán los pasos a realizar para tareas concretas del despliegue del sistema, como creación de imágenes ISO, instalación de máquinas, notificaciones... Además en este punto, se retoma la discusión sobre si es necesario emplear un sistema de integración continua ya existente en el mercado y utilizarlo.

En el capitulo cuatro se procederá a detallar el diseño de dichas aplicaciones. Este detallado se hará mediante los directorios, tecnología y *scripts* utilizados en cada aplicación dedicada para cada tarea. Además, se explicará cómo se realiza la integración continua y su funcionamiento detallado.

El quinto capítulo mostrará la validación del sistema empleado. Es decir, se analizará con detalle el sistema desarrollado y se comprobará tanto su funcionamiento cómo las mejoras que implementa al proyecto.
El sexto capítulo mostrará las conclusiones tomadas a partir de haber realizado y probado el sistema de integración continua, y se propondrán sus líneas futuras.

Por último, a modo de apéndices, se añadirá un manual en el cual se detalla lo necesario para desplegar este trabajo de fin de grado en cualquier máquina con un sistema operativo *Linux Centos 7*, y un segundo apéndice en el que se almacenan las imágenes tomadas para la validación del sistema desarrollado.

Capítulo 2

Tecnologías base

En este capítulo se van a estudiar las tecnologías existentes hoy día en el mercado, y (cómo se ha adelantado en la introducción) por qué no sirven actualmente para el tipo de despliegue que se maneja en la empresa. A continuación se hace una breve introducción sobre las tecnologías existentes en el proyecto *MSF* y comprender de dónde parte este trabajo de fin de grado.

2.1. Soluciones existentes

Hoy en día podemos encontrar un amplio catálogo de sistemas que proporcionan una integración continua del código, entre los cuales los más destacados son:

- *Jenkins*.
- *Travis CI*.
- *Bamboo*.
- *GitLab CI*.
- *RunDeck*.
- etc.

Estos son los principales sistemas de integración continua utilizados hoy día. Sin embargo existe una gran cantidad de ellos en el mercado, pues la integración continua se está convirtiendo en una pieza básica de todo proyecto de desarrollo de *software*

Jenkins es la herramienta más conocida del mercado. Este *software* viene del proyecto *Hudson* que comenzó a desarrollarse en 2005 con el lenguaje de programación *Java*, y que cuenta en la actualidad con varias funciones adicionales disponibles (alrededor de 1000 extensiones). *Jenkins* también permite, además de un despliegue continuo del proyecto, una entrega continua del proyecto a producción. Además es compatible con diversos sistemas de control de versiones y, además de poder ser manejada mediante ordenes en un interprete, dispone de una interfaz gráfica que facilita este trabajo. Una de las principales características que lo definen es que se trata de un sistema de código

abierto, gratuito y comúnmente utilizado por la comunidad informática.

Travis CI es un *software* estrechamente relacionado con el conocido sistema de control de versiones *GitHub*. Esta herramienta se puede configurar con un fichero *YAML*, como los que se utilizan en este mismo trabajo de final de grado, guardado en el directorio raíz del proyecto. Se trata de un sistema programado en *Ruby*, multiplataforma, que proporciona una conexión directa con *GitHub*. El precio de este *software* en el mercado es de alrededor de 68 dólares al mes para usuarios particulares y de 489 dólares al mes para corporaciones, y es de código abierto al igual que *Jenkins*.

Bamboo es el sistema de integración continua desarrollado por el equipo *Atlassian*, que gestiona también el servicio de alojamiento de ficheros *BitBucket*. Se trata de un *software* que comenzó a desarrollarse en 2007, y que no solo sirve de ayuda para la integración continua, sino que también permite automatizar el despliegue y la gestión de los lanzamientos a producción. Este sistema, al igual que los anteriores, también presenta una interfaz web, accesible desde cualquier máquina conectada a la red. Está escrito en *Java*, es multiplataforma y tiene un precio de entre 10 y 1100 dólares dependiendo del número de servidores utilizados.

Al igual que *Travis CI* está estrechamente relacionado con *GitHub*, *GitLab CI* forma parte del conocido sistema de control de versiones *GitLab*. Al igual que los anteriores, ofrece además de una integración continua, un despliegue y entrega continuos. Está programado en *Ruby* y *Go* y dispone de un fichero de configuración tipo *YAML*. Existen dos tipos, la versión gratuita que presenta muy pocas funciones adicionales, y las versiones de pago que sí implementan estas funciones. Su precio ronda entre los 4 y 99 dólares al mes por usuario.

Por último *RunDeck* es otro sistema de integración continua, que permite tanto el despliegue como la entrega automatizada. Dispone de una versión de pago y otra versión gratuita. La versión gratuita dispone de un menor catálogo de posibilidades frente a la que ofrece la versión de pago, la cual presenta características similares a los anteriores sistemas.

Como se puede ver, existe gran variedad de sistemas de integración continua, los cuales ofrecen un catálogo de posibilidades muy similar, en donde lo único en lo que se diferencian es en el precio y en las compatibilidades con otros sistemas.

2.2. PXE (*Preboot eXecution Environment)*

Para la instalación desatendida y paralela de las plataformas del proyecto se hará uso de la tecnología PXE [Int99].

La especificación del *Preeboot eXecution Environment (PXE)* describe un entorno cliente-servidor estandarizado que inicia un conjunto de software, principalmente una imagen de un sistema operativo, obtenido de una red. En el lado del cliente solo se requiere un controlador de interfaz de red (*Network Interface Controller, NIC*) com-

patible con *PXE*, y un pequeño conjunto de protocolos de red, como *DHCP* y *TFTP*.

Los orígenes de protocolo *PXE* se remontan a los primeros días de los protocolos *BOOTP/DHCP/TFTP* y, debido a su gran utilidad, a partir de 2015, forma parte del estándar *Unified Extensible Firmware Interface (UEFI)*, por lo que lo hace el estándar de facto en la industria. En los centros de datos modernos y otros grandes sistemas en red, *PXE* es la opción más frecuente para el inicio, la instalación y la implementación del sistema operativo.

El entorno *PXE* se basa en una combinación de protocolos de Internet estándar de la industria, a saber, *UDP/IP*, *DHCP* y *TFTP*. Estos protocolos se seleccionaron porque se implementan fácilmente en el *firmware* de la *NIC* del cliente, lo que da como resultado *ROMs PXE* estandarizadas de tamaño reducido. La estandarización, el pequeño tamaño de las imágenes de *firmware* de *PXE* y su bajo uso de recursos han favorecido que el lado del cliente del estándar *PXE* se implemente de forma idéntica en una amplia variedad de sistemas, desde potentes computadoras cliente a máquinas de una sola placa e incluso dispositivos "*System on Chip*".

Como se ha mencionado anteriormente, *PXE* hace uso de los protocolos *DHCP* y *TFTP* cuyas funciones son:

- *DHCP* se usa para proporcionar los parámetros de red del cliente apropiados y específicamente la ubicación (dirección *IP*) del servidor *TFTP*, donde se encuentra disponible el *Network Bootstrap Program (NBP)* y otros ficheros complementarios. Para iniciar una sesión de arranque *PXE*, el componente *DHCP* del *firmware PXE* del cliente difunde un paquete *DHCPDISCOVER* que contiene opciones específicas de *PXE* al puerto *67/UDP* (puerto del servidor *DHCP*); solicita la configuración de red requerida y los parámetros de arranque de red. Las opciones específicas de *PXE* identifican la transacción *DHCP* iniciada como una transacción *PXE*. Los servidores *DHCP* estándar (no habilitados para *PXE*) podrán responder con un *DHCPOFFER* normal que lleve información de red (es decir, dirección *IP*) pero no los parámetros específicos de *PXE*. Un cliente *PXE* no podrá iniciarse si solo recibe una respuesta de un servidor *DHCP* no habilitado para *PXE*.

- Una vez configuradas las propiedades de red, el cliente puede acceder a los ficheros de arranque del servidor *TFTP*, ya que conoce su dirección y el nombre del fichero de arranque. A continuación, el cliente lo transfiere a su propia memoria *RAM*, verifica que es un fichero de arranque válido, y finalmente lo ejecuta. Estos ficheros de arranque contienen una lista con un pequeño conjunto de ficheros complementarios para ejecutar un SO minimalista (*WindowsPE*, un núcleo de *Linux* básico + *initrd*, etc). Esta instancia del sistema operativo carga sus propios controladores de red y la pila *TCP/IP*. En este punto, las instrucciones restantes necesarias para iniciar o instalar un sistema operativo completo no se proporcionan a través de *TFTP*, sino que utilizan un protocolo de transferencia robusto (como *HTTP*, *CIFS* o *NFS*) y ya no es gestionada por *PXE*, si no por el

propio sistema operativo, y dependiendo del que se escoja, empleará una implementación diferente. En este caso, se ha seleccionado como sistema operativo *Red Hat Enterprise Linux 7*, por lo que este tecnología será la de *Kickstart* [Red19a].

2.3. Kickstart

Kickstart es la tecnología empleada por la distribución *Red Hat Enterprise Linux* para automatizar instalaciones de su sistema operativo. Consta de un fichero *Kickstart* de texto sin formato, que contiene unas palabras clave con su propia sintaxis que sirven como indicaciones para la instalación. Cualquier editor de texto capaz de guardar ficheros como texto *ASCII* (como *Gedit* o *vi* en sistemas Linux, o *Notepad* en sistemas Windows) se puede utilizar para crear y editar ficheros *Kickstart*.

El enfoque recomendado para crear este tipo de ficheros es realizar primero una instalación a mano en un sistema. Una vez que la instalación se complete, todas las elecciones hechas en el instalador se guardarán en el denominado `anaconda-ks.cfg`, localizado en el directorio `/root/` en el sistema de ficheros instalado. Partiendo de la base de este fichero se puede modificar la instalación del sistema operativo a gusto del administrador, pudiéndose automatizar así todos los pasos que se han realizado a mano anteriormente.

Además, este fichero *Kickstart* permite la ejecución de *scripts* antes y después de la instalación. Estos son los dos apartados clave en el despliegue de este trabajo fin de grado, pues es aquí donde se realizan los pasos necesarios (anteriormente realizados a mano) para desplegar el código en el sistema operativo. Además, se configuran algunas directivas como la instalación de paquetes, los repositorios, etc.

Una de las cosas que, como se ha dicho anteriormente, no se hacen automáticamente, es la configuración de red, pues dependiendo del servidor donde insertemos el DVD, este tendrá una configuración IP u otra. Este punto es el que se tratará de automatizar, además de el envío de los resultados de la instalación al sistema, para tener una instalación completamente desatendida.

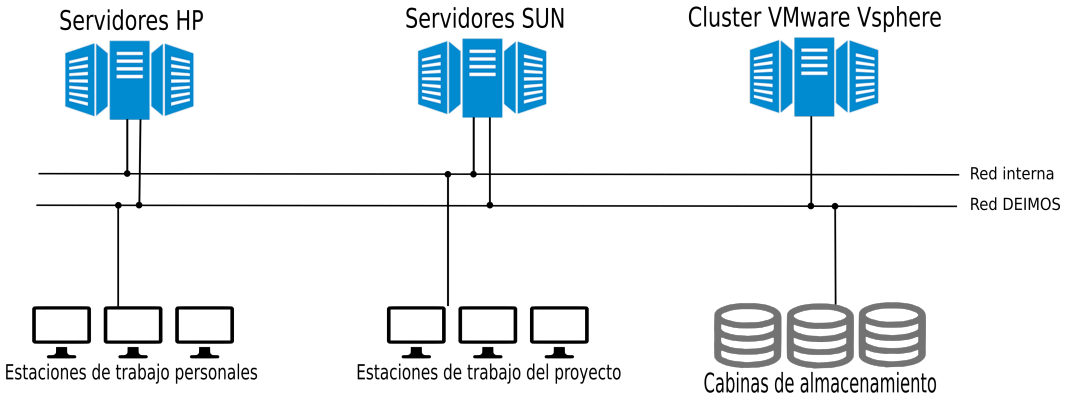
2.4. Tecnología existente en el proyecto *MSF*

Como se ha dicho anteriormente, el proyecto *MSF* presenta un método de despliegue costoso en el que se malgasta gran cantidad de tiempo y recursos, por lo que huelga decir que este trabajo de fin de grado no parte desde cero, si no que existen ciertas tecnologías y desarrollos ya realizados.

2.4.1. Red corporativa y máquinas disponibles

En primer lugar se hace una pequeña introducción sobre la estructura de la red corporativa, en la cual se realizará este trabajo de fin de grado. Como se puede ver en la figura 2.1, en la red corporativa existen diversos dispositivos:

Figura 2.1: Topología de la red local Deimos.



*Nota: En la red interna existen diferentes VLANs para que solo sean visibles entre sí un servidor con sus respectivas estaciones de trabajo.

- **Servidores**. En ellos se desarrolla el código del proyecto. Posteriormente, este código se ve embebido en un disco de instalación de sistema operativo, y se despliega de nuevo en estos mismos servidores. Estos servidores van acompañados de sus correspondientes estaciones de trabajo, las cuales sirven de interfaz para el cliente, y se comunican con su servidor correspondiente a partir de una red interna privada. Esta red, en la fase de desarrollo del proyecto, se lleva a cabo con la tecnología *VLAN*[1], de esta forma recreamos una situación de enlace directo, privado y seguro entre el servidor y sus respectivas estaciones de trabajo. Estos servidores, además, tienen conexión al exterior, en este caso la red corporativa. Dentro de estos servidores podemos encontrar varios tipos:

  1. Servidores *HPE PROLIANT*. Se trata de servidores de marca *Hewlett Packard, HP*, con 4 tarjetas de red (todas con soporte de arranque por red) y una tarjeta de gestión tipo *iLO*. *iLO* es un procesador de mantenimiento, que permite la monitorización y el control del propio servidor, embebido en la placa madre de los mismos [Hew18].
  2. Servidores *Oracle Sun*. Servidores de marca *Oracle Sun*, también en este caso con cuatro tarjetas de red, todas con arranque por red soportado, y una tarjeta de red aparte para el sistema *ILOM*. Este sistema es análogo al de servidores *HPE* [Ora13].
  3. Servidores virtualizados con el software de virtualización *KVM*. Aparte de estos servidores físicos, también se dispone de servidores virtualizados. En este caso, están desplegados con el software de virtualización *KVM* de *Red Hat Enterprise Linux*. Este software permite virtualizar diversas plataformas (servidores y estaciones de trabajo), e incluso permite crear una plataforma híbrida, con un servidor físico y su correspondiente estación de trabajo virtualizada o viceversa [Red19d].

[1]VLAN (Virtual Local Area Network) es de mecanismo que permite a los administradores de red crear redes virtuales aisladas a partir de una misma red. Esta práctica es útil para la reducción de tráfico en la red, para poder mantener terminales conectados sin la necesidad de que estén físicamente en el mismo lugar y para incrementar la seguridad de la red de nuestro sistema. [Cis]

4. *Cluster VMWare Vsphere*. Se trata de una agrupación de servidores, conectados en paralelo, los cuales comparten entre sí memoria y recursos de procesador (*Cluster*) de varios servidores tipo *Hewlett Packard, HP*, cada uno de ellos con el software *VMware ESXi* [VMW18b], y coordinados con el software de virtualización *VMware vSphere 6.7, vCenter Server 6.7* [VMW18a]. Esta plataforma permite virtualizar tanto servidores como estaciones de trabajo, conexiones, topología de redes, almacenamiento, etc.

- Estaciones de trabajo. Existen dos tipos de estaciones de trabajo en el proyecto *MSF*:

  1. Estaciones de trabajo del proyecto. Son las directamente conectadas a los servidores, y que se encuentran instaladas con el software desarrollado.
  2. Estaciones de trabajo personales. Ordenadores corporativos, desde los cuales los empleados desarrollan el código del proyecto, y se conectan a los servidores y estaciones de trabajo para su mantenimiento, desarrollo y lanzamiento de simulaciones (que es el principal uso que se realiza a los servidores y estaciones de trabajo instalados con el software del proyecto *MSF*).

- Cabinas de almacenamiento. Se trata de un sistema de discos de almacenamiento, como su propio nombre indica, donde se almacenan los ficheros relevantes para el proyecto. Por ejemplo, los entornos de trabajo de cada empleado, resultados de simulaciones, ficheros de intercambio, etc. Estos discos se encuentran accesibles en red, por lo que los usuarios pueden acceder a ellos desde cualquier servidor o estación de trabajo a partir de la *red DEIMOS*, como se ha llamado en la figura 2.1.

- Servidor *Atlassian*. Esta máquina se encuentra conectada a la *red DEIMOS*. En él se encuentran todos los servicios utilizados en el proyecto *MSF* (*BitBucket* como servidor de control de versiones *Git*, *Bamboo* como herramienta de integración continua, *Jira* como control de tareas, etc.). De este servidor se recoge el código desarrollado del proyecto para su posterior despliegue.

2.4.2. Método actual de despliegue

El código del proyecto presenta una gran variedad de lenguajes, entro los cuales destacan por orden de importancia: *Java*, *C++*, *Bash*, *Perl* y *Python*. Por tanto, el despliegue del código no es único, pues se necesita compilar el código *Java* con su compilador específico incluido en el *JDK*, en el caso del código *C++* necesita ser compilado en el compilador *gcc*, los lenguajes interpretados deben de ser organizados en directorios específicos, etc. Por tanto, esto deriva en que el despliegue no se va a traducir en una simple orden, si no que son necesarios una serie de pasos específicos.

Además, cómo se ha comentado en la introducción, no solo se entrega este *software*, sino que se debe entregar junto a un sistema operativo. En este caso se utiliza *RedHat Enterprise Linux*, y para su despliegue se crean una imágenes de disco (ISO) en las cuales va embebido el sistema operativo, el *software* desarrollado, y unas directivas de

instalación que permiten desplegar este *software*.

Estas directivas, en *Red Hat Enterprise Linux*, están recogidas en un fichero *Kickstart* [Red19a], en el cual se indica el idioma del sistema, el teclado, zona horaria, *scripts* a ejecutar antes y después de la instalación del sistema operativo, etc. El funcionamiento de esta tecnología *Kickstart* se explica en la sección 2.3.

Estas imágenes de disco se crean específicamente para el servidor y para la estación de trabajo, siendo distintas entre ellas. Mediante estas imágenes de disco creadas previamente es cómo se despliega el proyecto *MSF*, por lo que para conseguir una integración continua del proyecto (o poder utilizar un sistema externo), es necesario automatizar y centralizar la creación de estas imágenes de disco, e instalar los servidores y estaciones de trabajo remotamente y de forma completamente desatendida (sin tener que introducir el DVD manualmente).

Capítulo 3

Análisis

Una vez asentadas las bases de las que se parte en este trabajo fin de grado, y conociendo una breve introducción de lo que es necesario desarrollar para conseguir el sistema de integración continua deseado, se propone ya en detalle las partes específicas a desarrollar:

- Descripción del entorno corporativo sobre el que se desarrollará el sistema de integración continua.
- Definición de la máquina virtual en la que se alojará el sistema de integración continua.
- Desarrollo de la aplicación encargada de la instalación desatendida de las máquinas por red (servir ficheros necesarios para el arranque *PXE*, que incluye la gestión del servidor *DHCP*, *TFTP* y *HTTP* entre otros).
- Desarrollo de una interfaz gráfica para mejorar la experiencia de usuario, control de los errores y modificar parámetros de configuración.
- Desarrollo de una aplicación encargada de lanzar la creación de las imágenes ISO del servidor y la estación de trabajo (*build*) y alojarlas en un repositorio centralizado (cabinas de almacenamiento) y accesible por el sistema de integración continua.
- Desarrollo o utilización de una aplicación encargada de gestionar y automatizar la ejecución de las anteriores aplicaciones y así permitir una integración continua del proyecto.

3.1. Descripción del entorno corporativo

En los siguientes apartados se debatirá una solución para implementar el anteriormente mencionado sistema de integración continua. Para ello se debaten varios aspectos de este sistema:

1. Lugar de despliegue.
2. Objetivos específicos.
3. Tecnologías a tener en cuenta.

3.1.1. Lugar de despliegue

Teniendo en consideración los medios disponibles desarrollados en el punto anterior, las opciones disponibles en el proyecto *MSF* para desarrollar el sistema son las siguientes: servidor *HPE*, servidor *Sun*, máquina virtual con tecnología *KVM*, máquina virtual con tecnología *VMWare*, estación de trabajo del proyecto y estación de trabajo personal.

El sistema de integración continua, como se ha dicho anteriormente, precisa ser un sistema completamente automatizado y desatendido, por lo que no puede desplegarse sobre una estación de trabajo. La forma óptima es ejecutarlo en un servidor.

Los servidores *HPE* y *Sun* están reservados exclusivamente para el desarrollo del proyecto, por lo que en ellos no se puede desplegar el sistema de integración continua. Sin embargo, existen dos servidores, uno *HPE* y otro *Sun* utilizados para el soporte de máquinas virtuales tipo *KVM*.

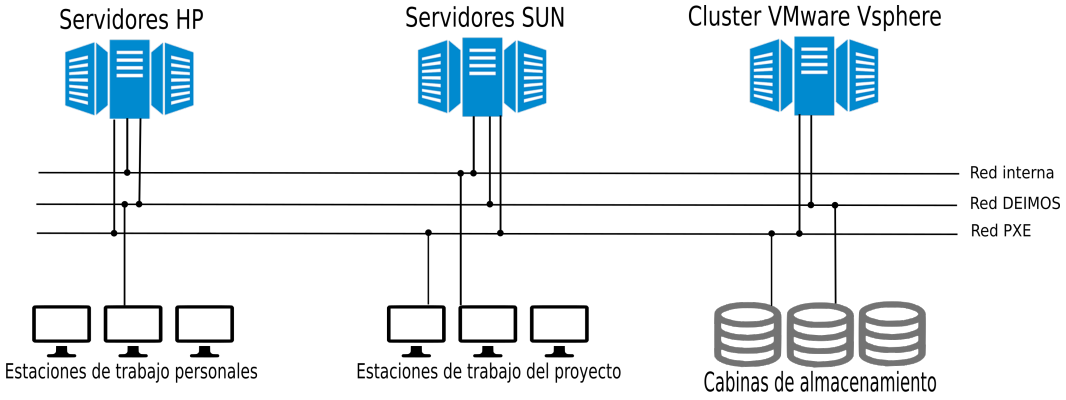
Por último existen las opciones de ejecutarlo en una maquina virtual con tecnología *KVM* o con tecnología *VMWare*. En un primer momento, las máquinas virtuales VMWare no se encontraban disponibles, por lo que se utiliza la tecnología *KVM*. Sin embargo, a lo largo del desarrollo de este trabajo de fin de grado, se migra el sistema a una máquina virtual *VMWare*, que tiene un mayor soporte y mejores características respecto a las máquinas virtuales *KVM*, por lo que la decisión final de desarrollo de este sistema será en una **máquina virtual con tecnología *VMWare***, almacenada en el *cluster* de servidores *HPE*.

3.1.2. Objetivos específicos

Como objetivos específicos del sistema se puede destacar:

- **Automatización de la instalación** (o despliegue) del software, sin necesidad de ninguna interacción humana.
- Capacidad de **detección de fallos** en cualquier momento del proceso, notificando en ficheros *logs* (los cuales recogen toda la información de un sistema en ejecución [Red19c]) o en una interfaz gráfica.
- Permitir la instalación (o despliegue) **simultánea** en las plataformas del proyecto *MSF* (servidor y varias estaciones de trabajo) que creamos convenientes, decidiendo su modo de instalación. Respecto a esto, el despliegue de este proyecto puede ser realizado de varias maneras: instalación de producción, que es el modo de instalación que se utiliza cuando el proyecto ya está entregado al cliente; o instalación de desarrollo, que se utiliza para el desarrollo del proyecto. Ambas instalaciones tienen cosas en común y cosas diferentes, como por ejemplo la instalación de herramientas de programación como Eclipse, Octave, etc.
- **Unificar** la automatización de la creación de las imágenes ISO contenedoras del código (***build***) con el despliegue automático de las **instalaciones**, permitiendo

Figura 3.1: Topología de la red local Deimos con las nuevas conexiones para nuestro sistema.



así una interacción casi nula del integrador del sistema.

3.1.3. Tecnologías a tener en cuenta

Como se ha comentado anteriormente, para el desarrollo del sistema de integración continua se debe tener en cuenta ciertos aspectos ya existentes en el proyecto *MSF*:

1. Sistema operativo.
2. Creación de las imágenes ISO.
3. Estructura de la red.
4. Variedad de servidores.
5. Instalación semi-automatizada con las imágenes ISO.

El sistema operativo utilizado es *Red Hat Enterprise Linux 7*, por lo que utilizaremos para la instalación desatendida su herramienta propia llamada *Kickstart*. Esta herramienta se utiliza en el proyecto para automatizar parte de la instalación por DVD (automatiza el despliegue del código, pero es necesaria su existencia física y la interacción de un integrador para su instalación), por lo que en este caso solo se modificarán ciertas configuraciones en estos ficheros *Kickstart* (servidor y estación de trabajo) para conseguir una automatización completa.

El objetivo entonces del sistema a desarrollar es automatizar este proceso, crear una herramienta centralizada que indique estas directivas a los servidores, y que sirva los ficheros contenidos en la imagen ISO a cada uno de ellos. Para ello la mejor solución es desarrollar un sistema en red, en el cual estén conectadas todas las máquinas del proyecto (tanto servidores como estaciones de trabajo), con el servidor centralizado. Se utilizará la tecnología *PXE* para servir de los ficheros necesarios de la imagen ISO a cada máquina, y se creará una nueva estructura en red, añadiendo una nueva

conexión llamada *Red PXE* como se indica en la figura 3.1. A esta red se conectarán los servidores y estaciones de trabajo en las que se desplegará el código, los servidores *HPE* y *Sun* encargados de las máquinas virtuales *KVM* y el *Cluster VMWare*. En este *cluster* se alojará en una máquina virtual el sistema gestor (como se ha decidido anteriormente) encargado de las instalaciones desatendidas. Además también se añaden a esta red las cabinas de almacenamiento, pues en ellas se almacenan y centralizan las imágenes ISO creadas. De esta forma se permite compartir los ficheros necesarios mediante el protocolo *NFS* a todas las máquinas pertenecientes a esta red.

3.2. Definición de la máquina virtual

Partiendo de la base de que el *cluster* de máquinas virtuales *VMWare* está correctamente configurado y funcionando, con una topología de red virtual idéntica a la red física externa 3.1, se definirá una nueva máquina virtual, conectada a la red DEIMOS (para poder acceder a ella y para que tenga acceso al exterior en el caso de que necesite recursos de la red Internet), y a la red *PXE* (para poder gestionar todas las máquinas conectadas a ella). Esta máquina virtual se definirá con un sistema operativo *CentOS 7* con una instalación básica sin interfaz gráfica. Se elige el sistema operativo *CentOS 7* debido a que utiliza las mismas tecnologías que el sistema operativo utilizado en el proyecto (*Red Hat Enterprise Linux 7*), y por tanto permitirá una mejor compatibilidad entre sistemas. Posteriormente se realizarán una serie de operaciones sobre este sistema operativo, asentando la base necesaria para al creación de las aplicaciones previamente mencionadas.

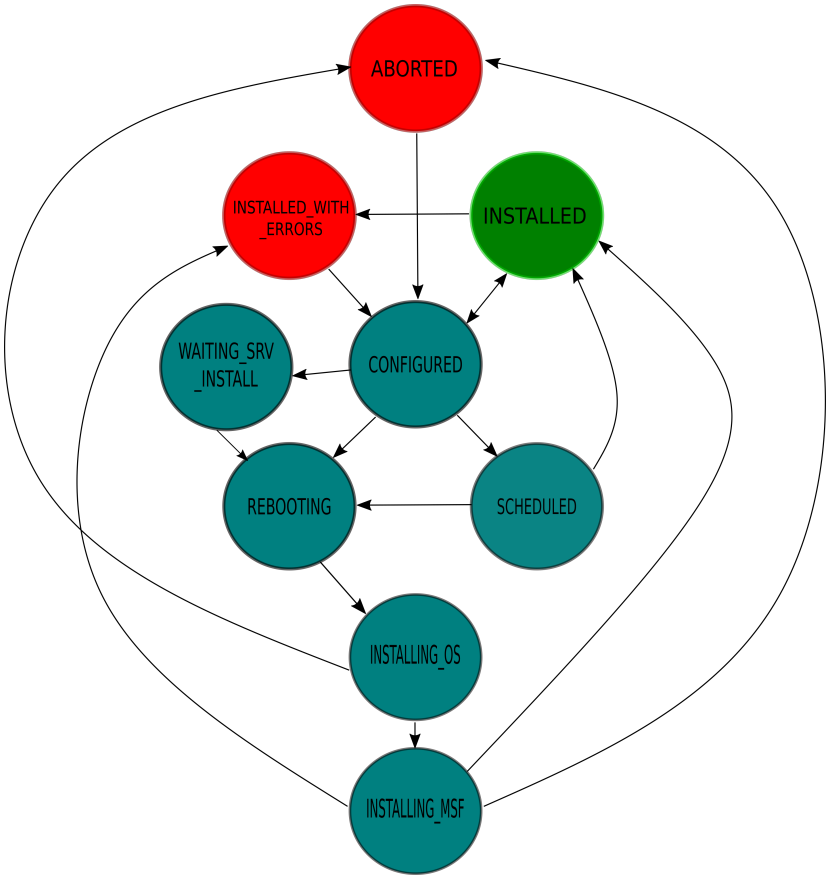
3.3. Desarrollo de la aplicación de gestión y automatización de instalaciones por red (*MSFAdt*)

Esta primera aplicación será la encargada de la instalación desatendida de las máquinas conectadas a la *red PXE*. Se encargará de gestionar el protocolo *PXE*, por lo que es necesario que modifique el servicio *DHCP*, *TFTP* y *HTTP* para el servicio de ficheros y configuración de red a las máquinas de la misma. También tiene que ser la encargada de ordenar el reinicio de las máquinas y su posterior arranque desde la tarjeta de red conectada a esta *red PXE*. A esta primera aplicación se la ha bautizado como *MSFAdt (MSF Automatic Deployment)*.

Como método de desarrollo de la aplicación, se ha optado por desarrollarla como una máquina de estados. Cada máquina disponible añadida en la aplicación podrá tener diferentes estados, y podrá pasar de uno a otro si y sólo si está permitido, como se indica en la figura 3.2. Estos estados serán:

- ***INSTALLED***: el sistema se ha desplegado correctamente en la máquina sin ningún error.
- ***INSTALLED_WITH_ERRORS***: el sistema ha sido desplegado, pero se han producido errores en el mismo. Aun así el sistema se ha desplegado y se puede trabajar con él. En este estado se deberá pasar los ficheros donde se ha almacenado

Figura 3.2: Máquina de estados de la aplicación *MSFAdt*.



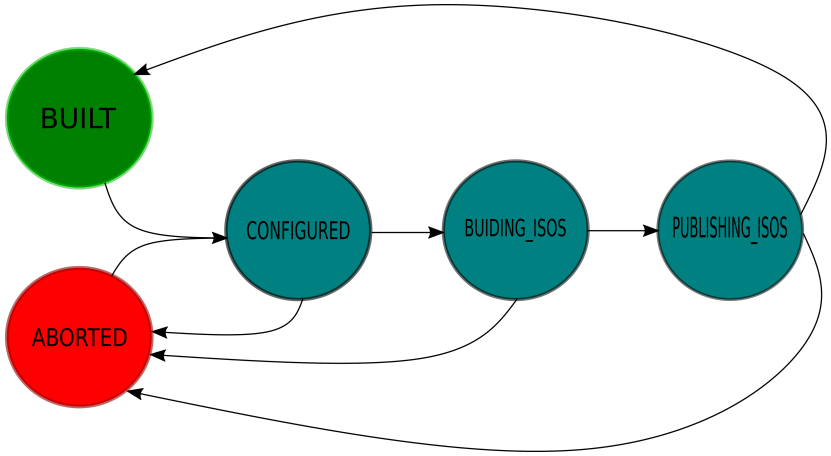
toda la información de la instalación (*logs*) a la máquina gestora, y descubrir qué errores han surgido.

- ***CONFIGURED***: la máquina esta lista para ser reiniciada e instalada. Para ello se ha configurado previamente el servicio *DHCP* para que le proporcione una configuración de red y un fichero de arranque y se han almacenado los ficheros necesarios en el servidor *TFTP* y *HTTP* para una correcta instalación. Además, se ha configurado la máquina para que arranque desde la tarjeta de red conectada a la *red PXE* al ser reiniciada.

- ***SCHEDULED***: la instalación de la máquina seleccionada ha sido programada para una fecha y hora concretas. Esto permite dejar la tarea del despliegue lista y preparada para cuando sea necesario, o para cuándo no sean horas de trabajo y no se interrumpa a los empleados.

- ***WAITING_SRV_INSTALL***: el orden correcto de la instalación de una plataforma es en primer lugar el servidor y tras esto las estaciones de trabajo. Por tanto, las estaciones de trabajo tendrán este estado intermedio entre *CONFIGURED* y *REBOOTING*, en el que se espera que el servidor se haya instalado para comenzar su instalación.

Figura 3.3: Máquina de estados de la aplicación *MSFAdt_build*.



- ***REBOOTING***: la máquina está siendo reiniciada y arrancando desde los ficheros proporcionados por *DHCP*, *TFTP* y *HTTP*.
- ***INSTALLING_OS***: la máquina está instalando el sistema operativo *RedHat Enterprise Linux*.
- ***INSTALLING_MSF***: la máquina ha terminado de instalar el sistema operativo y ahora está desplegando el software del proyecto *MSF*. Esto es gracias a que en el fichero *Kickstart* proporcionado se han indicado que se ejecuten una serie de *scripts* tras su instalación.
- ***ABORTED***: durante el despliegue ha surgido algún problema y no se ha podido finalizar correctamente. Difiere del estado *INSTALLED_WITH_ERRORS* en que en este caso el sistema no ha sido desplegado y no esta operativo para su uso.

Además, el lenguaje de programación que se utilizará para su desarrollo será **Perl**, debido al gran número de módulos disponibles en la comunidad de gestión de sistemas. Un aspecto a tener en cuenta en el desarrollo de esta aplicación es el cómo sabemos en qué estado se encuentra cada máquina. Para ello se va a desarrollar la aplicación como un servicio web. Esto quiere decir que las máquinas se podrán comunicar con ella mediante peticiones *HTTP* (en concreto peticiones *POST*). Es decir, cuando la máquina empiece a instalar el sistema operativo, se indicará en el *Kickstart* que se envíe una petición *HTTP* a la aplicación; por ejemplo `http://<ip>/estoy-instalando-el-so:<maquina>`, de esta manera en la máquina de estados correspondiente a la máquina pasada como parámetro se modificará (si se permite la transición del estado), y se tendrá un seguimiento de su estado actual.

3.4. Desarrollo de la aplicación de gestión de builds (*MSFAdt_build*)

Esta aplicación, llamada *MSFAdt_build*, será la encargada de automatizar la creación de nuevas imágenes ISO con el código del proyecto (lo que se ha bautizado como

build) desde la máquina gestora la cual está encargada de instalar las máquinas, del servicio *DHCP*, etc.

Su desarrollo será idéntico al anterior, como una máquina de estados, desarrollada en *Perl*. Sin embargo, esta tendrá una transición entre estados diferente mostrada a continuación:

- ***BUILT***: es uno de los estados base. Indica que la creación de las imágenes ISO se ha completado satisfactoriamente, y se han publicado en el servidor de almacenamiento. Desde este estado se pueden lanzar nuevos *builds*, indicando las ramas o etiquetas de los proyectos *Git* de los cuales se debe de coger el código del proyecto. De esta forma se pueden hacer *builds* del código disponible en cualquier momento del tiempo, seleccionando una etiqueta específica o una rama del proyecto (como la rama de desarrollo, la rama *master*, etc.).

- ***ABORTED***: este es el otro estado base desde el cual se puede lanzar de nuevo un *build*. Este estado indica que ha habido algún problema, ya sea configurando las ramas, creando las imágenes o publicándolas en el servidor de almacenamiento.

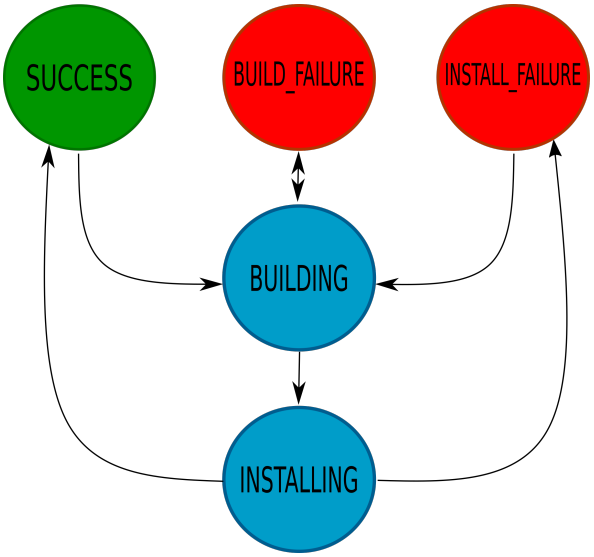
- ***CONFIGURED***: en este estado se configuran las ramas o etiquetas de las cuales se quiere realizar la imagen ISO. El *build* no solo recoge el código de un solo repositorio, sino que existen varios repositorios para cada funcionalidad del proyecto, como por ejemplo, el proyecto `csf.git` contiene la mayoría de código Java encargado de las funcionalidades del servidor y la interfaz gráfica de las estaciones de trabajo, el proyecto `msf.git` contiene directivas para la instalación, etc.

- ***BUILDING_ISOS***: en este estado se lanza la creación de las imágenes ISO con las ramas o etiquetas seleccionadas en el estado *CONFIGURED*.

- ***PUBLISHING_ISOS***: en este estado se transfieren las imágenes ISO creadas durante el estado *BUILDING_ISOS* al servidor de almacenamiento. Desde ahí serán accesibles desde todos los servidores y estaciones de trabajo disponibles, para su uso en la instalación o como respositorio de paquetes. Además, se publican con una estructura específica, organizados por día de creación, tipo del *build*, versión del mismo, etc. De esta manera, conseguimos un pseudosistema de control de versiones, dónde se almacenan todas las imágenes ISO de cada versión.

Al igual que en la anterior aplicación, esta también se diseñará como un servicio web para mantener una homogeneidad en el código. Actualmente esta aplicación web únicamente recibe peticiones desde el propio servidor *localhost*, y las transiciones entre estados se realizan dependiendo del resultado de las operaciones hechas internamente, por lo que no sería necesario estructurarlo como un servicio web. Sin embargo, por mantener una homogeneidad en el código, y tener la posibilidad de implementar más funcionalidades o peticiones en un futuro, se ha decidido esta implementación.

Figura 3.4: Máquina de estados de la aplicación *MSFAdt_CI*.



3.5. Aplicación de integración continua

Teniendo en cuenta las aplicaciones ya existentes mencionadas en la sección 2.1, se realizará un estudio mas intensivo respecto a las tecnologías *Jenkins*, *RunDeck* y *Bamboo*. Adicionalmente, también se propondrá la realización de esta aplicación encargada de la integración continua (y que gestionará las anteriores aplicaciones).

Si se decide utilizar una de estas herramientas externas, las aplicaciones *MSFAdt* y *MSFAdt_build* son también necesarias, pues, como se ha comentado al comienzo de este trabajo de fin de grado, son las encargadas de abstraer y simplificar las tareas pertinentes para la integración continua en una simple llamada o ejecución de un *script*.

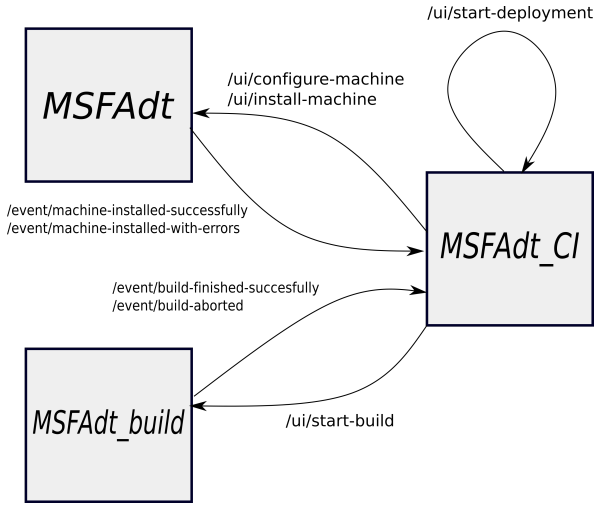
3.5.1. Desarrollo propio: *MSFAdt_CI*

Esta aplicación será la encargada de realizar las pertinentes llamadas y de gestionar los pasos necesarios para tener una integración continua en el proyecto. Esta aplicación, al igual que *MSFAdt* y *MSFAdt_build*, estará estructurada como una máquina de estados, donde la transición entre estados se muestra en el cuadro 3.4, y la relación entre las aplicaciones la que se muestra en el cuadro 3.5.

Los estados por los que pasará esta aplicación son:

- ***SUCCESS***: la creación del disco de instalación y el despliegue del código ha transcurrido con éxito y no se ha producido ningún fallo.
- ***BUILDING***: se están construyendo las imágenes de disco del servidor y estación de trabajo con el código correspondiente del proyecto. En este estado se hace uso de la aplicación *MSFAdt_build*.

Figura 3.5: Relación entre aplicaciones *MSFAdt*, *MSFAdt_build* y *MSFAdt_CI*



- ***BUILD_FAILURE***: la creación de las imágenes de disco no se ha producido o ha fallado, por lo que se aborta el despliegue. Se debe informar en este caso de los motivos por los que ha fallado o se ha abortado esta creación.
- **INSTALLING**: se está instalando la plataforma dedicada para la integración continua. Esta plataforma se compone de servidor y estación de trabajo. En este estado se hace uso de la aplicación *MSFAdt*.
- **INSTALL_FAILURE**: durante la instalación del servidor o de la estación de trabajo se ha producido un error, o ha sido abortada su instalación. En este estado también se debe de informar de los errores producidos por el que se ha pasado a él.

3.5.2. Reutilización: Jenkins, RunDeck o Bamboo

Jenkins Como se ha comentado anteriormente, es un sistema de integración continua de código abierto que nos permite automatizar y programar despliegues de proyectos de manera completamente desatendida y realizar un seguimiento de los mismos. El único código extra que se debe de desarrollar con este sistema son los *scripts* encargados de configurar las instalaciones, el *build* y el propio *pipeline* de *Jenkins* que muestra los pasos a seguir. Como se puede observar a simple vista, es mucho más fácil y menos costoso que realizar todo esto desde cero mediante una aplicación propia.

Se puede ver además, en la estructura del *pipeline*, que se siguen los estados y pasos definidos que en la aplicación *MSFAdt_CI* previamente definida (fig. 3.4).

RunDeck Al igual que *Jenkins*, es un software de código abierto, el cual nos proporciona una solución de integración continua. Nos sirve de gestor y programador de tareas, y de recolector de datos y estadísticas de igual manera que lo hace *Jenkins*. Sin

Función	**Bamboo**	**Jenkins**
Compatibilidad con JIRA, BitBucket, Git ...	Por defecto	Mediante extensiones
Programación de tareas	Por defecto	Mediante extensiones
Desarrollo	Mediante *pipelines*	Usa *stages*, *jobs* y *tasks* (similar a un *pipeline* pero seccionado)
Generación de estadísticas	Incorporado	Incorporado
Documentación	Extensa	Concisa y de fácil comprensión
Precio	10 $ al año por 10 trabajos o 1100 $ al año por trabajos ilimitados	Gratuito y de código abierto

Tabla 3.1: Jenkins vs. Bamboo.

embargo, *RunDeck* dispone de una versión *Enterprise* (de pago), aparte de la versión gratuita. Esta versión de pago da muchas más opciones de las que da el gratuito, como por ejemplo estadísticas extendidas, creación de más tareas, etc. [Run19]:

1. *Clustering* y *high-availability*, cosa que no aporta nada al sistema de integración continua que se va a desarrollar, pues solo se utiliza una máquina virtual.

2. Visión y entorno de trabajo avanzado para empresas.

3. Extensiones exclusivas para subscriptores.

4. Reglas de políticas de acceso extra.

5. Seguimiento de las tareas desde el pasado. Esto es una característica que *Jenkins* ofrece gratuitamente, por lo que es un gran punto negativo el tener que pagar por ella.

6. Soporte para *Windows*, que tampoco interesa al sistema ya que se trabaja con *Red Hat Enterprise Linux 7* o *CentOS 7*.

Respecto a la forma de funcionamiento, es muy similar a *Jenkins*. Sigue el mismo patrón de funcionamiento y no ofrece ninguna ventaja a mayores, por lo que, al tener la desventaja de que exista un software de pago y que no se proporcionen las mismas extensiones y ventajas que en *Jenkins*, no se tendrá en cuenta.

Función	MSFAdt_CI	Jenkins + Bamboo	RunDeck
Flexibilidad:	Lo podemos diseñar a nuestro gusto e implementar con *MSFAdt_web*	Gran cantidad de utilidades disponibles, como correos automáticos de información, programación de builds...	Gran cantidad de utilidades disponibles pero solo en la versión de pago; en la versión gratuita es más limitado que Jenkins
Estadísticas:	Recogida de estadísticas, estados del despliegue, seguimiento muy complejo, que requerirá gran cantidad de recursos al proyecto	Recogida automática de estadísticas, hora de despliegue, estado del despliegue	Recogida automática de estadísticas, hora de despliegue, estado del despliegue solo en la versión de pago
Programación de tareas:	Necesidad de uso de *cron*	Automatización de despliegue integrado en el sistema	Automatización de despliegue integrado en el sistema
Disponibilidad:	Gratuito	Gratuito + licencia disponible de *Bamboo*	Gratuito y de pago
Trabajo:	Elevado	Bajo	Medio
Modificación de APIs existentes:	Si	No	No
Comunicación:	Correo + Skype	Correo + Skype	Correo + Skype
Despliegue:	Bastante complejo	Muy sencillo	Sencillo

Tabla 3.2: Conclusiones sobre el sistema de integración continua.

Bamboo Por último, el sistema *Bamboo*, que pertenece a *Atlassian* [Atl19], también es un caso de estudio. Este sistema, aunque sea de pago, no conlleva un problema, pues la empresa dispone de licencia ilimitada para este servicio, al ser la herramienta corporativa actual. En el caso de este sistema de integración continua, todas y cada una de las ventajas que proporciona *Bamboo* (véase la tabla 3.2)) frente a *Jenkins*, no se lleva a cabo en él. Es decir, como el despliegue del proyecto *MSF* es muy particular, se ha abstraído lo suficiente mediante las aplicaciones previamente desarrolladas para que la integración continua del proyecto se desarrolle de manera más sencilla con la ejecución ordenada de varios *scripts* y llamadas *HTTP*. Por tanto, el gran número de funcionalidades que proporciona *Bamboo* no será de utilidad a la hora del despliegue del sistema de integración continua.

Aunque a la automatización de pruebas, en *Bamboo* vienen incorporadas por defecto, en *Jenkins* pueden ser conseguidas muy fácilmente mediante la instalación de varios *plugins*. Estas extensiones proporcionan funcionalidades extra que pueden ser añadidas

a nuestro sistema, y que tienen un sinfín de utilidades, al ser desarrolladas por los usuarios[1]. Por tanto, no supone un problema el añadir estas extensiones manualmente.

En cuanto al método de uso de *Bamboo*, se puede destacar que es similar al de *Jenkins*. Es más, permite importar proyectos ya existentes de *Jenkins* fácilmente. Esto da opción a que, en caso de que se seleccione el sistema *Bamboo*, desarrollar el sistema de integración continua completamente de forma gratuita en *Jenkins*, realizar las pruebas pertinentes y posteriormente migrar a *Bamboo* con la licencia corporativa proporcionada.

Discusión En la tabla 3.2 se muestra una comparativa entre los sistemas *MSFAdt_CI*, *Jenkins + Bamboo* y *RunDeck*. Se puede concluir que la solución más factible es la utilización del sistema ***Jenkins + Bamboo***, debido a su facilidad de uso y despliegue y a que da muchas más funcionalidades de las que podríamos conseguir con el mismo nivel de trabajo con una aplicación personal. Como se ha comentado anteriormente, si se elige este sistema, se requerirá una migración a *Bamboo* desde *Jenkins* en el momento en el que este esté diseñado, pues *Bamboo* es la aplicación corporativa exigida para el desarrollo de este tipo de aplicaciones, y es la utilizada en los servidores de la empresa. Por ello, se llevará a cabo y se comentará este desarrollo en *Jenkins* y finalmente se mostrarán los pasos para su importación en el sistema *Bamboo* de la empresa (que se encuentra en otro servidor ajeno al definido en este trabajo de fin de grado, por lo que para ejecutar los diferentes *stages* se hará uso del servicio *SSH*) en el capítulo 4. Adicionalmente, en el servidor *Bamboo* corporativo se pueden configurar las notificaciones y el seguimiento del código añadido respecto a los usuarios ya existentes, es decir, se hace uso de los usuarios ya definidos en el ámbito corporativo (ventaja muy significativa que se ha de tener en cuenta).

3.6. Desarrollo de la interfaz web (*MSFAdt_web*)

Tras haber desarrollado *MSFAdt*, *MSFAdt_build*, y haber unificado su ejecución en una sola aplicación, ahora es necesaria una interfaz web para el lanzado de instalaciones, configuración de máquinas, etc. En definitiva, una interfaz amigable para el usuario, evitando la modificación de ficheros de configuración manualmente.

Esta interfaz web deberá permitir seleccionar qué máquinas o grupo de máquinas se quieren instalar, con qué versión del *software*, si se trata de una instalación de desarrollo o de producción, el estado actual de todas las máquinas, los errores en caso de que los haya, programación de instalaciones, posibilidad de añadir o borrar nuevas plataformas o máquinas individuales, lanzar nuevos *builds*, etc. En definitiva, una interfaz dónde se pueda ver si algo ha ido mal con un solo golpe de vista, y se pueda configurar e instalar las plataformas de forma más fácil e intuitiva haciendo uso de las aplicaciones *MSFAdt*

[1] *Jenkins*, como se ha comentado anteriormente, es un sistema de código abierto, en el cual los usuarios pueden desarrollar nuevas funcionalidades del sistema en forma de extensiones o modificar su propio código fuente para adecuarlo a su propio caso.

y *MSFAdt_build*.

Debido a que las anteriores aplicaciones se desarrollaron con el lenguaje de programación *Perl*, esta interfaz web se desarrollará también con este mismo lenguaje. Para ello existe una herramienta llamada **Mojolicious** [Seb19], que se encarga de la gestión de este servicio web y de su interfaz. Esta herramienta utiliza plantillas (*templates*), realizadas en el lenguaje *Perl*, el cual lo traduce al lenguaje *HTML* para su posterior muestra en el navegador pertinente. Además, las rutas se gestionan desde la misma herramienta, por lo que resulta muy intuitiva la creación de un buen árbol de rutas HTML.

Capítulo 4

Diseño

En este capitulo se va a desarrollar en profundidad las tres aplicaciones comentadas anteriormente y el despliegue del sistema *Jenkins + Bamboo* como gestor de las anteriores aplicaciones, que nos permitirá desplegar el sistema de integración continua. Estas aplicaciones, salvando el sistema *Jenkins + Bamboo*, han sido desarrolladas con el lenguaje de programación *Perl*, como máquinas de estados y como aplicaciones de servicios web, como se ha explicado en el apartado anterior, y en una máquina virtual independiente con el sistema operativo *CentOS 7*. Conforme se han ido desarrollando las aplicaciones, se ha necesitado una configuración específica en el sistema operativo *CentOS 7*. Se comenzará entonces con la explicación de estas directivas, recogidas paso a paso en el manual *MSFAdt Installation guide* (apéndice A).

4.1. Configuración del sistema operativo *CentOS 7*

Fijados el lenguaje de programación, el tipo de aplicación (servicio web), la estructura de la misma (como máquina de estados) y los servicios necesarios por *PXE*, es necesario adecuar el sistema operativo para que permita dichas operaciones.

Para el desarrollo de las aplicaciones se debe trabajar y también ejecutar las tareas, como superusuario, ya que la aplicación va a tener que modificar varios ficheros de configuración sensibles, entre los que se incluyen los de los servicios *DHCP* (`/etc/dhcp/dhcpd.conf`), *TFTP* (`/var/lib/tftpboot`) y *HTTP* (`/etc/httpd/httpd-.conf`). Este último servicio solo es necesario configurarlo una vez, por lo que las aplicaciones no modificarán este fichero.

Ya teniendo en cuenta que es necesario trabajar como usuario *root*, se procede a instalar los paquetes necesarios en el sistema, como pueden ser:

- `git`: es necesario ya que el código se almacena en un repositorio existente del proyecto, llamado *msf.git*. De esta forma se consigue un seguimiento del desarrollo por versiones, y la posibilidad de un desarrollo paralelo gracias a las ramas de este sistema.
- Repositorio `epel`: repositorio de paquetes adicionales, de dónde se sacarán los paquetes necesarios de *Perl*.

- `Perl`: lenguaje elegido para el desarrollo de las aplicaciones.
- Módulos `Perl` necesarios para el desarrollo de las aplicaciones: por ejemplo, se necesitarán módulos para la gestión del servicio web, como `LWP`. Para la instalación de módulos también es necesaria la instalación de una herramienta llamada `cpanm`. Esta herramienta se conecta a los repositorios *CPAN*, en los cuales se encuentran almacenados estos módulos, y los descarga en la máquina local.
- Paquetes necesarios para la configuración de servidores `httpd`, `tftp` y `dhcp`. También se necesita el paquete `xinetd`, para tener una mejor gestión del servidor `tftp`, y apagar y encender el servicio sólo cuando sea necesario.

Ya con estos paquetes instalados, se procede a la configuración de los servidores para el arranque por *PXE*. El servicio `dhcpd` será gestionado por la aplicación encargada de la instalación de las máquinas.

El servicio `httpd` necesita ser configurado antes. Para ello se añade un nuevo fichero al directorio `/etc/httpd/conf.d/` donde se indica el árbol de directorios que se comparte por *HTTP*. Este fichero se llamará `msf-releases.conf` y tendrá la siguiente sintaxis:

```
1 Alias "/releases" "/vm/releases"
2  <Directory "/vm/releases">
3   Require all granted
4   Options Indexes
5  </Directory>
```

De esta forma se indica que al directorio `/vm/releases`, que es dónde estarán montadas las imágenes ISO generadas del proyecto, se podrá acceder por *HTTP* bajo el alias de `<ip_servidor>/releases`. De esta forma, las máquinas a instalar podrán descargar los ficheros necesarios con un `wget`.

El servicio `tftp` también debe ser configurado con anterioridad, no respecto a ficheros de configuración, sino respecto a los ficheros necesarios para el arranque por *PXE*. Para ello se creará un directorio en `/var/lib/tftpboot` llamado `boot.save` (fig. 4.1). En este directorio se introducen los ficheros necesarios para el arranque por red para sistemas con *BIOS Legacy* y sistemas *EFI*. Estos ficheros serán `initrd.img`, `pxelinux.0`, `vesamenu.c32`, `vmlinuz` y el fichero de configuración `pxelinux.cfg/default` para el arranque por *BIOS Legacy*, mientras que para el arranque por *EFI* se crea un directorio aparte dentro del mismo llamado `efi` y en el se introduce `grupx64.efi`, `initrd.img`, `shim.efi`, `vmlinuz` y por último el fichero de configuración `grub.cfg`. Estos ficheros `grub.cfg` y `pxelinux.cfg/default` serán lo que se vean modificados por la aplicación gestora de instalaciones.

Esta aplicación será la encargada de crear, partiendo de la base de este directorio de muestra `boot.save`, los directorios específicos para cada máquina.(Ej.: `msf-srv41`, `msf-ws411`, etc.), cada uno con su configuración específica de los ficheros `pxelinux.cfg/default` y `grub.cfg`.

Figura 4.1: Estructura del directorio *boot.save*.

```
[root@pxe-platform boot.save]# ll -R
.:
total 60180
drwxr-xr-x. 2 root root       90 mar 15 07:46 efi
-rw-r--r--. 1 root root 54799220 feb 15 07:55 initrd.img
-rw-r--r--. 1 root root    26826 feb 15 07:55 pxelinux.0
drwxr-xr-x. 2 root root       21 mar 15 07:46 pxelinux.cfg
-rw-r--r--. 1 root root   153104 feb 15 07:55 vesamenu.c32
-rwxr-xr-x. 1 root root  6635920 feb 15 07:55 vmlinuz

./efi:
total 62312
-rw-r--r--. 1 root root      700 mar 15 07:46 grub.cfg
-rwxr-xr-x. 1 root root  1091024 mar 15 07:46 grubx64.efi
-rw-r--r--. 1 root root 54799220 mar 15 07:46 initrd.img
-rw-r--r--. 1 root root  1301312 mar 15 07:46 shim.efi
-rwxr-xr-x. 1 root root  6635920 mar 15 07:46 vmlinuz

./pxelinux.cfg:
total 4
-rw-r--r--. 1 root root 488 mar 15 07:46 default
```

Para el uso de estos servicios de red, es necesario abrir los puertos que utilizan en el *firewall* del servidor. Además, para nuestras aplicaciones, que como hemos dicho anteriormente serán servicios web, también se debe abrir un puerto por cada aplicación. En concreto se eligen los puertos 3001, 3002 y 3003 para estas aplicaciones.

4.2. *MSFAdt*

En primer lugar se desarrolla la aplicación *MSFAdt* siguiendo las especificaciones que se han tomado en el capítulo 3. El directorio dónde se desarrollará la aplicación dentro de la máquina virtual *CentOS 7* es `/root/DEIMOS-MSF/msf/scripts/automatic-deployment`, siendo `msf` el repositorio *git* también mencionado anteriormente.

En primer lugar se asientan las bases del servicio web. Para ello se utiliza el módulo `Dancer2` de Perl [Dan19]. Con ello instalado, tal y como se especifica en el manual de instalación (apéndice A), se procede a crear un entorno de trabajo para esta aplicación web *Dancer2* crea una pequeña aplicación funcional de prueba, que puede ser ejecutada y visualizada desde un navegador web. Para ello se ejecuta la orden:

```
[root@pxe-platform ~]# dancer2 -a MSFAdt::API -d automatic-deployment -p /root/DEIMOS-
    MSF/msf/scripts/
```

Esto creará un entorno de trabajo de la aplicación con la estructura que se indica en

Figura 4.2: Entorno de trabajo de la aplicación web creada por *Dancer2*.

```
[root@pxe-platform automatic-deployment]# ll
total 20
drwxr-xr-x. 2 root root   22 Mar 18 05:03 bin
-rw-r--r--. 1 root root 1433 Mar 18 05:03 config.yml
-rw-r--r--. 1 root root  300 Mar 18 05:03 cpanfile
drwxr-xr-x. 2 root root   51 Mar 18 05:03 environments
drwxr-xr-x. 3 root root   20 Mar 18 05:03 lib
-rw-r--r--. 1 root root  802 Mar 18 05:03 Makefile.PL
-rw-r--r--. 1 root root  445 Mar 18 05:03 MANIFEST
-rw-r--r--. 1 root root  178 Mar 18 05:03 MANIFEST.SKIP
drwxr-xr-x. 5 root root  142 Mar 18 05:03 public
drwxr-xr-x. 2 root root   49 Mar 18 05:03 t
drwxr-xr-x. 3 root root   37 Mar 18 05:03 views
```

la figura 4.2. Sin embargo, solo se utilizarán ciertas cosas de este entorno de trabajo generado. Se adecuará a la estructura indicada en la figura 4.3, para utilizar `Starman` como servidor web. El contenido de los diferentes directorios del entorno de trabajo es:

- Directorio `apache`: en este directorio se almacena la configuración del servidor *Apache*, tal y como se ha modificado en la figura 4.1. De esta forma, si en un futuro se cambia de servidor o es necesaria la reinstalación de la máquina, tan solo es necesario copiar el fichero de configuración almacenado aquí a a la configuración del servidor *Apache*.

```
[root@pxe-platform apache]# cat msf-releases.conf
Alias "/releases" "/vm/releases"
<Directory "/vm/releases">
Require all granted
Options Indexes
</Directory>
```

- Directorio `api`: en este directorio se almacenan los ficheros necesarios para arrancar el servicio web, gestionado por `Starman` y el fichero `.psgi` generado por `Dancer2`.
- Directorio `api-test`: en este directorio se encuentran los ficheros *Perl* encargados de realizar peticiones *HTTP* a la aplicación web. De esta forma se puede simular el funcionamiento de la aplicación fácilmente, sin tener que reinstalar una máquina y realizar peticiones desde otras máquinas para comprobar su correcto funcionamiento (esto se realizará más tarde, cuando se tenga una aplicación medianamente funcional y adecuada a las especificaciones).
- Directorio `bin`: aquí se almacenan los binarios necesarios para la aplicación web. Por el momento no ha sido necesario ningún fichero binario para esta aplicación, por lo que este directorio está vacío.
- Directorio `boot.save`: tal y como se ha comentado en la sección 4.1 se necesita este directorio dónde se almacenan todos los ficheros necesarios para el arranque

Figura 4.3: Entorno de trabajo modificado para *MSFAdt*.

```
[root@pxe-platform automatic-deployment]# ll
total 18
drwxr-xr-x.  2 root root   31 Feb 15 06:42 apache
drwxr-xr-x.  2 root root   62 Mar 18 05:02 api
drwxr-xr-x.  2 root root 4096 Feb 21 06:48 api-test
drwxr-xr-x.  2 root root   53 Feb 15 06:42 bin
drwxr-xr-x.  4 root root   88 Feb 26 06:18 boot.save
drwxr-xr-x.  2 root root   41 Feb 15 06:42 code-templates
drwxr-xr-x.  2 root root   66 Feb 26 06:18 config-templates
drwxr-xr-x.  2 root root   58 Feb 15 06:42 doc
drwxr-xr-x.  2 root root  160 Mar 13 09:02 etc
drwxr-xr-x.  3 root root   37 Mar 18 04:54 lib
drwxr-xr-x.  2 root root  103 Mar  7 04:29 log
drwxr-xr-x. 44 root root 4096 Mar  1 03:33 machine-logs
-rw-r--r--.  1 root root  366 Feb 15 06:42 README
drwxr-xr-x.  2 root root 4096 Mar 11 06:25 run
drwxr-xr-x.  2 root root 4096 Mar 13 09:03 tools
```

por red de las máquinas. Además de añadirlo al servicio *TFTP*, se almacenará en este directorio por el mismo motivo que el directorio `apache`, para que en un futuro no se tenga que volver a definir manualmente, sino copiarlo directamente donde sea necesario.

- Directorio `code-templates`: aquí se almacenan las plantillas de las cuales partir para crear nuevos *scripts* o nuevos paquetes Perl `.pm` de la aplicación web.

- Directorio `config-templates`: al igual que el anterior directorio, este cumple la misma función pero para ficheros de configuración, como puede ser el fichero de configuración `/etc/dhcp/dhcpd.conf`, donde se especifica la configuración del servidor `dhcp`. También aquí se encuentran las plantillas para los ficheros `pxelinux.cfg/default` y `grub.cfg` que se mencionan en la sección 4.1, los cuales varían para cada máquina. Usando estas plantillas, se modifica la configuración de estos ficheros con la aplicación web.

- Directorio `doc`: aquí se almacenan los manuales de la aplicación, tanto manuales de instalación como de funcionamiento y de usuario. Estos manuales se adjuntan en el apéndice **??**.

- Directorio `etc`: aquí se almacenan los ficheros de configuración necesarios para la aplicación web. Entre ellos se encuentra el fichero de configuración global, el que almacena las variables necesarias para correr la aplicación, el fichero de configuración de logs (`Log4perl`), el fichero de configuración de las máquinas y el correspondiente a las versiones (donde se indica el numero de versión de la imagen ISO, el nombre de la misma, etc.). El lenguaje utilizado para estos ficheros de configuración es *YAML*, debido a su fácil sintaxis y facilidad de uso junto con

Perl, pues se puede volcar todo el contenido de un fichero *YAML* en un objeto, y de esta forma tener toda esta configuración en una única variable.

- Directorio `lib`: en este directorio se almacena el "corazón" de esta aplicación web. Se encuentran todos los paquetes *Perl* creados, con las funciones que realizará la aplicación web con cada petición *HTTP* recibida. Este directorio se estudiará detenidamente más adelante.

- Directorio `log`: almacén de logs de la aplicación web. Aquí se encuentra el log de errores de la aplicación web, el log de acceso (donde se especifica qué peticiones *HTTP* se ha recibido, desde qué máquina, etc.) y los logs de seguimiento. Estos ficheros han sido cruciales a la hora del desarrollo de la aplicación, pues en ellos se recoge toda la información necesaria para el trazado de los errores, de las funciones ejecutadas, etc.

- Directorio `machine-logs`: aquí se almacenan los ficheros logs generados en la instalación de la máquina, y recibidos mediante una petición *POST* de la aplicación web. Se estructuran en directorios separados por máquina y el nombre contiene la fecha y hora de la instalación, la *release* utilizada, el modo de instalación, etc.

```
[root@pxe-platform automatic-deployment]# ll machine-logs/
total 76
drwxr-xr-x. 2 root root 4096 Mar  8 07:41 msf-srv41
drwxr-xr-x. 2 root root 4096 Mar 11 21:44 msf-srv81
(...)
```

```
[root@pxe-platform automatic-deployment]# ll machine-logs/msf-srv41/
total 15668
(...)
-rw-r--r--. 1 root root  386419 Mar  7 07:56 20190307.125610-V3A.01-DR1
    -20190307.1213-production-full.log
-rw-r--r--. 1 root root 1313122 Mar  8 07:41 20190308.124140-V3A.01-DR1
    -20190308.1142-development-full.log
(...)
```

  Estos ficheros permitirán trazar errores causados durante la instalación del servidor o estación de trabajo. Estos errores se refieren a los surgidos durante el despliegue del código, pues este despliegue no es bloqueante, sino que si se trata de errores menores, el despliegue puede seguir adelante y notificar de estos errores en este fichero log.

- `README`: pequeño fichero en el que se especifica el directorio en el que se está, y la utilidad de la aplicación:

```
[root@pxe-platform automatic-deployment]# cat README
MSFADT: MSF Automatic Deployment Tool

This is a system for automatic deployment of MSF ISO images to bare metal
hardware.  It is intended to automate reinstallation of MSF servers and
workstations, and ultimately to allow for continuous deployment and testing
of MSF software on an automated, periodic basis.

See docs in doc/ directory for implementation details.
```

- Directorio `run`: aquí se almacena el estado actual de cada máquina en forma de ficheros *YAML*. Como se ha comentado anteriormente, la aplicación web se

Figura 4.4: Estructura del directorio *api*.

```
[root@pxe-platform api]# ll
total 12
-rw-r--r--. 1 root root 154 Feb 15 06:42 api.psgi
-rwxr-xr-x. 1 root root 651 Feb 15 06:42 launch.sh
-rwxr-xr-x. 1 root root 191 Feb 15 06:42 stop_service.sh
```

estructura como una máquina de estados, donde cada máquina definida en el sistema (concretamente en `/etc/machines.yaml`) se encuentra en un estado. Este estado se almacena en ficheros de configuración dentro de este directorio para cada máquina.

- Directorio `tools`: scripts externos a la aplicación web, que ésta utiliza en casos concretos. Estos scripts se explicarán en mayor detalle también en las secciones siguientes.

4.2.1. Directorio *api*

En la figura 4.4 se indican los ficheros existentes en este directorio. La funcionalidad de cada uno de ellos es:

1. `api.psgi`: se trata del fichero *PSGI*(*Perl Web Server Gateway Interface*) generado por `Dancer2`. Este fichero se encarga de gestionar las peticiones *HTTP* recibidas al servidor web, y realizar una función u otra dependiendo de qué petición se haya recibido [Met19a]. En el caso de la aplicación *MSFAdt*, este fichero se utilizará como intermediario, es decir, no se definirán aquí las rutas a las que realizar las peticiones, si no que se definirá que cuando se reciba una petición de la ruta global /, que se ejecuten diversas funciones definidas en los paquetes creados en `lib`. De esta forma, las rutas se definen en los paquetes contenidos en `lib`, más concretamente en `MSFAdt::API`[1] y no en este mismo fichero. Su sintaxis, teniendo en cuenta estos detalles, es la siguiente:

```
1  #!/bin/env perl
2  use Plack::Builder;
3  use FindBin;
4  use lib "$FindBin::Bin/../lib";
5
6  use MSFAdt::API;
7
8  builder {
9  mount '/'    => MSFAdt::API->to_app;
10 };
```

[1]El nombre de los paquetes en *Perl* indica una ruta. Es decir, bajo el directorio *lib* se encuentra el fichero *MSFAdt/API.pm*, que será el fichero que se ejecutará cuando se reciba cualquier tipo de petición [Sim05].

2. `launch.sh`: este fichero escrito en `bash` es el encargado de arrancar la aplicación web. Se utiliza para ello `Starman`. Esta herramienta `Starman` es capaz de hacer correr aplicaciones *PSGI*, por lo que se pasará por parámetro el fichero `api.psgi`, para poder así ejecutar las funciones y rutas necesarias.

```
#!/bin/bash

(...)
starman --listen "$ADDRESS:$PORT" --workers $WORKERS \
--access-log "$ACCESSLOG" --error-log "$ERRORLOG" \
--daemonize --pid "$PIDFILE" \
./api.psgi

echo "MSFAdt API launched - Pid: " $( cat "$PIDFILE" )
```

   Como se puede ver, las variables que se le pasan a la herramienta `Starman` se recogen de los ficheros de configuración anteriormente mencionados. Estas variables son de la dirección IP y el puerto en el que se ejecutará el servicio web, el numero de trabajadores, los logs de acceso y errores (anteriormente mencionados, y que se almacenan en `log`) y el fichero dónde se almacenará el ID del proceso. Este ID es necesario para luego poder parar el servicio web, como se hace en el fichero `stop_service.sh`. Además, se indica que el proceso se trate como un demonio, es decir, que no sea bloqueante en el sistema y se ejecute en segundo plano. Por último se indica la aplicación *PSGI* a ejecutar [Met19b].

3. `stop_service.sh`: fichero encargado de parar el servicio web. Haciendo uso del ID del proceso previamente guardado en un fichero, este script recoge dicho ID y mata el proceso con `kill`. De esta forma se para el servicio web:

```
#!/bin/bash

(...)
PIDFILE="$HOMEDIR/../run/api.pid"
SERVICE=‘cat $PIDFILE‘

kill $SERVICE
(...)
```

4.2.2. Directorio *etc*

Los ficheros que aquí se encuentran, especificados en la figura 4.5, son los ficheros de configuración de la aplicación web:

1. `api.yaml`: recoge la configuración para el arranque del servicio. En concreto, contiene la dirección IP y el puerto en el que se arrancará la aplicación web, y el número de trabajadores. Esta configuración se carga en el script `api/launch.sh` como se ha indicado anteriormente, para el arranque del servicio.

2. `main.yaml`: contiene la configuración global de la aplicación web, por ejemplo rutas específicas de scripts para su ejecución, ruta donde se montan las imágenes

Figura 4.5: Estructura del directorio *etc*.

```
[root@pxe-platform etc]# ll
total 60
-rw-r--r--. 1 root root    79 Feb 15 08:26 api.yaml
-rw-r--r--. 1 root root   398 Feb 15 06:42 log4perl.conf
-rw-r--r--. 1 root root 27879 Mar 11 15:09 machines.yaml
-rw-r--r--. 1 root root 10889 Feb 15 06:42 machines.yaml.
   example
-rw-r--r--. 1 root root  4075 Mar  6 08:33 main.yaml
-rw-r--r--. 1 root root   213 Mar 13 09:52 releases.yaml
-rw-r--r--. 1 root root  1019 Feb 15 06:42 releases.yaml.
   example
```

ISO, ruta donde se almacenan estas mismas, tiempos de espera entre instalaciones, etc. Sin embargo, lo más importante de este fichero de configuración es que es donde se especifican los estados y sus transiciones:

```
(...)
allowed_states:
 CONFIGURED:
  description: 'System configured and ready for installing the machine'
  allowed_from:
   - INSTALLED
   - INSTALLED_WITH_ERRORS
   - ABORTED
   - SCHEDULED
 SCHEDULED:
  description: 'Machine is scheduled for its installation'
  allowed_from:
   - CONFIGURED
 REBOOTING:
  description: 'Machine is rebooting for PXE installation'
  allowed_from:
   - WAITING_SRV_INSTALL
   - CONFIGURED
   - SCHEDULED
 WAITING_SRV_INSTALL:
  description: 'Waiting server install'
  allowed_from:
   - SCHEDULED
   - CONFIGURED
 INSTALLING_OS:
  description: 'Machine is installing the Operating System'
  allowed_from:
   - REBOOTING
 INSTALLING_MSF:
  description: 'Machine is installing the MSF Software'
  allowed_from:
   - INSTALLING_OS
 INSTALLED:
  description: 'Machine has been installed'
  allowed_from:
   - INSTALLING_MSF
   - INSTALLED_WITH_ERRORS
 INSTALLED_WITH_ERRORS:
  description: 'Machine has been installed with errors'
  allowed_from:
   - INSTALLED
 ABORTED:
  description: 'Installation has been aborted on the machine'
```

```
  allowed_from:
   - WAITING_SRV_INSTALL
   - REBOOTING
   - INSTALLING_OS
   - INSTALLING_MSF
(...)
```

De esta forma se indica, con una sintaxis *YAML*, el diagrama de estados definido en la figura 3.2.

Además, también se especifican las variables de configuración para el servidor *DHCP*, como pueden ser el rango de direcciones IP que se dará a las máquinas, la dirección del *gateway*, la red, la máscara de subred, etc.

3. `machines.yaml` y `machies.yaml.example`: este fichero guarda toda la configuración referida a las máquinas de la aplicación. Va acompañado de un fichero de ejemplo, en el cual se encuentran definidas algunas máquinas de prueba, para que a partir de esa sintaxis, se genere un fichero de configuración con las máquinas requeridas.

   Para cada máquina se asignan diversas configuraciones, referidas a la red *PXE*, red externa, red de mantenimiento, contraseñas, usuarios, versión, modo y tipo con el que se instalará, etc. Se adjunta un pequeño ejemplo de definición de una máquina:

```
---
 machines:
  msf-srv41:
   installation:
    mode: full
    release: V3A.01
    type: development
   management:
    ipaddress: 172.23.15.30
    password: ****
    type: vmware
    user: administrator@msf-vsphere.local
    vmname: MSF-SRV41
   network:
    external:
     gateway: 172.23.15.1
     ipaddress: 172.23.15.41
     netmask: 255.255.255.0
    pxe:
     ipaddress: 10.40.1.41
     macaddress: 00:50:56:86:c7:2a
     netmask: 255.255.255.0
   number_of_workstations: '1'
   platform_number: '41'
   type: server
   ws:
    - msf-ws411

(...)
```

Como se puede ver, en el apartado de `installation` se indica la versión a instalar, el modo y el tipo de la instalación. En *management* se especifica el tipo

Figura 4.6: Estructura del directorio *run*.

```
[root@pxe-platform run]# ll
total 260
-rw-r--r--. 1 root root   6 Mar  7 03:29 api.pid
-rw-r--r--. 1 root root  96 Feb 15 08:40 msf-srv1-state.yaml
-rw-r--r--. 1 root root  96 Feb 15 08:40 msf-srv2-state.yaml
(...)
-rw-r--r--. 1 root root  74 Feb 15 06:42 README
-rw-r--r--. 1 root root  97 Feb 15 06:42 template-state.yaml
```

de máquina que es (vmware, kvm, hp...) y cuál es la dirección IP, usuario y contraseña del servidor que aloja esa máquina. En el caso de las máquinas virtuales VMWare, este servidor de mantenimiento sera el Cluster; en el caso de las máquinas físicas, este servidor de mantenimiento es la interfaz iLO, desde la cual se puede gestionar el servidor. Con estas directivas se puede gestionar su reinicio, arranque por red, etc. En el apartado `network` se indica la configuración de red de la máquina, tanto la externa como la de red PXE. Por último se especifican el número de estaciones de trabajo asociadas al servidor y su nombre (si la máquina fuese una estación de trabajo estos campos no los tendría), y el tipo de la máquina (si es servidor o estación de trabajo).

4. `releases.yaml` y `releases.yaml.example`: al igual que en el caso de las máquinas, este fichero también dispone de un ejemplo del cuál partir para poder configurar un caso particular.

   Este fichero contiene las versiones disponibles del código del proyecto, y las imágenes ISO correspondientes a cada versión, tanto para servidor como para estación de trabajo. Además se almacena el identificador de *build* de cada ISO, para así saber, sin tener que realizar *scrapping*[2] de las imágenes ISO generadas, la versión (que corresponde con el tipo de *build* y la fecha del mismo) de las imágenes ISO a utilizar.

```
---
releases:
V3A.01:
build_number: DR2-20190312.0044
server_iso: msf-server-V3A.01-DR2-20190312.0044.iso
workstation_iso: msf-workstation-V3A.01-DR2-20190312.0044.iso
```

4.2.3. Directorio *run*

En la figura 4.6 se puede ver la estructura de este directorio. Estos ficheros son, como se ha comentado anteriormente, los que almacenan los estados de las máquinas.

[2]Técnica utilizada para la extracción de información de un sitio en concreto, leyendo lo que allí se encuentra y quedandose con lo que interesa. En este caso, se daría al leer el nombre de la imagen ISO generada, y recoger el número de build que allí se encuentra.

Figura 4.7: Estructura del directorio *lib*.

```
[root@pxe-platform lib]# ll -R
.:
total 4
drwxr-xr-x. 2 root root  127 Mar 12 12:19 MSFAdt
-rw-r--r--. 1 root root 1759 Feb 15 06:42 MSFAdt.pm

./MSFAdt:
total 40
-rwxr-xr-x. 1 root root 13927 Mar 12 12:19 API.pm
-rw-r--r--. 1 root root  3779 Feb 19 04:57 MachineReboot.pm
-rw-r--r--. 1 root root   963 Feb 15 06:42 MSFLogConfiguration.
   pm
-rw-r--r--. 1 root root  3114 Feb 15 06:42 StateMachine.pm
-rw-r--r--. 1 root root 10547 Mar 12 12:19 SystemConfiguration.
   pm
```

Además, aquí se almacena también el fichero `api.pid`, que es el que contiene el ID del proceso del servicio web. Contiene un fichero plantilla `template-state.yaml`, que es desde el cual se generan nuevos ficheros de estado de las máquinas, cuando estas se acaban de definir y no existe su fichero específico de estado.

Un pequeño ejemplo de fichero de configuración de estado, por ejemplo `msf-srv1-state.yaml`, tiene la siguiente sintaxis:

```
---
last_change_time_utc: '2019/02/15 14:40:34'
machine_name: msf-srv1
machine_state: INSTALLED
```

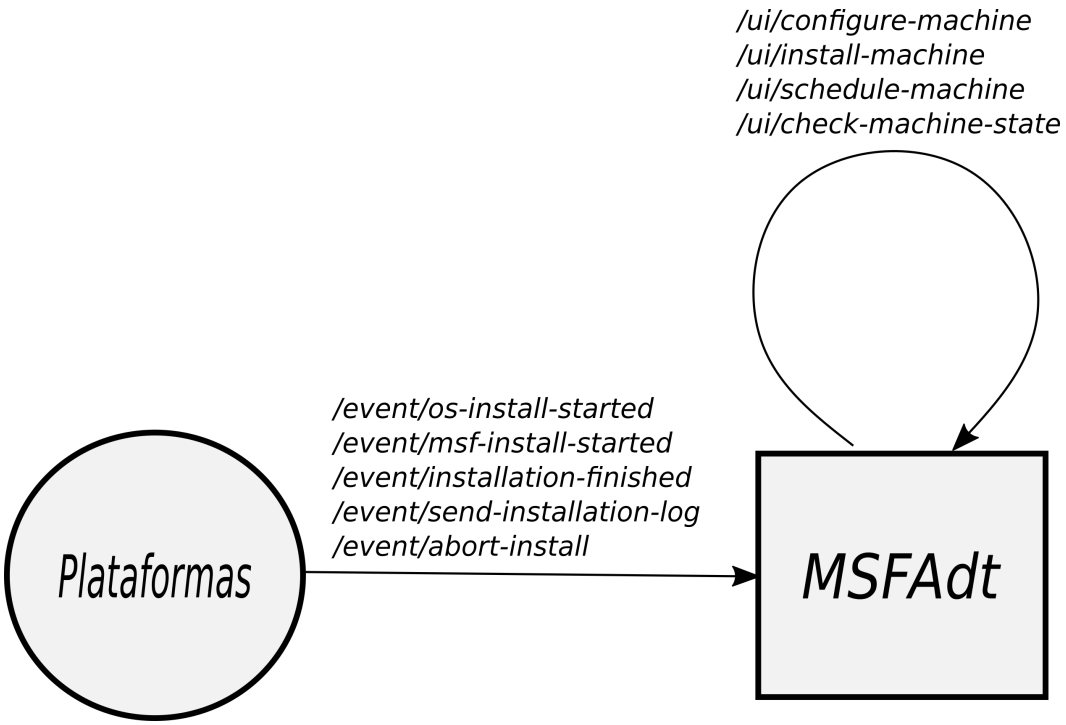
4.2.4. Directorio *lib*

En la figura 4.7 se indica la estructura del directorio `lib`, el ***corazón*** de la aplicación *MSFAdt*.

Como se ha dicho anteriormente, el fichero que se ejecuta el primer lugar es `API.pm`, el cuál será el encargado de definir las diferentes rutas para las peticiones *HTTP*, y las funciones que realizará la aplicación en el caso de recibir una determinada petición [Tom12]. Los demás paquetes almacenan estas funciones, encargadas de reiniciar máquinas, cargar la configuración *YAML*, recibir los logs de instalación, cambiar entre estados y realizar tareas en el sistema (cómo configuración DHCP, TFTP, etc.).

El fichero `MSFAdt.pm`, recoge las funciones encargadas de leer los ficheros de configuración alojados en `etc`, y almacenarlos en un objeto *Perl* llamado `$cfg` [Tom03] [Bri09]. Este objeto contiene la configuración general de `main.yaml` bajo `$cfg`, la confi-

Figura 4.8: Peticiones */event* y */ui* de la aplicación *MSFAdt*.



guración de cada máquina bajo `$cfg->{'machines'}{$machine}` y la configuración de las versiones en `$cfg->{'releases'}{$release}`. Y esto es así para todos los ficheros de *MSFAdt*.

Como ya se ha dicho, en el fichero `API.pm` se definen las diferentes rutas que puede haber en el servicio web. Si la ruta que se pide no existe, se mostrará un error `404 Not Found` desde dónde se quiera acceder a dicha ruta. Las rutas que definimos en esta aplicación se definen en la tabla 4.1[3], y las transiciones entre estados que originan, en la tabla 4.2. Esta transición entre estados requiere previamente que la máquina se encuentre en el estado definido en primer lugar. Si no se encuentra en dicho estado, el servicio web devolverá un error.

Cada una de estas peticiones desencadena una serie de funciones dentro del sistema. En concreto, las peticiones bajo el prefijo */ui* hacen referencia a peticiones realizadas desde el propio servidor que contiene la aplicación, es decir, se trata de una interacción desde la interfaz de usuario (*user interface*), mientras que las peticiones realizas desde las máquinas a instalar se alojan debajo de */event*, debido a que son eventos recibidos desde la red *PXE*. Para entender mejor esta característica, se muestra en la figura 4.8 un breve bosquejo de esta situación. Veamos estas funciones en detalle:

1. ***/ui/configure-machine***: esta petición debe de ir acompañada de una variable

[3]La ruta */ui/check-machine-state/¡machine¿* se usó mientras no existía la interfaz web. De esta manera era posible saber el estado actual de la máquina sin leer los ficheros de configuración */run/maquina-state.yaml*.

Prefijo	Ruta	Función
/ui	/configure-machine	Configura la máquina para el arranque por red.
	/schedule-machine	Programa la instalación para una fecha concreta.
	/install-machine	Reinicia la máquina desde la tarjeta de red PXE.
	/check-machine-state/:machine	Devuelve el estado actual de la máquina.
/event	/abort-install	La instalación se ha abortado.
	/os-install-started	Ha comenzado la instalación del SO.
	/msf-install-started	Ha comenzado la instalación del software MSF.
	/installation-finished	Ha terminado la instalación.
	/send-installation-log/:machine	Envía el log de la instalación.

Tabla 4.1: Rutas *MSFAdt*.

en formato *JSON*[4] dónde se indica la máquina a configurar (por ejemplo, *msf-srv41*, *msf-ws411*, etc.). Para realizar esta petición se recurre a un script *Perl*, alojado en el directorio `tools`, llamado `configure-machine.pl`. La ejecución de este script requiere como argumento el nombre de la máquina a configurar y envía una petición al servidor web:

```
1  (...)
2  my $ua = LWP::UserAgent->new;
3  my $req = HTTP::Request->new( POST => "http://$address:
       $port/ui/configure-machine" );
4  $req->content_type( 'application/json' );
5  $req->content( to_json( { machine => $machine } ) );
6
7  my $res = $ua->request( $req );
8  if ( $res->is_success ) {
9    my $data = from_json( $res->content );
10   if ( $data->{'error'} ) {
11     printf "error: %s\n", $data->{'message'};
12   } else {
13     printf "success: %s: %s\n", $data->{'machine'}, $data
         ->{'message'};
14   }
15 } else {
16   die $res->status_line . "\n";
17 }
18 (...)
```

Si no se ha producido ningún error en el servidor web, el script muestra un texto

[4] *(JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo* [Ecm17].

Prefijo	Ruta	Estado Origen	Estado Destino
/ui	/configure-machine	**INSTALLED** **INSTALLED_WITH_ERRORS**	**CONFIGURED**
	/schedule-machine	**CONFIGURED**	**SCHEDULED**
	/install-machine	**CONFIGURED** **SCHEDULED**	**REBOOTING**
/event	/abort-install	**INSTALLING_OS** **INSTALLING_MSF**	**ABORTED**
	/os-install-started	**REBOOTING**	**INSTALLING_OS**
	/msf-install-started	**INSTALLING_OS**	**INSTALLING_MSF**
	/installation-finished	**INSTALLING_MSF**	**INSTALLED**
	/send-installation-log	**INSTALLED**	**INSTALLED** **INSTALLED_WITH_ERRORS**

Tabla 4.2: Rutas y sus transiciones de estados de *MSFAdt*.

de éxito durante la operación.

En la parte del servidor, cuando recibe esta petición, lo que hace en primer lugar es comprobar que la máquina pasada por *JSON* existe en la base de datos. Si no existe, el proceso acaba en error y no se realiza la transición entre estados.

```
1  (...)
2  # check machine is valid
3  if ( not is_valid_machine($machine) ) {
4          $log->error("Machine '$machine' not found in
              inventory");
5          return {
6                  machine => $machine,
7                  error   => 1,
8                  message => "Machine '$machine' not found in
                       inventory",
9          };
10 }
11 (...)
```

El segundo paso que se realiza es comprobar que la transición entre el estado actual de la máquina y al que se quiere acceder está permitido en el fichero de configuración `etc/main.yaml`. En caso contrario no se realiza la transición entre estados y se devuelve un error:

```
1  (...)
2  # get origin and end state
3  my $current_state = get_machine_state($machine);
```

```
4  my $new_state     = $param_new_state;
5
6  # check that the state transition is valid
7  if ( not state_transition_is_allowed( $current_state =>
       $new_state ) ) {
8          $log->error("State transition not allowed (
               $current_state -> $new_state)");
9          return {
10                 machine => $machine,
11                 error   => 1,
12                 message => "$machine: State transition not
                       allowed ($current_state -> $new_state)",
13         };
14 }
15 (...)
```

Acto seguido se comprueba si hay que realizar alguna función específica para esa transición de estados. Para `configure-machine` se realiza una función almacenada en el fichero `SystemConfiguration.pm`, llamada `generate_system_configuration_for_machine($machine)`, y lo que hace es añadir la configuración de esta máquina al servicio *DHCP* y los ficheros necesarios para su arranque en el servicio *TFTP*.

Entrando en detalle en la función `generate_system_configuration_for_machine`, lo primero que se hace es poner la máquina en el estado ***CONFIGURED***, para acto seguido configurar el servicio *DHCP*, añadiendo las máquinas que se encuentren en un estado distinto a ***INSTALLED***, ***INSTALLED_WITH_ERRORS*** ó ***ABORTED***:

```
1  (...)
2  # 2.1. Regenerate DHCPD configuration
3  my @hosts;
4  foreach my $m ( keys %{ $cfg->{'machines'} } ) {
5    if ( ( get_machine_state( $m ) ne 'INSTALLED' ) && (
         get_machine_state( $m ) ne 'INSTALLED_WITH_ERRORS' )
         && ( get_machine_state( $m ) ne 'ABORTED' )) {
6      push @hosts, $m;
7    }
8  }
9  (...)
```

De esta manera, usando la plantilla previamente mencionada en `config-templates`, se modifica el fichero de configuración `/etc/dhcp/dhcpd.conf`, añadiendo la configuración de todos los *hosts* que se encuentren en esos estados. El contenido de la platilla es:

```
ddns-update-style interim;
ignore client-updates;
authoritative;
allow booting;
```

```
allow bootp;
option arch code 93 = unsigned integer 16;

subnet [% config.pxeboot.dhcpd.network %] netmask [% config.pxeboot.dhcpd.
    netmask %] {
  range [% config.pxeboot.dhcpd.address_range.join(' ') %] ;
  default-lease-time 600;
  max-lease-time 7200;
  deny unknown-clients;

  [% FOREACH host IN hosts %]

  host [% host %] {
    hardware ethernet [% config.machines.$host.network.pxe.macaddress %];
    next-server [% config.pxeboot.dhcpd.server_address %];
    fixed-address [% config.machines.$host.network.pxe.ipaddress %];
    if option arch = 00:07 {
    filename "[% host %]/efi/shim.efi";
    } else {
    filename "[% host %]/pxelinux.0";
    }
  }
[% END %]
}
```

Donde las variables `[% VAR %]` se sustituyen en la función por los valores específicos de cada máquina.

Después se configura el servicio *TFTP*. En primer lugar se copia la carpeta `boot.save` con el nombre específico de la máquina en el caso de que no existiese previamente. Tras esto se modifican los ficheros `pxelinux.cfg/default` y `grub.cfg` haciendo uso de plantillas, como en el caso de *DHCP*:

`default`:

```
default linux
timeout 0

label linux
timeout [% time %]
menu default
kernel vmlinuz
append initrd=initrd.img ip=dhcp inst.repo=nfs:[% nfs_server %]:/export/isos/[%
    release %]/[% buildnumber %]/[% type %]/ inst.ks=http://[% server_IP_address
     %]/releases/[% version %]/[% type %]/MSF/Kickstart/[% type %]-ks.cfg msf.
    machinetype=[% machine_type %] msf.[% type %]type=[% server_type %] msf.
    updateinstallation=[% mode %] msf.ipaddress=[% ipaddress %] msf.netmask=[%
    netmask %] msf.gateway=[% gateway %] msf.platform=[% platform %] [% wsnumber
     %] msf.release=[% release %] msf.pxeinstallation=1 msf.pxe_macaddress=[%
    pxe_macaddress %] [% sec_ipaddress %] [% sec_netmask %] [% wra_ipaddress %]
    [% wra_netmask %] msf.imageiso=[% imageiso %]
```

`grub.cfg`:

```
set default="0"

function load_video {
        insmod efi_gop
        insmod efi_uga
        insmod video_bochs
        insmod video_cirrus
        insmod all_video
}
```

```
load_video
set gfxpayload=keep
insmod gzio
set timeout=1

menuentry 'linux' --class fedora --class gnu-linux --class gnu --class os {
        linuxefi (tftp)/[% machine %]/efi/vmlinuz ip=dhcp inst.repo=nfs:[%
            nsf_server %]:/export/isos/[% release %]/[% buildnumber %]/[% type
            %]/ inst.ks=http://[% server_IP_address %]/releases/[% version %]/[%
             type %]/MSF/Kickstart/[% type %]-ks.cfg msf.machinetype=[%
            machine_type %] msf.[% type %]type=[% server_type %] msf.
            updateinstallation=[% mode %] msf.ipaddress=[% ipaddress %] msf.
            netmask=[% netmask %] msf.gateway=[% gateway %] msf.platform=[%
            platform %] [% wsnumber %] msf.release=[% release %] msf.
            pxeinstallation=1 msf.pxe_macaddress=[% pxe_macaddress %] [%
            sec_ipaddress %] [% sec_netmask %] [% wra_ipaddress %] [%
            wra_netmask %] msf.imageiso=[% imageiso %]
        initrdefi (tftp)/[% machine %]/efi/initrd.img
}
```

Donde `nfs_server` es la dirección IP de las cabinas de almacenamiento (donde se guardan las imágenes ISO generadas) y `server_IP_address` es la dirección IP del servidor *Apache* de *MSFAdt*, donde se encuentran montadas las imágenes ISO.

Una vez realizadas estas configuraciones, se reinicia el servicio *DHCP* y ya habría quedado configurada la máquina y en el estado ***CONFIGURED***.

2. ***/ui/schedule-machine***: análogamente a como se hace en `configure-machine`, al comienzo se comprueba la existencia de la máquina pasada por *JSON* y si la transición entre estados está permitida. En este caso, se pasa otro parámetro por *JSON*, y este es la fecha en la que se desea programar la instalación. En este caso, se ejecuta la función `schedule_installation($machine, $installation_date)`, alojada en el fichero `SystemConfiguration.pm`.

   En esta función, que recibe como argumento la máquina a programar y su fecha, se programa la ejecución del script `tools/install-machine.pl`[5] mediante la herramienta de Linux `at`. *Al igual que la herramienta cron, esta herramienta permite la posibilidad de ejecución de scripts o comandos a una fecha y hora específicas* [Red19b]. La programación de la ejecución de la llamada al sistema `install-machine`, la conseguimos con el siguiente comando en la función `schedule_installation`:

```
1 (...)
2 $log->info("Machine '$machine' was scheduled for PXE
     installation at '$installation_date'") if system("at -t
     $installation_date <<EOF;
3 $ENV{ 'MSFADT_HOME' }/$cfg->{'scripts'}/install-machine.pl
     -m $machine
4 EOF
5 (...)
```

[5]Que es el encargado de enviar la petición *install-machine* al servicio web, como se describe en la sección 4.2.5

Hecho esto, se realiza la transición de estados al estado ***SCHEDULED*** y se devuelve una señal de éxito.

3. ***/ui/install-machine***: de igual forma que las anteriores peticiones, esta se envía mediante un script en `tools/install-machine.pl`, es decir, se realizan localmente. En el fichero `API.pm`, cuando se recibe la petición a esta ruta, se realizan las comprobaciones pertinentes (existencia de máquina y comprobación de transición de estados), y además se realiza una comprobación extra. Cuando se instala una plataforma completa (servidor + estaciones de trabajo), el orden adecuado de instalación es en primer lugar el servidor y después la estación de trabajo. Por ello, se añade un estado intermedio llamado ***WAITING_SRV_INSTALL***, al cual se llega únicamente si la máquina es una estación de trabajo y su servidor va a ser instalado junto con ella (en caso contrario, no se pasa a este estado y se instala con normalidad). Esto lo comprobamos en el código con:

```
(...)
#By default the next state and the function which the system has to run are the next ones.
my $new_state  = 'REBOOTING';
my $param_code = \&reboot_machine_for_installation;

#If the machine is a workstation, and its server is ready to be installed, the new states will be WAITING_SRV_INSTALL and the installation don't starts for them.
# (it starts when the server has finished).
foreach my $m ( keys %{ $cfg->{'machines'} } ) {
  if ( ( $cfg->{'machines'}{$m}{'type'} eq 'server' ) && ( get_machine_state($m) ne 'INSTALLED' ) && ( get_machine_state($m) ne 'INSTALLED_WITH_ERRORS' ) ) {
    foreach my $ws_machine ( @{ $cfg->{'machines'}{$m}{'ws'} } ) {
      if ( ( $cfg->{'machines'}{$machine}{'type'} eq "workstation" ) && ( $ws_machine eq $machine ) ) {
        $new_state  = 'WAITING_SRV_INSTALL';
        $param_code = undef;
        last;
      }
    }
  }
}
(...)
```

En el primer caso (es un servidor o una estación de trabajo y su servidor no va a ser instalado), el nuevo estado será ***REBOOTING***, y se ejecutará la función `reboot_machine_for_installation($machine)`. En el otro caso (es una estación de trabajo y su servidor va a ser instalado también), el nuevo estado de esta máquina será ***WAITING_SRV_INSTALL***, y no se realizará ninguna función extra; simplemente se cambiará de estado.

La función `reboot_machine_for_installation`, alojada en el fichero `MachineReboot.pm` es la encargada de reiniciar la máquina seleccionada y configurar su arranque para que lo haga desde la tarjeta de red conectada a la red PXE. En ella existen varios métodos de reinicio y configuración, dependiendo del tipo de máquina a instalar. Es decir, para máquinas físicas basadas en iLO (HPE o Sun Oracle), se ingresa en la iLO remotamente mediante *SSH*, se configura el orden de arranque y se reinicia. Para máquinas virtuales KVM, se ingresa mediante *SSH* en el servidor dónde se aloja dicha máquina y mediante línea de comandos se reinicia[6]. Por último, en el caso de máquinas virtuales VMWare, se ejecuta el script `configure_boot_and_reboot_vm.sh`. Este script esta alojado en un directorio aparte de este sistema, pues no se utiliza únicamente en este sistema. Este script está escrito en *PowerShell*, pues utiliza un módulo específico para la administración de máquinas virtuales *VMWare Vsphere*. Lo que se hace en él es apagar la máquina virtual, configurar la máquina para que arranque desde la tarjeta de red de PXE y volver a iniciar la máquina virtual. Recibe como argumentos el usuario y contraseña para ingresar en el servicio *VCenter*, encargado de la gestión de las máquinas virtuales emphVMWare, y la máquina virtual a reiniciar.El código utilizado es:

```
1  (...)
2
3  pwsh -Command "Connect-VIServer -Server $SERVER -User $USER -Password $PASSWD;
       Stop-VM -VM $VM -Confirm:\$False; \$vm = Get-VM -Name $VM ; \
       $intNICDeviceKey = (\$vm.ExtensionData.Config.Hardware.Device | ?{\$_.
       DeviceInfo.Label -eq ((Get-NetworkAdapter -VM $VM)[-1].Name)}).Key; \
       $oBootableNIC = New-Object -TypeName VMware.Vim.
       VirtualMachineBootOptionsBootableEthernetDevice -Property @{"DeviceKey" = \
       $intNICDeviceKey}; \$spec = New-Object VMware.Vim.VirtualMachineConfigSpec -
       Property @{ "BootOptions" = New-Object VMware.Vim.VirtualMachineBootOptions
       -Property @{ BootOrder = \$oBootableNIC } }; \$vm.ExtensionData.
       ReconfigVM_Task(\$spec); Start-VM $VM -Confirm:\$False"
4
5  (...)
```

Reiniciada la máquina, se pasa al estado ***REBOOTING*** y comienza su instalación.

La herramienta utilizada para la comunicación entre todos los integrantes del proyecto *MSF* es *Skype*. Los avisos y notificaciones sobre instalaciones y despliegues se realizan mediante esta herramienta. Cómo función adicional al sistema, se ha implementado la ejecución de un *script* escrito en *Python* que informa al grupo del proyecto de que una máquina a empezado su instalación. De esta forma, los trabajadores se percatarán de dicha información, notificada por el propio sistema de forma automática, y esperarán a que se haya reinstalado para volver a entrar en dicha máquina:

```
1  (...)
```

[6]En este caso, el arranque por red esta configurado por defecto en ellas, por lo que no es necesario modificarlo previamente.

```
2  system("ssh -t dev\@172.22.10.186 \'/home/dev/skype_script/
      $cfg->{'send_skype_script'} \"Machine $machine started
      its installation in mode-type: $mode-$type and version:
      $install_release ( $build_number )\"\'");
3  (...)
```

4. ***/event/abort-install***: esta ruta, a diferencia de las anteriores, es llamada desde las máquinas a instalar. Por tanto, se debe implementar en el *Kickstart* empleado para la instalación de las mismas. La información que nos da esta llamada es que la instalación de la máquina ha sido abortada, y por tanto no se ha desplegado el código correctamente (e incluso no se ha podido instalar el sistema operativo). La función que realizará esta petición desde las máquinas será:

```
1  (...)
2  function abort_and_reboot {
3    echo "$1"
4    echo ""
5    echo "============================================================"
6    echo "  INSTALLATION ABORTED - PRESS ANY KEY TO REBOOT SYSTEM"
7    echo "============================================================"
8
9    if [ "$pxeinstallation" == "1" ]; then
10     wget --header="Content-type:application/json" \
11     --post-data='{"machine":"'$hostname'"}' \
12     http://10.40.1.2:3001/event/abort-install
13   else
14     read rubbish
15   fi
16
17   /sbin/reboot
18
19 }
20
21 (...)
```

Como se puede ver, esta llamada se ha añadido a una función ya existente en el *Kickstart* empleado en el proyecto anteriormente. Sin embargo hay que diferenciar cuándo una instalación se realiza por *PXE* y cuando por disco, pues cuando se realiza por disco estas llamadas no se deben de realizar. Esto se hace mediante una variable del *kernel*, cuando la instalación se ve abortada, se envía la petición a la aplicación web. En la aplicación, esta llamada simplemente modifica el estado a ***ABORTED***, y no realiza ninguna acción especial.

Para la comprobación de esta llamada al sistema, existe un script en `api-test/abort-install.pl`, que se encarga de simular esta llamada al sistema, pudiendo así comprobar su funcionamiento.

5. ***/event/os-install-started***: esta ruta indica al servicio web que ha empezado la instalación del sistema operativo en la máquina pasada por parámetro *JSON*. Al igual que en el caso de */ui/abort-install*, a esta ruta se la llama desde el *Kickstart* alojado en las máquinas durante su instalación. En concreto, se llama en el apartado de *%pre*, en el cual se ejecutan los scripts de antes de instalar el sistema operativo:

```
1  (...)
```

```
case $pxeinstallation in

   1 )
     (...)
     wget --header="Content-type:application/json" \
     --post-data='{"machine":"'$hostname'"}' \
     http://10.40.1.2:3001/event/os-install-started
   ;;
(...)
```

Como podemos ver, el *Kickstart* existente previamente en el proyecto se debe ver modificado, tanto para añadir estas llamadas al servicio web como para desplegar el código remotamente (sin tener introducido el DVD de instalación). La parte más importante a modificar viene a continuación.

La única función que se realiza cuando se recibe esta llamada es la transición entre estados.

6. ***/event/msf-install-started***: mismo caso que */event/os-install-started*. Aquí se indica que ha comenzado la instalación del software *MSF* y se ejecuta al comenzar la sección *%post* del *Kickstart*; zona donde se ejecutan los *scripts* de post instalación.

```
(...)
#echo "Copying Installation DVD contents to the hard disk. Please wait..."
mkdir -p /mnt/sysimage/media/MSFDVD
pxeinstallation=`cat /proc/cmdline|grep -Po 'msf.pxeinstallation=[^\s]+'|cut -f2
     -d=`
imageiso=`cat /proc/cmdline|grep -Po 'msf.imageiso=[^\s]+'|cut -f2 -d=`
if [ "$pxeinstallation" == "1" ]; then
   mkdir /tmp/isos
   mount 10.40.1.3:/export/isos/ /tmp/isos
   mount -o ro /tmp/isos/$imageiso /mnt/sysimage/media/MSFDVD
   chroot /mnt/sysimage /media/MSFDVD/MSF/Kickstart/api-calls/msf-install-
       started.sh
else
   mount -o bind /run/install/repo /mnt/sysimage/media/MSFDVD
fi
(...)
```

Esta modificación en el *Kickstart* es la más importante para el despliegue del código. Una vez instalado el sistema operativo y configurado a nuestro gusto con la directivas pertinentes (y ya existentes) del *Kickstart*, se procede al despliegue del código *MSF*. Si se trata de una instalación en red, el software no se encuentra en el DVD, y por tanto este no sera la fuente del montaje, por lo que tendremos que introducir un condicionante que permita, en el caso de una instalación por red, montar el contenido del DVD (y por tanto el software MSF) localmente desde la red PXE. Una vez hecho esto, ya tenemos los ficheros del DVD disponibles, por lo que podemos ejecutar las llamadas al servicio web *MSFAdt* mediante la ejecución de un script, y es lo que hacemos en este caso para indicar que ha comenzado la instalación del software *MSF*.

```
msf-install-started.sh:

#!/bin/bash

MACHINE=`hostname -s`
ADDRESS="10.40.1.2"
```

```
7  PORT=3001
8
9  curl --header "Content-type: application/json" \
10 --request POST \
11 --data '{"machine":"'$MACHINE'"}' \
12 http://$ADDRESS:$PORT/event/msf-install-started
```

Al igual que en el caso anterior, la única función que se realiza cuando se recibe esta llamada es la transición entre estados.

7. ***/event/installation-finished***: ruta que indica que la instalación y despliegue de código ha terminado en la máquina pasada por parámetro *JSON*. Esta llamada se realiza, al igual que las anteriores, desde el *Kickstart* asociado a la máquina en cuestión; concretamente al final de la sección *%post*, tras haber desplegado todo el software. Su ejecución se realiza mediante un script, al igual que en */event/msf-install-started*.

   En el servicio web, cuando se recibe esta petición, además de realizar las pruebas rutinarias (existe la máquina, transición entre estados, etc.), se ejecuta una función llamada `regenerate_system_config($machine)`. Esta función esta alojada en el fichero `SystemConfiguration.pm` y se encarga de restablecer la configuración por defecto de la máquina a la de antes de haber sido instalada. En concreto, se cambia el estado de la máquina a ***INSTALLED***, y se realiza la misma operación que en `generate_system_configuration_for_machine`, se comprueba que máquinas no están en los estados ***INSTALLED***, ***INSTALLED_WITH_ERRORS*** y ***ABORTED***, y se añaden a la configuración del servicio *DHCP*. Como en este caso la máquina a regenerar su configuración se encuentra en el estado ***INSTALLED***, no se añadirá al servicio *DHCP*.

   Adicionalmente, se realizan dos acciones importantes más: instalar la estación de trabajo asociada al servidor (en el caso de que la máquina instalada haya sido un servidor y tenga asociada una estación de trabajo en el estado ***WAITING_SRV_INSTALL***) y configurar de nuevo la máquina instalada para que arranque desde el disco duro y no desde la tarjeta de red (como habíamos configurado previamente). Esto lo hacemos con dos funciones: `install_ws` y `set_redhat_bootorder`.

   En el caso de `install_ws`, se comprueba previamente que se haya instalado un servidor para su ejecución. Seguidamente se comprueba que su estación de trabajo asociada (indicada en los ficheros *.yaml*) está en el estado ***WAITING_SRV_INSTALL***; en caso afirmativo, comienza la instalación de la estación de trabajo cambiando su estado a ***REBOOTING***.

   En `set_redhat_bootorder` hay que diferenciar el tipo de máquina instalada. Si se trata de un servidor físico, la operación se realiza mediante una conexión *SSH* al *iLO* (al igual que hicimos para configurar el arranque por red). Si se trata de una máquina VMWare, existe un script, análogo al utilizado para configurar el

arranque por red, el cual configura la máquina virtual VMWare para que arranque desde el disco duro. El código de este *script* `configure-redhat-boot.sh` es:

```
(...)
pwsh -Command "Connect-VIServer -Server $SERVER -User $USER -Password $PASSWD; \
    $vm = Get-VM -Name $VM ; \$intHDDeviceKey = (\$vm.ExtensionData.Config.
    Hardware.Device | ?{\$_.DeviceInfo.Label -eq (Get-HardDisk -VM $VM).Name}).
    Key; \$oBootableHD = New-Object -TypeName VMware.Vim.
    VirtualMachineBootOptionsBootableDiskDevice -Property @{"DeviceKey" = \
    $intHDDeviceKey}; \$spec = New-Object VMware.Vim.VirtualMachineConfigSpec -
    Property @{ "BootOptions" = New-Object VMware.Vim.VirtualMachineBootOptions
    -Property @{ BootOrder = \$oBootableHD } }; \$vm.ExtensionData.
    ReconfigVM_Task(\$spec)"
(...)
```

8. ***/event/send-installation-log***: por último está la petición encargada de enviar mediante una petición *POST* el log completo de la instalación. Este log se encuentra alojado en la máquina mientras que va siendo instalada, y es donde se vuelcan todos los resultados de los scripts realizado para el despliegue del código. Mediante este fichero, podemos comprobar si la instalación ha ido correctamente o si se ha producido algún fallo simplemente buscando por el fichero si existe alguna palabra clave como *error*, *failed*, *aborted*, etc. Esta comprobación la realizamos mediante el script llamado `check-errors.sh` y que existía previamente en el proyecto[7].

   Esta llamada se realiza en el *Kickstart* justo después de */event/installation-finished*, y se ejecuta con el script `send-installation-log.sh`:

```
#!/bin/bash

MACHINE=‘hostname -s‘
ADDRESS="10.40.1.2"
PORT=3001

echo -e "{\n\t\"logfile\" : \"‘base64 /root/msf-software-install-full.log | tr
    ’\n’ ’ ’‘,\n\"}" > /root/msf-software-install-full-json.log

curl -X POST -H "Content-type: application/json"  \
-d @/root/msf-software-install-full-json.log \
http://$ADDRESS:$PORT/event/send-installation-log/$MACHINE

rm -rf /root/msf-software-install-full-json.log
```

   La función de este *script* es enviar todo el contenido del fichero `/msf-software-install-full.log` al servidor web *MSFAdt* en una variable *JSON*. Para ello antes adecuamos su forma en un fichero `msf-software-install-full-json.log`, y se envía.

   En el servidor web, al recibir esta llamada, además de las funciones rutinarias de comprobación, se recoge el fichero enviado y se decodifica (debido a la codificación previa realizada en la máquina instalada):

```
(...)
```

[7]Para esta aplicación, se ha realizado un enlace simbólico desde su ruta original a nuestra ruta *tools*, para tenerlo accesible desde una ruta más cercana.

Figura 4.9: Estructura del directorio *tools*.

```
[root@pxe-platform automatic-deployment]# ll tools/
total 52
lrwxrwxrwx. 1 root root   59 Feb 15 06:42 check_errors.sh ->
   ../../automated-install/server-iso/MSF/util/check_errors.sh
-rwxr-xr-x. 1 root root  941 Feb 15 06:42 configure-machine.pl
-rwxr-xr-x. 1 root root 7826 Mar 13 09:03 edit_wiki.pl
-rw-r--r--. 1 root root   49 Feb 15 06:42 groupid
-rwxr-xr-x. 1 root root  939 Feb 15 06:42 install-machine.pl
-rwxr-xr-x. 1 root root 3188 Feb 26 06:18 mount-releases.pl
-rw-r--r--. 1 root root   44 Feb 15 06:42 redmine
-rwxr-xr-x. 1 root root 1033 Feb 15 06:42 schedule-machine.pl
-rwxr-xr-x. 1 root root  416 Mar 12 12:19 send_skype_message.py
-rwxr-xr-x. 1 root root 1037 Feb 15 06:42
   set_installed_state_all.pl
-rwxr-xr-x. 1 root root  976 Mar 12 12:19 set_installed_state.
   pl
-rw-r--r--. 1 root root 6850 Mar 19 08:36 wiki_content.save
```

```
2  #Gets the log recieved and decodes it
3  my $log_data = decode_base64( body_parameters->get('logfile') );
4  (...)
```

Una vez con el fichero volcado en una variable, realizamos la ejecución del script `check_errors` para comprobar si ha habido algún fallo en la instalación. Si este no devuelve nada, el nuevo estado será ***INSTALLED***, mientras que si este devuelve algún error, se pasará al estado ***INSTALLED_WITH_ERRORS***. Tras haber hecho esto, se almacena el log recibido en un fichero dentro del directorio `machine-logs/<maquina>/`, creando así una base de datos de logs para cada máquina, por si en un futuro nos son necesarios.

Esta es la función más importante de esta llamada. Sin embargo se realizan algunas funciones extra, como por ejemplo la notificación por Skype de que se ha instalado correctamente o con fallos la máquina, o la modificación de la *Wiki* de la empresa automáticamente con la versión y fecha pertinentes mediante el script `tools/edit-wiki.pl`. En el proyecto existe una página de información para los trabajadores alojada en *Redmine* [Jea19]. En esta página se indica la fecha de instalación de la máquina, la versión, el nombre, cómo acceder, etc. Esta aplicación se encarga de modificar esta página automáticamente con cada instalación, ahorrando el tener que modificar este recurso web asiduamente.

4.2.5. Directorio *tools*

Además de los *scripts* previamente mencionados en la sección 4.2.4, existen algunos otros cómo se puede ver en la figura 4.9. Dando por explicados los scripts *check_errors.sh, configure-machine.pl, edit-wiki.pl, install-machine.pl, schedule-machine.pl*

Figura 4.10: Estructura del proyecto *MSFAdt_build*.

```
[root@pxe-platform msfadt_build]# ll
total 4
drwxr-xr-x. 2 root root   62 Feb 21 07:05 api
drwxr-xr-x. 2 root root 4096 Mar 11 08:32 build-log
drwxr-xr-x. 2 root root   27 Feb 15 06:42 doc
drwxr-xr-x. 2 root root   84 Mar 11 08:45 etc
drwxr-xr-x. 3 root root   49 Feb 15 06:42 lib
drwxr-xr-x. 2 root root   67 Feb 18 03:20 log
drwxr-xr-x. 2 root root   45 Mar 12 12:19 run
drwxr-xr-x. 2 root root   28 Mar  7 06:01 tools
```

y send_skype_message.py, existe:

1. `groupid`: fichero donde se almacena el identificador del grupo de *Skype* empleado en el proyecto. Es usado por el script `send_skype_message.py`, para enviar el mensaje al grupo especificado.

2. `mount-releases.pl`: script encargado de montar las imágenes ISO especificadas en el fichero de configuración `releases.yaml` en el directorio correspondiente `/vm/releases/<version>/<server/workstation>`. Recoge las ISOs desde la cabina de almacenamiento (montada localmente en el sistema operativo) y las despliega en el directorio especificado (que se trata del directorio compartido por *HTTP*).

3. `redmine`: fichero que guarda las credenciales de acceso para ingresar en la plataforma *Redmine*, sitio donde se encuentra la previamente mencionada *Wiki* del proyecto, y que se ve modificada por el servicio web automáticamente (recogiendo estas credenciales de acceso de este fichero).

4. `set_installed_state.pl` y `set_installed_state_all.pl`: *scripts* encargados de poner una máquina (o todas) en el estado por defecto *INSTALLED*, independientemente del estado en el que se encuentre. Estos scripts son utilizados tanto para reiniciar los estados de todas las máquinas en caso de catástrofe, o para cancelar la instalación de una máquina (pues se cambia su estado de *CONFIGURED*, y por tanto al regenerar la configuración del servicio *DHCP*, cosa que se hace en el script, se borrará la máquina y no será instalada).

5. `wiki_content.save`: fichero que almacena el contenido de la *Wiki* alojada en Redmine, y es usado por el script `edit-wiki.pl`.

4.3. *MSFAdt_build*

Una vez explicado el funcionamiento de la aplicación *MSFAdt*, ya se ha podido probar la instalación de máquinas y la transición entre estados de las mismas. Viendo que esto funciona correctamente, se puede pasar al desarrollo de la aplicación web

Figura 4.11: Estructura del directorio *etc* de *MSFAdt_build*.

```
[root@pxe-platform etc]# ll
total 16
-rw-r--r--. 1 root root   79 Feb 15 08:26 api.yaml
-rw-r--r--. 1 root root   87 Mar 11 08:45 buildconfig.yaml
-rw-r--r--. 1 root root  398 Feb 15 06:42 log4perl.conf
-rw-r--r--. 1 root root 1068 Feb 19 04:57 main.yaml
```

MSFAdt_build, que se encargará de realizar la creación de estas imágenes ISO, contenedoras del software del proyecto, de forma remota y desatendida (pues son necesarios varios pasos para su correcta creación).

Como se ha comentado anteriormente, esta aplicación la desarrollaremos de forma análoga a la aplicación web *MSFAdt*, aunque no es estrictamente necesario para su funcionamiento, por lo que su estructura de directorios y funcionamiento es completamente igual, salvo por el puerto en el que se pone a funcionar el servicio web. Esta aplicación la crearemos bajo el directorio `automatic-deployment/msfadt_build`, utilizando las mismas herramientas que usamos para *MSFAdt*, salvo que en este caso la aplicación se llamará *MSFAdt_build*.

Fijándose en la jerarquía de directorios mostrada en la figura 4.10, vemos que existe en su mayor parte los mismos directorios que en *MSFAdt*. El directorio `api` contiene los ficheros de arranque y parada del servicio, en `doc` se encuentran los manuales de esta aplicación, en `etc` los ficheros de configuración, en `lib` las librerias Perl encargadas de realizar las funciones y la recepción de peticiones HTTP de nuestra aplicación, en `log` los ficheros logs al igual que en *MSFAdt*, en `run` el estado del *build*[8] y por último en `tools` simplemente se almacena el script `start-build.pl` que lanza la petición HTTP al servicio web `/ui/start-build`.

4.3.1. Directorio *etc*

Al igual que en *MSFAdt*, se dispone de los ficheros `api.yaml` y `log4perl.yaml`, donde se especifica la configuración IP del servicio web y la configuración de los logs del programa. Además existe el fichero `main.yaml`, donde se especifica la configuración necesaria para la aplicación, como puede ser la dirección IP del servidor de *build*, usuario y contraseña, y la transición entre estados (figura 3.3), como en *MSFAdt*:

```
(...)
allowed_states:
 CONFIGURED:
  description: 'System configured for building'
  allowed_from:
   - ABORTED
   - BUILT
```

[8]Pues en este caso no existen varias maquinas, existe simplemente un *build* y presenta un único estado que varía entre los estados definidos en la figura 3.3.

Figura 4.12: Estructura del directorio *run* de *MSFAdt_build*

```
[root@pxe-platform run]# ll
total 8
-rw-r--r--. 1 root root   6 Mar  7 06:16 api.pid
-rw-r--r--. 1 root root 182 Mar 12 12:19 build-state.yaml
```

```
 BUILDING_ISOS:
  description: 'System is building all the packages and isos'
  allowed_from:
   - CONFIGURED
 PUBLISHING_ISOS:
  description: 'System is publishing isos into the iso-directory'
  allowed_from:
   - BUILDING_ISOS
 ABORTED:
  description: 'Build failed'
  allowed_from:
   - CONFIGURED
   - BUILDING_ISOS
   - PUBLISHING_ISOS
 BUILT:
  description: 'Build ok'
  allowed_from:
   - PUBLISHING_ISOS

# if a machine has no state associated, it is set to this state
initial_state: BUILT
(...)
```

Adicionalmente existe el fichero `buildconfig.yaml`, donde se especifican las ramas (o etiquetas) de los principales repositorios git de los que recoger el código para la realización del build.

```
---
buildconfig:
csf_branch: develop
msf_branch: develop
msfcots_branch: develop
```

Estos principales repositorios git son `msf` (dónde se encuentran los Kickstart utilizados para la instalación), `csf` (código Java del servidor y estación de trabajo) y `msf-cots` (programas y paquetes externos al sistema operativo). La rama *develop* es la rama principal donde se suben los cambios de nuestro código, y por tanto suele ser la más utilizada en estos builds (salvo la rama *master* que es dónde se sube el código que irá a producción y se ve etiquetado con un número de versión).

4.3.2. Directorio *run*

En este caso solo se dispone de un fichero donde se alojará el estado actual del *build*, pues solo puede existir un *build* simultaneo[9]. Este fichero `build-state.yaml` tiene la misma sintaxis que los ficheros `msf-srvx-state.yaml`, salvo que en este caso también

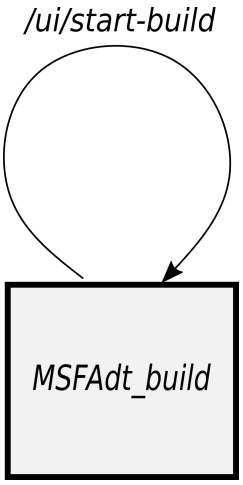
[9]Para poder matar el proceso y parar el servicio web, dado que en *MSFAdt* el fichero *api.pid* guarda el ID del proceso del servicio.

Figura 4.13: Estructura del directorio *lib* de *MSFAdt_build*

```
[root@pxe-platform lib]# ll -R
.:
total 4
drwxr-xr-x. 2 root root   61 Mar 20 08:04 MSFAdt_Build
-rw-r--r--. 1 root root 1274 Feb 15 06:42 MSFAdt_Build.pm

./MSFAdt_Build:
total 16
-rwxr-xr-x. 1 root root 2314 Feb 15 06:42 API.pm
-rw-r--r--. 1 root root 5581 Mar 12 12:19 Functions.pm
-rw-r--r--. 1 root root 2799 Feb 15 06:42 StateBuild.pm
```

Figura 4.14: Peticiones de la aplicación *MSFAdt_build*



se almacena el fichero log del *build* actual, donde escribirá este proceso, para así poder trazar los fallos en el caso de ocurran:

```
---
current_log: /root/DEIMOS-MSF/msf/scripts/automatic-deployment/msfadt_build/build-log/
    build-20190228.155926-develop.log
last_change_time_utc: '2019/02/28 17:33:23'
state: BUILT
```

4.3.3. Directorio *lib*

Este directorio alberga el "corazón" de la aplicación. En este caso, solo existe una ruta disponible a la que realizar la petición HTTP, y esta ruta es `/ui/start-build` (como se puede ver esquematizado en la figura 4.14). Una vez hecha esta petición, la propia aplicación se encarga de conmutar entre los estados dependiendo del resultado de la función realizada en cada estado.

En el fichero `MSFAdt_build.pm` se encuentran las funciones necesarias para cargar

los ficheros de configuración `.yaml` en un objeto tipo *Perl* (exactamente igual que en *MSFAdt*). En `StateBuild.pm` se encuentran las funciones encargadas de leer el estado del *build* y de cambiarlo cuando sea necesario. El fichero `API.pm` es el primer fichero ejecutado con el servicio web, y es el encargado de gestionar que funciones realizar con la petición HTTP recibida. Como en este caso únicamente existe una ruta, cuando reciba una petición a esta, ejecutará el proceso del build (además de comprobar si se encuentra en un estado en el que pueda comenzar el mismo):

```
(...)
# ui routes
prefix '/ui';

post '/start-build'     => state_change_function( 'CONFIGURED',
    \&start_build );
(...)
```

El comienzo de este *build* se realiza en la función `start_build()`, que se encuentra en el fichero `Functions.pm`. Lo primero que hace esta función es fijar el nuevo fichero de log donde se almacenará el resultado del *build*, línea por línea. Una vez hecho esto, se pasa a configurar las ramas y etiquetas de las que se recoge el código del proyecto, y remotamente se configuran los repositorios del servidor de *builds* con esas etiquetas/ramas. Adicionalmente, se debe de ejecutar un *Makefile* en el repositorio `msf-cots`, que realiza la sincronización de los paquetes y ficheros externos dentro de un directorio en concreto del servidor de *builds*:

```
(...)
$log->info("Recovering top msf status");
if ( system("ssh -t deliveries\@$ipaddress 'cd src/DEIMOS-MSF/
   msf/; git checkout -- *; git checkout master' >> $build_log")
    ) {
  return abort_and_return( $build_log );
}

if ( qx(ssh -tq deliveries\@$ipaddress 'cd src/DEIMOS-MSF/msf/;
    git pull -q ; git branch -a --list $msf_branch') ) {
  $log->info("A branch was selected...");
  configure_branches($msf_branch, $csf_branch, $msfcots_branch,
      $ipaddress, $build_log);
} else {
  $log->info("A tag was selected...");
  configure_tags($msf_branch, $csf_branch, $msfcots_branch,
     $ipaddress, $build_log);
}

(...)
```

Como se ve, si en algún momento falla uno de los comandos al servidor de *builds*, se aborta la instalación y pasa al estado ***ABORTED***. Además, todo lo que se va haciendo se vuelca en el fichero de logs del *build* actual.

Hay que tener en cuenta que si se selecciona una rama en el fichero de configuración, se requiere realizar un proceso ligeramente distinto al que se llevaría a cabo si se seleccionase una etiqueta. Por ello se separa en dos funciones.

Una vez se haya realizado esta configuración correctamente, se pasa al estado ***BUILDING_ISOS*** y se ejecuta la función `build_all_packages_and_isos()`. Esta función simplemente ejecuta una orden del *Makefile* existente en el proyecto previamente, que se encarga de realizar estas imágenes ISO. La aplicación se encarga de monitorizar este proceso, y en el caso de que falle en algun momento de pasar al estado ***ABORTED***:

```
(...)
if ( system("ssh -t deliveries\@$ipaddress 'cd src/DEIMOS-MSF/
   msf/scripts/automated-install; make build-all-packages-and-
   isos' >> $build_log") ) {
   return abort_and_return();
}

(...)
```

Una vez esto se haya realizado correctamente, se pasa el estado ***PUBLISHING_ISOS***, en el que se envían las imágenes ISO generadas a la cabina de almacenamiento. Este paso también esta definido en una orden del mismo *Makefile*, por lo que su ejecución es análoga a la anterior:

```
(...)
if ( system("ssh -t deliveries\@$ipaddress 'cd src/DEIMOS-MSF/
   msf/scripts/automated-install; make publish-isos' >>
   $build_log") ) {
   return abort_and_return();
}
(...)
```

Si todo ha ido correctamente se pasará al estado inicial ***BUILT***, y el proceso habrá quedado guardado en el fichero de log especificado al comienzo del *build*.

4.4. Desarrollo y despliegue de *Jenkins*, y migración a *Bamboo*.

En primer lugar, se desarrollará el sistema de integración continua en *Jenkins* debido a que se trata de una herramienta de código abierto, accesible por cualquier usuario. Tras haber realizado un sistema funcional, se realizará su migración a la herramienta corporativa de integración continua *Bamboo*, ya que este sistema permite importar proyectos existentes de *Jenkins*, y ser interpretados de la misma forma.

4.4.1. Desarrollo en Jenkins

Son necesarios ciertos pasos para su despliegue, los cuales están recogidos de [**?**].

En primer lugar se descarga el paquete que proporciona el servicio y se arranca:

```
sudo wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/redhat/jenkins.
    repo

sudo rpm --import https://jenkins-ci.org/redhat/jenkins-ci.org.key

sudo yum install jenkins

sudo service jenkins start
```

Ya con el servicio en funcionamiento, ingresaremos en un navegador la IP del servidor desde el cual hemos arrancado el servicio, y hacia el puerto 8080[10].

Desde la interfaz web instalamos los *plugins* por defecto y configuramos a nuestro gusto el sistema. En ella, también creamos una nueva tarea, de tipo *Pipeline*, la cual se encargará de realizar todos los pasos para el despliegue de nuestro proyecto haciendo uso de *scripts* ya existentes, como ejecutar un *build* a partir de *MSFAdt_build*, configurar la instalación, ejecutar una instalación con *MSFAdt*, etc. Este *Pipeline* tendrá la forma:

```
pipeline {
 agent any
 environment {
  //Ficheros donde se almacena el estado actual de la maquina y nombres de scripts y
      variables
 }
 options {
  skipStagesAfterUnstable()
 }
 stages {

  stage('Configure Build') {
   steps {
    echo 'Configuring build...'
    //Script para configurar el build
   }
  }

  stage('Build') {
   steps {

    echo 'Building...'
    //Script para lanzar el build
    waitUntil {
     //Comprobamos cada cierto tiempo el estado del build, hasta que termine bien o
         mal
    }
    //Si ha ido mal se aborta con error
   }
  }

  stage('Configure, mount ISOs, and configure CI machines') {
   steps{
    echo 'Configuring and mounting ISOs and machines...'
    //Script para configurar ISOs y parametros de las instalaciones.
```

[10]Hay que tener cuidado con el *firewall*, ya que si está habilitado hay que permitir las conexiones TCP por el puerto 8080. También hay que tener en cuenta añadir el usuario *jenkins* al fichero de *sudoers* [Tod19], ya que va a tocar ficheros con permisos de *root* (todas las aplicaciones están desarrolladas con este usuario).

```
    }
   }

   stage('Install') {
    steps {
     echo 'Installing CI machines...'

     //Scripts de configurar e instalar maquinas

     waitUntil {
      //Comprobamos cada cierto tiempo el estado del servidor
     }

     //Esperamos 2 minutos a que se estabilice el estado
     sleep 120

     //Vemos si ha ido mal, y si es asi abortamos

     //Lo mismo para la estacion de trabajo

    }
   }
  }
}
```

Después de configurar el *Pipeline*, se selecciona una hora a la que queramos que se realice el despliegue cada día y el sistema se encarga de hacerlo. Además, este sistema guarda los estados de cada despliegue, el momento en el que ha fallado, el paso que ha fallado, el resultado por consola, las ISOs generadas y también permite enviar un correo de notificación en el caso de que haya ido mal en algún momento. Como se puede ver en el *Pipeline*, añadir nuevos pasos es muy sencillo. Tan solo hay que añadir un nuevo *Stage* en el cual se realice lo deseado.

Con el servicio ya listo y configurado, y con las aplicaciones propias desarrolladas y en funcionamiento, podemos añadir en el *Pipeline* los scripts que harán estas tareas. Estos, que añadiremos bajo el directorio `CI_tools`, serán los encargados de modificar los ficheros de configuración de la plataforma (servidor + estación de trabajo) dedicada para la integración continua, aplicación del *build*, de notificar vía *Skype* el estado de la integración, etc.

Estos scripts serán:

- `configure-build.pl`: configurará la rama de la cual se realizará el próximo *build*.
- `mount_CI_release_and_configure_modetype.pl`: encargado de recoger el nombre de las ISOs recién creadas, montarlas en el correspondiente directorio bajo `/vm/releases`, copiar el nombre de estas ISOs en un fichero `.csv`[11] y por último configurar la versión y el modo de instalación en la plataforma de integración continua.
- `send_skype_message_CI.py`: envía un mensaje por el grupo dedicado de Skype para la integración continua del proyecto.

[11]Esto se hará para permitir un mejor seguimiento del nombre de las ISOs creadas para integración continua, y así, en el momento de que se tengan problemas de espacio disponible en la cabina de almacenamiento, poder borrar estos ficheros en primer lugar.

Stage	**Script**
Configure build branch	*configure-build.pl*
Start build	Llamada HTTP POST a *MS-FAdt_build* *send_skype_message_CI.py* y *send_build_log_by_skype.py*
Configure installation tools	*mount_CI_release_and_configure_modetype.pl*
Install CI machines	Llamada HTTP POST a *MS-FAdt*, *send_skype_message_CI.py* y *send_install_log_by_skype.py*

Tabla 4.3: Scripts ejecutados en cada *Stage* de *Jenkins*.

- `send_install_log_by_skype.py`: envía el fichero de resultados completo de la instalación del servidor o de la estación de trabajo al grupo de Skype dedicado para la integración continua.
- `send_build_log_by_skype.py`: análogo al anterior script, salvo que en este se envía el fichero de resultados del *build*.

El *Stage* en el cual se ejecutan estos scripts se muestra en la tabla 4.3.

4.4.2. Migración a *Bamboo*

Una vez configurado *Jenkins*, y con él funcionando correctamente, procedemos a migrar este sistema a *Bamboo* (concretamente a la versión 6.5 instalada en los servidores corporativos) que, como se ha dicho anteriormente, es la herramienta corporativa [Atl19]:

- Se importará el proyecto *Jenkins*, y se reconfigurará el *Pipeline* en forma de *Stages* y *Jobs*. Para importar el proyecto, son necesarios varios pasos explicados en [Atl19], y que no se detallarán en este documento.
- Una vez importado el proyecto en *Bamboo* este se distribuye en varias partes:
  1. Plan: el proyecto de integración continua al completo.
  2. *Stages*: se definen los pasos mas generales para el despliegue, en este caso *build* e instalación.
  3. *Jobs*: aquí se definen los pasos que hay que realizar en cada *Stage*. Estos trabajos se pueden llevar a cabo en paralelo, pero como en este sistema interesa hacerlo todo paso a paso, se creará un trabajo por *Stage*.
  4. *Tasks*: las tareas que realizará cada trabajo, por ejemplo ejecutar varios *scripts* en orden (configurar rama, lanzar *build*, etc.).

5. Variables de entorno: las variables de entorno se configuran para cada plan. En ellas pondremos las necesarias para la ejecución de las llamadas, scripts, etc. En concreto, al estar esta herramienta corporativa en un servidor externo, las tareas serán ejecuciones de *script* por *SSH*, donde la máquina destino será la máquina virtual utilizada para el despliegue de las aplicaciones *MSFADt* y *MSFAdt_build*.
6. *Triggers*: sistema que se utiliza para configurar las horas y el día de despliegue. En este caso, se seleccionará los días de diario, ya que los fines de semana, al no haber cambios en el código, no es necesario su despliegue.
7. *Notifications*: notificaciones a ciertas personas del proyecto, entre las que se incluye el desarrollador de este trabajo final de grado, en el caso de que el despliegue haya ido correctamente, en que *stage* ha fallado, por qué, etc.

Se configuran los *Stages* tal y como estaban en *Jenkins* y ya se dispondrá del sistema de integración continua desplegado y funcionando:

1. ***Build***: se configura la rama de la que queremos recoger el código desarrollado y, por tanto, construir las imágenes ISO. Se llama a la aplicación *MSFAdt_build* mediante una petición *HTTP POST*, para que comience a construir las imágenes. Se comprueba que la tarea de creación de imágenes se ha producido correctamente, y en caso contrario se notifica y se para el sistema de integración.
2. ***Installation***: se configura la aplicación *MSFAdt* para que instale las imágenes ISO recién generadas en el servidor y estación de trabajo dedicados a la integración continua. Se llama a la aplicación *MSFAdt* para que comience la instalación. Se comprueba si el servidor se instalado sin errores y en caso contrario se notifica y se pone como fallida la integración. Se comprueba si la estación de trabajo se ha instalado sin errores y en caso contrario también se notifica y se pone como fallida la integración.

Una de las razones por las que migramos este sistema a *Bamboo* es debido a que existe un proyecto de pruebas de calidad de código con *SonarQube* [Son19]. Por tanto, se puede añadir este paso al proyecto de integración continua, y así automatizar la mayor cantidad de pruebas posibles. Con todo esto configurado, el sistema de integración continua queda desplegado y en funcionamiento.

4.5. *MSFAdt_web*

De manera adicional, también interesa tener una interfaz gráfica desde la cual se pueda configurar todas las plataformas (no sólo las dedicadas a la integración continua) y poder realizar sus instalaciones de forma completamente desatendida. Además se podría visualizar el estado de todas estas plataformas (incluida la de integración continua) con un solo golpe de vista. Esta interfaz de usuario amigable permitirá gestionar las dos aplicaciones *MSFAdt* y *MSFAdt_build* al gusto del administrador. Para ello se utiliza el módulo `Mojolicious` de *Perl*, que da un servicio web en tiempo real adecuado a las necesidades del proyecto debido a su amplio catálogo de herramientas de

Figura 4.15: Entorno de trabajo de la aplicación web creado por *Mojolicious*.

```
[root@pxe-platform msfadt_web]# ll
total 4
drwxr-xr-x. 3 root root 45 Mar 21 06:05 lib
-rw-r--r--. 1 root root 62 Mar 21 06:05 msfadt_web.conf
drwxr-xr-x. 2 root root 24 Mar 21 06:05 public
drwxr-xr-x. 2 root root 24 Mar 21 06:05 script
drwxr-xr-x. 2 root root 21 Mar 21 06:05 t
drwxr-xr-x. 4 root root 36 Mar 21 06:05 templates
```

Figura 4.16: Entorno de trabajo modificado para *MSFAdt_web*.

```
[root@pxe-platform msfadt_web]# ll
total 8
drwxr-xr-x.  2 root root  21 Feb 15 06:42 doc
drwxr-xr-x.  2 root root  43 Mar  4 05:33 etc
drwxr-xr-x.  3 root root  45 Mar 12 12:19 lib
drwxr-xr-x.  2 root root  65 Feb 15 08:53 log
-rw-r--r--.  1 root root  63 Feb 15 06:42 msfadt_web.conf
drwxr-xr-x.  2 root root  57 Mar 11 21:12 public
-rw-r--r--.  1 root root  69 Feb 15 06:42 README
drwxr-xr-x.  2 root root  69 Mar 12 12:19 script
drwxr-xr-x.  2 root root  21 Feb 15 06:42 t
drwxr-xr-x. 12 root root 196 Mar 12 12:19 templates
drwxr-xr-x.  2 root root  71 Mar 13 08:59 tools
```

desarrollo. Permite la utilización de plantillas escritas en *Perl* para desarrollar código *HTML*, *CSS*, *Javascript*, *AJAX*... Herramientas necesarias para el desarrollo de una buena interfaz web [Seb19].

Este módulo `Mojolicious` [Pav09] permite generar una aplicación completamente funcional a partir de su herramienta `mojo`. Habilita, a partir de una aplicación inicial, su modificación a *posteriori* para adecuarse a las funcionalidades del usuario. Se ejecuta en el directorio raíz de *MSFAdt*:

```
[root@pxe-platform automatic-deployment]# mojo generate app Msfadt_web
```

Así se genera una primera aplicación predefinida bajo el directorio `msfadt_web`, que crea una estructura de directorios como la que se indica en la figura 4.15.

A partir de esta, se desarrolla la interfaz web y se añaden los directorios necesarios como se especifica en la figura 4.16. Los directorios añadidos en este caso serán `doc`, `etc`, `log` y `tools`, que a continuación se comenta su contenido:

- Directorio `doc`: este directorio se crea posteriormente, es decir, no viene por defecto al generar la aplicación. En él se almacenan los manuales de usuario y

funcionamiento de la interfaz web *MSFAdt_web*.

- Directorio `etc`: también se trata de un directorio nuevo, que no viene definido por la aplicación original, y en el que se va a almacenar los ficheros de configuración necesarios por la aplicación. En concreto estarán los ficheros `api.yaml` y `log4perl.yaml`, que (al igual que en las anteriores aplicaciones) albergan la dirección IP y el puerto por el que arrancar el servicio web (y por el que se podrá acceder a él a través de un navegador), y la configuración del módulo `Log4Perl` encargado de generar los ficheros *log* del sistema.

- Directorio `lib`: de nuevo los ficheros incluidos en este directorio son el "corazón" de la aplicación, y los encargados de generar las páginas HTML (con ayuda de unas plantillas) que serán visibles desde un navegador web y conectarlas con la ejecución de las peticiones HTTP de las anteriores aplicaciones, o de modificación de configuraciones. Estas funciones son muy variadas, y permitirán gran cantidad de operaciones con las anteriores aplicaciones, como modificar la configuración de X máquinas a la vez, lanzar su instalación, programar instalaciones, etc. Se verá en profundidad en los siguientes apartados.

- Directorio `log`: al igual que en las anteriores aplicaciones, este directorio almacena los ficheros *logs* de la aplicación. En ellos se recogen todos los accesos a la interfaz web, inicios de sesión, operaciones realizadas, etc.

- Directorio `public`: en él se almacenan las páginas web estáticas de la interfaz. Es decir, las páginas que no se ven modificadas con relativa frecuencia, si no que siguen un carácter estático. Estas son la página en la que se muestra una breve ayuda de funcionamiento de la interfaz web y las páginas de errores generadas por la aplicación. Estas páginas de errores son específicas para cada máquina en caso de que esta se haya instalado con errores.

- Directorio `script`: se encuentran los *scripts* que inician y detienen la aplicación web.

- Directorio `templates`: se almacenan todas las plantillas escritas en Perl + HTML, con las cuales se generan las diversas páginas a mostrar en la interfaz web. Mediante estas plantillas se consigue que la interfaz web sea interactiva, y se modifique conforme un fichero de configuración de las anteriores aplicaciones se vea alterado.

- Directorio `tools`: igual que en las anteriores aplicaciones, en este directorio se almacenan los *scripts* utilizados por la aplicación para realizar diversas tareas específicas.

Debido a que varios directorios de esta aplicación presentan las mismas funcionalidades y estructuras que en las aplicaciones anteriores, se pasará a comentar lo verdaderamente distinto de esta, alojado en los directorios `lib`, `public` y `templates`.

Figura 4.17: Página web *help.html*.

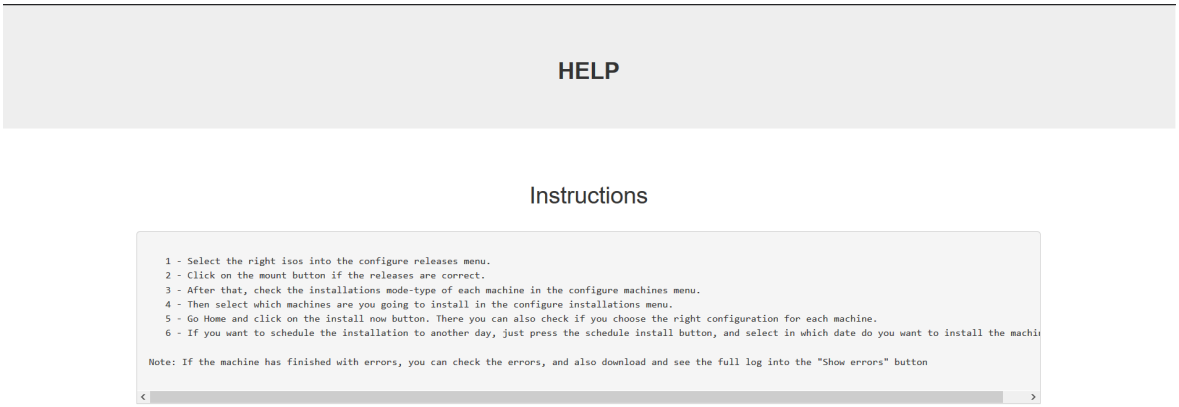


Figura 4.18: Página web *errors.html*.

4.5.1. Directorio *public*

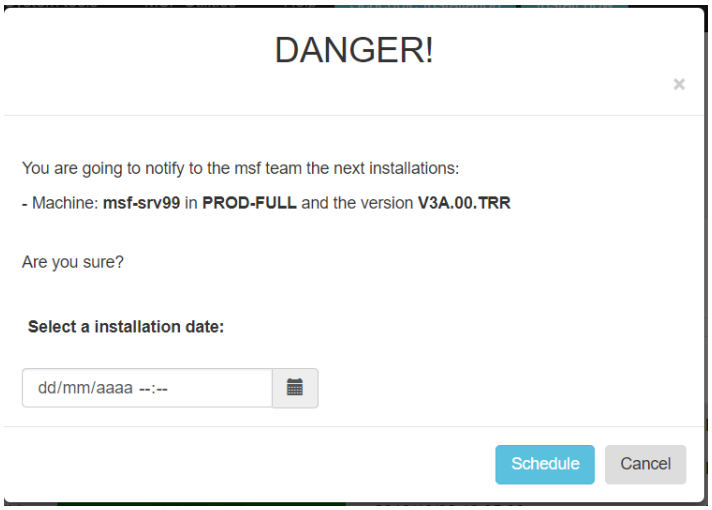
En este directorio se almacenan las páginas estáticas de la interfaz web. Entre ellas se encuentran las páginas de errores de las máquinas (generadas mediante una plantilla alojada en el directorio `templates`, explicado en la sección 4.5.2), mostradas en la figura 4.18. En esta página se muestra la información del nombre de la máquina instalada, fecha de la instalación, versión instalada, número de *build* de la imagen ISO instalada, modo y tipo de la instalación y los errores producidos durante la misma. Esta página da opción a descargar el fichero *log* al completo (donde se muestran todas las acciones ocurridas durante la instalación) y ver este mismo fichero en línea.

También se almacena la página web de ayuda `help.html`, en la cual se detallan una serie de pasos para la utilización de la interfaz web, como se puede ver en la figura 4.17.

Figura 4.19: Directorio *templates*.

```
[root@localhost msfadt_web]# ll templates/
total 44
drwxr-xr-x. 2 root root 4096 may 28 14:48 api
drwxr-xr-x. 2 root root 4096 may 28 14:48 build
drwxr-xr-x. 2 root root 4096 may 28 14:48 configuremachines
-rw-r--r--. 1 root root 3752 may 28 14:48 errors.html.tt
drwxr-xr-x. 2 root root 4096 may 28 14:48 global
drwxr-xr-x. 2 root root 4096 may 28 14:48 installations
drwxr-xr-x. 2 root root 4096 may 28 14:48 layouts
drwxr-xr-x. 2 root root 4096 may 28 14:48 releases
drwxr-xr-x. 2 root root 4096 may 28 14:48 wiki
```

Figura 4.20: Formulario para programar las instalaciones.



4.5.2. Directorio *templates*

En el directorio `templates` mostrado en la figura 4.19 se almacenan todas las plantillas Perl+HTML que generan las páginas web dinámicas de la interfaz. Estas páginas están divididas según sus funciones en las aplicaciones de *MSFAdt* y *MSFAdt_build*, y adicionalmente una plantilla para el *layout* de cada página. Este *layout* añade las opciones de inicio de sesión y navegación entre las páginas de las interfaz.

Plantilla *layouts*

Esta plantilla se incluye al comienzo de todas las plantillas definidas en la interfaz web. En ella se definen las cabeceras `<head>` HTML [AR19c], además del formulario de *login* a la plataforma, y los botones que permiten iniciar las instalaciones.

En concreto, se definen las siguientes características:

- Cabeceras HTML.
- *Scripts* globales de JavaScript [AR19d] y peticiones AJAX.
- Estilo CSS [AR19b].
- Botón que permite instalar las plataformas configuradas. Se muestra únicamente en el caso de que exista alguna máquina en el estado *CONFIGURED*.
- Botón que permite realizar un *build* y posteriormente una instalación con las imágenes ISO generadas en ese *build*. Esto permite que el sistema se encargue de los pasos intermedios entre *build* e instalaciones, y por tanto automatizar del todo la herramienta.
- Programar las instalaciones para una fecha concreta, pasando del estado *CONFIGURED* al estado *SCHEDULED* a las máquinas que se encuentren en este estado previamente. Esto muestra una ventana, mostrada en la figura 4.20, que permite seleccionar la fecha y hora de las instalaciones.
- Botón para cancelar las instalaciones programadas. Permite abortar las instalaciones de las máquinas que se encuentren en el estado *SCHEDULED*.
- Botón para instalar las máquinas en el estado *SCHEDULED* inmediatamente, sin esperar a la fecha destinada para ello.

Plantilla *api*

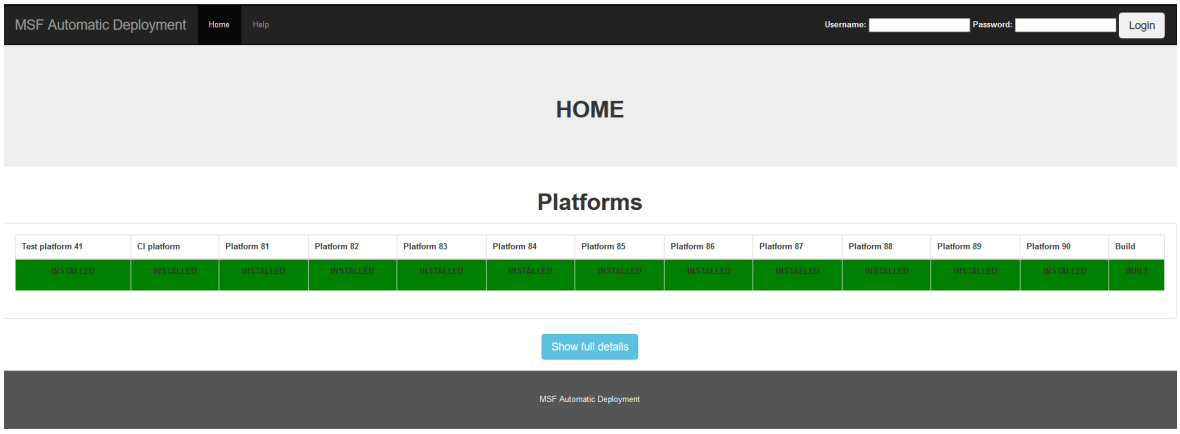
Con esta plantilla se crea la página web hogar de la aplicación *MSFAdt_web* como se puede ver en la figura 4.21. En ella se muestran todas las plataformas definidas en los ficheros de configuración *MSFAdt*, así como su estado, fecha de instalación, modo y tipo, etc. Estas plataformas están compuestas por un servidor y varias estaciones de trabajo, por lo que el estado de estos también puede ser visto por separado.

Únicamente esta página puede ser vista (además de la página de ayuda) sin estar dado de alta en el sistema. Por tanto, para poder acceder a más páginas de configuración o información es necesario ingresar en el sistema con un usuario y contraseña que se crean en una base de datos (concretamente en un fichero de configuración).

Es también desde esta página donde se puede ver si la plataforma (o servidor/estación de trabajo) se ha instalado correctamente o con errores. En el caso de que los haya, se muestra un botón que redirige automáticamente a una página estática como la que se puede ver en la figura 4.18, donde se muestran los errores surgidos durante la instalación.

Estas plantillas, como se ha comentado en anteriores ocasiones, están desarrolladas en Perl + HTML + JavaScript + AJAX. Su diseño es similar a una página común desarrollada con HTML + JavaScript + AJAX, salvo que en este caso se puede utilizar Perl para añadir condicionantes o cargar variables por parte de la aplicación Mojolicious. Todas las páginas web de este sistema se basan en un modelo de tablas, en las

Figura 4.21: Página web *home.html*.



cuales se muestra determinada información alojada en los ficheros de configuración de las anteriores aplicaciones. La mayoría de ellas siguen la siguiente forma:

1. Carga del *layout*: se inserta la plantilla *layout* al comienzo del fichero (ver sección 4.5.2).

```
% layout 'default', machines => $machines, auth => $auth,
   user => $user, page => 'home', run => $run, buildrun =>
   $buildrun, testsrv => $testsrv, testws => $testws, CIsrv
    => $CIsrv, CIws => $CIws;
% title 'MSFAdt';
```

2. Título de la página.

```
<div class="jumbotron">
 <h2>HOME</h2>
</div>
```

3. Tabla de visión rápida del estado de las plataformas: en esta tabla simplificada se puede ver el estado de la plataforma en cuestión. Estas plataformas, como bien se ha dicho, están compuestas por 1 servidor y 2 estaciones de trabajo. Si uno de ellos está siendo instalado, o ha sido instalado con errores o abortada su instalación, se mostrará en estas celdas.

```
<h2>Platforms</h2>
<div class="panel panel-default">
 <div class="panel-body">
  <table id="mytable3" class="table table-bordered" width="
     100%">
   <thead>
    <tr>
     % foreach my $m ( sort keys %{$machines}) {
      % if ( $machines->{$m}{'type'} eq "server" ) {
```

```
          % if ( $m eq $testsrv ) {
           <th>Test platform <%= substr($m,7,9) %></th>
          % } elsif ( $m eq $CIsrv ) {
           <th>CI platform </th>
          % } else {
           <th>Platform <%= substr($m,7,9) %></th>
          % }
         % }
        % }
        <th>Build</th>
       </tr>
      </thead>
      <tbody>
       <tr>
        % foreach my $m (sort keys %{$machines}) {
         % if ( $machines->{$m}{'type'} eq "server" ) {
          % my $platform = substr($m,7,9);
           <td style="font-weight: bold;" id="<%= $platform
              %>-color"><p id="<%= $platform %>-state"></p></
              td>
         % }
        % }
        <td id="build-color" style="font-weight: bold; text-
           align: center;"><p id="build-state"></p></td>
       </tr>
      </tbody>
     </table>
    </div>
   </div>
  </div>
```

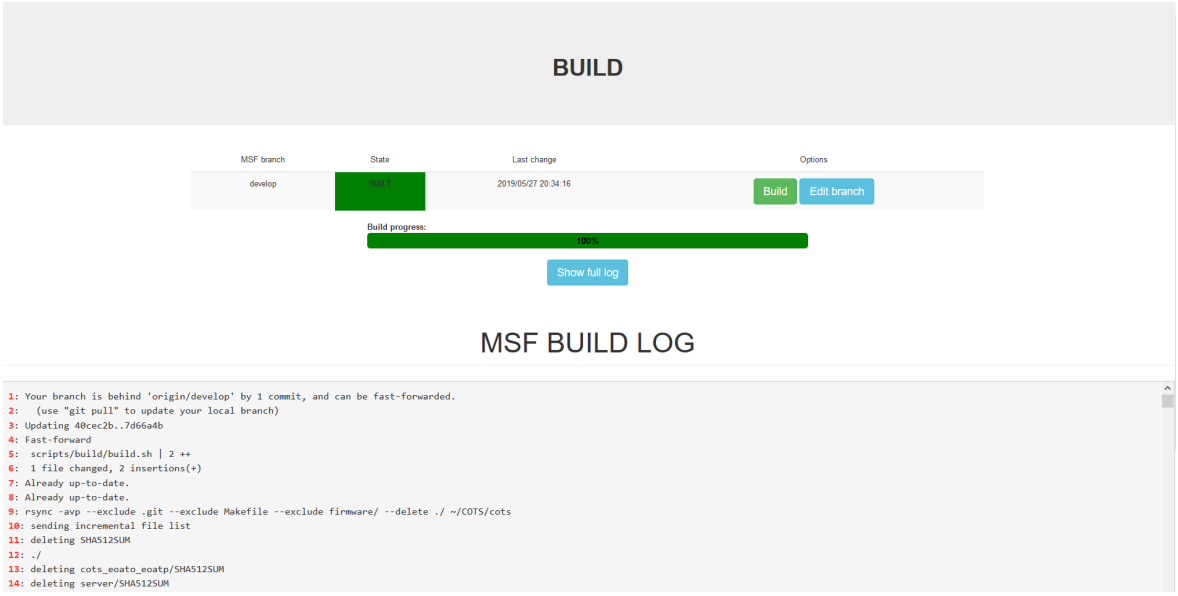
4. Por último se muestra otra tabla con los detalles específicos de cada máquina y el pie de página.

Todas las variables utilizadas en este código (y por tanto en las demás plantillas también) son variables cargadas desde la aplicación de Mojolicious, y que son generadas por funciones desarrolladas en el directorio `lib` que se detallará su utilidad en la sección 4.5.3. Algunas de estas variables son pasadas por AJAX [AR19a] a la interfaz web, permitiendo así una modificación de determinadas celdas de las tablas sin tener que refrescar completamente la página. Estas funciones AJAX se implementan por parte del servidor en las funciones definidas en `lib`, y por parte del cliente en las plantillas añadiendo el código:

```
<script type="text/javascript">

 $(function(){
  setInterval(function(){
   $.ajax({
    type:'GET',
    url:'/ajax/get-ready-state',
```

Figura 4.22: Página web *build.html*.



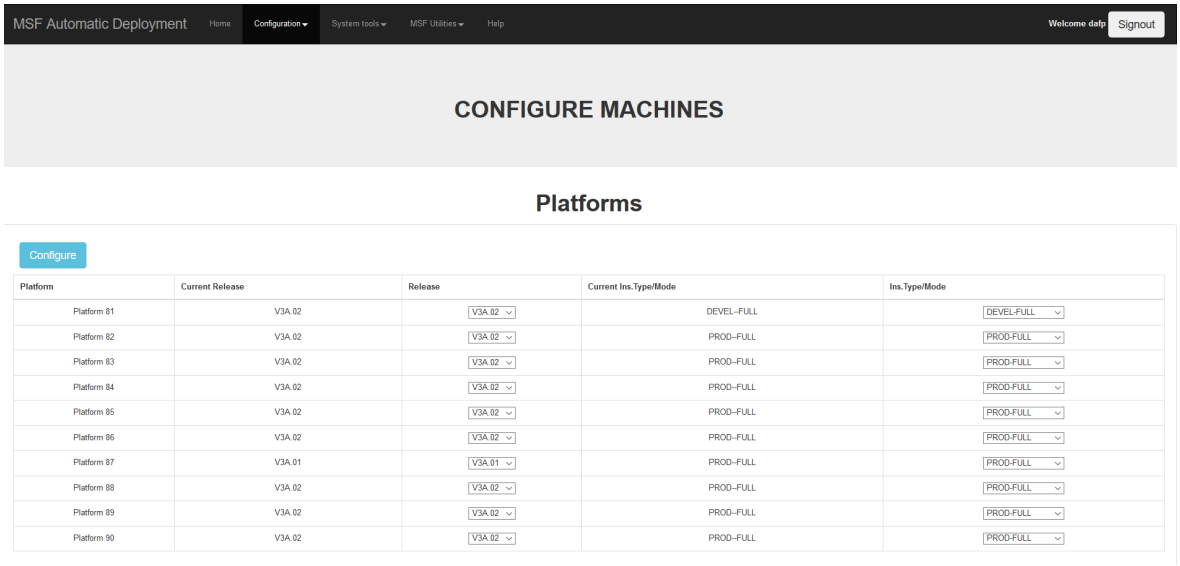

```
8       dataType: 'json',
9       success: function(json){
10       if ( json.state ) {
11         $('#install-button').hide();
12         $('#schedule-button').hide();
13       }
14      }
15     });
16    }, 1000);
17  });
18 </script>
```

Plantilla *build*

En la página generada por esta plantilla se muestran las opciones relacionadas con la aplicación *MSFAdt_build*. En ella se puede seleccionar la rama desde la cual se realizará el *build*, lanzar uno nuevo y seguir su estado a tiempo real en la ventana inferior, mostrada en la figura 4.22. Cuando se esta realizando un *build*, el estado del mismo se va escribiendo en un fichero (como se ha dicho en la aplicación *MSFAdt_build*). Lo que hace esta interfaz web es leer cada cierto tiempo (mediante una petición AJAX) el estado de dicho fichero, y de esta forma poder mostrar por la interfaz web, línea por línea, lo que está sucediendo. Además se ha añadido una barra de progreso estimada en función del número de lineas que se han escrito en el *log* (este número de lineas esta distribuido de forma Gaussiana con una varianza muy pequeña, por lo que casi siempre tendrá el mismo valor), para darle mejor visibilidad al estado de las operaciones.

Una vez haya finalizado el proceso, en los *logs* se puede ver resaltado en rojo las dos nuevas imágenes ISO generadas.

Figura 4.23: Página web *configuremachines.html*.



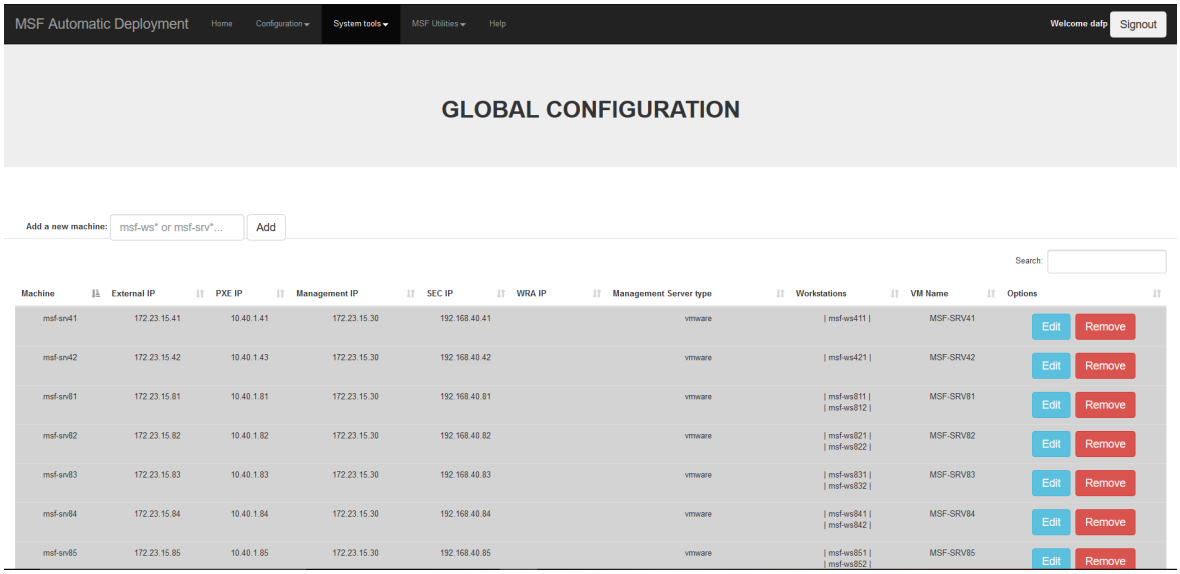
Plantilla *configuremachines*

Esta plantilla genera la página dedicada para la configuración del modo, tipo y versión a instalar en las plataformas del proyecto. Sigue la estética de tablas, y consta de celdas de opciones para escoger las configuraciones oportunas para cada plataforma. Adicionalmente se puede modificar la configuración de cada máquina individual si es necesario.

Al modificar estos valores existe el botón *Configure*, que lo que hace es llamar a una función definida bajo el directorio `lib` que lee estos valores y los modifica en el fichero de configuración `machines.yaml` perteneciente a la aplicación *MSFAdt*.

```
<form action="/select-platforms-config" id="platforms-config"
   method="post">
...
<select name="release-platform<%= $platform %>">
 % foreach my $v (keys %{$releases}) {
  % if ( $v eq $machines->{$m}{'installation'}{'release'} ) {
   <option value="<%= $v %>" selected><%= $v %></option>
  % } else {
   <option value="<%= $v %>"><%= $v %></option>
  % }
 % }
</select>
...
<select name="modetype-platform<%= $platform %>">
 % if ( $type eq 'PROD' ) {
  <option value="production-full" selected> PROD-FULL </option>
 % } else {
  <option value="production-full"> PROD-FULL </option>
```

Figura 4.24: Página web *globalconfig.html*.




```
18    % }
19    % if ( $type eq 'DEVEL' ) {
20     <option value="development-full" selected> DEVEL-FULL </
          option>
21    % } else {
22     <option value="development-full" > DEVEL-FULL </option>
23    % }
24    <option value="production-update"> PROD-UPDATE </option>
25    <option value="development-update"> DEVEL-UPDATE </option>
26  </select>
27  ...
28  <input type="Submit" class="btn btn-info" style="margin: 8px;"
       value="Configure">
```

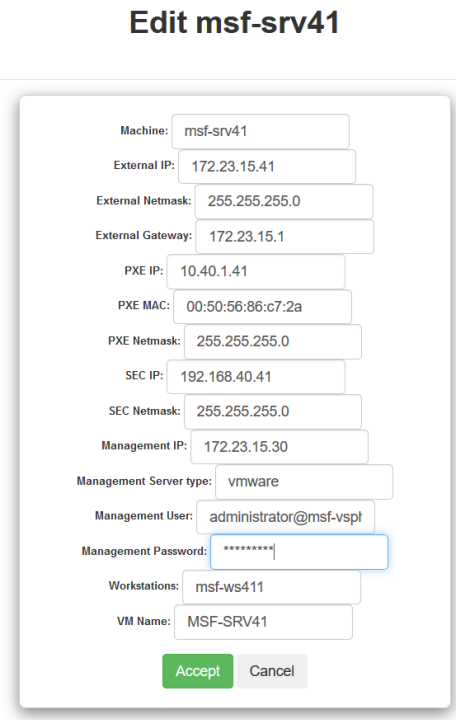
Plantilla *global*

La página que genera la plantilla global (fig. 4.24) sigue la misma estructura de tablas que en los casos anteriores. En esta página se pueden modificar los parámetros más sensibles de configuración de las máquinas, además de poder eliminar, editar y añadir nuevas máquinas. Para eliminar y añadir una máquina se utiliza una función asociada al *click* del botón *delete* o *add*, que pasa como argumento el nombre de la máquina a las funciones de `lib`, y estas se encargan del resto. Para editar se sigue el mismo procedimiento, salvo porque en este caso se utiliza un formulario para ello.

Los parámetros a modificar son: nombre de la máquina, dirección IP, máscara y ruta por defecto para la red de DEIMOS, dirección IP, dirección MAC y máscara de la tarjeta de red conectada a la red PXE, otras configuraciones IP para el proyecto y configuraciones para su mantenimiento (fig. 4.25).

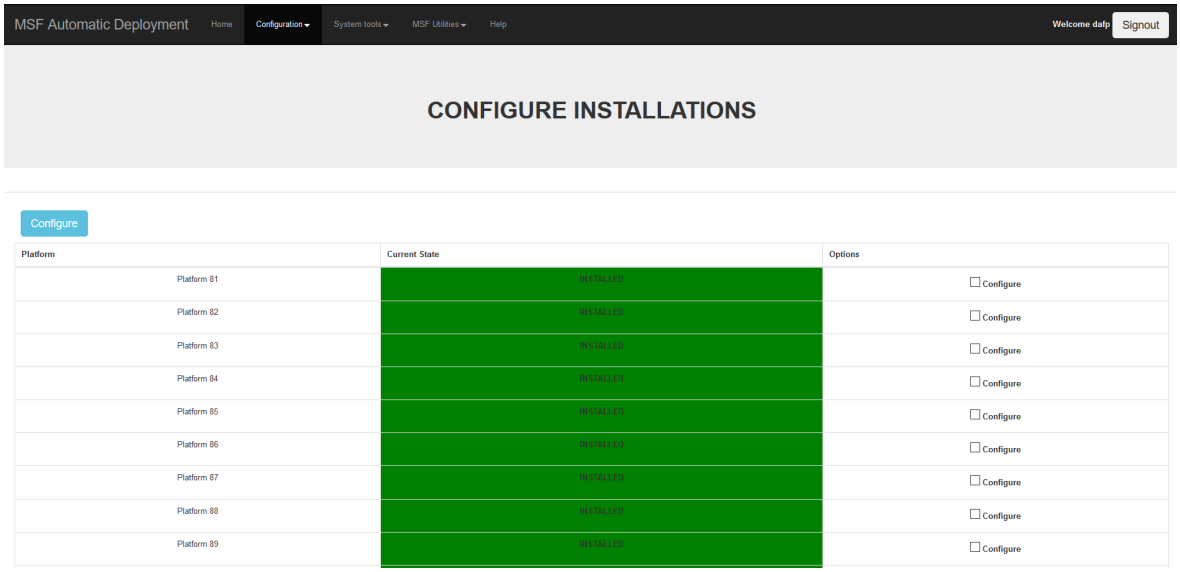
El código empleado para crear este formulario es:

Figura 4.25: Formulario para editar una máquina existente.




```
<form style="text-align: center; display: inline-block;" action
   ="/global-config/<%= $m %>/send" method="post">
...
 <div class="modal-content" style="width: 70%; text-align:
    center; margin: auto; padding: 3%;">
  <div>
   <label>Machine: </label>
   <input class="form-control" type="text" name="machine_new"
      value="<%= $m %>">
  </div>
  <div>
   <label>External IP: </label>
   <input class="form-control" type="text" name="
      external_ipaddress" value="<%= $machines->{$m}{'network
      '}{'external'}{'ipaddress'} %>">
  </div>
  <div>
   <label>External Netmask: </label>
   <input class="form-control" type="text" name="
      external_netmask" value="<%= $machines->{$m}{'network'}{'
      external'}{'netmask'} %>">
  </div>
  <div>
   <label>External Gateway: </label>
   <input class="form-control" type="text" name="
```

Figura 4.26: Página web *configureinstallations.html*.




```
        external_gateway" value="<%= $machines->{$m}{'network'}{'
        external'}{'gateway'} %>">
19    </div>
20  ...
21    <div>
22     <button class="btn btn-success">Accept</button>
23     <button type="button" class="btn btn-secondary" data-dismiss
          ="modal">Cancel</button>
24    </div>
25  </form>
```

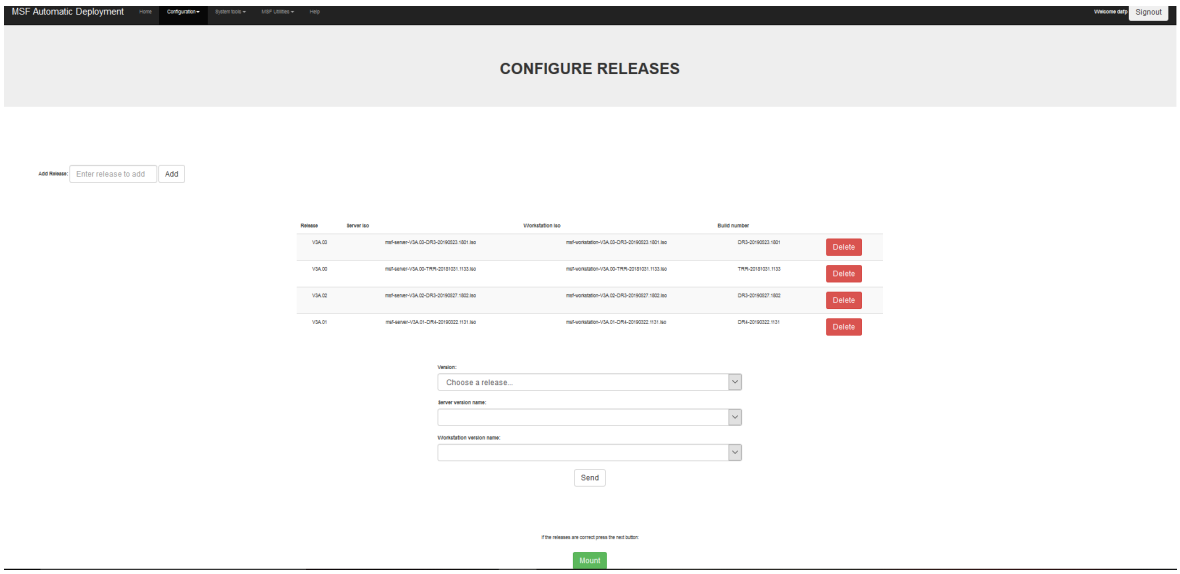
Plantilla *installations*

Esta plantilla genera una página web que permite seleccionar qué máquinas van a ser reinstaladas. Como se puede ver en la figura 4.26 se muestran en primer lugar los estados de las plataformas (más abajo se puede configurar cada máquina por separado) seguidos de un *check* que permite seleccionar las plataformas a preparar.

Configurar una máquina se entiende, en este aspecto, a que el estado de la máquina (en este caso máquinas) pasen de *INSTALLED* a *CONFIGURED*, y por tanto que se configure el servicio TFTP y DHCP para que en el próximo reinicio de la máquina pueda arrancar por PXE y por tanto ser reinstalado (como se ha explicado en profundidad en la sección 4.2).

El botón *configure* lo que hace es llamar a una función de `lib`, que se encarga de ejecutar el *script* `configure-machine.pl` de la aplicación *MSFAdt* con las plataformas que tengan marcada la casilla en la web.

Figura 4.27: Página web *releases.html*.



El estado de cada plataforma (y por tanto de cada máquina), al igual que en la página inicial, se recoge del fichero de configuración de cada máquina mediante peticiones AJAX. Estas peticiones se realizan periódicamente, y son gestionadas por otra función definida en `lib`.

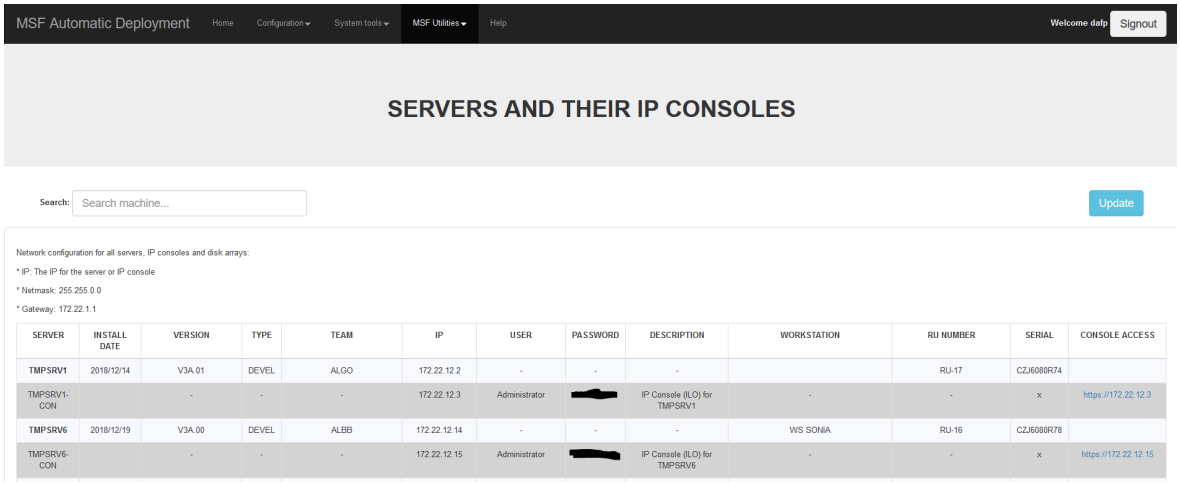
Plantilla *releases*

Esta plantilla genera una pagina para la configuración las diferentes versiones (fig. 4.27), y por tanto imágenes ISO asociadas, montadas en el servidor en el cual están desplegadas estas aplicaciones. Desde aquí se pueden seleccionar, para cada versión, las imágenes ISO a montar en el punto de montaje desde el cual las máquinas, por PXE, cogerán los ficheros necesarios para su instalación y despliegue.

Existen dos pasos para el montaje de estas imágenes. En primer lugar se seleccionan los nombres de las imágenes y la versión a modificar en los desplegables. Posteriormente está el botón *send*, que modifica el fichero de configuración `releases.conf` (asociado a la aplicación *MSFAdt*), indicando las nuevas imágenes ISO para la versión seleccionada. Seguidamente está el botón *mount*, que realiza el montaje de todas las imágenes ISO de cada versión en los directorios correspondientes. Tras esto, cada celda correspondiente a una imagen ISO se coloreará de verde si el montaje ha sido satisfactorio o de rojo si no se ha podido realizar correctamente.

Además, desde esta página se pueden añadir nuevas versiones del *software* y eliminarlas. El hecho de poder añadir nuevas versiones es muy útil, debido de que cada cierto tiempo el *software* va evolucionando, y el número de estas versiones crece en un periodo muy corto. Por tanto el poder añadir nuevas versiones de manera tan sencilla es un punto muy positivo a tener en cuenta en esta página.

Figura 4.28: Página web *wiki.html*.



Plantilla *wiki*

Esta plantilla genera una copia de la *wiki* de la empresa alojada en el servidor *Redmine* mencionado anteriormente. Se trata de una página tan solo informativa a pesar de que puede modificarse desde la propia interfaz web. A efectos del funcionamiento de la aplicación, no aporta nada relevante.

4.5.3. Directorio *lib*

Este directorio (fig. 4.29) almacena los paquetes de Perl que contienen las funciones que se realizan mediante peticiones AJAX, envío de formularios, *clicks* en botones, etc. en la interfaz web. En concreto, en el fichero `MSFAdt_web.pm` se definen, además de los datos de usuarios y contraseñas del sistema, las rutas en las cuales se encuentran las anteriores páginas mencionadas (generadas por las plantillas).

Respecto al directorio `Controller`, en él se encuentran todos los paquetes en los que se almacenan las funciones ejecutadas por peticiones AJAX, clicks en botones, envío de formularios, generación de las páginas a partir de las plantillas, etc. En el directorio `Functions` se encuentran los paquetes en los que están las funciones adicionales para el sistema, como carga de variables almacenadas en los ficheros de configuración de las aplicaciones, generación de las páginas estáticas a partir de plantillas, modificación de configuraciones, etc.

Paquete *MSFAdt_web.pm*.

Se almacenan, además de los datos de usuarios y sus contraseñas para acceder al sistema, las rutas de la web, así cómo las funciones que se ejecutan al recibir una petición de cada ruta. Es decir, cuando se recibe una petición a una determinada ruta, este fichero establece una relación entre ruta y paquete + función a ejecutar. Esta función, partiendo de una plantilla alojada en el directorio `templates`, genera la pági-

Figura 4.29: Directorio *lib*.

```
[root@pxe-platform msfadt_web]# ll -R lib/
lib/:
total 12
drwxr-xr-x. 4 root root 4096 may 28 14:48 Msfadt_web
-rw-r--r--. 1 root root 5613 may 28 14:48 Msfadt_web.pm
lib/Msfadt_web:
total 8
drwxr-xr-x. 2 root root 4096 may 28 14:48 Controller
drwxr-xr-x. 2 root root 4096 may 28 14:48 Functions
lib/Msfadt_web/Controller:
total 60
-rw-r--r--. 1 root root 8129 may 28 14:48 Ajax.pm
-rw-r--r--. 1 root root 8477 may 28 14:48 Api.pm
-rw-r--r--. 1 root root 1570 may 28 14:48 Build.pm
-rw-r--r--. 1 root root 3557 may 28 14:48 Configuremachines.pm
-rw-r--r--. 1 root root 6802 may 28 14:48 Global.pm
-rw-r--r--. 1 root root 4250 may 28 14:48 Installations.pm
-rw-r--r--. 1 root root 3001 may 28 14:48 Releases.pm
-rw-r--r--. 1 root root 1640 may 28 14:48 Wiki.pm
lib/Msfadt_web/Functions:
total 20
-rw-r--r--. 1 root root 6926 may 28 14:48 RedmineFunctions.pm
-rw-r--r--. 1 root root 1429 may 28 14:48 ReleasesConfiguration
   .pm
-rw-r--r--. 1 root root 3095 may 28 14:48 StaticPage.pm
-rw-r--r--. 1 root root 3022 may 28 14:48 SystemConfiguration.
   pm
```

na web pasándola como argumentos variables de configuración cargadas anteriormente.

Algunas de las rutas definidas en este fichero:

```
1  ...
2
3  # Normal route to controller
4  $r->get('/')->to('api#msfadt');
5
6  # Login
7  $r->post('/login')->to('api#create_session');
8
9  # sends the log file to the client
10 $r->get('send-log')->to('api#send_log');
11
12 # Ajax get machine states
13 $r->get('/ajax/get-machine-states/:machine')->to('ajax#
      get_machine_states');
```

```
14
15  # Ajax get platform states
16  $r->get('/ajax/get-platform-state/:platform')->to('ajax#
        get_platform_state');
17
18  # Get build state
19  $r->get('/ajax/get-build-state')->to('ajax#get_build_state');
20
21  # Get CI state
22  $r->get('/ajax/get-CI-state')->to('ajax#get_CI_state');
23
24  ######################################################
25  # members only area
26  ######################################################
27
28  my $auth = $r->route('/')->over(authenticated => 1);
29
30  # Normal route to configuremachines
31  $auth->get('configure-machines')->to('configuremachines#home');
32
33  # Get via AJAX if there are any configured machine
34  $auth->get('/ajax/get-ready-state')->to('ajax#get_ready_state')
        ;
35
36  ...
```

Paquete *Ajax.pm.*

En este paquete se definen las funciones utilizadas para el envío de respuestas AJAX. En concreto se definen las funciones:

- Función que lee el *log* actual del *build*, y envía su contenido.
- Función que lee las imágenes ISO asociadas a una versión concreta y las envía recogidas en una variable (tanto para servidor como para estación de trabajo).
- Función que devuelve los estados de las plataformas y máquinas, leyendo estos de los ficheros de configuración `run` enunciados en la sección 4.2.
- Función que devuelve el estado del *build*, leyendo el estado del fichero `run` de la aplicación *MSFAdt_build*.

Paquete *Api.pm.*

Paquete destinado a la página principal de la interfaz web. En el se definen las funciones dedicadas a:

- Descargar el *log* de instalación.
- Iniciar y cerrar una sesión en el sistema (con sus respectivos usuario y contraseña).

- Programar la instalación de una o varias máquinas. En este caso se hace uso del script `schedule-machine.pl` de la aplicación *MSFAdt*. También se define la función para cancelar esta programación de instalaciones.
- Funciones para la instalación de las máquinas y para la función de realizar el *build* e instalación conjuntamente.
- Funciones que notifican al equipo de programadores de la programación de las instalaciones (envía un correo automáticamente y un mensaje de Skype).

Paquete *Build.pm.*

En este paquete se definen las funciones para configurar la rama/etiqueta de la que se recogerá el código para generar las nuevas imágenes ISO. En este caso se modifica el fichero de configuración `build-config.yaml`, explicado en detalle en la sección 4.3. También se define la función para arrancar la aplicación *MSFAdt_build* ejecutando el *script* `start-build.pl`.

Paquete *Configuremachines.pm.*

Paquete en el que se definen las funciones dedicadas a:

- Configurar modo y tipo de una máquina en concreto, modificando el fichero `machines.yaml`.
- Configurar versión a instalar en una máquina en concreto, modificando el mismo fichero que en el caso anterior.
- Configurar modo, tipo y versión para una o varias plataformas (servidor + estación de trabajo). En este caso se modifica el fichero `machines.yaml` para todas las máquinas pertenecientes a las plataformas a modificar.

Paquete *Global.pm.*

Paquete en el que se definen las funciones dedicadas a la modificación de parámetros específicos de cada máquina. Como se ha visto en la sección 4.24, en esta página se pueden modificar parámetros específicos de cada máquina, como configuración de red, nombre de la máquina, etc. En este fichero se almacenan las funciones para modificar estos parámetros tras rellenar el formulario mostrado en la figura 4.25. Además también se almacenan las funciones para añadir o borrar una nueva máquina al sistema.

Paquete *Installations.pm.*

Paquete que define las funciones para comunicarse directamente con la aplicación *MSFAdt*:

- Configurar una máquina o plataforma ejecutando el *script* `configure-machine.pl`.
- Desconfigurar (pasar de estado *CONFIGURED* a *INSTALLED*) una máquina o plataforma ejecutando el *script* `set-installed-state.pl`.

Paquete *Releases.pm*.

Paquete que contiene las funciones dedicadas para la página descrita en la figura 4.27:

- Añadir o borrar versiones.
- Modificar las imágenes ISO para cada versión.
- Montar las imágenes ISO de cada versión en su respectivo punto de montaje ejecutando el *script* `mount-releases.pl`, y recogiendo su resultado para modificar la página acorde al mismo (verde si se ha montado correctamente, rojo si no).

Paquete *Wiki.pm*.

En este paquete se encuentran las funciones para poder editar la wiki y poder verla. Recoge los datos de la verdadera wiki de la empresa alojada en *Redmine* y los vuelca la plantilla `wiki.html` mostrada en la figura 4.28.

Capítulo 5

Validación del funcionamiento

Para validar la necesidad y el funcionamiento de este sistema se ha realizado una comparativa entre el tiempo utilizado y la metodología necesitada en la forma de despliegue anterior y actual. Para una mejor exposición de esta evolución, y por tanto de la validación del sistema, se hablará acerca de los siguientes puntos:

1. Metodología de despliegue anterior y estudio del tiempo necesitado.
2. Pequeño resumen de lo que permite el sistema desarrollado en este trabajo fin de grado.
3. Metodología de despliegue actual y estudio del tiempo necesitado.
4. Comparativa.

5.1. Metodología de despliegue anterior y estudio del tiempo necesitado

En la fase previa al desarrollo de este sistema, para el despliegue del proyecto *MSF* era necesario realizar todos los pasos manualmente. Es decir, la creación de las imágenes ISO, tanto para el servidor como para la estación de trabajo, se debía de hacer manualmente ejecutando ordenes alojadas en un *Makefile*. Seguidamente, con estas imágenes ISO se grababan dos DVDs físicos y se introducían en los respectivos servidores y estaciones de trabajo, para su posterior instalación (indicando manualmente la configuración de red y estando físicamente frente a la máquina). Esto suponía que, para el despliegue del código, debía de estar presente un empleado, encargado del despliegue, el cual solo podría instalar una sola plataforma a la vez.

Lamentablemente no se disponen datos sobre los tiempos de despliegue mediante esta metodología, sin embargo se ha realizado una pequeña entrevista con el encargado de estas operaciones durante esta fase del tiempo. Mediante esta entrevista se ha podido saber que los tiempos de despliegue venían a durar una media de 3 horas, con una varianza muy elevada debido a que todas las operaciones se realizaban a mano, por lo que el factor humano tenía gran influencia en el tiempo de despliegue.

5.2. Resumen del sistema de integración continua desarrollado

Con el desarrollo de este sistema de integración continua se ha podido automatizar la creación de las imágenes ISO y las instalaciones de las máquinas, centralizando toda su ejecución en un solo servidor encargado de proporcionar los ficheros necesarios mediante la tecnología *PXE* y *Kickstart*. Esta centralización permite, además de automatizar completamente estos pasos, y por tanto poder realizar un sistema de integración continua (y además, una plataforma desde la cual se puedan instalar todas las máquinas deseadas en cualquier momento), una reducción considerable del tiempo de despliegue y una posibilidad de despliegues paralelos. Para ello ha sido necesario desarrollar tres aplicaciones, una encargada de la creación y publicación de las imágenes ISO de forma automatizada (*MSFAdt_build*), otra encargada de gestionar las instalaciones y proporcionarles los paquetes necesarios a las máquinas conectadas a la red de PXE (*MSFAdt*), y una última que proporcione una interfaz gráfica para el usuario final, pudiendo ejecutar instalaciones y configurar las máquinas de forma fácil e intuitiva (*MSFAdt_web*). Por último, también se ha desarrollado el sistema de integración continua en *Bamboo* haciendo uso de los mismos *scripts* utilizados en la aplicación web, pues al fin y al cabo lo que permite la interfaz web es realizar el despliegue del código en cualquier máquina, en vez de en la máquina destinada para la integración continua (utilizada por *Bamboo* y la que se realiza diariamente).

5.3. Metodología de despliegue actual y estudio del tiempo necesitado

Como se ha comentado anteriormente, el despliegue actual se ha visto completamente automatizado, y es posible su ejecución mediante la interfaz web desarrollada en *MSFAdt_web*. Por tanto, la metodología utilizada para el despliegue del código se basa simplemente en configuraciones realizadas en dicha interfaz web.

Son posibles dos maneras de despliegue, una completamente automatizada que no se explicará, debido a que su ejecución simplemente consiste en pulsar el botón `Build+install` en la interfaz web. El *script* que ejecuta este botón se trata del mismo que ejecuta la aplicación de *Bamboo*, permitiendo así el despliegue completamente automatizado. La otra opción se trata de ejecutar las aplicaciones manualmente, que al fin y al cabo es lo que realiza el anteriormente mencionado *script*, pero de manera ordenada. Estas configuraciones se detallan a continuación, además de en el apéndice B:

1. En primer lugar, para la generación de la imágenes ISO se hará uso de la aplicación *MSFAdt_build*, que se puede gestionar en la interfaz web desde la página dedicada a los *builds* `System tools` → `build`. En esta página se especifica la rama o etiqueta de la cual se va a recoger el código desarrollado pulsando el botón *Edit branch*, y se lanza el *build* pulsando en el botón *Build*. En esta misma ventana se verá que el proceso ha comenzado cuando se empiecen a escribir líneas

Muestra:	1	2	3	4	5	6	7	8	9	10	11	12	13
Minutos:	87.3	85.98	91.1	81.7	86.56	84.05	88.55	82.43	84.53	81.28	80.73	86.68	86.98

Tabla 5.1: Tiempo empleado en el despliegue del código.

de *log* en la ventana inferior. Cuándo este proceso termine se podrá ver el nombre de las nuevas imágenes ISO generadas resaltado en rojo (fig. B.1).

2. Una vez la generación de las imágenes ISO haya finalizado, estas podrán ser seleccionadas en la ventana de `Configure` → `Configure releases`. Tras ser seleccionadas en su correspondiente versión, pasan a ser montadas en la máquina virtual que hace las veces de servidor de las instalaciones (fig. B.2).

3. En la página de `Configure` → `Configure machines` se selecciona la versión, el modo y el tipo de la instalación de la plataforma en cuestión, o del servidor/estación de trabajo a instalar (fig. B.3).

4. En la ventana de `Configure` → `Configure Installations` se seleccionan la(s) plataforma(s) a instalar o el servidor/estación de trabajo en concreto, y se configuran. En este paso, es cuándo su dirección MAC e IP estática se configuran en el servidor DHCP alojado en la máquina virtual servidora de instalaciones, y de esta forma en el próximo arranque por red, el servidor les dará esta IP y los ficheros de arranque por red (fig. B.4).

5. Por último en la pantalla de inicio se selecciona la opción `Install now` y se inicia el proceso de instalación (fig. B.5). Tanto si este proceso acaba satisfactoriamente como si no, se podrá ver en los estados de esta misma página. Si acaba con errores, se podrá consultar la página dedicada a estos para más información del error sucedido (fig. B.6).

Para la comprobación del tiempo de despliegue, se han recogido los tiempos de inicio y final de despliegue de los *logs* generados por las aplicaciones, y se han almacenado en un fichero de datos. Se han podido recoger 15 muestras mostradas en la tabla 5.1.

Con estas muestras se ha realizado el test de *Lillietest* [The19], que decide si un número pequeño de muestras (cómo es este caso) proviene de una población Gaussiana o no. Si esto es cierto, se podrá estimar una media y una varianza de las instalaciones, y por tanto comparar con el tiempo de despliegue utilizado anteriormente.

Para realizar el test, en *MatLab* se ha ejecutado el siguiente *script*:

```
1 %Se cargan los tiempos de instalacion almacenados en el fichero
      .dat
2 t_instalaciones = load('instalaciones.dat');
3
4 %Se normalizan estos valores
5 t = (t_instalaciones - mean(t_instalaciones))/std(
     t_instalaciones);
6
```

```
%Se comprueba su normalidad
[h,p] = lillietest(t)

---

h =

   0
p =

   0.5000
```

Este *script* devuelve dos valores `h` y `p`. El valor `h` indica, cuando es 0, que la población de la que provienen las muestras es una población Gaussiana, y el valor p devuelve la fiabilidad del resultado, es decir, cuanto menor es el valor `p` (entre el rango `[0.001,0.5]` [The19]), mayor es la duda de que la población de la que provengan las muestras sea Gaussiana. En este caso, este valor `p` es el máximo (se sale de las tablas utilizadas), por lo que se puede deducir que, sin ninguna duda, la población de la que provienen las muestras es Gaussiana.

Una vez comprobado que se comporta este tiempo de despliegue de manera Gaussiana, se puede sacar una media y una varianza de este tiempo:

$$\eta_{despliegue} = 84{,}8400\ minutos \quad \sigma^2_{despliegue} = 9{,}9256\ minutos^2 \tag{5.1}$$

5.4. Comparativa

Tras haber visto la evolución que proporciona este sistema en el despliegue del código, se va a realizar una comparativa acerca de los tiempos del mismo. Cómo se ha comentado en la sección 5.1, no se ha podido obtener una serie de muestras sobre el tiempo de despliegue requerido anteriormente al este sistema, sin embargo tras una pequeña entrevista con el encargado de ello se ha podido estimar que tenía una media de 3 horas, lo que son 180 minutos. En el caso actual, tras el desarrollo del sistema, el despliegue ha quedado reducido a una media de 84,84 minutos con una varianza de 9,9256 minutos^2.

Además de verse reducido el tiempo de despliegue, cabe destacar que este puede realizarse de manera paralela en todas las plataformas deseadas, cosa que anteriormente sin este sistema no podía realizarse, por lo que el tiempo de despliegue, si se realizase para varias plataformas, se vería incrementado en el primer caso, mientras que con el sistema actual el tiempo de despliegue no variaría.

Se puede concluir en que este sistema, además de proporcionar un despliegue **completamente desatendido** y con la posibilidad de realizar varios **despliegues paralelos**, proporciona un despliegue mucho más eficiente en tiempo respecto al anterior, concretamente se realiza el despliegue casi tres veces más rápido (en el caso de un solo despliegue), proporcionando un ahorro de recursos a la empresa muy elevado.

Capítulo 6

Conclusiones y líneas futuras

6.1. Conclusiones

Haber desarrollado las aplicaciones *MSFAdt*, *MSFAdt_web* y *MSFAdt_build*, y el sistema de integración continua en *Bamboo* (haciendo uso de las anteriores aplicaciones), provee una serie de beneficios hacía el usuario final:

- El sistema de integración continua, objetivo principal de este trabajo final de grado, queda completamente desarrollado y configurado para que el despliegue del código se realice diariamente de manera completamente automatizada. Informa de los resultados mediante la herramienta de comunicación en la empresa, si bien han sido satisfactorios o se ha producido algún error. En el caso de que se hayan producido errores, informa del fallo en concreto y del responsable (personas que han subido código nuevo al repositorio respecto a la última ejecución satisfactoria del sistema). Esto permite una mejora de calidad en el desarrollo y despliegue del sistema desarrollado, pudiendo mantener un control de errores continuo en el código. Además este sistema proporciona a la empresa un ahorro de tiempo significativo en el despliegue de su proyecto.

- La aplicación *MSFAdt_web* proporciona hacia los usuarios una interfaz amigable de los resultados de las instalaciones, *builds* y demás. A mayores permite ejecutar tareas aisladas al sistema de integración continua, como por ejemplo instalación de una plataforma para el desarrollo del código, creación de unas nuevas imágenes ISO para su posterior instalación en las plataformas, ejecutar un escenario de integración continua en una plataforma determinada (o varias), etc. Proporciona información sobre posibles fallos en las instalaciones y *builds*. En definitiva, centraliza todas las operaciones y permite un seguimiento de errores de los posibles despliegues, otro de los objetivos de este sistema.

- El método de despliegue del proyecto *MSF* ha quedado completamente simplificado, pudiendo realizar el mismo con el *click* de un botón en la interfaz web. Además la eficiencia se ha visto aumentada, ya que, además de mejorar el tiempo de despliegue, con este sistema son posibles despliegues en varias plataformas de manera paralela, liberando así de una gran carga de trabajo para el encargado del despliegue en la empresa.

6.2. Líneas futuras

Respecto a las líneas futuras que puede tomar este sistema desarrollado se basan en la implementación de nuevos pasos en este proceso de integración continua. En este trabajo final de grado se han mencionado y automatizado los pasos de creación de las imágenes ISO con el código y el sistema operativo desarrollado, y su posterior instalación en las plataformas. Sin embargo, a este sistema se le pueden añadir diversos pasos a priori y posteriori que permiten una mayor comprobación de la validez y errores del código. Estos pasos pueden ser:

- **Comprobaciones de calidad del código mediante *SonarQube***: Permiten comprobar la calidad del código desarrollado por los empleados con la herramienta corporativa *SonarQube*. Esta herramienta comprueba fichero a fichero su sintaxis comparándola con una serie de reglas alojadas en el servidor *SonarQube* corporativo. Automatizar dichas pruebas ahorraría mucho tiempo a los empleados, pues semanalmente se realizaría esta comprobación y no sería necesaria su realización a mano. Este paso sería el primero en realizarse, previo a la creación de la imagen ISO.

- **Tests unitarios**: La mayor parte del código se encuentra desarrollado en *Java*. En este existen ficheros desarrollados por los empleados que comprueban la validez del código parte por parte. Estos ficheros de comprobación (*tests*), se ejecutan mediante herramientas que comprueban su cobertura (*tests* ejecutados correctamente y *tests* fallidos). La comprobación previa de estas coberturas permite una comprobación de las funcionalidades del código necesaria en cada momento. Añadiendo y automatizando estos *tests* al sistema de integración continua (paso previo a la creación de la imagen ISO, y tras haber realizado las comprobaciones de calidad del código), permitirá a los desarrolladores una liberación de carga de trabajo extra, debido a que a partir de ese momento las comprobaciones de cobertura se realizarán diariamente y de forma automática.

- **Comprobaciones de integridad del sistema**: Una vez instalada la plataforma, para comprobar su validez, se realizan una serie de comprobaciones sobre el sistema para comprobar que es seguro y que se encuentra todo tal y como debe de estar. Estas comprobaciones se encuentran ya desarrolladas, por lo que el paso siguiente sería automatizarlas y añadirlas al sistema de integración continua, liberando así de trabajo y proporcionando información diaria a los validadores de código que realizan estas operaciones con cada sistema.

De forma adicional, para la mejora del sistema de despliegue (ya no del propio sistema de integración continua, si no lo pasos requeridos para desplegar), se realizarán estudios de los tiempos que tarda en realizar cada tarea, por ejemplo, en crear compilar determinada parte, en publicar la imagen ISO, en instalar el sistema operativo, etc. Teniendo el conocimiento de estos tiempos se podrán realizar gráficas (que se mostrarán en una nueva página de la aplicación *MSFAdt_web*) que permitan comprobar qué tareas son las que retrasan más el proceso de despliegue, y por tanto poder tomar decisiones para la optimización del proceso de despliegue.

Apéndice A

MSFAdt Installation Guide

A.1. Installing basics

1. Install a minimum CentOs 7.x install and upgrade to latest. We need first to configure hostname, statics IPs (PXE network → 10.40.1.0/24), and DNS servers (nameserver 8.8.8.8 and DEIMOS servers).

2. Since MSFADT touches several system config files, it must be used and run as root.

3. Update the OS `yum update` and also install the kernel headers

```
[root@pxe-platform ~]# yum install kernel-devel kernel-$(uname -r)
```

4. Install GIT for downloading MSFAdt software `yum install git`

5. Clone the git repository `git clone https://<url\_git>/msf.git` into `/root/DEIMOS-MSF` directory

6. Install EPEL repository `yum install epel-release`

7. Create a local::lib private perl environment:

```
[root@pxe-platform ~]# yum install perl-local-lib

[root@pxe-platform ~]# perl -Mlocal::lib >> ~/.bashrc

(exit from the shell and enter again)

[root@pxe-platform ~]# yum install perl-core

[root@pxe-platform ~]# curl -L https://cpanmin.us | perl - App::cpanminus
```

8. Install redhat-rpm-config, it is required for LWP and HTML::HeadParser modules and also gcc package

```
[root@pxe-platform ~]# yum install redhat-rpm-config gcc
```

9. Install required modules in the local::lib environment:

```
[root@pxe-platform ~]# cpanm YAML:Tiny Log::Log4perl Dancer2 Starman LWP
Datetime Template JSON File::Slurp Net::SSH::Perl Mojolicious
Mojolicious::Plugin::Authenticantion Mojolicious::Plugin::RenderFile
HTML::Template HTML::FromANSI::Tiny Switch
```

10. Install the required packages for HTTP, DHCP and TFTP services

```
[root@pxe-platform ~]# yum install httpd dhcp tftp-server xinetd syslinux
```

11. Copy Apache config file to destination, and enable+restart apache:

```
[root@pxe-platform ~]# cp $MSFAdt\_dir/apache/msf-releases.conf /etc/httpd/conf.
    d/
```

12. On the api.yaml config files, we have to put our server ip address `<MSFAdt\_dir>/etc/api.yaml`, `<MSFAdt\_dir>/msfadt\_web/etc/api.yaml` and `<MSFAdt\_dir>/msfadt\_build/etc/api.yaml`.

13. Enable the httpd service

```
[root@pxe-platform ~]# systemctl enable httpd
[root@pxe-platform ~]# systemctl restart httpd
[root@pxe-platform ~]# chkconfig httpd on
```

14. Configure the firewall to allow required services:

```
[root@pxe-platform ~]# firewall-cmd  --add-service=http
[root@pxe-platform ~]# firewall-cmd  --add-service=dhcp
[root@pxe-platform ~]# firewall-cmd  --add-service=nfs
[root@pxe-platform ~]# firewall-cmd  --add-port=3001/tcp
[root@pxe-platform ~]# firewall-cmd  --add-port=3002/tcp
[root@pxe-platform ~]# firewall-cmd  --add-port=3003/tcp
[root@pxe-platform ~]# firewall-cmd --runtime-to-permanent
```

    NOTE: If you have problems with permissions into the apache server, disable SELinux:

```
[root@pxe-platform ~]# setenforce 0

Open /etc/selinux/config file and set the SELINUX mod to disabled
```

15. TFTP is configured for serving the boot files, and xinetd manage this service, so at /etc/xinetd.d/tftp, the “disabled” option must be “no”.

16. Copy the `$MSFAdt_dir/boot.save` files into `/var/lib/tftpboot`

```
[root@pxe-platform ~]# cp -rf $MSFAdt\_dir/boot.save /var/lib/tftpboot/
```

17. Create directories where we are going to serve the MSF files to the machines.

```
[root@pxe-platform ~]# mkdir -p /vm/releases
```

18. `/vm/images-iso` must be mounted in our server (disk array where are stored the iso files):

```
[root@pxe-platform ~]# echo '<ip\_array>:/export/isos /vm/images-iso nfs
    defaults
 0 0' >> /etc/fstab
[root@pxe-platform ~]# mkdir /vm/images-iso
[root@pxe-platform ~]# mount -a
```

19. The msfadt api needs external scripts to work properly. One of them needs a config file with our redmine user and password. By default it uses the dafp credentials, but if dafp is not now in the msf project, just put your username and password there, into `$MSFAdt\_dir/tools/redmine`

20. Install at software with `yum install at`

21. The msfadt api also needs a python script to notify the machines installations. So we need to install python and a Skype module for it:

```
[root@pxe-platform ~]# yum install -y https://centos7.iuscommunity.org
    /ius-release.rpm
[root@pxe-platform ~]# yum install -y python36u python36u-libs python36u-devel
    python36u-pip
[root@pxe-platform ~]# pip3.6 install SkPy
```

22. To manage remote vmware installations, we need to install powershell and the powerCLI module into it.

```
[root@pxe-platform ~]# sudo rpm --import https://packages.microsoft.com/keys
    /microsoft.asc
[root@pxe-platform ~]# curl https://packages.microsoft.com/config/rhel/7/prod.
    repo
   | sudo tee /etc/yum.repos.d/microsoft.repo
[root@pxe-platform ~]# yum update
[root@pxe-platform ~]# yum install powershell
[root@pxe-platform ~]#  pwsh
   >> Set-PSRepository -Name "PSGallery" -InstallationPolicy "Trusted"
   >> Find-Module "VMware.PowerCLI" | Install-Module -Scope "CurrentUser"
        -AllowClobber
   >> Set-PowerCLIConfiguration -InvalidCertificateAction "Ignore"
```

A.2. Running MSFAdt

1. Create the config files

```
[root@pxe-platform ~]# cp $MSFAdt\_dir/etc/machines-example.yaml
    $MSFAdt\_dir/etc/machines.yaml
[root@pxe-platform ~]# cp $MSFAdt\_dir/etc/releases-example.yaml
    $MSFAdt\_dir/etc/releases.yaml
```

2. Switch on the APIs

```
[root@pxe-platform ~]# $MSFAdt\_dir/api/launch.sh
[root@pxe-platform ~]# $MSFAdt\_dir/msfadt\_build/api/launch.sh
[root@pxe-platform ~]# $MSFAdt\_dir/msfadt\_web/script/launch.sh
```

3. Execute one api-test to load the run config files for each machine:

```
[root@pxe-platform ~]# $MSFAdt\_dir/api-test/install\_machine.pl -m msf-srv99
```

(after that, check if there are the config files into the run directory).

4. Create the scheduled-machines.yaml file

```
[root@pxe-platform ~]# touch $MSFAdt\_dir/run/scheduled-machines.yaml
```

5. Create the msfadt_build log directories

```
[root@pxe-platform ~]# mkdir $MSFAdt\_dir/msfadt\_build/log
[root@pxe-platform ~]# mkdir $MSFAdt\_dir/msfadt\_build/build-log
```

6. If the servers don't work properly, check logs into log directories for each api.

7. Log in into the msfadt_web api in any browser

```
$msfadt_ip:3002
```

Apéndice B

Proceso de despliegue del código.

Figura B.1: Paso 1: Creación de las imágenes ISO.

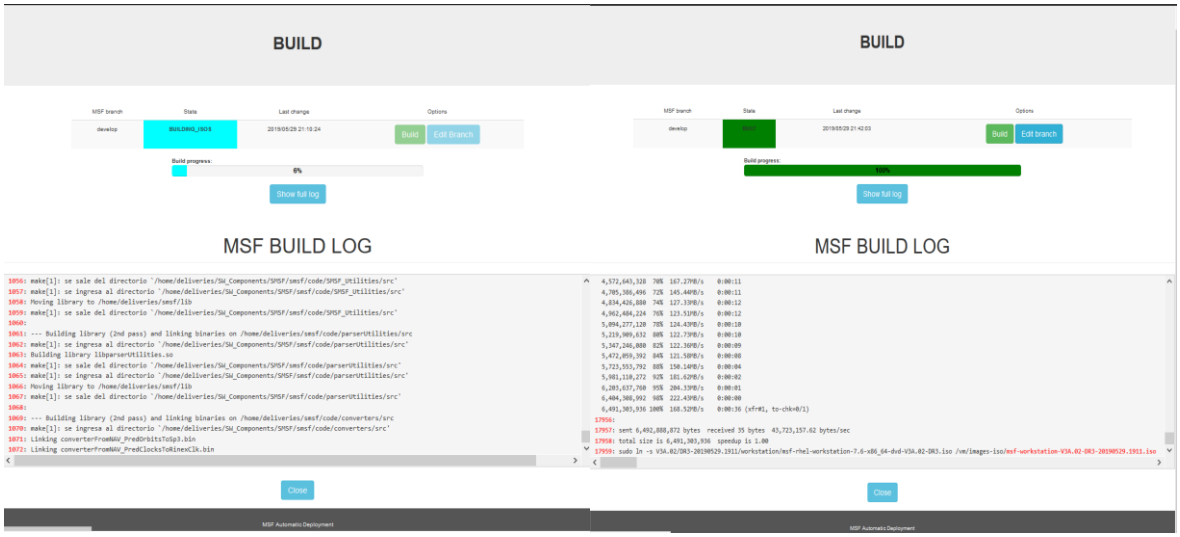


Figura B.2: Paso 2: Selección y montaje de las nuevas imágenes ISO.

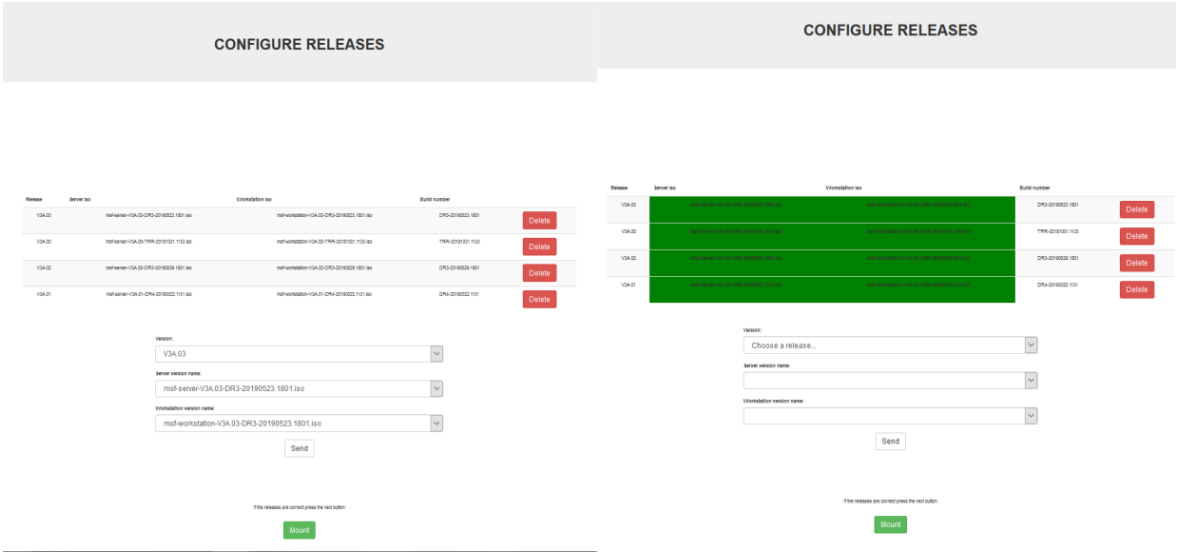


Figura B.3: Paso 3: Configuración de modo y tipo de instalación de las plataformas.

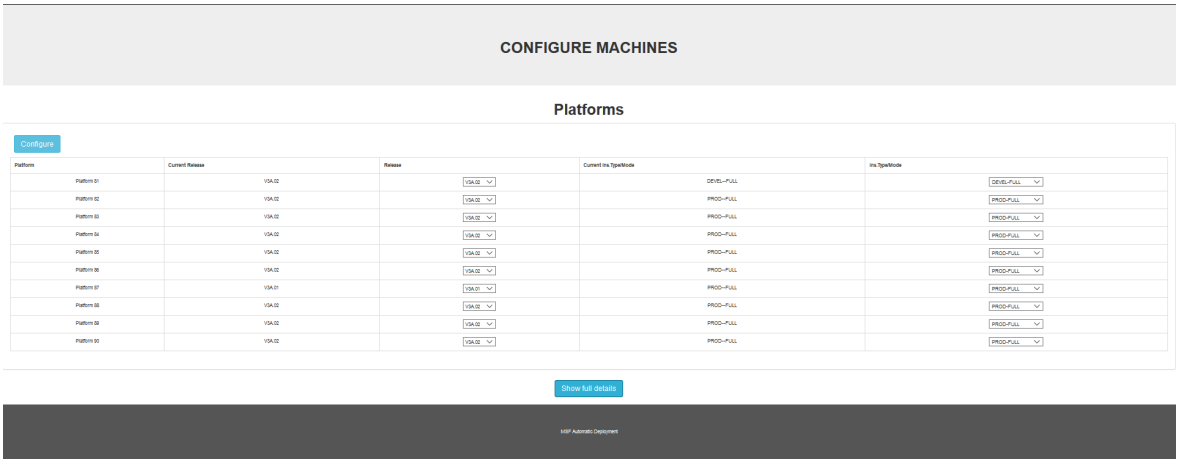


Figura B.4: Paso 4: Selección de plataformas a instalar.

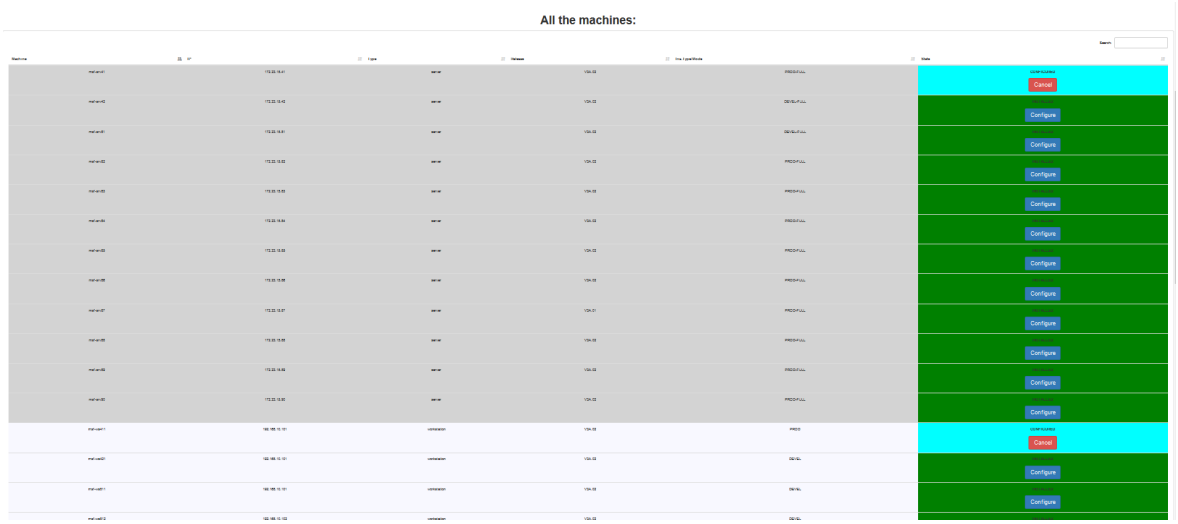


Figura B.5: Paso 5: Instalación de plataformas.

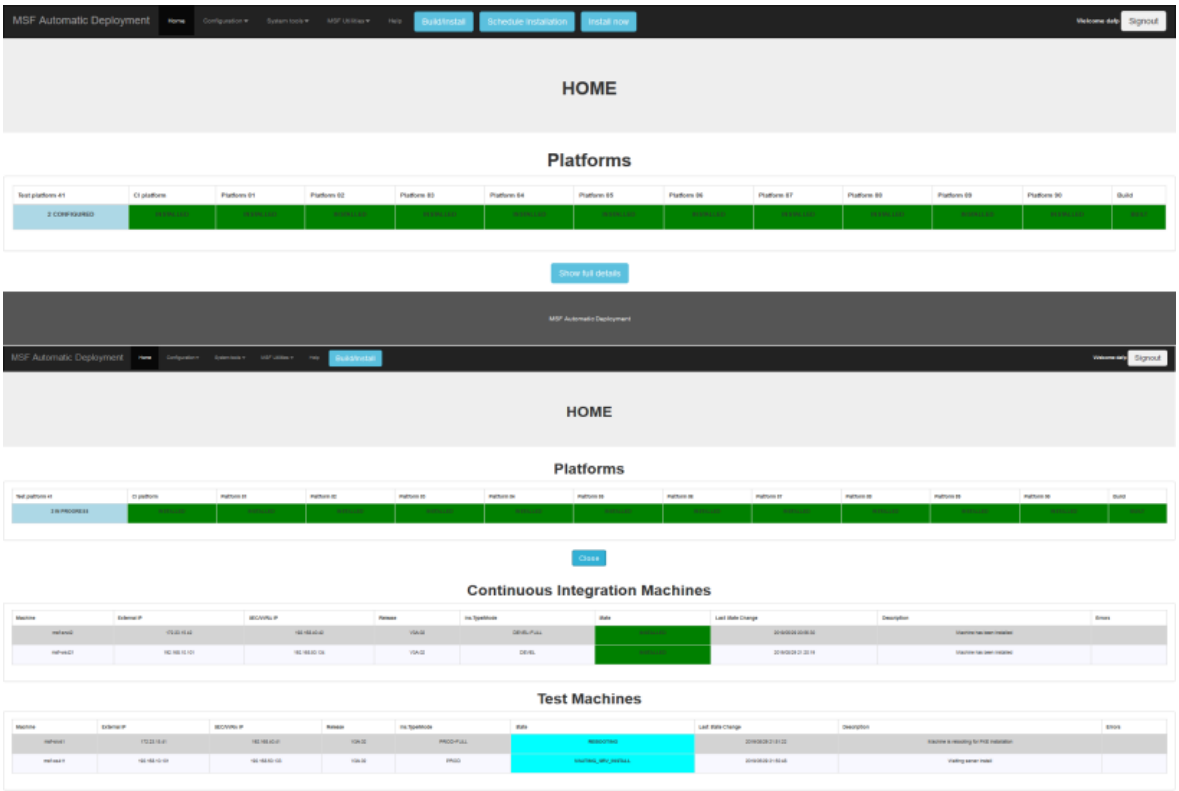
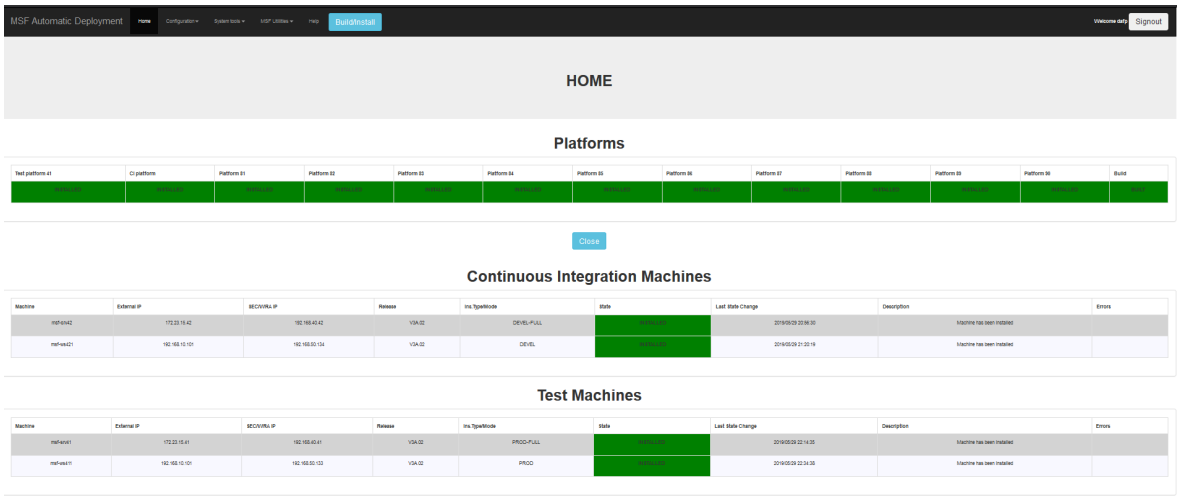


Figura B.6: Paso 6: Finalización del proceso de instalación.



Bibliografía

[AR19a] M. Antón Rodríguez y M. Ángeles Pérez Juárez. *Manual de AJAX, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad de Valladolid*, 2019.

[AR19b] M. Antón Rodríguez y M. Ángeles Pérez Juárez. *Manual de CSS, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad de Valladolid*, 2019.

[AR19c] M. Antón Rodríguez y M. Ángeles Pérez Juárez. *Manual de HTML, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad de Valladolid*, 2019.

[AR19d] M. Antón Rodríguez y M. Ángeles Pérez Juárez. *Manual de JavaScript, Escuela Técnica Superior de Ingenieros de Telecomunicación, Universidad de Valladolid*, 2019.

[Atl19] Atlassian. Página oficial de Bamboo. `https://es.atlassian.com/software/bamboo`, Última visita abril 2019.

[Bri09] Brian Foy. *Mastering Perl*. O'Reilly Media, Inc, 2009.

[Cis] Cisco Systems Inc. Creating Ethernet VLANs on Catalyst Switches. `https://www.cisco.com/c/en/us/support/docs/lan-switching/vlan/10023-3.html`.

[Dan19] Dance Perl Web Framework. The easiest say to create web applications in Perl. `http://perldancer.org/quickstart`, Última visita abril 2019.

[Ecm17] Ecma International. The JSON Data Interchange Syntax, Diciembre 2017.

[Hew18] Hewlett Packard Enterprise. Hewlett Packard Enterprise, HPE iLO 5 User Guide, 2018.

[Int99] Intel Corporation, SystemSoft TM. Preboot Execution Environment (PXE) Specification, 1999.

[Jea19] Jean-Philippe Lang. Página oficial de Redmine. `https://www.redmine.org/`, Última visita mayo 2019.

[Jen19] Jenkins Community. Jenkins Official Page and Documentation. `https://jenkins.io/`, Última visita abril 2019.

[Met19a] MetaCPAN. PSGI - Perl Web Server Gateway Interface Specification. https://metacpan.org/pod/PSGI, Última visita abril 2019.

[Met19b] MetaCPAN. Starman - High-performance preforking PSGI/Plack web server. https://metacpan.org/pod/Starman, Última visita abril 2019.

[Ora13] Oracle Corp. *Oracle Integrated Lights Out Manager (ILOM) 3.0. Daily Management — Concepts Guide*, 2013.

[Pav09] Pavel Kulchenko, Randy Ray. *Programming Web Services with Perl Practical Advice for Rapid Web Services Development*. O'Reilly Media, Inc, 2009.

[Red] Red Hat Inc. ¿Qué es la integración continua/distribución continua (CI/CD)? https://www.redhat.com/es/topics/devops/what-is-ci-cd.

[Red19a] Red Hat Inc. 23.2. ¿Cómo realizar una instalación de Kickstart? https://access.redhat.com/documentation/es-es/red_hat_enterprise_linux/7/html/installation_guide/sect-kickstart-howto, Última visita abril 2019.

[Red19b] Red Hat Inc. 27.2. At and Batch. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/deployment_guide/s1-autotasks-at-batch#sect-At_and_Batch_Installation, Última visita abril 2019.

[Red19c] Red Hat Inc. Chapter 40. Log files. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/5/html/deployment_guide/ch-logfiles, Última visita abril 2019.

[Red19d] Red Hat Inc. Installing, configuring, and managing virtual machines on a Red Hat Enterprise Linux physical machine. https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/virtualization_deployment_and_administration_guide/, Última visita abril 2019.

[Run19] RunDeck Community. Rundeck Official Page and Documentation. https://www.rundeck.com/open-source, Última visita abril 2019.

[Seb19] Sebastian Riedel. A next generation web framework for the Perl programming language. https://mojolicious.org/, Última visita abril 2019.

[Sim05] Simon Cozens. *Advanced Perl Programming: The Worlds Most Highly Developed Perl Tutorial*. O'Reilly Media, Inc, 2005.

[Son19] SonarQube. Página oficial SonarQube. https://www.sonarqube.org/, Última visita mayo 2019.

[The19] The MathWorks, Inc. Lillietest. https://es.mathworks.com/help/stats/lillietest.html?s_tid=srchtitle#btodixy-2, Última visita junio 2019.

[Tod19] Todd C. Miller. Manual oficial de Linux, capítulo 5, Última visita mayo 2019.

[Tom03] Tom Christiansen, Nathan Torkington. *Perl Cookbook: Solutions and Examples for Perl Programmers*. O'Reilly Media, Inc, 2003.

[Tom12] Tom Christiansen, Brian Foy, Larry Wall, Jon Orwant. *Programming Perl*. O'Reilly Media, Inc, 2012.

[VMW18a] VMWare Inc. Vcenter Server Installation and Setup, 2018.

[VMW18b] VMWare Inc. VMware ESXi Installation and Setup, 2018.