UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Automatización de despliegues mediante VMware, Jenkins y Robot Framework

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Pablo Borja Hernández
Tutor: Francesc D. Muñoz-Escoí

Curso 2016-2017

Resumen

El objetivo principal de este proyecto es conseguir una automatización completa a la hora de implantar una solución software compleja. La automatización es una herramienta poderosa que da la posibilidad de tener el proceso controlado ahorrando costes.

En el ámbito de la integración de una solución software, la automatización basada en virtualización nos permite implantar nuestro software de forma desatendida en entornos 'limpios' de una manera sencilla y rápida. Un entorno limpio permite a los desarrolladores disponer de entornos equivalentes a los de producción. Los entornos de prueba suelen estar viciados por el coste de recrearlo desde cero, gracias a esta automatización se pueden explorar fallos en la solución con mayor exactitud. Así se consigue software de mayor calidad.

Además, si enlazamos la automatización del despliegue junto a baterías de pruebas también automatizadas, conseguimos una herramienta potente para mejorar la robustez de nuestra solución con un bajo coste.

En el núcleo del proyecto se utilizan principalmente tres herramientas: Jenkins, como orquestador de todo el proceso; Orchestrator, una herramienta de VMware que permite orquestar procesos para administrar entornos de virtualización y Robot Framework, un framework de automatización de testing que en el contexto del proyecto se utiliza para automatizar la instalación de software y controlar si el flujo de trabajo se desarrolla correctamente.

Este proyecto está desarrollado dentro de una empresa privada y por lo tanto las herramientas utilizadas están condicionadas para adaptarse a esta. Como objetivo secundario del proyecto, es importante mencionar que también es utilizado para comprobar el procedimiento de implantación de la solución. Esto condiciona la forma en la que se ha automatizado.

Pablo Borja Hernandez
pabborh1@upv.es

Palabras clave: Despliegue de aplicaciones, Despliegue automatizado, Virtualización, Orquestación

Resum

L'objectiu principal d'aquest projecte és aconseguir una automatització completa a l'hora d'implantar un sistema de programari complex. L'automatització és una eina poderosa que permet tenir el procés control·lat, estalviant costos.

En l'àmbit de la integració d'una solució de programari, l'automatització basada en virtualització ens permet implantar el nostre programari de manera desatesa en entorns nets d'una manera senzilla i ràpida. Un entorn net permet als programadors disposar d'entorns equivalents als de producció. Els entorns de prova solen estar viciats pel cost de recreació des de zero. Gràcies a aquesta automatització es poden explorar fallades en la solució amb major exactitud. Així s'aconsegueix programari de major qualitat.

A més, si combinem l'automatització del desplegament amb bateries de proves també automatitzades, aconseguirem una eina potent per a millorar la robustesa de la nostra solució amb un cost reduït.

En el nucli d'aquest projecte s'utilitzen principalment tres eines: Jenkins, com a orquestrador de tot el procés; Orchestrator, una eina de VMware que permet orquestrar processos per a administrar entorns de virtualització; i Robot Framework, un framework de gestió de testeig que en el context del projecte s'utilitza per a automatitzar la instal·lació de programari i controlar si el fluix de treball es duu a terme correctament.

Aquest projecte ha estat fet en el context d'una empresa privada i, per aquest motiu, les eines utilitzades estan condicionades per a adaptar-se a ella. Com a objectiu secundari del projecte, és important recordar que també s'ha utilitzat per a comprovar la qualitat del procediment d'implantació de la solució. Això també condiciona la manera en la que s'ha fet l'automatització.

Paraules clau: Desplegament d'aplicacions, Desplegament automatitzat, Virtualització, Orquestració.

Abstract

The main goal of this project consists in automating the deployment of a multi-component software product. Automation is a powerful strategy that allows controlling a deployment procedure reducing its costs.

In the scope of the integration of a multi-component software project, automation –based on virtualisation– allows us to implement that software in controlled environments, in an easy and fast way. This allows developers to use development environments, equivalent to the final production environment, where design faults and development bugs may be easily found and fixed. Thus, a software of better quality can be built.

Additionally, when automated deployment is combined with automated testing, their result is a powerful tool for improving the robustness of a software product, with a minimal staff effort.

This project uses three main software tools: (1) Jenkins, as its global orchestrator, (2) VMware Orchestrator, as its VM manager, and (3) Robot Framework, for automating software installation and controlling whether the intended forkflow is correctly applied.

This project has been carried out in a private company. This has conditioned the set of tools to be used and the set of stages to be automated.

Key words: Application deployment, Automated deployment, Virtual machines, Orchestration

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

1.1 Motivación

El proyecto se ha realizado en el ámbito empresarial, por lo tanto debido a motivos de confidencialidad no se puede contar detalladamente cual es la arquitectura de la solución software la cual es automatizada, pero esta no importa en exceso ya que podemos aplicar esta automatización a casi cualquier solución software.

La herramienta que desarrolla la empresa ayuda a las compañías telefónicas a optimizar su red. Realiza mediciones, obtiene datos de la red, calcula datos de interés, muestra mapas con datos sobre la red, entre otras muchas funcionalidades.

Para realizar todas estas funciones, la herramienta utiliza varios servidores Linux y Windows conjuntamente. Cada servidor desempeña una función específica y se comunican entre ellos para llevarla a cabo.

Debido a la complejidad en el despliegue de la solución *software* de la empresa, la automatización de esta era una tarea necesaria. Con ella se consigue reducir el tiempo total necesario en desplegar la solución, pasando de dos días de media, a apenas tres horas.

No solo el factor tiempo total es una de las grandes mejoras, también el tiempo necesario de un trabajador para realizarlo disminuye de manera drástica. De dos días de trabajo necesario, instalando, configurando, esperando a componentes que dependen de otros para poder instalarse, etc. se pasa a diez minutos de trabajo en rellenar el XML de configuración donde decimos al automatizador lo que queremos desplegar.

La solución maneja una gran cantidad de datos. Si quisiéramos probar un entorno funcional de pequeño tamaño, necesitaremos como mínimo 500 GB de memoria RAM y alrededor de 10 TB de espacio de almacenaje. Por este motivo es caro mantener varios despliegues funcionando de forma simultánea. Por lo tanto es importante ser capaz de eliminar y crear un entorno en poco tiempo y con un coste bajo. Además debido a la arquitectura de la solución la reutilización del entorno es costosa.

Además de dar entornos de forma más eficaz a los validadores del software, también es interesante la integración continua y la automatización de testing. De esta forma, cuando se realice un cambio en el código de la solución, se puede lanzar de forma autónoma un despliegue y a continuación ejecutar una batería de pruebas.

1.2 Objetivos

El objetivo de este proyecto es conseguir automatizar completamente el despliegue de una solución *software* compleja. Esta solución puede no estar preparada para instalaciones sin interacción usuario computador. De forma que se simulará esta interacción para conseguir una automatización completa.

Para conseguir un despliegue completo de una solución software se va a utilizar máquinas virtuales para aprovechar la flexibilidad que ofrece. Después se conseguirá un flujo de trabajo controlado el cual interaccionará con la máquina virtual con el fin de dejarla en el estado deseado.

El objetivo principal es ahorrar el tiempo que conlleva configurar máquinas desde cero y evitar posibles fallos humanos en la instalación, consiguiendo un entorno limpio y controlado.

La automatización tendrá como parámetro de entrada un fichero XML. En este definiremos qué componentes *software* se van a desplegar y qué versión de los mismos. Los componentes son módulos los cuales se pueden desplegar de manera independiente. Así se consigue flexibilidad a la hora de sustituir un componente en un entorno, es decir, si tenemos un entorno compuesto por 15 servidores distintos y deseamos sustituir uno o un grupo de ellos, podemos eliminarlos y redesplegarlos de cero.

También se tendrá en cuenta ofrecer flexibilidad dando la posibilidad de desplegar solo la máquina virtual con el *hardware* y la red configurada, sin el *software* instalado. Con esto conseguimos probar instalaciones que no sean estándar o necesiten de un cuidado especial.

Cada componente o módulo debe tener la configuración Hardware y la configuración de red. Además también se incluye en qué disco de almacenaje y en qué Host ESXi se va a crear, así como el tipo de aprovisionamiento de estos. En el fichero de configuración XML, gracias a la flexibilidad de la virtualización, también se puede indicar si se desea sobrescribir una máquina o un entorno entero.

Además ofrece la posibilidad de generar un fichero de sesiones para MobaXterm, una suite de conexiones SSH y RDP, pudiendo importarla y acceder así al entorno de forma rápida y sencilla.

1.3 Estructura de la memoria

El resto de este trabajo está estructurado de la siguiente forma. En el capítulo 2 se describen las herramientas que se utilizan. El capítulo 3 contiene la descripción de la arquitectura del sistema de automatización. Describiendo cómo se relacionan las principales herramientas. En el capítulo 4 se muestra la infraestructura del sistema de automatización. El capítulo 5 explica cómo se desarrolla la aplicación de forma detallada. En el capítulo 6 podemos ver diferentes *workaround* necesarios en la automatización. El capítulo 7 expone un caso de uso junto a su traza de ejecución para dar una mayor visión del sistema de automatización. En el capítulo 8 podemos encontrar las conclusiones y los resultados del aprendizaje.

CAPÍTULO 2
Herramientas

2.1 Introducción

Debemos diferenciar dos fases en el proceso de la automatización. La primera es el aprovisionado de la máquina virtual, es decir, la creación de la máquina con unas especificaciones Hardware determinadas y un determinado SO. La segunda es la instalación del *software* que deseamos automatizar y la configuración del mismo. Puede consultar un artículo publicado por el IEEE sobre Máquinas Virtuales (VM) en [1].

Gracias a la flexibilidad que otorga la virtualización, la primera fase se basa en esta. La virtualización es el proceso de crear una representación basada en *software* (o virtual), en lugar de una física. La virtualización se puede aplicar a servidores, aplicaciones, almacenamiento y redes, y es la manera más eficaz de reducir los costos de TI y aumentar la eficiencia y la agilidad de los negocios de cualquier tamaño.

Concretamente y debido a que ya se encontraba presente en la empresa donde se desarrolla el proyecto, se ha utilizado VMware [15], una herramienta de virtualización de máquinas. Gracias a esta podemos simular un ordenador físico con diferentes características de memoria RAM, número de procesadores y memoria de almacenamiento.

Para la segunda fase, se ha elegido Robot Framework[11], una herramienta utilizada para automatizar testing que ya estaba presente en la empresa. Si hacemos pequeñas modificaciones, esta herramienta nos permite conseguir un control remoto de una máquina virtual y definir diferentes pasos que se ejecutarán de manera autónoma.

Para englobar estas dos fases se ha utilizado Jenkins[13].

Jenkins es un servidor de integración continua[2], gratuito, open-source y actualmente uno de los más empleados para esta función. La integración continua (continuous integration en inglés) es un modelo informático propuesto inicialmente por Martin Fowler que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes.

2.2 VMware y ESXi

VMware es un sistema de virtualización por *software*. Un sistema virtual por *software* es un programa que simula un sistema físico (un computador) con unas características determinadas.

Cuando se ejecuta el programa (simulador), proporciona un ambiente de ejecución similar a todos los efectos a un computador físico (excepto en el puro acceso físico al *hardware* simulado), con CPU (puede ser más de una), BIOS, tarjeta gráfica, memoria

RAM, tarjeta de red, sistema de sonido, conexión USB, disco duro (pueden ser más de uno), etc.

Un virtualizador permite ejecutar (simular) varios computadores (sistemas operativos) dentro de un mismo *hardware* de manera simultánea, permitiendo así el mayor aprovechamiento de recursos.

El rendimiento del sistema virtual varía dependiendo de las características del sistema físico en el que se ejecute, y de los recursos virtuales (número de procesadores asignados, memoria RAM, etc.) asignados al sistema virtual.

VMware virtualiza de forma que la mayor parte de las instrucciones en VMware se ejecutan directamente sobre el *hardware* físico.

La base de la arquitectura son servidores físicos con ESXi[16], un hipervisor[17] nativo especialmente diseñado para albergar máquinas VMware. ESXi se instala directamente en el servidor físico, lo que permite dividirlo en varios servidores lógicos denominados máquinas virtuales. La arquitectura de la virtualización es la representada en la figura 2.1.

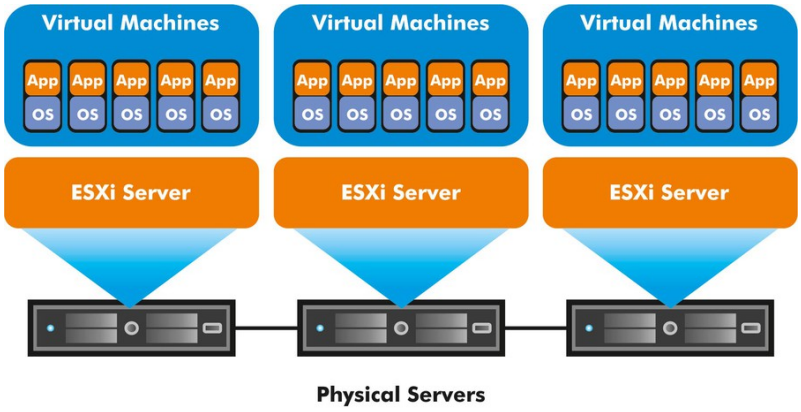

Figura 2.1: Arquitectura ESXi VMware

Para gestionar estos servidores existen herramientas de la misma compañía como vCenter[6], que ofrece una interfaz web para administrar estos servidores ESXi y las máquinas virtuales que contienen. Con funcionalidades tan potentes como pasar una máquina virtual de un servidor a otro sin que la máquina virtual pare de trabajar.

En vCenter podemos tener una visión centralizada de todos los servidores que tengamos, pudiendo consultar el uso del procesador, memoria o disco duro (HD) de los servidores, así como las diferentes máquinas que albergan.

2.3 vRealize Orchestrator (vRO)

Esta herramienta[8] es la principal de la primera fase, donde crearemos las máquinas virtuales donde poder instalar nuestro *software*. También es propietaria de VMware y es un automatizador de tareas (Orquestadora).

Una herramienta orquestadora[9] es una herramienta que permite definir un flujo de trabajo *workflow*, el cual podrá ser ejecutado de forma autónoma. Estas herramientas son típicamente usadas por los administradores de sistemas para automatizar tareas de mantenimiento en las máquinas de trabajo. En nuestro caso, esta herramienta nos servirá para gestionar la creación y configuración de las máquinas virtuales.

Gracias a la integración de vRO con vCenter, el cual contiene los servidores anfitriones ESXi que disponemos y a su vez todas las máquinas virtuales que contienen, podemos acceder a todos estos recursos desde vRO.

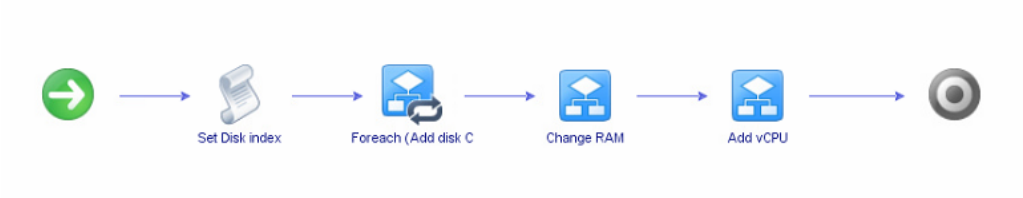

Figura 2.2: Workflow para configurar una máquina virtual

En la figura 2.2 podemos ver un *workflow* sencillo, el cual a partir de una máquina virtual, un vector de discos, un número de procesadores y una cantidad de memoria RAM, modifica el equipo de una máquina virtual.

2.4 Jenkins

Jenkins es un *software open source* de integración continua. La integración continua es un modelo informático que consiste en hacer integraciones automáticas de un proyecto lo más a menudo posible para así poder detectar fallos cuanto antes. La estructura de Jenkins es sencilla, consiste principalmente en proyectos que contienen trabajos (*Jobs*) que pueden relacionarse entre ellos consiguiendo un flujo de trabajo. Estos *Jobs* se componen de una sección inicial donde puedes descargar código de un repositorio como GIT[18] o TFS[25], definir los parámetros del trabajo o por ejemplo si se debe borrar el *Workspace* antes de cada ejecución, por ejemplo, puedes definir un trabajo que se descargue código de un repositorio, como en la figura 2.3.

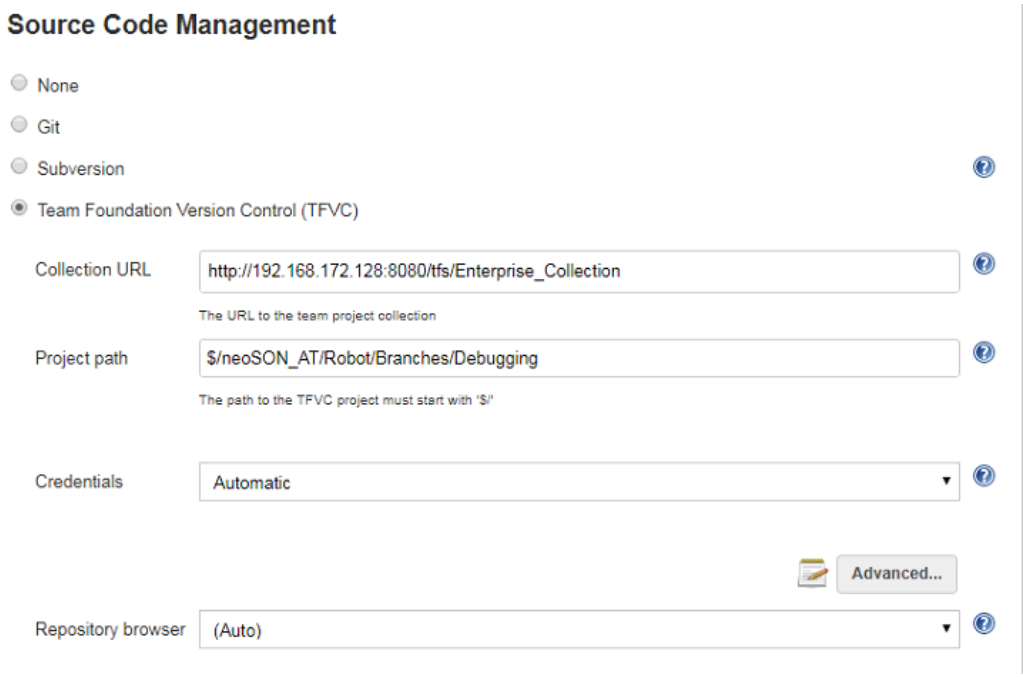

Figura 2.3: Ejemplo de conexión de Jenkins a TFS desde un trabajo

También poseen una sección de construcción *"Build"* donde definir los pasos que va a realizar el trabajo. En esta zona podemos definir que ejecute un script descargado previamente y llame a un segundo trabajo como en la figura 2.4.

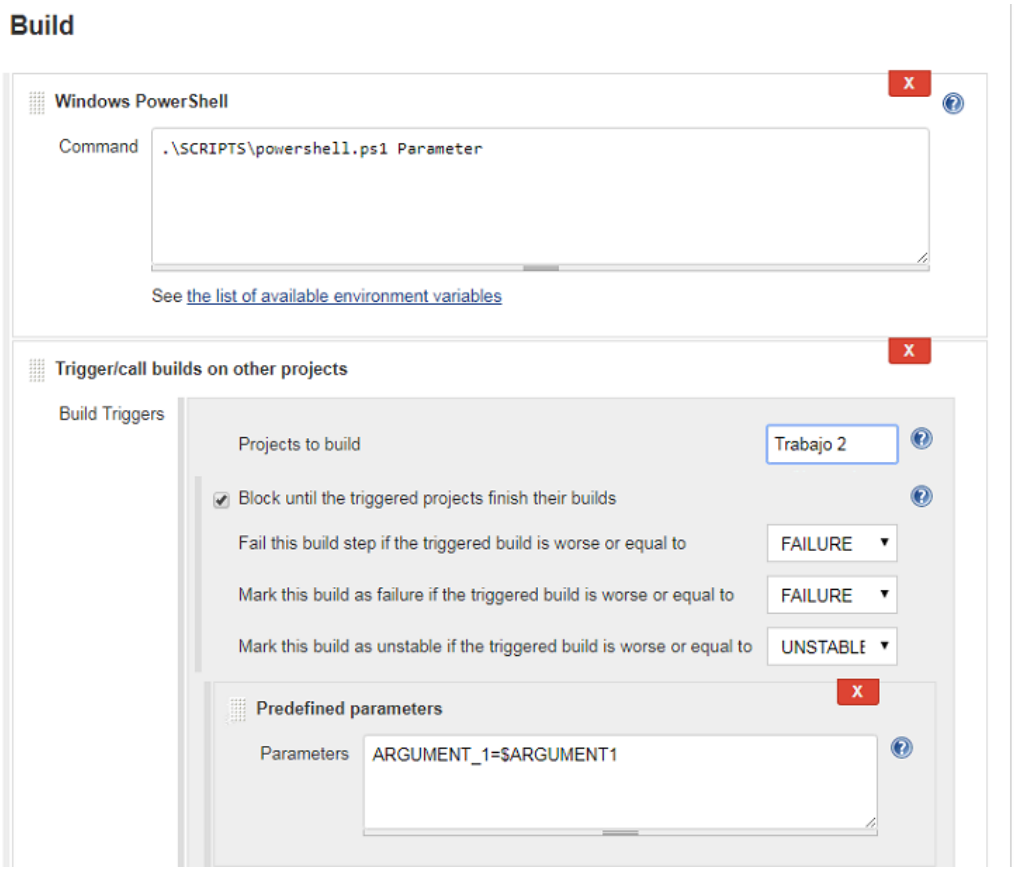

Figura 2.4: Zona de construcción de un trabajo

Cuando termina la zona de construcción el trabajo se da por finalizado y dependiendo de si los pasos han ido correctamente o no, se marcará el trabajo como completado o como fallido.

Jenkins también gestiona una zona de post-construcción *"Post-Build"* que se ejecuta una vez finalizado el trabajo, aquí podemos incluir más pasos, como por ejemplo, llamar a otro trabajo si el trabajo no se completa correctamente "Build Failed"

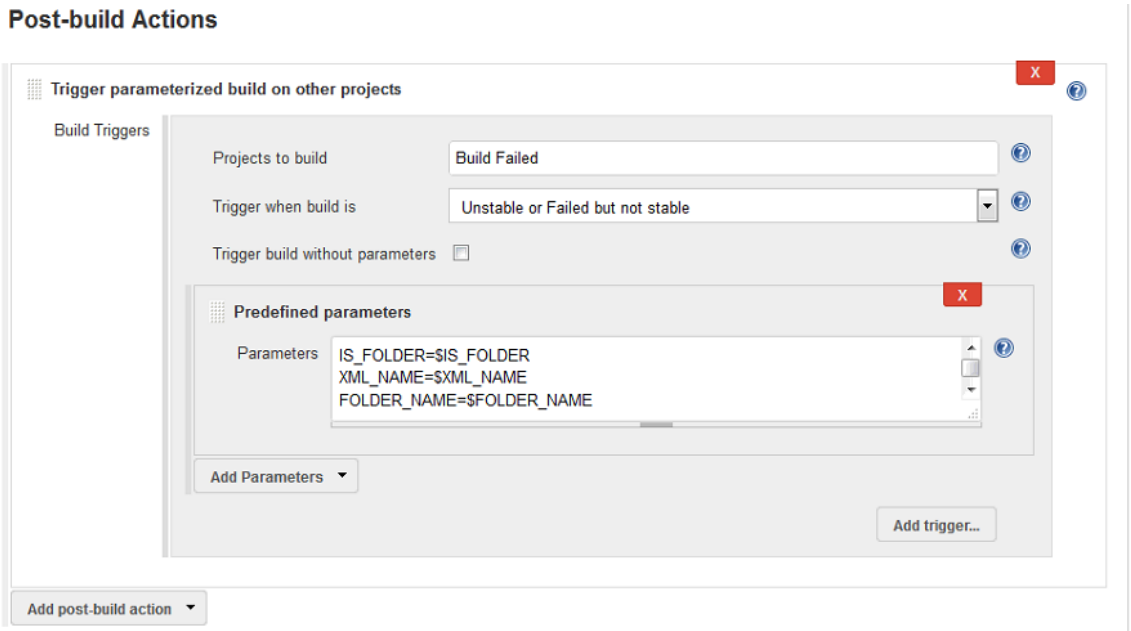

Figura 2.5: Zona post-construcción

Jenkins se basa en complementos (plugins) los cuales añadiremos dependiendo de qué herramientas estemos utilizando. Gracias a la gran comunidad que dispone Jenkins, podemos encontrar toda clase de complementos que harán posible desarrollar el flujo de trabajo que deseemos.

En el caso concreto de este proyecto, se utiliza Jenkins para controlar todo el proceso de ejecución. Desde la creación de las máquinas virtuales a la instalación del *software* y testeo del mismo.

Algunos de los componentes que se utilizan en Jenkins son TFS (Team Foundation Server), que permite descargar código de un repositorio compartido, Orchestrator plugin, que permite llamar a flujos de trabajo definidos en VROrchestrator, Trigger Jenkins, que permite la ejecución paralela de trabajos, entre otros.

2.5 Robot Framework

Para crear el flujo de trabajo que supone la instalación del *software* y el uso presente de esta herramienta en la empresa, se ha utilizado Robot Framework. Esta es una herramienta típicamente de testing. Está hecha en python[19] y está basada en *keywords* y *suits*. Una keyword es una función que se define, la cual se puede llamar desde una *suite*. Gracias a bibliotecas como SSHLibrary[20] o WinRM[22] y a unas *keywords* especialmente definidas para las políticas de seguridad de la empresa, se consigue el control remoto necesario sobre las máquinas para instalar el *software*.

Robot Framework tiene un editor visual llamado RIDE, con el que puedes consultar las bibliotecas y ver de forma visual el código. Su estructura se basa en *"suits"* y *"keywords"*. Una suite es un conjuto de *"Test"* y los *"Test"* se componen de llamadas a *keywords*. Estas *keywords* son funciones que podemos encontrar en las librerías que queramos utilizar. También podemos definir nuestros propios *keywords*.

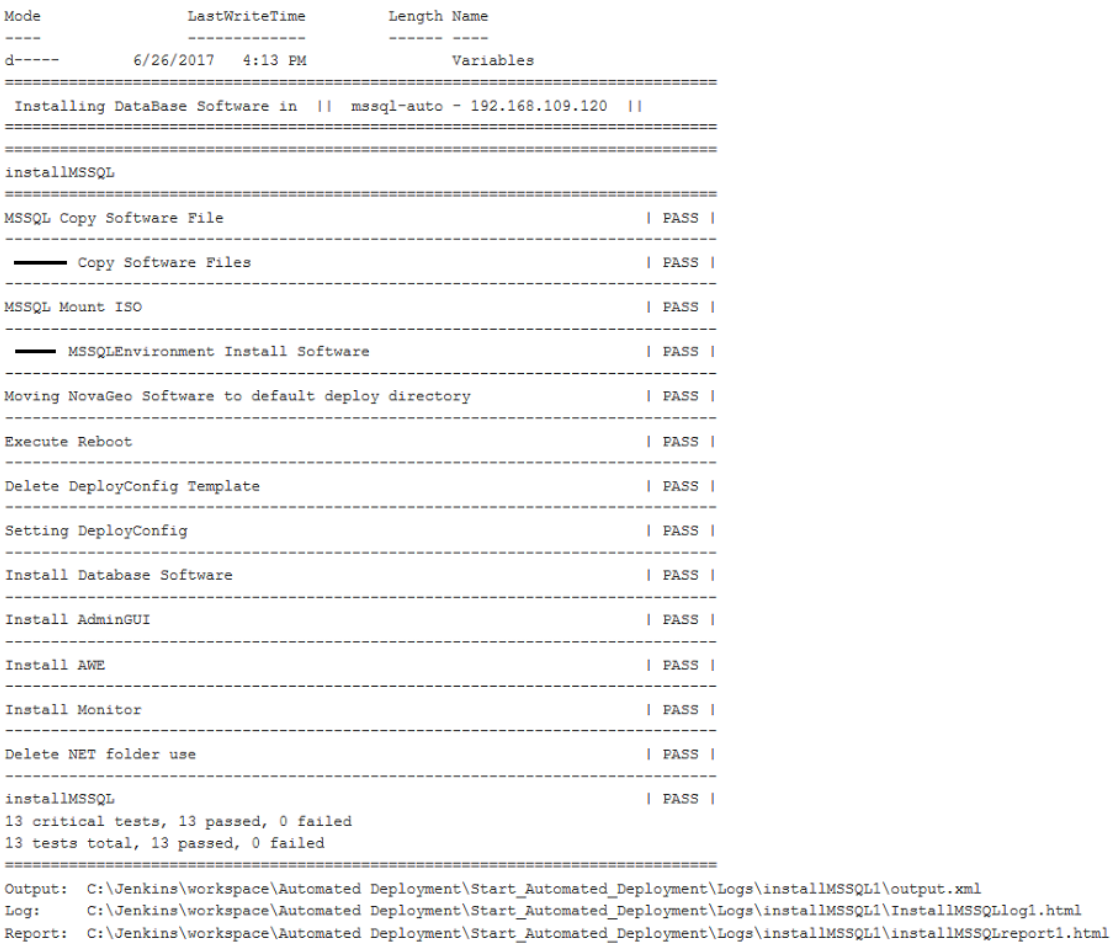

Figura 2.6: Output de un Trabajo Jenkins que llama a Robot Framework

Aquí podemos ver una ejecución de una *suite* llamada installMSSQL que contiene varios test. En nuestro caso nos servirán para copiar *software*, mover y modificar archivos y ejecutarlos.

2.6 Otras Herramientas

Además de estas herramientas principales, también se utilizan algunas herramientas secundarias. Estas se usan para resolver problemas que fueron apareciendo a lo largo del desarrollo debido a que no todos los componentes de la herramienta están preparados para una instalación desatendida, es decir, sin interacción con el usuario.

También y aunque no sea estrictamente una herramienta, sino más bien una funcionalidad, se ha hecho uso de las Tareas de Windows (*Scheduled Task*)[23] para conseguir ejecuciones que requerían ser locales ya que necesitaban de credenciales que no funcionaban por la problemática de salto múltiple de credenciales debido a las conexiones remotas.

2.6.1. AutoIT

AutoIT[24] es un herramienta de scripting. Su uso habitual es el testeo de aplicaciones en Windows. En nuestro contexto la utilizamos para conseguir una instalación desatendida a partir de un instalador que requiere una interacción con el usuario.

De esta forma, como el instalador siempre requiere de los mismos pasos, a través del script que nos proporciona AutoIT, podemos controlar las ventanas de un entorno Windows, pudiendo enviar controladamente clicks de ratón y señales de teclado.

Una de las cosas potentes de AutoIT es que no necesitas tener la herramienta instalada en la máquina donde vayas a ejecutar el script, ya que este se puede compilar fácilmente en un ejecutable ".exe". Por lo tanto para conseguir una instalación desatendida a partir de un *software* que no está preparado, basta con copiar el *software* y el ".exe" y ejecutar el ejecutable del script, el cual lanza la ejecución del *software*.

2.6.2. PuTTy

PuTTy[28] es un cliente SSH, Telnet, rlogin, y TCP raw con licencia libre. En el contexto del automatizador, al igual que AutoIT, se utiliza debido a los problemas ocasionados por instalaciones que no soportan la ejecución desatendida.

En este caso, a diferencia de AutoIT, se utiliza PuTTy para instalaciones en entornos Linux. No obstante, debido a que las máquinas Linux no disponen de entorno gráfico y que algunos componentes *software* que no tienen instalación desatendida utilizan la herramienta *"Dialog"*[29], se realizó una solución híbrida entre AutoIT y PuTTy.

El sistema operativo de la máquina Jenkins es un Windows, por lo tanto, para resolver la problemática de convertir una instalación que requiere interacción y que utiliza *Dialog*, se optó por abrir una conexión desde Jenkins a la máquina Linux a través de PuTTy, lanzando mediante AutoIT la instalación.

Esta solución planteaba varios problemas, AutoIT es una herramienta preparada para aplicaciones Windows y no es fácil leer una consola que realiza una conexión SSH[30]. Para saber en qué punto de la instalación se está, se monitoriza el log de instalación a través de un powershell que se descarga el log cada 10 segundos de la máquina Linux, luego lo lee e indica en qué punto de la instalación se encuentra.

Esto es debido a que los instaladores preguntan datos a mitad de la instalación, siendo un tiempo variable e indeterminado, por lo que no se podía automatizar secuencialmente una serie de pasos espaciados por un tiempo definido.

Sumado al control de flujo de la instalación aparece otro problema, conseguir un entorno gráfico remotamente a la máquina Jenkins. El sistema está pensado para lanzarse desde la web que proporciona Jenkins introduciendo un XML[31].

Para conseguir una sesión gráfica de forma remota en Windows Server 2012, es necesario realizarlo a través de los servicios de Windows, además de habilitar la sesión gráfica para estos y también permitir guardar las credenciales del servicio para poder ejecutarlo desde una conexión remota.

De esta forma se convierte el ejecutable del AutoIT en un servicio, el cual un Powershell lo instalará, controlará su ejecución y borrará de la máquina Jenkins.

Si bien es cierto que es una solución enrevesada, se debe de recordar que el proyecto se ha realizado en una empresa, y los componentes son los que son, algunos con instalaciones desatendidas y otros no.

CAPÍTULO 3
Arquitectura de la solución

3.1 Introducción

Como ya se ha mencionado con anterioridad, Jenkins es quien controla toda la ejecución. Tiene tres tipos de trabajo: un lanzador, trabajos de despliegue[32] de VM y trabajos de instalación.

El trabajo lanzador (*Start Automated Deployment*) controla la ejecución de los demás trabajos y los va llamando en bloques de dependencias. Los trabajos de despliegue de las VM son los que llaman a orchestrator y los trabajos de *"install"* llaman a unos *script* de powershell los cuales realizan las llamadas a Robot Framework.

- Trabajo Lanzador o *Launcher*.

  Este trabajo recibe como parámetro de entrada el XML de configuración y es el que llama al resto de trabajos respetando las dependencias internas entre componentes. Para realizar esta tarea, Jenkins dispone de un plugin *"trigger"* que sirve para lanzar otros trabajos. Además también se encarga de realizar una validación de los parámetros introducidos.

- Trabajo de Despliegue de la VM.

  Este tipo de trabajo es el que crea la máquina virtual.Se encarga de llamar a vRO a través de su *plugin* para Jenkins y espera su ejecución. Cuando este termina el trabajo ejecuta una comprobación para asegurarse que la VM cumple con los parámetros con los que se definió, zona horaria, red, discos, etc.

- Trabajo instalador. Es el encargado de instalar el *software*.

  Este trabajo solo tiene llamadas a *scripts* de powershell. Dependiendo del número de componentes que se instalan, tendrá el mismo número de llamadas a *scripts* powershell. Este *script* recoge las variables necesarias en tiempo de instalación generando un fichero que luego utilizará para llamar a Robot Framework. Después de cada instalación también se realiza una comprobación para saber si la instalación ha ido correctamente.

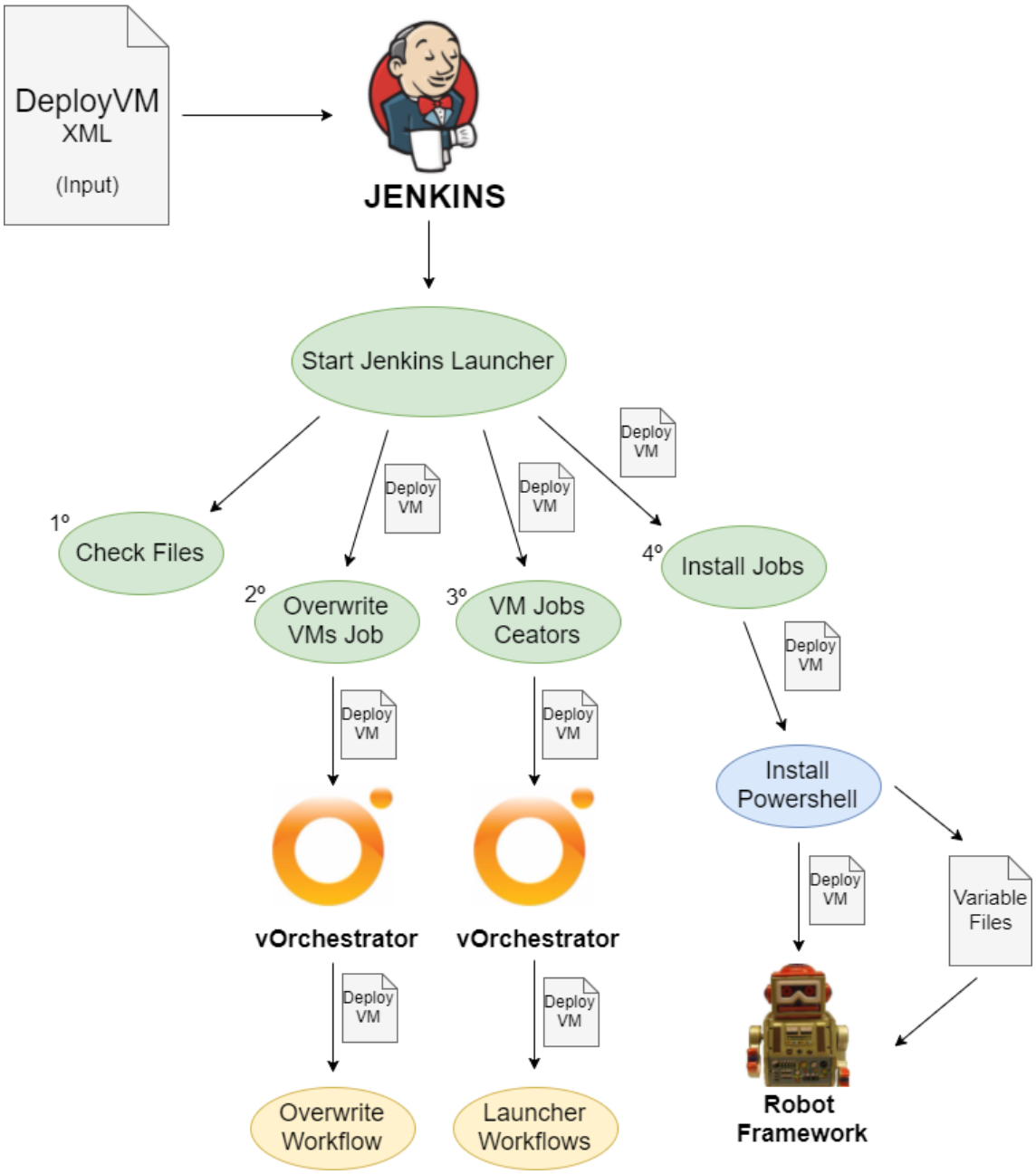

Figura 3.1: Arquitectura global del Automatizador

Los trabajos de Jenkins agrupan las instalaciones que se hacen en la misma máquina virtual y están en la misma fase.

Por ejemplo, la Máquina Virtual "Linux portal" tiene 4 trabajos. El trabajo de creación de la máquina virtual (cada máquina virtual tiene un trabajo de creación), el trabajo de instalación de un repositorio, el trabajo de instalación del entorno (Apache[33], MySQL[34], Tomcat[35] y MongoDB[36]) y un portal Web y el trabajo de instalación de un gestor y un "explorador", el cual hace una visualización de los datos del sistema en un mapa.

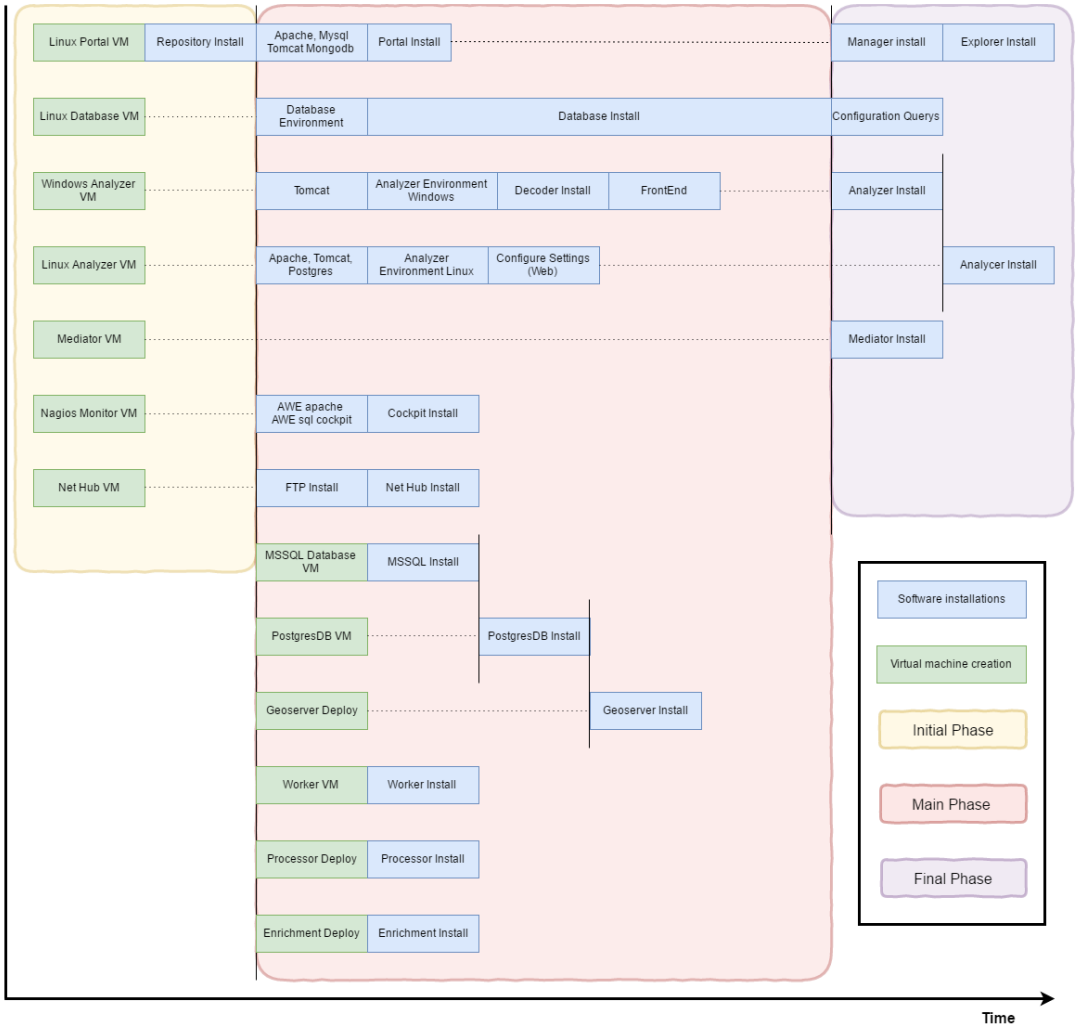

Figura 3.2: Secuencialidad de los trabajos Jenkins

Los módulos se han hecho lo más independientes posible. Ofreciendo la posibilidad de desplegar solo los componentes que queramos testear o cambiar en un entorno.

En la figura 3.3 se muestra un esquema básico de la organización de los trabajos de despliegue o creación de la VM y de instalación.

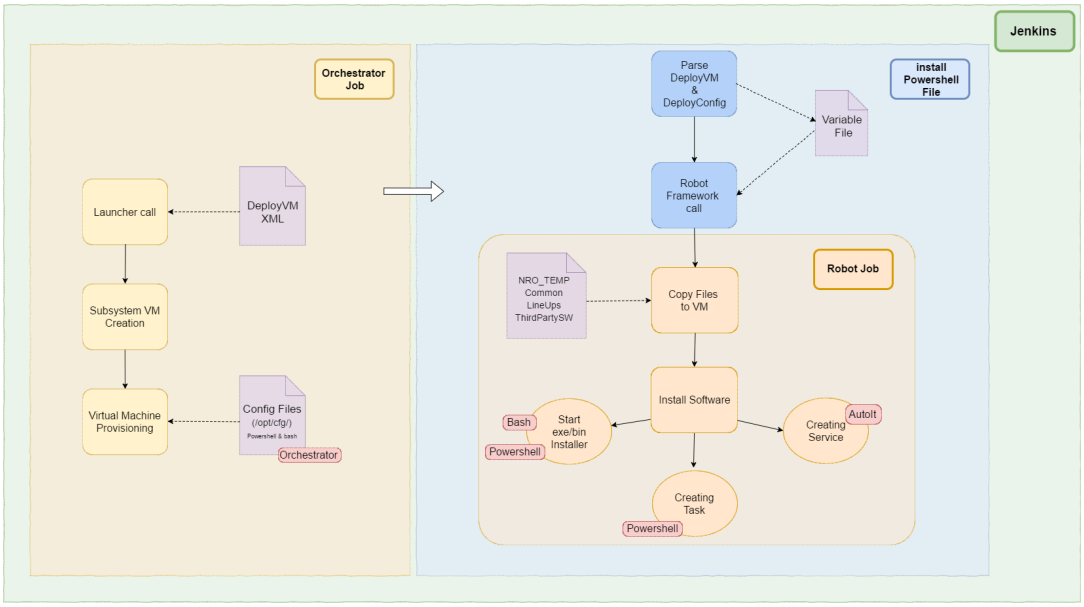

Figura 3.3: Arquitectura interna de los trabajos Jenkins

Aquí podemos ver la arquitectura del despliegue de un componente. Esta estructura está replicada N veces dependiendo del número de componentes y a su vez la parte referida a ”install powershell file” se repite N veces dependiendo de cuántas fases de instalación tenga el componente.

Como podemos ver en la figura 3.2, la arquitectura está englobada en tres fases. La fase inicial o primera fase, la segunda fase o fase principal y la tercera fase o la fase final. Estas fases están definidas en el trabajo ”Lanzador”, trabajo que controla el flujo de trabajo en Jenkins.

3.2 Fase Inicial

La Fase Inicial o *Initial Phase* es la primera fase de la automatización, en ella se comprueba que estén todos los ficheros necesarios y si se debe borrar máquinas que vayan a ser sobrescritas.

En esta fase también se crean la primera mitad de las Máquinas Virtuales (VMs) y se instala en una de ellas un repositorio el cual usarán el resto de VMs para resolver dependencias e instalar un paquete de seguridad hecho por la empresa.

3.2.1. Check Files

El trabajo ”Lanzador” recoge el XML de configuración de las máquinas. El primer trabajo que se llama es uno llamado *”Check Files”*. Este trabajo comprueba que todos los instaladores y ficheros de configuración necesarios en el despliegue que se va a realizar estén presentes y accesibles. En caso contrario el trabajo termina e interrumpe la ejecución indicando qué ficheros o instaladores faltan.

3.2.2. Sobrescritura de VMs

Si el trabajo anterior se ejecuta sin fallos, se pasa al trabajo de sobrescritura de las máquinas. Dependiendo de como se hayan configurado comprueba si las máquinas ya

existían previamente o son de nueva creación. Si ya existían y se ha indicado que se desean sobrescribir son borradas para dejar paso a las nuevas. Esta comprobación se puede realizar a nivel de cada máquina virtual.

3.2.3. Creación de la primera mitad de las VMs

Una vez las máquinas hayan sido eliminadas (si las hay), se procede a crear la primera mitad de las máquinas virtuales. El trabajo *Launcher* llama a los trabajos de creación de las VMs correspondientes de forma paralela. Estos trabajos llaman a vRO y este se encarga de crearlas y aprovisionarlas.

La creación de las máquinas virtuales depende del componente a desplegar, ya que dependiendo de este se utiliza una u otra plantilla de la cual partirá la VM. Actualmente hay seis plantillas, tres plantillas Linux, una RedHat 7.2 y dos RedHat 6.7 con configuraciones distintas y tres Windows Servers 2012 con diferentes configuraciones.

Cuando las máquinas están creadas, aprovisionadas y correctamente configuradas, se comprueba mediante un pequeño *testing* que todo haya ido correctamente: IP, número de discos, zona horaria y nombre de la máquina. Si todo ha ido correctamente se dan por finalizados los trabajos y se marcan como trabajos completados de forma satisfactoria.

3.2.4. Instalación del repositorio YUM

Esta primera fase también incluye la instalación de un repositorio yum, herramienta libre de gestión de paquetes para sistemas Linux basados en RPM. Este repositorio será utilizado por el resto de máquinas Linux las cuales lo utilizarán para instalar dependencias y el software de seguridad de la empresa. Esta instalación se realiza a través de un trabajo que se inicia cuando la VM en la cual se va a instalar ha terminado el trabajo de creación. Este trabajo llama a una *suite* de Robot Framework la cual realiza la instalación, si todo va bien el trabajo finaliza de forma satisfactoria.

Cuando todos los trabajos finalizan se da por completada la primera fase y se pasa a la segunda fase, o "fase principal".

3.3 Fase Principal

Cuando la fase inicial finaliza, se lanza la segunda fase. Esta llama de forma paralela a la instalación de las VMs que han sido desplegadas en la primera fase y lanza también, de forma paralela, la creación del resto de máquinas virtuales. Esta fase también lanza la instalación de estas últimas VMs.

3.3.1. Creación de la segunda mitad de las VMs

El entorno entero lo forman dos partes que funcionan de forma conjunta pero pueden trabajar ellas solas, por este motivo y para liberar carga en vCenter y los servidores anfitriones ESXi separé la creación de las máquinas virtuales, decrementando el tiempo de clonación de las plantillas.

Aquí se crean las VMs restantes mediante la llamada de cada trabajo a vRO. Cuando finalizan también pasan un *test suite* de Robot Framework que comprueba que el aprovisionado haya funcionado de forma correcta. Cuando finalizan llaman al trabajo correspondiente de instalación del software.

3.3.2. Instalación del Software

Cabe destacar las dependencias en tiempo de instalación que posee el entorno. Para solucionar este problema se diseñó un flujo de trabajo el cual indica qué podía instalarse de forma paralela y qué debe hacerlo de manera secuencial. En la figura 3.2 podemos ver dicho flujo.

Como podemos observar Geoserver[37] depende de la base de datos Postgres ya que necesita realizar una conexión en tiempo de instalación a dicha base de datos para finalizar su instalación. A su vez Postgres se despliega a través de un shell script el cual se genera cuando finaliza el despliegue de la base de datos de Microsoft SQL.

Las instalaciones software consisten en un powershell que recoge las variables necesarias a partir del XML de configuración y del XML de despliegue de la base de datos de Microsoft SQL. Después llama a la *suite* de Robot Framework la cual abre una conexión remota con la VM para realizar los pasos necesarios para conseguir la instalación automatizada.

Para completar configuraciones necesarias en la solución *software* se automatizan modificaciones que se realizan a través de portales web. Estas automatizaciones utilizan Robot Framework con una librería especial para automatizaciones web llamada Selenium[21]. Biblioteca típicamente usada en el *testing* web.

Gracias a la biblioteca Selenium y al controlador para el navegador web que deseemos utilizar, somos capaces de ejecutar comandos javascript, obteniendo el control del portal web para realizar las configuraciones necesarias.

En el caso concreto de la solución software, hay dos servidores que realizan análisis de las mediciones. El primero actúa como maestro y el segundo como esclavo. Para que los dos puedan interactuar, el esclavo debe registrarse en un portal web servido por el maestro. Además debe configurarse el modo el cual van a comunicarse entre ellos, entre otras modificaciones menores.

Todas estas configuraciones se hacen a través del portal web utilizando Robot Framework a través de la biblioteca de Selenium.

3.4 Fase Final

En la tercera fase se instalan los componentes que dependen en tiempo de instalación del primer bloque de máquinas virtuales. En esta fase también hay una pequeña dependencia entre el "Analizador" Windows y el Linux.

Algunos componentes se comunican con las bases de datos insertando datos, otros son paquetes personalizados por la empresa que configuran software de terceros como en el caso de los Analizadores los cuales se basan en Microstrategy, una plataforma de análisis empresarial. Estos componentes son los que engloban la fase final.

Por otra parte, debido a que algunos componentes necesitan de configuración después de su instalación, por ejemplo indicar el número de fuentes de datos disponibles para el sistema o activar ciertos parámetros en ficheros de configuración, se configuran en última instancia una vez el software está completamente instalado.

3.5 vRealize Orchestrator

Dentro de vRO disponemos de dos tipos de *workflows*. Un *workflow* es un flujo de trabajo definido, dentro de él podemos hacer uso de otros *workflows*. Internamente los he distribuido en dos secciones (”Miscellaneous”), trabajos misceláneos y”Launchers and Deploy”, donde se encuentran todos los trabajos Launcher, estos trabajos son los que se llaman desde Jenkins, y el trabajo ”Deploy Subsystem” del cual hablaremos más adelante.

3.5.1. Launchers and Deploy

Cada componente (tipo de servidor) tiene su ”Launcher”. Estos trabajos tienen dos fases diferenciadas, la primera gestiona si la máquina virtual definida en el XML se desplegará o ya existe. En el fichero de configuración XML que proporcionamos al sistema, damos la posibilidad de sobrescribir una carpeta o una máquina virtual. Esta primera fase se encarga de buscar de entre todos los Host ESXi que están asociados a vCenter, si encuentra la máquina o la carpeta y esta está marcada como ”Overwrite”, Orchestrator la elimina. En caso de que la máquina exista y esta no esté marcada con la opción overwrite el sistema la elimina del vector que contiene los entornos a desplegar.

Después, en la segunda fase se realiza la creación y configuración básica de la máquina virtual mediante la llamada al *workflow* ”Deploy Subsystem”. Este workflow es el que crea todas las máquinas virtuales y lo único que cambia de una ejecución a otra son las variables introducidas.

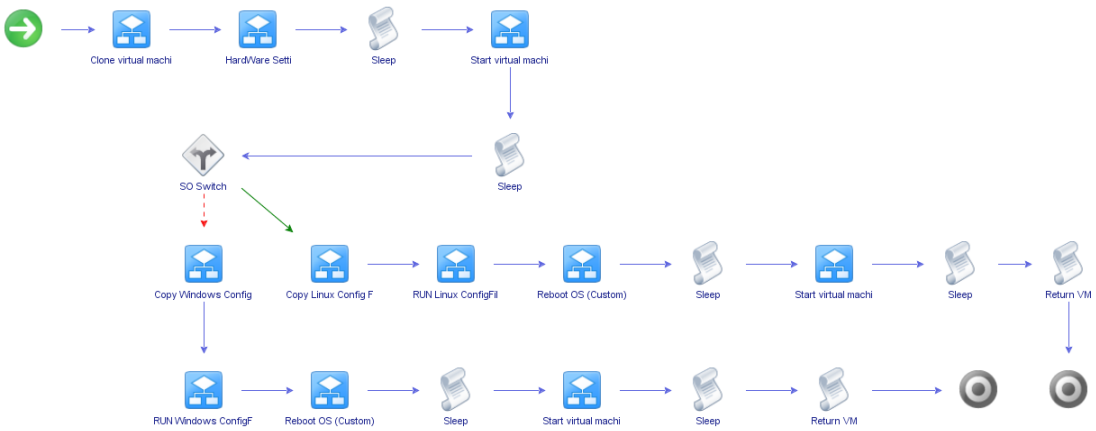

Figura 3.4: Workflow de despliegue

El *deploy subsystem* es válido tanto para Linux como Windows. Lo primero que realiza es el clonado de la plantilla que corresponde al componente a desplegar. Después se configura el *hardware* y se enciende la máquina. Después se copian unos ficheros de configuración, los cuales configuran la franja horaria del servidor, la configuración de red, el idioma del teclado, el hostname, los host que deben aparecer en el fichero de hosts (tanto Windows como Linux) y particionan y configuran los discos añadidos. Como estos ficheros se copian y ejecutan de forma distinta dependiendo del sistema operativo, el *workflow* se divide y la ejecución irá por una rama u otra dependiendo del SO.

3.5.2. Miscellaneous

El segundo conjunto contiene los *workflows* customizados que se utiliza en Deploy Subsystem y los launchers. vRealize Orchestrator, ya tiene definidos ciertos *workflows* básicos, como pueda ser *"Add Disk"*, que añade un disco a una máquina virtual, pero podemos unir este *workflow* a uno que modifique la memoria RAM y el número de procesadores y podemos llamarlo *"Hardware Settings"*. Así conseguiremos mayor nivel de abstracción y tendremos un código más modular y reutilizable.

Dentro de este grupo cabe destacar el *workflow (*Parse XML), este trabajo es el encargado de leer el XML y rellenar los vectores de tipo compuesto *"ServerType"*. Este Tipo es un tipo customizado que recoge todas las variables necesarias para desplegar un servidor, desde el número y tamaño de los discos, servidor anfitrión donde se creará, configuración de red, etc.

3.6 Robot Framework y Powershell

Una vez finalizado el *workflow* de orchestrator, se pasa a la fase de instalación del *software*. Esta parte está controlada por Robot Framework el cual es llamado y configurado a través de un *script* powershell. A este tipo de powershell lo llamaremos *install powershell file*.

3.6.1. Install Powershell files

Cada componente posee un *install powershell file*. Los ficheros *install powershell file* se dividen en dos partes, la primera crea un fichero de variables parseando el XML de configuración y la segunda llama a Robot Framework.

Los *install powershell file* se llaman a través de los trabajos de instalación de Jenkins.

Recogida de variables

Cada *install powershell file* comprueba si el servidor donde va a instalar el componente tiene activada la opción de *Install* a *True*. En caso contrario termina la ejecución.

Si continúa con la instalación, se genera un fichero de variables clave/valor. Estas se recogen a través del XML de configuración y del XML de despliegue de la base de datos de Microssoft SQL.

LLamadas a Robot Framework

La segunda sección de los *install powershell file* realiza la llamada a una *suite* de Robot Framework utilizando el fichero generado de variables. Esta llamada instalará el *software* necesario.

Una vez terminada la instalación el *install powershell file* lanza otra *suite* de Robot Framework. Esta vez el objetivo será testear la correcta instalación del componente.

Desde Jenkins podemos seguir el estado de las instalaciones a través de la salida de consola del trabajo. Un ejemplo de salida lo podemos ver en la figura 2.6.

3.6.2. Estructura de Robot Framework

Suites de Instalación

Las instalaciones de los diversos componentes están divididas en *suites*, estas *suites* contienen diferentes *Tests*, cada *Test* es una fase de la instalación y a su vez están compuestos de *keywords*. Desde aquí se copia el *software* necesario y se lanza la ejecución de este.

Gracias a la orientación de este *framework* al testing, posee una fuerte estructura para comprobar la correcta ejecución de cada *keyword* contenida en los test. Por lo tanto a medida que vamos copiando *software* o instalándolo, podemos comprobar el código de retorno de la ejecución.

El objetivo de estas *suites* de Robot Framework es conseguir una shell o powershell con privilegios. Desde esa conexión remota se toma el control de la máquina para proceder con la instalación.

En el caso de Windows y algunas instalaciones de Microsoft se debe conseguir ejecuciones locales, por ello se ha hecho uso del Scheduler Task. También ha sido necesario el uso de sesión gráfica, utilizando servicios con la cuenta de sistema se ha conseguido este objetivo.

Keywords

Las *keywords* son las instrucciones que ejecutan los *Test*. Una de las fortalezas de Robot Framework es que estas *keywords* se comportan como funciones y podemos definir nuestras propias *keywords*. Para ello podemos utilizar otras *keywords* de bibliotecas oficiales de Robot Framework o bibliotecas que otros desarrolladores han compartido.

Para conseguir las conexiones remotas he construido keywords que abstraen esta conexión. Robot Framework ofrece la posibilidad de definir entre otras cosas *Suite setups* y *Suite Teardowns*. Estos dos elementos son "precondiciones" y "postcondiciones". De esta manera se llama a la *keyword* que crea la conexión en el *Suite setup* y en el *Suite Teardown* se llama otra *keyword* que destruye la conexión.

Gracias a estas conexiones podemos conseguir por ejemplo *keywords* que ejecutan comandos powershell de forma remota. Un ejemplo simplificado de instalación en Windows sería:

- Comando Powershell que resuelve el Path del instalador. (Esto es así debido a que las carpetas que contienen los instaladores pueden variar la versión)
- Comando Powershell que recoge el Path de la *keyword* anterior y copia el instalador a la VM destino.
- Comando Powershell que lanza la ejecución con las variables recogidas en los *install powershell file* y espera a que finalice.

CAPÍTULO 4
Infraestructura de la solución

4.1 Infraestructura de la solución

Hemos hablado de diversas herramientas, Jenkins, Robot Framework, Powershell, etc. Pero ¿quién aporta la infraestructura para ejecutarlas?

Para facilitar la implantación del automatizador de despliegues, se ha creado una imagen de una máquina virtual (OVA)[4] la cual contiene todas estas herramientas configuradas y las dependencias resueltas.

De esta forma podemos transportar la herramienta donde queramos sin tener que preocuparnos. Además de dos scripts de configuración que al ejecutarlos pondrán a punto el automatizador de forma sencilla.

También es necesaria la infraestructura de vCenter y la máquina vRealize Orchestrator. Esta infraestructura la proporciona VMware y es necesaria junto a vCenter para poder utilizar el automatizador.

4.2 VMware: vRealize Orchestrator y vCenter

Lo primero que necesitamos es un vCenter. Recordamos que vCenter es una interfaz web que nos facilita la administración de servidores ESXi. VMware ofrece este entorno a través de una OVA la cual deberemos convertir a máquina virtual dentro de un Host anfitrión. Su configuración es sencilla y está documentada[7] por la misma empresa VMware.

Una vez tengamos un vCenter que gestione al menos un Host ESXi debemos desplegar el segundo componente necesario, vRealize Orchestrator. VMware también ofrece esta herramienta a través de una OVA. Deberemos convertirla a máquina virtual de la misma forma que vCenter. Después deberemos enlazar vRO con vCenter.

Estas dos son máquinas basadas en Linux las cuales contienen de forma embebida las herramientas y necesitan de una licencia para su uso.

4.3 Máquina Automatizadora

Toda la automatización recae sobre la máquina automatizadora. Esta es una máquina Windows donde están instaladas y configuradas todas las herramientas necesarias. Además también posee todas las dependencias resueltas.

Esta máquina contiene: Jenkins, Robot Framework, Powershell 4.1, Selenium Library con el driver correspondiente para Google Chrome, PuTTy, NSSM (Una herramienta para gestionar servicios de Windows), entre otros. Además de tener configuradas de forma correspondiente todas las políticas de seguridad y permisos para los movimientos de ficheros, ejecuciones remotas, etc.

Esta máquina dispone de una carpeta con el repositorio de todos los *test suite* de Robot Framework y todos los ficheros y *scripts* de configuración necesarios.

Esta implementación ha sido necesaria ya que la herramienta se está utilizando en diferentes áreas de la empresa. Cada área dispone de su propio vCenter, vRO y una máquina automatizadora. Además uno de los objetivos es la portabilidad de la herramienta para poder utilizarla fuera del entorno habitual de la empresa.

4.3.1. Configuración del Automatizador

Para configurar la herramienta automatizadora se han creado dos scripts que facilitan la tarea. El primero configura vRO y el segundo configura Jenkins.

Como ya se dijo en apartados anteriores, vRO necesita de *workflows* creados especialmente para la automatización. Estos *workflows* se encuentran en una carpeta en la máquina automatizadora.

Para configurar vRO simplemente se debe llamar al script y pasarle como argumento la IP de la máquina vRO. Este script se encarga de importar todos los *workflows* necesarios a la máquina vRO a través de un módulo de Powershell, PowervRO [26, 27], creado especialmente para gestionar máquinas de vRO. Los *workflows* necesarios están incluidos en una carpeta dentro de la máquina automatizadora.

Jenkins viene sin trabajos definidos en la OVA de la máquina automatizadora, solo incluye los plugins necesarios. Para configurar Jenkins debemos proporcionar al script la ruta de los trabajos, presentes en una carpeta dentro de la máquina automatizadora. Además debemos proporcionarle la IP de la máquina de vRO ya que los trabajos de Jenkins hacen llamadas a vRO y necesitan su IP. Esta IP se modifica en los trabajos de Jenkins, definidos en XML y posteriormente se importan a Jenkins.

4.3.2. Actualizaciones y mantenimiento

Debido al cambio de componentes, funcionalidades o posibles mejoras, se ha diseñado para que una actualización fuera rápida y simple. La máquina automatizadora se compone de dos núcleos. El repositorio de código donde se incluyen todos los scripts, ficheros de configuración y *suites* de Robot Framework forma el primer núcleo y se encuentra en una carpeta en la OVA.

El segundo núcleo lo forman los trabajos de Jenkins. Estos trabajos están definidos en XML y también están guardados en una carpeta dentro de la OVA.

Para actualizar o cambiar de versiones lo único que hay que hacer es sustituir la carpeta del repositorio con el nuevo y llamar al *script* de configuración de Jenkins indicándole la ruta de los trabajos que queremos sustituir.

De esta forma se consigue poder hacer cambios en la automatización sin que suponga un coste elevado.

CAPÍTULO 5
Desarrollo de la solución

5.1 Fichero XML de configuración (DeployVM)

El XML de configuración es el único parámetro de entrada al sistema de automatización, aparte del *software* que se quiera instalar. También cabe destacar otro XML de configuración de despliegue de la base de datos de SQL Server[38], el cual también se debe proporcionar si se quiere desplegar dicho componente.

La primera zona del XML es una zona general común a todos los entornos y donde se especifica qué *software* vamos a instalar.

```xml
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->
    <!--                         Automatic Deployments CFG                           -->
    <!--                                                                             -->
    <!--                                Version 1.0                                  -->
    <!-- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -->


<!-- TimeZone must be like "Europe/Madrid" -->
<!-- If RepositoryIP is setted all environments will likend with that Repository, You should remove Attribute if you want new repository -->
<!-- Autoconfigure_etc_Hosts will get INTRA IP. If there is not in XML will get ADMIN IP -->
<!-- ThinProvisioned False means Eager Zeroed Provisioning, If use ThinProvisioned True deploy will be faster, but disk usage will be slower -->
<AUTO TimeZone="Europe/Madrid" Autoconfigure_etc_Hosts = "True" ThinProvisioned = "False" MobaSessions = "False">

    <!-- /!\ C A U T I O N  /!\ -->
    <!-- IF YOU OVERWRITE VM_FOLDER, YOU WILL DELETE ALL VM CONTAINED -->
    <VM_FOLDER Folder = "XXXXXXXXXX" Overwrite = "False" />


    <NETWORK>

        <!-- NETWORK ADAPTERS CONFIGURATION --><!-- Delete or comment interfaces u will not use -->
        <ADMIN_NETWORK Gateway ="XXX.XXX.XXX.XXX" Mask = "XXX.XXX.XXX.XXX" DNS = "XXX.XXX.XXX.XXX" />
        <USER_NETWORK Gateway ="XXX.XXX.XXX.XXX" Mask = "XXX.XXX.XXX.XXX" DNS = "XXX.XXX.XXX.XXX" />
        <INTRA_NETWORK Gateway ="XXX.XXX.XXX.XXX" Mask = "XXX.XXX.XXX.XXX" DNS = "XXX.XXX.XXX.XXX" />
        <INTER_NETWORK Gateway ="XXX.XXX.XXX.XXX" Mask = "XXX.XXX.XXX.XXX" DNS = "XXX.XXX.XXX.XXX" />

    </NETWORK>

    <!-- If you set "True" InstallAll attribute, you will install solution in all NovaGeo servers deployed -->
    <!-- SRTM avaiables ZONES -> All|"Africa","Australia","Eurasia","Islands","North_America","South_America"-->
    <SOFTWARE LineUp = "XXXXX" RepositoryIP = "X.X.X.X" SRTM="XXXXX" />
```

Figura 5.1: Zona de parámetros generales del XML de configuración

En esta zona podemos configurar la zona horaria de los servidores. Elegir el tipo de aprovisionamiento, el tipo de aprovisionamiento define cómo vamos a crear la máquina virtual, si deseamos reservar todo el espacio de almacenaje o va a ir creciendo dinámicamente. También tenemos la opción de configurar el fichero de *Hosts* añadiendo en todos los servidores todos los servidores con su hostname (igual al nombre de la máquina virtual) con su correspondiente IP.

MobaXterm[39] es una herramienta de administración remota, pudiendo guardarte conexiones a máquinas remotas, entre otras funciones, como la de ejecutar simultáneamente lo mismo en varias máquinas. La automatización da la posibilidad, mediante un pequeño script, de generar un fichero que después podremos importar desde MobaXterm con todas las conexiones guardadas y configuradas, preparadas para administrar las máquinas que estamos desplegando.

vCenter da la posibilidad de agrupar en carpetas máquinas virtuales. Aprovechando esta funcionalidad, se da la opción de elegir el nombre de la carpeta y además, si queremos borrar las máquinas que contiene una carpeta para crearlas de nuevo, podemos indicar *Overwrite* a *True*.

En la zona general es donde configuraremos las cuatro interfaces de red, pudiendo elegir la puerta de acceso, máscara de subred y DNS.

Por último elegiremos qué *software* vamos a instalar. Para ello indicaremos qué versión de LineUp vamos a utilizar. Este lineup es una carpeta que contiene todos los componentes *software* con la versión que queramos de cada uno de ellos.

El entorno dispone de un repositorio *yum* interno el cual se utiliza para instalar pequeños paquetes y actualizaciones de los entornos Linux además del *software* de seguridad de la empresa. Podemos elegir la dirección de este repositorio. Si queremos instalar y utilizar uno incluido en el despliegue basta con poner su IP aquí, en cambio, si queremos uno externo debido a que no vamos a desplegar uno, podemos poner la IP que deseemos.

Como ya hemos mencionado con anterioridad. Los SRTM[10] son una serie de ficheros generados por la NASA los cuales contienen la topología de la tierra. Una de las funciones del *software* es dibujar mapas con una serie de datos de interés. Por esto, es aquí donde elegiremos qué zonas geográficas queremos cargar en el sistema, ofreciendo la posibilidad de elegir una, combinar las que queramos o directamente todas.

El XML DeployVM se ha intentado hacer lo más versátil posible, pudiendo borrar servidores a desplegar o incluso poniendo varios de este tipo.

```
<!--
If you dont want deploy a server u must comment or delete it.
-->

    <DATABASE_SERVERS>

        <!-- IF YOU OVERWRITE SERVER YOU WILL DELETE VM WITH SAME NAME -->
        <DATABASE Overwrite = "False" Install = "True" >

        <!-- VMWARE SETTINGS-->

            <VM_NAME Name = "XXXXXXXXXX" />
            <HOST Host = "XXX.XXX.XXX.XXX"/>
            <DATASTORE Datastore = "XXXXXXXXXX" />
            <POOL Pool = "Resources" />


        <!--HARDWARE SETTINGS -->

            <!-- RAM in MB -->
            <RAM_MB Ram = "XXXX" />
            <!-- Number of Virtual Cores -->
            <VCORES VCores = "X" />
            <!-- EXTRA Hard Disk Size in GB -->
            <DISK_GB_1 DSize ="XXX" Datastore = "XXXXXXXXXX"/><!-- D: CallList -->
            <DISK_GB_2 DSize ="XXX" Datastore = "XXXXXXXXXX"/><!-- E: MessageFlow -->
            <DISK_GB_3 DSize ="XXX" /><!-- F: Data -->
            <DISK_GB_4 DSize ="XXX" /><!-- G: Logs -->
            <DISK_GB_5 DSize ="XXX" /><!-- H: TempDB -->
            <DISK_GB_6 DSize ="XXX" /><!-- I: TempDB -->
            <DISK_GB_7 DSize ="XXX" /><!-- J: TempDB -->
            <DISK_GB_8 DSize ="XXX" /><!-- K: TempDB -->
            <DISK_GB_9 DSize ="XXX" /><!-- L: Backups -->


        <!-- NETWORK SETTINGS-->

            <!-- Admin IP will be use for share folder with Parser & Processor -->
            <ADMIN_IP IP = "XXX.XXX.XXX.XXX" />
            <USER_IP IP = "XXX.XXX.XXX.XXX" />
            <INTRA_IP IP = "XXX.XXX.XXX.XXX" />
            <INTER_IP IP = "XXX.XXX.XXX.XXX" />

        </DATABASE>

    </DATABASE_SERVERS>
```

Figura 5.2: Ejemplo de configuración de un servidor en el XML de despliegue

Aquí podemos ver una zona de la plantilla de despliegue donde configuramos los servidores "Database" correspondientes al servidor de MSSQL. En esta zona se encuentran las diferentes opciones configurables.

Aparte de lo básico como: nombre de la máquina virtual, Host anfitrión, Datastore (Discos de almacenamiento del servidor anfitrión, normalmente hay varios), etc. también se ofrece la posibilidad de elegir si sobrescribimos la máquina, es decir, si hay una máquina con el mismo nombre, el sistema la borra y despliega la nueva, si no se desea sobrescribir no se despliega la máquina. También disponemos de la opción de install. Con ella decidiremos si queremos instalar el *software* o solo una máquina virtual configurada.

El objetivo de este XML es ser lo más flexible posible. De esta manera si queremos reinstalar *software* en una máquina virtual ya configurada podemos elegir *"Overwrite"* a *false* e *"Install"* a *True*.

Las máquinas disponen siempre de cuatro interfaces de red. El XML da la opción de configurar la combinación que deseemos, siendo la de *Admin* la única obligatoria.

En el caso concreto de un despliegue del servidor *Database* no tiene sentido poner más de uno, pues la solución *software* no está preparada para trabajar concurrentemente.

No obstante otros servidores sí que están preparados, es por ello que todos los componentes sin excepción (con vistas a futuro) tienen la posibilidad de desplegarse el número de instancias que queramos.

Para realizar esto siguiendo con el ejemplo del servidor *Database* sería tan sencillo como duplicar tantas veces como queramos toda la etiqueta *"DATABASE"* y añadirlo dentro de *"DATABASE SERVERS"*, modificándolo pertinentemente.

5.2 Aprovisionamiento de la Máquina Virtual

La primera etapa es el aprovisonado de las VMs, es decir, la creación y configuración hardware. Esta tarea la realiza vRealize Orchestrator.

5.2.1. Parse XML

Parse XML es el workflow que procesa el XML de entrada y lo transforma en variables que después se utilizarán para crear las máquinas virtuales. Este workflow solo consta de un script el cual recibe como string o como fichero el XML. El script llama a la función *"parseServer"* por cada servidor definido, aquí se crea un vector de servidores con las variables necesarias para ser desplegados.

Este vector es de un tipo compuesto personalizado el cual recoge todas las variables necesarias. Entre ellas se encuentra, el disco de almacenaje donde será alojada la VM, qué plantilla de VM va a ser utilizada, qué host anfitrión alojará la VM, el número de memoria RAM y procesadores, las IPs de la futura VM, entre otras variables internas al proceso.

5.2.2. Deploy Subsystem

Una vez definidos los servidores en variables, se llama al workflow *"Deploy Subsystem"*. Este workflow es el eje angular del aprovisonado y recibe como argumento de entrada un servidor definido por el tipo compuesto.

Como primer paso clona una plantilla mínima definida en el tipo compuesto, después cambia la memoria RAM, modifica el número de procesadores que utilizará y le añade el número de discos correspondientes con el tamaño deseado.

Una vez clonada y hecha la configuración *hardware* se enciende la máquina virtual. Es aquí donde se bifurca el flujo de trabajo, dependiendo de qué sistema operativo contenga la VM.

Los dos caminos realizan la misma funcionalidad, copian scripts de configuración (powershell o shell) los ejecutan y reinician las máquinas. La diferencia de caminos es debido a la forma de copiar y ejecutar estos scripts.

Linux necesita de dos scripts de configuración. El primero da formato a los discos y añade sus puntos de montaje. Este script es único para cada tipo diferente de servidor.

El segundo configura la configuración de red, la zona horaria, el idioma de la consola, el *hostname* y el fichero de *hosts*. El sistema soporta servidores RedHat 6 y RedHat 7. Debido al cambio de algunos ficheros de configuración y el tratamiento de los servicios, hay dos scripts de configuración, uno por versión de SO.

Windows en cambio se configura con tres scripts de Powershell. Además requiere de un reinicio extra a diferencia de los Linux. Los dos primeros scripts son similares en funcionalidad a los del Linux, en cambio para definir una zona horaria y un idioma para

una sesión se debe ejecutar con las credenciales de esta sesión. VRO utiliza la cuenta de ”localsystem” es por ello que hay un tercer script que es llamado con las credenciales correctas.

Orchestrator lanza la ejecución de estos scripts, pero no sabe cuando finaliza la ejecución. Mediante la API de vRO se puede conseguir el PID de la ejecución y monitorizarla, pero el tiempo que hay entre que el script finaliza y que vRO sabe que ha finalizado es alto, del orden de minutos. Por este motivo se decidió crear un fichero de control al final del script. La monitorización de la ejecución consiste en buscar este fichero y eliminarlo, reduciendo a segundos este tiempo de espera.

Una vez finalizada la ejecución de los scripts, se reinicia la máquina y finaliza el *workflow*.

5.3 Instalación del Software

Una vez finalizado el aprovisionamiento de la máquina virtual, se procede a la instalación de *software*. Este proceso se realiza mediante powershell y Robot Framework.

Cada componente dispone de un script de Powershell el cual inserta las variables de la instalación en un fichero que luego será utilizado en la ejecución de Robot Framework.

Este powershell es el que comprueba la opción de *Install* del XML de despliegue, si el servidor está marcado como ”instalable” se produce la llamada a Robot Framework.

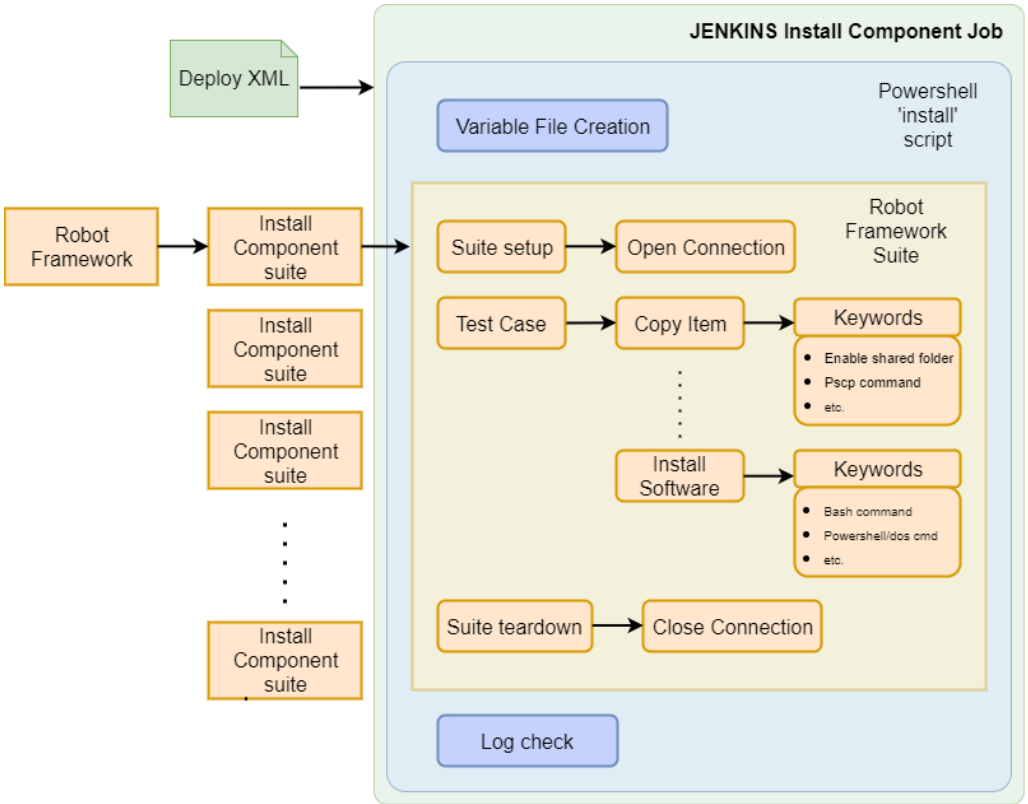

Figura 5.3: Organización de Robot Framework

Este diagrama muestra un ejemplo mínimo de un trabajo Jenkins de instalación.

5.3.1. Robot Framework

Robot Framework es una herramienta típicamente usada para testing, por lo tanto tiene una funcionalidad muy potente a la hora de comprobar ejecuciones. Cada paso de la ejecución comprueba el código de retorno. Este factor es importante ya que la automatización también comprueba la correcta integración de la solución *software* de los instaladores, no solo proporciona entornos para *testing*.

Además la herramienta posee una poderosa infraestructura de logs, los cuales a través de la integración con Jenkins podemos dar un trabajo como fallido si algo no funciona como se espera. Después si queremos ver en detalle dónde ha ocurrido el fallo podemos ver toda la traza de la ejecución.

Cada componente de instalación dispone de un suite en Robot Framework. Cada suite dispone de un *suit setup* y *teardown* que depende del SO donde se instale el componente. La función de esta ”precondición” y ”postcondición” es crear la conexión y destruirla.

Para realizar las conexiones se utilizan keywords que se han creado específicamente para el *software* de seguridad de la empresa.

Para Windows se utiliza la biblioteca WinRM y el *suit setup* consiste en habilitar remotamente esta funcionalidad en el servidor y crear una sesión remota de powershell, la cual se utilizará a lo largo de la instalación.

En Linux el *suit setup* se encarga de loguearse con la máquina utilizando la biblioteca SSH de Robot Framework. No obstante debido a las medidas de seguridad de la empresa, no se permite la conexión SSH de root. Para conseguir ejecutar comandos como root se ha realizado una keyword ”Su And CMD” que eleva privilegios ejecutando su y el comando que se le proporcione.

Esta keyword utiliza la biblioteca de SSH y escribe ”su” en la consola abierta mediante el *suit setup*, después se espera hasta leer la palabra ”Password:” después escribe la contraseña de root y espera hasta encontrar el carácter ”#” el cual dice que disponemos de una consola con privilegios de root. Una vez somos root ejecutamos el comando que pasamos como argumento a la keyword, y esperamos hasta volver a leer ”#” para saber que el comando ha finalizado. Cuanto termina la ejecución escribimos echo $? que nos proporciona el código de retorno y proporcionamos este código como salida de la keyword.

Esta keyword se utiliza en todas las instalaciones de los SO Linux menos en la propia instalación del *software* de seguridad,la que impone esta restricción.

La estructura de las diferentes suites es similar, primero se copia el *software* y ficheros necesarios para la instalación. Usando pscp y copy o xcopy dependiendo del SO destino.

Después se ejecutan los instaladores con las variables necesarias y si es necesario se configuran ciertos ficheros para completar la instalación.

5.3.2. Powershell y Jenkins

Jenkins está montado sobre una máquina Windows, es por esto que se utiliza Powershell para hacer las llamadas a Robot Framework y para ejecuciones ”locales” en Robot Framework que también es ejecutado en la máquina de Windows de Jenkins.

Cada componente a instalar dispone un Powershell. Este recoge las variables a partir del XML de despliegue, guardándolas en un fichero. Después hace la llamada a Robot Framework incluyendo este fichero de variables.

Estos Powershell de instalación disponen de la lógica que hace posible elegir en el XML de despliegue si queremos instalar el componente o no.

Powershell no sólo se utiliza para estos scripts de instalación. También es utilizado para dar funcionalidades a Robot Framework. Por ejemplo reiniciar máquinas de manera controlada de forma remota.

También aporta la lógica para crear servicios y tareas programadas, indispensables para las soluciones alternativas generados por los paquetes de *software* que no están preparados para una instalación sin interacción con el usuario.

CAPÍTULO 6
Workaround o Soluciones Alternativas

6.1 La importancia de las soluciones alternativas

Debido a usar software real diseñado por una empresa, no es fácil modificarlo o pedir que se modifique. Por lo tanto hay que ser flexible a la hora de imaginar soluciones que salten o sorteen los problemas que aparezcan.

El problema más común al que me he enfrentado ha sido componentes software no compatibles con una instalación en modo silencioso, es decir, sin interacción con el usuario. Pero también han aparecido otros problemas como el salto de credenciales en ejecuciones remotas en Windows o conseguir ejecuciones desde una ruta concreta en Linux.

6.2 Simular instalaciones silenciosas

Para realizar una automatización es necesario conseguir una instalación silenciosa. Si el instalador espera una interacción con el usuario ya no será automático y deberemos estar atentos en tiempo de ejecución.

Tenemos que diferenciar varios tipos de simulación. Instalaciones en Windows desde la propia máquina desplegada, instalaciones en Linux desde la máquina de Jenkins y configuraciones Web desde la máquina de Jenkins.

Cabe destacar que si bien se necesita modificar parámetros de seguridad para conseguir implementar estas soluciones alternativas, se revierten los cambios realizados para dejarlas en su estado original.

6.2.1. Instalaciones con ejecución local en Windows

Si el elemento que queremos instalar es para el Windows, los servidores disponen de sesión gráfica. Por lo tanto se puede utilizar herramientas como AutoIT que automatizan la GUI de Windows.

Usando este lenguaje de scripting podemos automatizar paso a paso una instalación que requiera de interacción con el usuario. Con ella podemos simular clicks en botones e incluso inputs de teclado. Además podemos monitorizar cambios en las ventanas para no perder el flujo de la instalación.

Gracias al compilador a fichero ejecutable del que dispone AutoIT no hace falta instalar ninguna librería adicional en la máquina donde se vaya a ejecutar teniendo que copiar simplemente el fichero compilado resultante.

Aun teniendo la instalación automatizada a través de un ejecutable, debido a que Windows Server 2012 (SO de las máquinas Windows) no permite sesiones gráficas remotas, tuve que instalar este ejecutable como servicio, realizando las correspondientes habilitaciones para los servicios con sesión gráfica.

De esta forma a través de un script Powershell llamado desde Robot Framework, el cual crea el servicio, lo ejecuta, monitoriza y borra cuando finaliza, se consigue la automatización de la instalación.

En este artículo [3] podemos ver problemas y soluciones alternativas típicas en las máquinas Windows causados por ejecuciones remotas vía WinRM o Powershell remoto.

En el caso de la Automatización también se ha tenido que utilizar el *Scheduler Task* de Windows para conseguir ejecuciones locales.

Después *Robot Framework* a través de *Powershell* monitoriza la ejecución de la tarea y se elimina para dejar la máquina virtual en el estado original.

6.2.2. Instalaciones con ejecución remota en Linux

Los servidores Linux que utiliza la empresa no disponen de entorno gráfico. Además utilizan la herramienta dialog, un entorno de consola que simula la interfaz gráfica. Por lo que no se puede automatizar la interacción mediante un simple script.

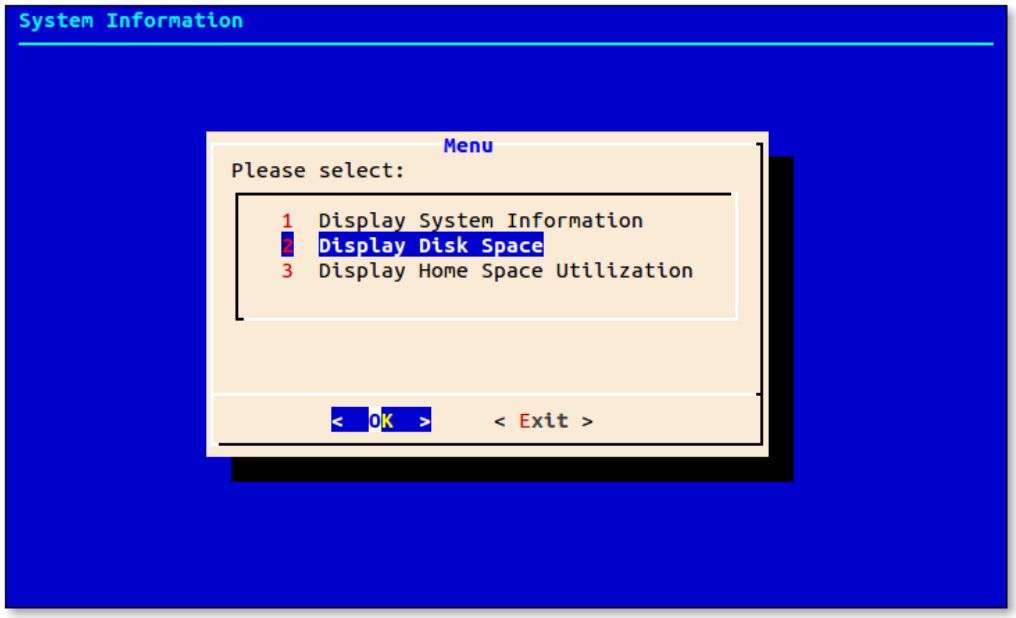

Figura 6.1: Ejemplo de interacción de la herramienta Dialog

Llegados a este punto se intentaron varias soluciones alternativas, como leer por consola intentado obviar toda la parte gráfica y buscando palabras clave, intentando enviar señales de teclado como espacio, *enter* o tabulación directamente por una conexión SSH, pero no resultó efectivo.

Por lo tanto decidí volver a utilizar AutoIT para la automatización. Para ello necesitaba abrir una conexión PuTTy desde un servidor Windows, por lo que volví a utilizar la máquina de Jenkins y Robot Framework.

AutoIT no es capaz de leer el contenido de una ventana de PuTTy, por lo que actúa como una caja negra a la que se puede enviar señales como espacio, intro, tabulación o cadenas de texto, pero no se puede controlar el flujo de la instalación, ya que no se sabe si ya está lista la pantalla donde se interactúa o todavía está cargando componentes de la instalación.

Para solventar este problema, decidí hacer un controlador que mediante un script Powershell monitoriza el log de instalación. Mediante el log de instalación toma puntos de referencia e informa a AutoIT cuando puede interactuar con el instalador.

De esta forma conseguimos nuevamente una instalación sin requerir de una interacción persona-computador.

Al igual que con el apartado anterior, para realizar este *workaround* se necesita interfaz gráfica por lo que de la misma forma se instala el servicio a través de un script Powershell el cual lo lanza, ejecuta, controla que finalice y lo elimina.

CAPÍTULO 7
Caso de uso

7.1 Introducción

Para finalizar e intentar completar la visión sobre el automatizador se va a exponer un caso de uso mínimo siguiendo la traza de la ejecución.

Este ejemplo está basado en una ejecución real. El servidor que se va a desplegar es una base de datos Microsoft SQL. Esta base de datos dispone de un XML propio donde se configuran diferentes opciones de la solución software y se quería comprobar que ciertas funcionalidades nuevas se configuraban correctamente.

Ya se ha hablado en el capítulo "Infraestructura de la solución" que el automatizador está embebido en una VM. Para facilitar su uso se comparte por red una carpeta de dicha máquina virtual (VM). Mediante esta carpeta podemos configurar la ejecución y lanzarla sin necesidad de entrar a ningún otro sitio. La estructura se puede ver en la figura 7.1.

Además también incluye una presentación PowerPoint la cual explica brevemente la arquitectura del automatizador y como se debe usar.

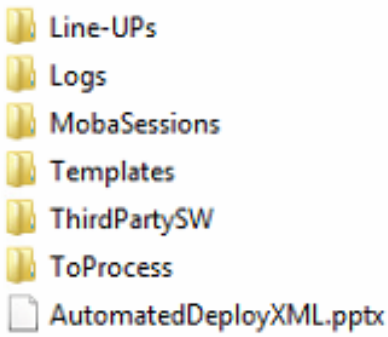

Figura 7.1: Estructura de carpetas del automatizador

7.2 Configurando la ejecución

7.2.1. Line-UP

Lo primero que necesitamos es un *Line-UP*. Un *Line-UP* es el conjunto de componentes software que se quieren instalar.

En nuestro caso solo queremos desplegar la VM de Microsoft SQL, por lo tanto solo vamos a necesitar dos componentes software. El primero es un configurador del entorno que prepara, instala y configura una instancia de SQL. El segundo componente software despliega dos bases de datos sobre la instancia previamente configurada, una con elementos de administración donde se encuentra la configuración que queremos comprobar y la segunda preparada para recibir y almacenar datos suministrados por otros elementos de la solución *software*.

Estos dos componentes software deben incluirse en una carpeta que llamaremos "LineUP-CasoDeUso" y la debemos incluir dentro de la carpeta "Line-UPs".

7.2.2. Fichero XML de configuración

Lo segundo que necesitamos es realizar el XML de configuración de las máquinas virtuales. Recordamos que el XML se puede diferenciar en dos zonas, una zona de configuración general y la otra donde se definen las máquinas virtuales.

En la figura 7.2 podemos ver la zona de configuración general.

```
<!-- TimeZone must be like "Europe/Madrid" -->
<!-- If RepositoryIP is setted all environments will likend with that Repository, You should remove Attribute if you want new repository -->
<!-- Autoconfigure_etc_Hosts will get INTRA IP. If there is not in XML will get ADMIN IP -->
<!-- ThinProvisioned False means Eager Zeroed Provisioning, If use ThinProvisioned True deploy will be faster, but disk usage will be slower -->
<AUTO TimeZone="Europe/Uzhgorod" Autoconfigure_etc_Hosts = "True" ThinProvisioned = "True" MobaSessions = "False">

    <!-- /!\ C A U T I O N  /!\ -->
    <!-- IF YOU OVERWRITE VM_FOLDER, YOU WILL DELETE ALL VM CONTAINED -->
    <VM_FOLDER Folder = "Despliegue Caso de Uso" Overwrite = "False" />


    <NETWORK>

        <!-- NETWORK ADAPTERS CONFIGURATION --><!-- Delete or comment interfaces u will not use -->
        <ADMIN_NETWORK Gateway ="192.168.171.254" Mask = "255.255.240.0" DNS = "192.168.171.80" />


    </NETWORK>

    <!-- If you set "True" InstallAll attribute, you will install solution in all NovaGeo servers deployed -->
    <!-- SRTM avaiables ZONES -> All|"Africa","Australia","Eurasia","Islands","North_America","South_America"-->
    <SOFTWARE LineUp = "LineUP-CasoDeUso" RepositoryIP = "X.X.X.X" SRTM="XXXXX" />
```

Figura 7.2: XML de configuración del caso de uso. Zona general.

Si nos fijamos en el XML de configuración, podemos ver una primera línea donde se configura ciertos parámetros generales. En ella podemos indicar la zona horaria. En nuestro caso he elegido la zona horaria de Uzhgorod (GMT+2/+3).

El segundo parámetro indica si se debe configurar el fichero de *host*, en caso de hacerlo se añadirán todas las máquinas virtuales que contiene el XML indicando su *host name* y dirección IP.

El tercer elemento indica si las máquinas virtuales se van a desplegar en *Thin Provisioned* o *Thick Provisioned, Eager Zeroed*. Aquí podemos ver más información acerca del aprovisionamiento [5]. Como solo queremos ver que se configuran ciertos parámetros correctamente en base de datos, es decir, no vamos a utilizar el entorno, he elegido *Thin Provisioned*, esto incrementará la velocidad del despliegue de la VM.

El último elemento de la primera línea indica si se quiere generar un fichero donde se configuran las conexiones remotas a las máquinas, incluyendo usuarios y contraseñas. En nuestro caso de uso no hace falta, ya que solo queremos una VM. Pero en situaciones

donde se despliega un entorno entero formado por unas 14 máquinas virtuales, es muy útil este fichero, ya que evita configurar 14 conexiones remotas, sustituyéndolo por la simple importación de un fichero.

El segundo elemento que nos encontramos es *VM FOLDER*. En él definimos el nombre de la carpeta de vCenter, suite de gestión de servidores y máquinas virtuales, donde se incluirán las máquinas virtuales. También podemos indicar si queremos "sobrescribir" la carpeta, en ese caso si existiese una carpeta con dicho nombre el automatizador borraría las máquinas virtuales que contiene para dar lugar a las nuevas que se van a desplegar.

En el siguiente elemento configuramos la red, en nuestro caso solo hemos configurado una interfaz de red de las cuatro disponibles.

El último elemento de la zona general es donde indicamos qué software vamos a instalar. ¿Recordáis la carpeta que hemos incluido dentro de *Line-UPs*?. Pues es aquí donde indicamos el *Line-UP* que vamos a utilizar. Los siguientes atributos son la IP del repositorio que vamos a utilizar y la Zona de SRTMs que vamos a utilizar. En nuestro caso no se utilizan ninguno de las dos, por eso no hace falta configurarlos. De todas formas recordamos que el repositorio es un servidor YUM el cual se utiliza para instalar paquetes en servidores Linux y los SRTM son unos ficheros proporcionados por la NASA los cuales contienen la topología de la Tierra, pero son utilizados por otro servidor distinto al que vamos a desplegar.

Ya tenemos la zona general configurada. Nos falta definir la VM que deseamos desplegar, la cual podemos ver en la figura 7.3

```
<!-- #-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#  SERVER CONFIGURATION  #-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-#-# -->
<!--
If you dont want deploy a server u must comment or delete it.
-->

    <DATABASE_SERVERS>

        <!-- IF YOU OVERWRITE SERVER YOU WILL DELETE VM WITH SAME NAME -->
        <DATABASE Overwrite = "False" Install = "True" >

        <!-- VMWARE SETTINGS-->

            <VM_NAME Name = "CasoDeUso-MSSQL" />
            <HOST Host = "192.168.172.12"/>
            <DATASTORE Datastore = "rds012.datastore2" />
            <POOL Pool = "Resources" />


        <!--HARDWARE SETTINGS -->

            <!-- RAM in MB -->
            <RAM_MB Ram = "24576" />
            <!-- Number of Virtual Cores -->
            <VCORES VCores = "6" />
            <!-- EXTRA Hard Disk Size in GB -->
            <DISK_GB_1 DSize ="500" <!-- D: CallList -->
            <DISK_GB_2 DSize ="500" <!-- E: MessageFlow -->
            <DISK_GB_3 DSize ="500" Datastore = "rds012.datastore1"/><!-- F: Data -->
            <DISK_GB_4 DSize ="90" /><!-- G: Logs -->
            <DISK_GB_5 DSize ="30" /><!-- H: TempDB -->
            <DISK_GB_6 DSize ="30" /><!-- I: TempDB -->
            <DISK_GB_7 DSize ="30" /><!-- J: TempDB -->
            <DISK_GB_8 DSize ="30" /><!-- K: TempDB -->
            <DISK_GB_9 DSize ="50" /><!-- L: Backups -->


        <!-- NETWORK SETTINGS-->

            <!-- Admin IP will be use for share folder with Parser & Processor -->
            <ADMIN_IP IP = "192.168.166.20" />


        </DATABASE>

    </DATABASE_SERVERS>

</AUTO>
```

Figura 7.3: XML de configuración del caso de uso. Configuración de la VM.

El servidor de base de datos de Microsoft SQL se llama *DATABASE*. Hay varias bases de datos en la solución pero esta al ser la principal es la llamada base de datos, propiamente. Digo esto para no dar lugar a confusión. Dentro del elemento *DATABASE SERVERS* solo se pueden definir máquinas virtuales *DATABASE*. Cada servidor dispone de su elemento XML, por ejemplo el servidor *MEDIATOR* dispondrá de su elemento *MEDIATOR SERVERS* donde dentro de él se podrá definir las instancias que queramos de dicho servidor. En nuestro caso solo queremos una instancia de *DATABASE*.

El primer atributo que debemos configurar es *Overwrite*. Este atributo indica si se quiere borrar previamente la VM si ya existiese para reemplazarla por la nueva. El segundo atributo indica si deseamos instalar el software.

Gracias a estos dos atributos podemos generar estas combinaciones:

- Borrar una VM, crear una nueva e instalar *software*.
- Borrar una VM, crear una nueva y dejarla preparada para instalar *software*.
- Instalar software en una VM que ya existiese y no deseamos borrarla.
- En el caso de no existir la VM simplemente se crearía y se instalaría o no, dependiendo del atributo.

El siguiente atributo a configurar es el nombre de la VM que aparecerá dentro de la carpeta definida en el vCenter, además este atributo se utiliza también para dar nombre

al *host name* interno de la máquina. El nombre de la VM que aparecerá en el vCenter es el utilizado para buscar la VM para el atributo *Overwrite*.

Para completar la zona de VMware se indica la IP del servidor anfitrión que albergará dicha VM, el nombre del disco de almacenaje donde se almacenará la VM y el *pool* de recursos donde se encuentra el servidor anfitrión.

El siguiente apartado es el de *hardware*. Aquí configuramos la cantidad de memoria RAM, el número de procesadores y el tamaño de los discos.

Como detalle cabe destacar que se puede incluir el *datastore*, es decir, el disco de almacenaje, para cada disco de la máquina virtual. En el ejemplo podemos ver como el disco "F" se incluye en un *datastore* distinto al resto. Esto es una opción útil, por ejemplo, si el disco no nos cabe en el *datastore* en el cual estamos desplegando la VM o si queremos incluir el disco en otro *datastore* con otra configuración RAID por motivos de redundancia de datos.

El último apartado es el de *Network Settings*. Aquí indicamos que queremos configurar la interfaz de "Admin", la cual habíamos configurado en la zona general. En el atributo IP indicaremos la dirección IP.

7.3 Traza de la ejecución

7.3.1. Lanzando la ejecución y comprobación de ficheros

Ya tenemos el fichero de configuración de las máquinas virtuales listo. Para lanzar una ejecución basta con copiar el fichero XML e introducirlo dentro de la carpeta *ToProcess* que podemos ver en la figura 7.1. Jenkins se encarga de monitorizar la carpeta y cuando detecta un fichero lanza la ejecución del trabajo maestro que comienza con toda la ejecución. Sería conveniente recordar la figura 3.1.

Nos encontramos en la zona de *Build* o Construcción del trabajo maestro. Lo primero que realiza el trabajo maestro es insertar en una variable global el XML de configuración para que el resto de trabajos que van a ser lanzados puedan ver la variable. Después lo elimina de la carpeta *ToProcess*.

El segundo *step* que tiene el trabajo maestro es un script de Powershell el cual procesa la variable del XML de configuración y genera un fichero de conexiones remotas compatible con una suite llamada MobaXterm. El propio script dispone de una plantilla para las conexiones Windows y otra para las Linux. Lo que hace es ir generando el fichero sustituyendo la dirección IP y el nombre de la conexión utilizando el nombre de la VM. Los puertos, usuarios y contraseñas ya están definidas en las dos plantillas.

El primer trabajo que es lanzado desde la zona de *Build* del trabajo maestro es un trabajo que comprueba los ficheros que se van a utilizar en las instalaciones. En nuestro caso comprueba que dentro de la carpeta de *LineUP* que hemos indicado están los dos componentes software que vamos a instalar.

El componente que prepara el entorno e instala Microsoft SQL necesita una imagen ISO con el Microsoft SQL, el trabajo también realiza esta comprobación. Esta ISO se encuentra en la carpeta de ThirdPartySW que podemos ver en la figura 7.1.

Si el trabajo no encuentra algún fichero terminará como fallido, el trabajo maestro está preparado para terminar su ejecución si este trabajo falla. Desde Jenkins se puede ver la salida por consola de los trabajos. Aquí el trabajo que comprueba los ficheros indicará las rutas que espera y si las ha encontrado o no.

7.3.2. Sobreescritura de entornos

Si todo ha ido correctamente el trabajo maestro lanzará la ejecución del trabajo de sobreescritura de máquinas virtuales. Este trabajo lo único que hace es llamar al *workflow* de vOrchestrator que realiza esta función proporcionándole el XML de configuración. Dicho *workflow* procesa el XML y busca las máquinas que es necesario borrar, si hay alguna la apagará y borrará.

Para evitar posibles sustos se organizó vCenter en dos grandes carpetas de máquinas virtuales. Una contiene subcarpetas con entornos fijos que no se desean borrar, la otra contiene subcarpetas con entornos habilitados para este tipo de tareas. Cuando creamos un entorno, la carpeta que genera el automatizador la genera dentro de esta segunda carpeta de entornos. Si deseamos cambiar un entorno de fijo a habilitado para ser borrado basta con moverlo de una carpeta a otra desde vCenter. Esta tarea se realiza instantáneamente.

En nuestro caso la máquina virtual que queremos desplegar tiene el atributo *Overwrite* a *False*. Por lo tanto el *workflow* finaliza.

7.3.3. Fase inicial

Han terminado los dos primeros trabajos que se ejecutan de manera secuencial. Ahora desde la zona de *Build* del trabajo maestro llama a una serie de trabajos de forma paralela. Dichos trabajos se pueden ver en la figura 3.2.

Recordamos que a partir de ahora hay dos tipos de trabajos. Los creadores de máquinas virtuales, que llaman a vOrchestrator y los instaladores de software, que llaman a los *install Powershell Files*. Recordamos la figura 3.3 donde se puede ver la estructura de dichos trabajos.

Los trabajos creadores de las máquians virtuales llaman a su *workflow* lanzador proporcionándole el XML de configuración. Cabe recordar que cada tipo de servidor dispone de un trabajo de creación diferente y a su vez dispone de un *workflow* lanzador distinto. Este lanzador dispone de la información de qué ficheros de configuración se van a utilizar en la creación de la VM, ya que cada máquina se configura de manera distinta. Este *workflow* lanzador llama al trabajo de creación de VMs (*Deploy Subsystem*) proporcionándole la información de dichos ficheros de configuración. Este *workflow* es único y se amolda tanto para Windows como para Linux. Podemos ver dicho *workflow* en la figura 3.4.

Como nuestra máquina no pertenece al grupo de VM que se crean en la primera fase los trabajos finalizan al momento sin realizar nada.

7.3.4. Fase Principal

Todos los trabajos de la Fase inicial han terminado de forma instantánea y la zona de *Build* del trabajo maestro pasa a llamar al segundo bloque de trabajos de forma paralela. Es aquí donde el trabajo maestro llama al trabajo instalador de *DATABASE* y este a su vez tiene como precondición el trabajo de creación de la máquina virtual.

Aquí el trabajo de creación de nuestra máquina llamará al *workflow* lanzador de nuestro tipo de servidor. Como solo deseamos desplegar una instancia, el trabajo lanzador solo llamará una vez al *workflow Deploy Subsystem*.

Deploy Subsystem

El trabajo lanzador de *DATABASE* procesa el XML de configuración mediante la llamada al *workflow* de Parseo del XML. Este *workflow* devuelve un vector de servidores a desplegar, en nuestro caso este vector solo contendrá un elemento. Después llama al *workflow Deploy Subsystem* tantas veces como elementos tenga el vector de forma paralela.

La ejecución de *Deploy Subsystem* comienza con el clonado de la plantilla de VM correspondiente, esta plantilla viene definida en el array. En este punto ya vemos en vCenter una carpeta llamada "Despliegue Caso de Uso" con una máquina virtual dentro con el nombre de "CasoDeUso-MSSQL".

Cuando finaliza este clonado se modifica el *hardware* acorde con lo especificado en el XML de configuración a través del *workflow* HardWare Settings, modificando el número de procesadores, la memoria RAM y añadiendo los discos con el tamaño especificado. Seguidamente se enciende la VM mediante un *workflow* que enciende la VM y espera a que esta esté encendida. Después de esto hay un pequeño *Sleep* de espera. Estos Sleeps que podemos ver en la figura 3.4 están puestos debido a que en la práctica las máquinas necesitaban un pequeño tiempo de espera para estar completamente funcionales.

Con la VM encendida se procede a copiar los ficheros de configuración específicos para cada tipo de servidor. En nuestro caso se copian dos *scripts*, uno de configuración de Windows, *script* común a todos los Windows, y un *script* propio del servidor *DATABASE*.

En el primer *script* se configura la zona horaria, el idioma del servidor, la configuración de las cuatro interfaces de red, en nuestro caso solo una, el *host name* de la máquina y el fichero de hosts.

Una vez copiados los ficheros se ejecutan de forma secuencial. En nuestro caso, a través de Powershell, configurará la zona horaria de Uzhgorod, modificará la interfaz de red de Administración, cambiará el *host name* por CasoDeUso-MSSQL y como el XML de configuración solo dispone de un servidor configurado, modificará el fichero de hosts añadiéndose a sí mismo, es decir, añadirá "CasoDeUso-MSSQL 192.168.166.20".

El último paso de este *script* es la creación de un fichero llamado "ConfigDone". El *workflow* que ejecuta los ficheros está buscando cada 5 segundos dicho fichero, cuando lo encuentra, lo borra y pasa a la ejecución del siguiente *script*.

Esto está implementado de esta forma debido a que era más rápido que utilizando la API de vOrchestrator, la cual puede controlar el PID de un proceso, pero tarda varios minutos en percibir cambios en los PIDs activos.

Finalizado el primer *script* común a todos los Windows, se lanza la ejecución del segundo, perteneciente al tipo específico del servidor. Aquí se formatean y se dan nombre a los discos de almacenaje. De la misma forma que el primer *script*, antes de finalizar genera el fichero ConfigDone para informar al *workflow* de su finalización. Todos los *scripts* generan un log, el cual se guardará con todos al final del proceso en una carpeta.

Cuando finalizan las ejecuciones de los *scripts* se aplica un reinicio de la VM para que parámetros configurados como el *host name* se apliquen. Una vez reiniciada y completamente encendida finaliza el *workflow Deploy Subsystem* y por tanto su Lanzador también.

Trabajo de instalación

En este punto ya tenemos la VM creada y aprovisionada correctamente. El *workflow* lanzador ha terminado y por tanto el trabajo Jenkins de creación también. Recordamos que el trabajo maestro ha llamado al trabajo instalador de *DATABASE* y este a su vez

tenía como precondición completar el trabajo de creación de la máquina. Como este ya ha finalizado su ejecución, da paso al trabajo de instalación.

El trabajo instalador llama al *script* de Powershell *installMSSQL*, este es uno de los *scripts* que en el contexto del TFG llamamos *install Powershell files* y podemos ver su estructura en la figura 3.3.

El *script*, que dispone del XML de configuración como variable de entorno, comprueba si la máquina de tipo *DATABASE* tiene el atributo de *install* a *True*. Si no fuera el caso, terminaría en este punto.

Como la VM que queremos desplegar sí que dispone del valor *True*, el *script* Powershell comienza a recoger las variables necesarias que necesitará Robot Framework a partir del XML de configuración. Estas variables son, la dirección IP de la máquina, la versión del software y otros parámetros que se recogen en el XML de despliegue de la base de datos. Este XML es distinto al de configuración de las VM y es propio de la base de datos Microsoft SQL. En él se encuentran variables necesarias para configurar rutas de instalación que necesitarán otros componentes que se instalan en el mismo servidor, además de otras variables y direcciones IP.

Cuando el *script* Powershell ha generado un fichero con todas las variables, llama a la *suite* de Robot Framework incluyendo dicho fichero en la llamada.

Robot Framework

El *script* Powershell ha llamado a la instalación de *DATABASE* junto con un fichero de variables. Esta instalación la realiza Robot Framework a través de la suite de instalación de dicho servidor.

Recordamos que la *suite* de instalación dispone de un *suite setup* y de un *suite teardown*, es decir, una precondición y postcondición que se encargan de establecer una conexión remota y de destruirla.

El *suite setup* y *suite teardown* son dos *keyword* incluidas en una librería creada especialmente para esta tarea que se incluye en todas las instalaciones Windows. Por otro lado Linux dispone de otra librería, también creada especialmente para abstraer esta funcionalidad.

En el caso de Windows la keyword se encarga de ejecutar un *script* Powershell que se encargará de configurar y crear una sesión de WinRM. Con esta dispondremos de una consola Powershell remota a través de un identificador de sesión que se utilizará a lo largo de todo el proceso.

Una vez Robot Framework abre la conexión a través de la precondición, comienza la instalación propiamente dicha. Recordamos que la *suite* está compuesta de *Tests* y estos a su vez están compuestos por *keywords*.

Recordamos que Robot Framework está instalado en la misma máquina que Jenkins. El primer Test que se ejecuta copia los ficheros necesarios para la instalación. Estos se encuentran disponibles en un directorio local, en las carpetas de ThirdPartySW y LineUPs mencionados en la figura 7.1. Para ello existe una *keyword* en Robot Framework que ejecuta comandos cmd locales. LLamando a esta *keyword* e introduciendo el comando *xcopy* conseguimos copiar los ficheros a la VM destino.

Si recordamos, la tarea principal de Robot Framework es la de *testing*. Es por ello que tiene una infraestructura muy potente para comprobar que todas las keywords se ejecutan de forma correcta. Por lo tanto después de cada keyword que se ejecuta hay otra

que comprueba que el código de retorno de la ejecución anterior sea 0, es decir, se haya ejecutado de forma correcta.

Con todos los ficheros software copiados en la VM destino, el siguiente *test* es montar la ISO de Microsoft SQL en la VM destino y nos guardamos como variable la letra de la unidad donde se ha montado, para ello se utiliza la sesión WinRM que se ha abierto al empezar la instalación.

El siguiente *test* es instalar el preparador del entorno, el cual configura ciertas carpetas, permisos y recursos compartidos de la VM y a su vez instala y prepara una instancia de Microsoft SQL. Como es un componente el cual llama Robot Framework y lanza la instalación de Microsoft SQL, existe un problema de salto de credenciales, para solventar el problema se utilizó un *workaround*, el cual en vez de lanzar desde Robot Framework la ejecución del preparador del entorno, crea una tarea en el *scheduler task* de Windows ejecutándola, monitorizándola y eliminándola al finalizar a través de un *script* Powershell.

Por lo tanto llamamos desde la suite de instalación a un *script* Powershell que crea la tarea, la ejecuta y cuando finaliza la elimina. Este script necesita, entre otras cosas, la ruta del *setup.exe* de la ISO de SQL. Es en esta llamada donde le pasamos estas variables y otras necesarias como rutas de instalación que dependerán del tipo de despliegue de base de datos que hayamos configurado en el XML de despliegue de base de datos. XML distinto al de configuración de las VM.

Una vez preparado el entorno e instalada la instancia de Microsoft SQL, la suite de Robot Framework ejecuta un reinicio necesario para continuar con la instalación. Este reinicio se hace a través de un *script* Powershell el cual ejecuta de forma remota un reinicio sumado a un *wait* el cual espera hasta que la máquina se reinicia y vuelve a estar completamente encendida.

Con el reinicio de la máquina finalizado se ejecuta el siguiente *test* el cual despliega las bases de datos en la instancia configurada previamente. Este despliegue se realiza a través de un *script* bat con la opción de instalación silenciosa, el cual es ejecutado a través de la sesión WinRM.

Después se instalan tres componentes software más que comparten servidor, estos tres componentes son tres *test* más las cuales afortunadamente disponen de instalación silenciosa (sin interacción persona-computador). Estas instalaciones también se realizan a través de la sesión abierta de WinRM.

Finalizando la instalación

La *suite* de Robot Framework termina. Esta hace terminar al Powershell de instalación que es llamado desde Jenkins. El trabajo de Jenkins de instalación termina y por lo tanto finaliza la "Fase Principal".

7.3.5. Fase Final

Desde la zona de *build* del trabajo maestro de Jenkins pasa al siguiente *step* que es la llamada al grupo de trabajos que componen la fase final. Esta fase son trabajos que dependen en tiempo de instalación de otros componentes que previamente se han instalado en la fase anterior.

En nuestro caso todos los trabajos finalizarán de forma inmediata ya que todo lo que debía ser instalado ya lo está.

7.3.6. Recogida de Logs

Tanto los primeros scripts de configuración de las VM que ejecuta vOrchestrator, como Robot Framework, generan unos ficheros de Logs los cuales se almacenan en la carpeta de Logs que podemos ver en la figura 7.1. Dentro de esa carpeta se genera una subcarpeta con el nombre de la ejecución y la fecha y hora del inicio de la ejecución, además se añade al final del nombre *Done* o *Failed* dependiendo de si algún trabajo ha fallado o no.

Solo hay dos trabajos que si fallan bloquean por completo la ejecución. Son los dos primeros, el que comprueba la existencia de todos los ficheros de instalación necesarios y el que borra las VM que van a ser reemplazadas. El resto de trabajos pueden fallar, pero continuarán con la instalación.

Cabe destacar el sistema de *log* del que dispone Robot Framework el cual al estar orientado al *testing* es un sistema muy potente. Este genera un log HTML el cual se puede ver de forma visual si todo ha ido correctamente, en caso contrario se puede seguir la traza de la ejecución para ver donde pudo estar el fallo.

En la siguiente figura podemos ver un ejemplo de log de Robot Framework con el *test* de reinicio de la VM desplegado para ver el detalle.

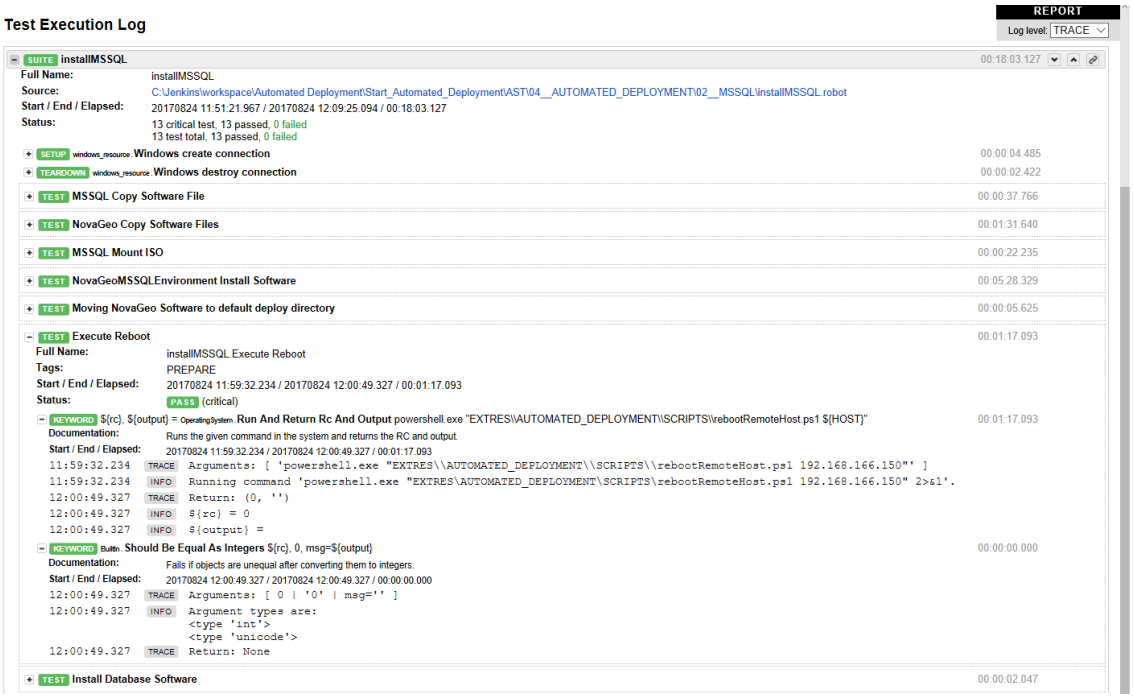

Figura 7.4: Log de Robot Framework de la suite de instalación

CAPÍTULO 8
Conclusiones

8.1 Conclusiones

La automatización de la instalación de tu software trae grandes ventajas. Ayuda a gastar menos tiempo en el despliegue, evita fallos humanos a la hora de desplegar la solución, como saltarse pasos, no configurar de forma correcta algún componente, no seguir el orden correcto de instalación, etc.

Además una vez implementado evita trabajo a la hora de conseguir un entorno nuevo.

En el caso concreto de la empresa donde se desarrolla este Trabajo Final de Grado, el software de la empresa es complejo, dispone como mínimo de 13 servidores entre Linux y Windows, los cuales necesitan cada uno al menos 3 componentes a instalar. Además el orden de instalación es crucial dependiendo unos de otros.

El despliegue de un entorno completo solía llevar dos días como poco y era frecuente despistes a la hora de instalar todos los componentes. De esta forma centralizas la lógica de la instalación haciendo que sea la misma. Sumado a la virtualización consigues mucha flexibilidad a la hora de crear entornos y destruirlos.

Como añadido, si enlazamos los despliegues automáticos con baterías de pruebas automatizadas, con Robot Framework, por ejemplo, conseguimos que de forma completamente desatendida se puedan probar toda clase de circunstancias y detalles que de forma manual sería completamente inviable, consiguiendo un software estable y robusto.

El claro ejemplo de la evolución del *testing* es pasar de cubrir todo el espectro de comprobaciones que confirma que tu software hace lo que tiene que hacer (testing funcional) a realizar testing para intentar romper o forzar tu solución y por consiguiente hacerlo más robusto a malos usos.

El despliegue en producción también ocasiona problemas, no siendo un punto fuerte de la empresa. El objetivo también es conseguir desplegar de forma automática en entornos de producción, reduciendo así problemas que puedan surgir derivados del mal despliegue de una solución software compleja.

En definitiva y como conclusión final, si una solución software es relativamente compleja de desplegar, es muy interesante contemplar la automatización de la misma, ya que con un poco de esfuerzo se pueden ahorrar costes en casi todas las áreas de una empresa.

8.2 Resultados del aprendizaje

Las herramientas utilizadas así como las tecnologías usadas han sido nuevas para mí, por lo que la realización del TFG me ha supuesto un aprendizaje muy amplio. La envergadura del proyecto, así como las diferentes versiones que he debido manejar, han hecho que ponga en práctica una serie de metodologías de trabajo que no había utilizado hasta ahora.

El eje central del proyecto es la virtualización y la automatización. A lo largo del Grado Universitario apenas se ha visto nada sobre virtualización y lo poco que se vio era sobre una tecnología diferente a la usada. Tampoco se explicaban herramientas de automatización como Jenkins o Robot Framework.

Si bien es cierto que toda la corriente troncal de las materias está presente en el proyecto. Desde metodologías de programación, como la programación por capas y programación concurrente a diseño de grafos están presentes en el proyecto y han sido de gran utilidad.

El proyecto se aleja en cierta medida de la rama de especialización que he cursado, Ingeniería de Computadores. Debido a esto quizá han sido relativamente nuevos varios aspectos del proyecto. Como por ejemplo la administración de sistemas, la cual tiene un gran peso en el trabajo que he realizado y prácticamente no he visto a lo largo de mi formación.

Esto ha supuesto un gran aprendizaje en lenguajes como Powershell o Bash. Además de aspectos propios de la administración de sistemas, como por ejemplo, tareas como los distintos formateados que se pueden aplicar a un disco de almacenaje hasta políticas de usuarios que han sido nuevas para mí.

Otras herramientas como Robot Framework también han sido un descubrimiento. El *testing* está prácticamente ausente en el Grado de Ingeniería Informática. Gracias a este trabajo final he descubierto aspectos de la informática hasta entonces desconocidos.

La envergadura del proyecto es sin duda la más grande que he tenido en comparación con trabajos anteriores. Acostumbrarse a manejar tantas líneas de código no ha sido una tarea fácil. Trabajos sencillos pueden desembocar en muchas líneas de código innecesarias. La magnitud del trabajo me ha ayudado a comprender lo importante que es la lógica a la hora de definir tareas para conseguir un código más compacto y más sencillo de mantener.

Por otro lado, el haber realizado el proyecto en un ámbito empresarial me ha supuesto enfrentarme a problemas reales fuera del ámbito académico. Me ha ayudado a fomentar el pensamiento lateral y la búsqueda de soluciones alternativas. Además de un primer contacto con el mundo laboral, el cual me ha dado un "baño" de realidad, dicho de forma cariñosa, y me ha dado una visión diferente y muy valiosa acerca de la profesión.

En definitiva, puedo afirmar que el Trabajo Final de Grado me ha enriquecido en muchos aspectos, tanto técnicos como humanos de forma muy positiva. Como me digo a mí mismo, ha sido un pequeño gran Master.

Desde aquí también me gustaría agradecer la ayuda que me han ofrecido mis compañeros de trabajo, así como la orientación e implicación que he recibido desde el tutor en la empresa como el tutor del proyecto en la UPV. Gracias.

Bibliografía

[1] James Smith ; Nair, Ravi. IEEE Computer Society. *The Architecture of Virtual Machines*, 32–38, 2005.

[2] Martil Fowler "Integración Continua" mayo de 2006. Disponible en https://martinfowler.com/articles/continuousIntegration.html. Consultado en mayo de 2017.

[3] Matt Wrock "Safely running windows automation operations that fail inside winrm or powershell remoting" enero de 2015. Disponible en http://www.hurryupandwait.io/blog/safely-running-windows-automation-operations-that-typically-fail-overwinrm-or-powershell-remoting. Consultado en abril 2017

[4] Mike English "File Formats and Tools for Virtualization" Open Virtual Appliance junio de 2013. Disponible en https://spin.atomicobject.com/2013/06/03/ovf-virtual-machine/. Consultado mayo 2017.

[5] Rickard Nobel "Eager thick vs Lazy thick disk performance" agosto de 2015. Disponible en http://rickardnobel.se/eager-thick-vs-lazy-thick-disk-performance/. Consultado en abril de 2017.

[6] José Maria Gonzalez "Todo lo que necesitas saber a cerca de vCenter" noviembre de 2013. Disponible en https://www.josemariagonzalez.es/2013/11/20/todo-necesitas-saber-vmware-vcenter-server.html. Consultado en abril 2017.

[7] Mike Brown "Mware vCenter Server 6.0 Deployment Guide" febrero de 2015. Disponible en https://www.vmware.com/content/dam/digitalmarketing/vmware/en/pdf/techpaper/vmware-vcenter-server6-deployment-guide-white-paper.pdf. Consultado en abril 2017

[8] Documentación vRealice Orchestrator, Disponible en https://docs.vmware.com/en/vRealize-Orchestrator/index.html.

[9] Wikipedia:" Orquestación (Computación)" junio de 2017. Disponible en https://en.wikipedia.org/wiki/Orchestration_(computing). Consultado en agosto 2017.

[10] Nasa SRTM, Web Oficial. Disponible en https://www2.jpl.nasa.gov/srtm/.

[11] Robot Framework, web Oficial. Disponible en http://robotframework.org.

[12] Robot Framework, biblioteca estándar. Disponible en http://robotframework.org/robotframework/latest/libraries/BuiltIn.html.

[13] CloudBees "About Jenkins". Disponible en https://www.cloudbees.com/jenkins/about.

[14] Jenkins, Web Oficial. Disponible en https://jenkins.io/.

[15] VMware Disponible en https://www.vmware.com/es/company.html. https://es.wikipedia.org/wiki/VMware.

[16] Wikipedia: "Servidores ESXi ". Disponible en https://es.wikipedia.org/wiki/VMware_ESXi. Enero 2017. Consultado en agosto 2017

[17] Wikipedia: "Hipervisor". Disponible en https://es.wikipedia.org/wiki/Hipervisor. Agosto 2017. Consultado en agosto 2017

[18] GIT, agosto de 2017. Disponible en https://es.wikipedia.org/wiki/Git. https://github.com/git/git.

[19] Wikipedia: "Python". Disponible en https://es.wikipedia.org/wiki/Python.

[20] SSHLibrary, Documentación. Disponible en http://robotframework.org/SSHLibrary/latest/SSHLibrary.html.

[21] Selenium Library Robot Framework, Documentación. Disponible en http://robotframework.org/Selenium2Library/Selenium2Library.html.

[22] WinRM, Documentación. Disponible en https://github.com/dmizverev/robot-framework-library/tree/master/doc. https://github.com/WinRb/winrm-elevated

[23] Wikipedia: "Windows Scheduler Task". Disponible en https://en.wikipedia.org/wiki/Windows_Task_Scheduler.

[24] Autoit, Documentacion. Disponible en https://www.autoitscript.com/autoit3/docs/functions.htm.

[25] TFS, Team Foundation Server. Web Oficial. Disponible en https://www.visualstudio.com/es/tfs/.

[26] PowervRO, Documentacion. Disponible en https://github.com/jakkulabs/PowervRO/tree/development/docs.

[27] Jonathan Medd "Automate vRealize Orchestrator with PowerShell: Introducing PowervRO" 15 de agosto 2016. Disponible en http://www.jonathanmedd.net/2016/08/automate-vrealize-orchestrator-with-powershell-introducing-powervro.html. Consultado en junio 2017

[28] Wikipedia:" Putty". Disponible en https://es.wikipedia.org/wiki/PuTTY.

[29] Dialog, Documentación. Disponible en https://bash.cyberciti.biz/guide/Bash_display_dialog_boxes.

[30] Wikipedia: "Protocolo SSH". Disponible en https://en.wikipedia.org/wiki/Secure_Shell.

[31] Wikipedia: "XML Format". Disponible en https://en.wikipedia.org/wiki/XML.

[32] Wikipedia: "Software Deployment". Disponible en https://en.wikipedia.org/wiki/Software_deployment.

[33] Sobre Apache, Web Oficial. Disponible en https://httpd.apache.org/ABOUT_APACHE.html.

[34] MySQL, Documentación. Disponible en https://dev.mysql.com/doc/.

[35] Tomcat, Tomcat Wiki Oficial. Disponible en https://wiki.apache.org/tomcat/FrontPage.

[36] Mongodb.com "Getting Started with MongoDB" Disponible en https://docs.mongodb.com/getting-started/shell/.

[37] Geoserver, Web Oficial. Disponible en http://geoserver.org/.

[38] Wikipedia: "SQL Server". Disponible en https://es.wikipedia.org/wiki/Microsoft_SQL_Server.

[39] MobaXterm, Web Oficial. Disponible en http://mobaxterm.mobatek.net/.