Universitat Oberta de Catalunya

uoc.edu

Automatización de Infraestructura IT con IaC

Carlos Martino Ojeda
Ingeniería Informática - master universitario
Computación de altas prestaciones

Joaquin Lopez Sanchez-Montañes
Josep Jorba Esteve

20 Enero 2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Automatizacion de Infraestructura IT con IaC*
Nombre del autor:	*Carlos Martino Ojeda*
Nombre del consultor/a:	*Joaquin Lopez Sanchez-Montañes*
Nombre del PRA:	*Josep Jorba Esteve*
Fecha de entrega (mm/aaaa):	01/2020
Titulación::	*Ingeniería Informática - master universitario*
Área del Trabajo Final:	*Computación de altas prestaciones*
Idioma del trabajo:	***Español***
Palabras clave	*IaC Automatización Infraestructura*

Resumen del Trabajo:

El trabajo de fin de master de Automatización de Infraestructura IT con IaC, consiste en diseñar e implementar el despliegue de aplicaciones con Fedora Server en una infraestructura virtual OpenStack, mediante Integración y Entrega Continuas.

Éstas técnicas, al usarse, solucionan los problemas de tiempo y fallos que se tienen en muchas empresas a la hora de crear y mantener servidores manualmente. La ventaja del método, es que la configuración de la infraestructura permanece inmutable, como código, en un repositorio central. Además se garantiza su estabilidad, gracias a los tests automatizados, que permiten detectar fallos.

La implementación, usa un flujo en una herramienta de automatización (Jenkins), que genera una imagen de máquina virtual lista para desplegar. Una vez la imagen está creada, con otro flujo se pueden crear máquinas virtuales, configurándolas con roles de Ansible, según el tipo de servidor que se quiera crear (por ejemplo una pequeña base de datos o un servidor web).

El objetivo del flujo de creación de imagen, es probar la configuración realizada con Lorax (usando la librería de Python TestInfra), antes de que las imágenes se creen y desplieguen en la infraestructura virtual. De igual forma, cada rol de Ansible dispone de un flujo Jenkins de pruebas, que incluye también pruebas automatizadas de TestInfra.

Todo el código de creación de infraestructura se encuentra alojado en repositorios públicos de GitHub, de forma que los flujos de Jenkins lo descargan de éstos cada vez que hay algún cambio para probar.

Abstract:

This master degree final project, Automatizacion de Infraestructura IT con IaC (IT Infrastructure Automation with IaC), designs and implements the application deployment with Fedora Server in an OpenStack based virtual infrastructure, using continuous integration and continuous delivery.

The use of these techniques, allows companies to save time and solve issues that occurs when configuring and maintaining servers manually. A benefit of this method, is that all the infrastructure configuration remains immutable, as code, in a central repository. Additionally, the infrastructure stability is guaranteed, thanks to the automated tests that can detect configuration bugs.

Project implementation, uses a Jenkins automation tool pipeline, that generates a virtual machine image ready to be deployed. Once the image is available, another pipeline can take it to create virtual machines and configuring them with Ansible roles. For example, depending on the server type, it may be configured as a database or a web server.

The image pipeline function, is to test the configuration (using TestInfra Python library), made with Lorax, before images are created and deployed in the virtual infrastructure. Furthermore, each Ansible role includes a Jenkins pipeline, that tests it automatically using TestInfra, as well.

Finally, all the infrastructure creation code, is hosted publicly in GitHub repositories. In that way, all Jenkins pipelines can download them to test each time a change is made.

Índice

1 Introducción

Durante mucho tiempo, las empresas y organizaciones han estado usando lo que podemos denominar infraestructura IT clásica, para ejecutar sus aplicaciones y dar servicio a sus clientes. La infraestructura IT clásica, está formada por un gran número de servidores, tanto físicos como virtuales, los cuales son mantenidos manualmente por equipos de operaciones. En la mayoría de los casos, los componentes de la infraestructura (servidores, almacenamiento, equipos de red, etc) están inventariados en lo que se conoce como Base de Datos de Gestión de la Configuración o CMDB (Configuration Management Database). A su vez, la información de la CMDB es utilizada por una aplicación de gestión del servicio o ITSM (IT Service Management), que permite la gestión de los cambios en la infraestructura y la comunicación entre los usuarios y los equipos que la mantienen. De entre las aplicaciones de gestión del servicio destacan Servicenow y Helix.

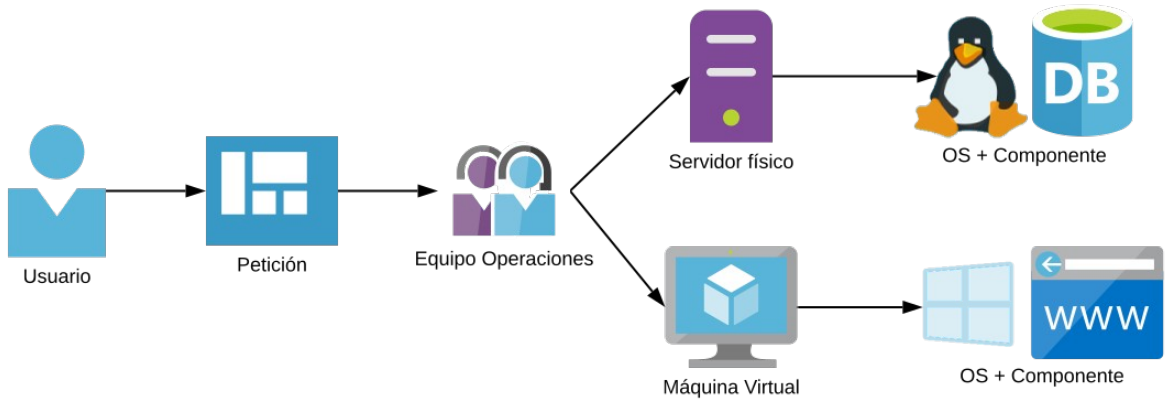


Figura 1: Proceso de instalación de un servidor en la infraestructura IT clásica

En una infraestructura clásica, el proceso de creación o de mantenimiento de un servidor arranca mediante una petición de un usuario en la aplicación ITSM. A la petición, se le asigna un equipo de operaciones y un elemento de configuración o CI (Configuration Item), que es la representación del servidor en la CMDB e incluye toda su información. Si el servidor es instalado por primera vez, se creará un CI nuevo con sus datos, en caso contrario se actualizará su información con los cambios realizados. Una vez la petición es recibida por el grupo de operaciones, se asigna a uno de sus miembros para realizarla. El responsable de llevar a cabo la petición, deberá encargarse de obtener un servidor físico (que incluye su colocación y cableado en un datacenter) o de crear una máquina virtual (en un servidor de máquinas virtuales), estas dos acciones puede que sean realizadas por otros equipos. Tras obtener el servidor, se podrá instalar el sistema operativo, configurarlo según la petición e instalar una aplicación si se ha requerido (Figura 1).

Aunque el modelo de infraestructura clásica ha servido para gestionar y organizar grandes entornos, presenta varios problemas:

- Tiempo de despliegue: Los equipos de operaciones suelen recibir multitud de peticiones de usuarios, las cuales quedan en espera hasta que son asignadas. A ese tiempo de espera se une el de la obtención del equipo, que como hemos dicho anteriormente, puede ser provisto por otro equipo operacional y el de instalación o configuración del sistema operativo y la aplicación. Como consecuencia, una tarea que duraría un día puede llegar a alargarse dos semanas.

- Configuraciones dispares: Al ser la gestión manual, servidores que a priori deberían configurarse con los mismos ajustes, se hacen de forma diferente, ya sea por errores humanos o malentendidos. Por lo que la infraestructura queda en un estado inconsistente.

- Mantenimiento: Las aplicaciones y los sistemas operativos evolucionan continuamente y los fabricantes publican mejoras y parches que solucionan fallos. Además, pueden aparecer vulnerabilidades del software, que afectan la seguridad de la infraestructura y deben corregirse a la brevedad. Todo lo anterior, hace que se tengan que establecer ventanas de mantenimiento, afectando la disponibilidad de las aplicaciones que componen el servicio.

- Incidentes: Ya sea por la inconsistencia de la infraestructura o por fallos de las aplicaciones, se producen incidencias que afectan la continuidad del servicio y que tienen que ser resueltas por los equipos operacionales.

- Documentación: Aunque se crean guías y documentos que explican cómo realizar las configuraciones e instalaciones, éstas suelen estar dispersas en multitud de lugares como wikis o repositorios de documentos y no siempre se actualizan a la vez que aparecen cambios en la configuración o nuevas versiones de los programas.

La idea de éste proyecto es solucionar los problemas anteriores de la infraestructura IT clásica, aplicando el modelo de infraestructura como código o IaC (Infrastructure as Code).

La infraestructura como código tiene como objetivo la automatización de la infraestructura, basándose en prácticas de desarrollo de aplicaciones. Pone el foco en rutinas consistentes y repetibles para la configuración, entrega y cambios en los sistemas. Los cambios son codificados usando herramientas de gestión de configuración y desplegadas en los sistemas mediante procesos desatendidos, que incluyen validación. Con lo anterior, se consigue tratar la infraestructura como si fuera software y datos. Otra ventaja, es que se pueden usar herramientas de desarrollo como gestores de código con sistema de control de versiones y repositorios, librerías de pruebas automatizadas y utilidades de orquestación. Por último, se abre la posibilidad de aplicar técnicas de desarrollo como integración continua o CI (Continous Integration), entrega continua o CD

(Continous Delivery) y desarrollo guiado por pruebas de software o TDD (test-driven development).

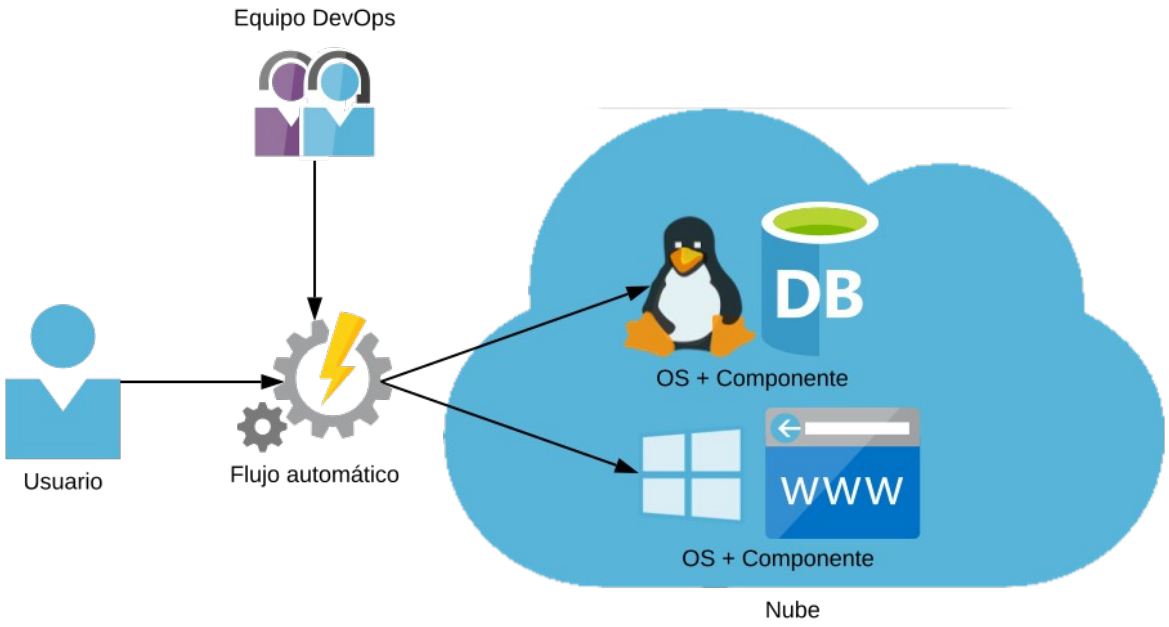


Figura 2: Proceso de creación de un servidor con IaC en un entorno de nube

A diferencia de la infraestructura IT clásica, una infraestructura basada en IaC se caracteriza por utilizar un servicio de nube, ya sea privado o público, que está preparado para que sus elementos de infraestructura sean creados y configurados automáticamente mediante código. Normalmente los servicios en nube ofrecen la creación de servidores virtuales, aunque también existen opciones para crear servidores físicos (como MAAS o AWS EC2). El proceso de despliegue y configuración de los servidores es automático, los usuarios pueden acceder directamente a los flujos de automatización, ya sea mediante mediante acceso directo al orquestador de la infraestructura, o vía API o portal que conecten con el primero. Los equipos de operaciones se transforman en equipos DevOps y en vez de atender peticiones de usuarios se encargan de mantener el código de la infraestructura (Figura 2). A su vez, toda la infraestructura creada con IaC en la nube, está pensada para ser masiva y efímera, de forma que las múltiples instancias de servidores se puedan crear durante un tiempo determinado y se destruyan cuando dejen de ser útiles. Lo anterior hace que el uso de una CMDB deje de tener sentido y se use en su lugar sistemas de monitorización automáticos provistos por la nube o de terceros (como Datadog) que generen métricas de la disponibilidad de todas las instancias.

Mediante la infraestructura como código se resuelven los problemas que padece la infraestructura clásica:

- Tiempo de despliegue: La creación de los servidores y configuración de aplicaciones se realiza en cuestión de minutos y es iniciada directamente por los usuarios. En consecuencia, el ahorro de tiempo con respecto a la infraestructura clásica es considerable.

- Configuraciones dispares: Los datos y las acciones de configuración de la infraestructura están codificadas en un gestor de código (como GitHub, Gitlab o Bitbucket), que incluye repositorios y control de versiones, por lo que todos los despliegues realizados usarán acciones y configuraciones idénticas. Además, el gestor de código permite identificar los cambios realizados en el código respecto a versiones anteriores por cada usuario, lo que permite identificar incongruencias o fallos y restaurar versiones anteriores. Por último, a la hora de promocionar nuevas versiones de código se puede añadir aprobaciones y pruebas automáticas, que garantizan que los todos cambios van a ser supervisados. En éste proyecto se ha usado la Semantic Versioning Specification para indicar el número de versión del código.

- Mantenimiento: A la hora de aplicar mejoras, parches a fallos o vulnerabilidades tanto a los sistemas operativos como a las aplicaciones, es posible codificar una nueva configuración que los recoja y crear instancias nuevas de los servidores. Las instancias antiguas, a medida que se crean las nuevas, pueden irse destruyendo, midiendo con sistemas de monitorización si existen fallos durante el despliegue. En caso de que se produzca algún error, las instancias nuevas se podrán eliminar y recrear las antiguas, sin que haya pérdida ni interrupción del servicio.

- Incidentes: Dado que las nuevas versiones de las instancias son probadas y revisadas anteriormente antes de desplegarse, los incidentes por cambios se evitan en su mayoría. Además, como se ha dicho anteriormente, existe la posibilidad de retornar a versiones anteriores en caso de que los sistemas de monitorización detecten un número considerable de instancias de servidores de una misma aplicación que fallen. En el caso de existir un fallo, éste se podrá revisar a parte en el código, sin consecuencias para el servicio.

- Documentación: Aprovechando las capacidades de los gestores de código, la documentación del código de la infraestructura, se almacena junto a éste. Por lo que toda la configuración queda documentada en un único lugar y se actualiza a la vez que el código ante cualquier cambio. También aprovecha el versionado del gestor del código y se pueden seguir y aprobar los cambios de la misma forma que con el código. En el caso de gestores de código basados en Git (como GitHub, Gitlab o Bitbucket) se usa el formato Markdown (.md) para presentar la documentación. Para documentar el código del proyecto se ha escogido la opción del formato Markdown y todos sus entregables poseen un fichero README.md con su respectiva documentación.

Plan de trabajo

El proyecto se ha organizado en tres entregas, comprendidas en dos prototipos y la entrega final. De ésta forma se ha tenido un desarrollo cíclico basado en técnicas de Extreme Programming. La programación extrema o Extreme Programming, es una metodología ágil de programación cuya finalidad es mejorar la calidad de los programas y la respuesta a la hora de cambiar los requerimientos del cliente. Dicha metodología, aboga por establecer ciclos cortos de entregas, con lo que se consigue dar al cliente pequeñas funcionalidades desde el primer momento e incluir pequeños hitos donde poder introducir nuevas mejoras o cambios. El uso de Extreme Programming, encaja con el proyecto, al haberse establecido varios periodos de pequeñas entregas y da la posibilidad de ir evolucionando el producto poco a poco.

Los entregables de cada prototipo y de la entrega final son los siguientes:

1.1 Prototipo 1

Flujo de Jenkins capaz de construir una máquina virtual a partir de una imagen creada con Lorax en OpenStack. El flujo descarga un fichero blueprint de git, con el que crea la imagen. Mediante playbooks de Ansible copia la imagen y crea la máquina virtual.

1.2 Prototipo 2

- **Flujo de Jenkins de creación de imágenes en OpenStack:** Realiza pruebas automáticas con TestInfra en una máquina virtual, generada a partir de una imagen, para validar el código del blueprint de Lorax y los roles de Ansible. Si la imagen es validada, se copia una nueva versión en OpenStack.

- **Roles de Ansible:** Los roles permiten configurar un servidor básico y servidores Apache y Mariadb.

- **Flujo de creación de máquina virtual:** Instancia una nueva máquina virtual, de un determinado tipo, usando la última versión estable de la imagen.

1.3 Entrega final

- **Entregables del Prototipo 2:** Integrados en el flujo de despliegue de aplicación.

- **Flujo Jenkins de despliegue de aplicación:** Crea una aplicación web de prueba conectada a una base de datos y realiza pruebas globales con TestInfra.

- **Documentación:** Documentos del proyecto, presentación del proyecto y vídeo.

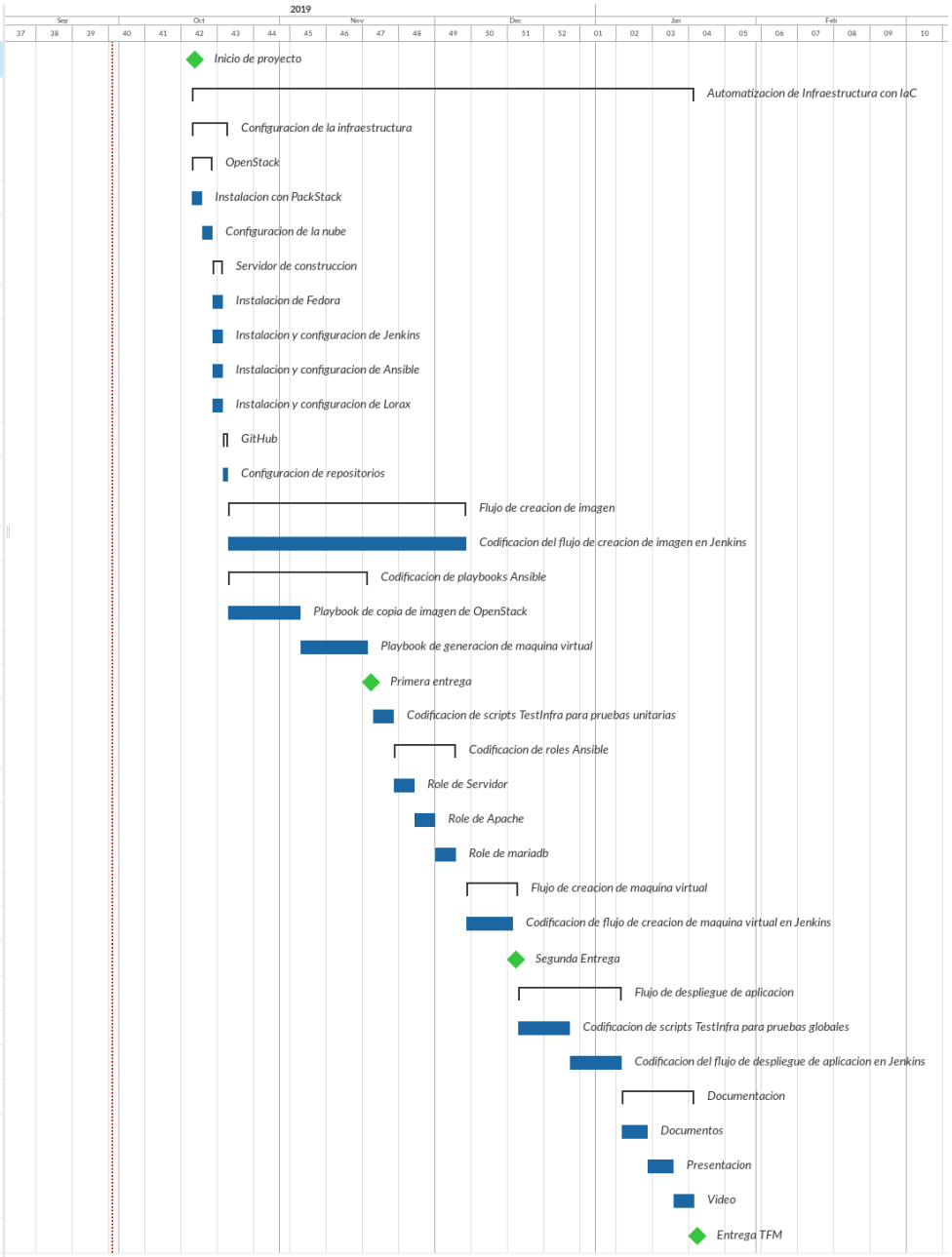
2 Tareas del proyecto

Las tareas con las que se han generado las entregas del proyecto, incluyendo las fechas de inicio y finalización de cada una, se muestran en la Tabla 1, junto con el diagrama de Gantt.

Tarea	Fecha inicio	Fecha fin	Estado
Inicio de proyecto	15.10.2019	15.10.2019	Finalizada
Automatizacion de Infraestructura con IaC	15.10.2019	19.01.2020	Finalizada
Configuracion de la infraestructura	15.10.2019	21.10.2019	Finalizada
OpenStack	15.10.2019	18.10.2019	Finalizada
Instalacion con PackStack	15.10.2019	16.10.2019	Finalizada
Configuracion de la nube	17.10.2019	18.10.2019	Finalizada
Servidor de construccion	19.10.2019	20.10.2019	Finalizada
Instalacion de Fedora	19.10.2019	20.10.2019	Finalizada
Instalacion y configuracion de Jenkins	19.10.2019	20.10.2019	Finalizada
Instalacion y configuracion de Ansible	19.10.2019	20.10.2019	Finalizada
Instalacion y configuracion de Lorax	19.10.2019	20.10.2019	Finalizada
GitHub	21.10.2019	21.10.2019	Finalizada
Configuracion de repositorios	21.10.2019	21.10.2019	Finalizada
Flujo de creacion de imagen	22.10.2019	06.12.2019	Finalizada
Codificacion del flujo de creacion de imagen en Jenkins	22.10.2019	06.12.2019	Finalizada
Codificacion de playbooks Ansible	22.10.2019	17.11.2019	Finalizada
Playbook de copia de imagen de OpenStack	22.10.2019	04.11.2019	Finalizada
Playbook de generacion de maquina virtual	05.11.2019	17.11.2019	Finalizada
Primera entrega	18.11.2019	18.11.2019	Finalizada
Codificacion de scripts TestInfra para pruebas unitarias	19.11.2019	22.11.2019	Finalizada
Codificacion de roles Ansible	23.11.2019	04.12.2019	Finalizada
Role de Servidor	23.11.2019	26.11.2019	Finalizada
Role de Apache	27.11.2019	30.11.2019	Finalizada
Role de mariadb	01.12.2019	04.12.2019	Finalizada
Flujo de creacion de maquina virtual	07.12.2019	16.12.2019	Finalizada
Codificacion de flujo de creacion de maquina virtual en Jenkins	07.12.2019	15.12.2019	Finalizada
Segunda Entrega	16.12.2019	16.12.2019	Finalizada
Flujo de despliegue de aplicacion	17.12.2019	05.01.2020	Finalizada
Codificacion del flujo de despliegue de aplicacion en Jenkins	27.12.2019	05.01.2020	Finalizada
Codificacion de scripts TestInfra para pruebas globales	17.12.2019	26.12.2019	Finalizada
Documentacion	06.01.2020	19.01.2020	Finalizada
Documentos	06.01.2020	10.01.2020	Finalizada
Presentacion	11.01.2020	15.01.2020	Finalizada
Video	16.01.2020	19.01.2020	Finalizada
Entrega TFM	20.01.2020	20.01.2020	Finalizada

Tabla 1: Tareas del Proyecto

ID	SUBJECT	START DATE	FINISH DATE
75	Inicio de proyecto	15/10/2019	15/10/2019
44	Automatizacion de Infraestructura con IaC	15/10/2019	19/01/2020
48	Configuracion de la infraestructura	15/10/2019	21/10/2019
58	OpenStack	15/10/2019	18/10/2019
59	Instalacion con PackStack	15/10/2019	16/10/2019
60	Configuracion de la nube	17/10/2019	18/10/2019
53	Servidor de construccion	19/10/2019	20/10/2019
54	Instalacion de Fedora	19/10/2019	20/10/2019
55	Instalacion y configuracion de Jenkins	19/10/2019	20/10/2019
56	Instalacion y configuracion de Ansible	19/10/2019	20/10/2019
57	Instalacion y configuracion de Lorax	19/10/2019	20/10/2019
61	GitHub	21/10/2019	21/10/2019
62	Configuracion de repositorios	21/10/2019	21/10/2019
49	Flujo de creacion de imagen	22/10/2019	06/12/2019
63	Codificacion del flujo de creacion de imagen en Jenkins	22/10/2019	06/12/2019
64	Codificacion de playbooks Ansible	22/10/2019	17/11/2019
65	Playbook de copia de imagen de OpenStack	22/10/2019	04/11/2019
66	Playbook de generacion de maquina virtual	05/11/2019	17/11/2019
76	Primera entrega	18/11/2019	18/11/2019
71	Codificacion de scripts TestInfra para pruebas unitarias	19/11/2019	22/11/2019
67	Codificacion de roles Ansible	23/11/2019	04/12/2019
70	Role de Servidor	23/11/2019	26/11/2019
68	Role de Apache	27/11/2019	30/11/2019
69	Role de mariadb	01/12/2019	04/12/2019
50	Flujo de creacion de maquina virtual	07/12/2019	16/12/2019
72	Codificacion de flujo de creacion de maquina virtual en Jenkins	07/12/2019	15/12/2019
77	Segunda Entrega	16/12/2019	16/12/2019
51	Flujo de despliegue de aplicacion	17/12/2019	05/01/2020
74	Codificacion de scripts TestInfra para pruebas globales	17/12/2019	26/12/2019
73	Codificacion del flujo de despliegue de aplicacion en Jenkins	27/12/2019	05/01/2020
52	Documentacion	06/01/2020	19/01/2020
79	Documentos	06/01/2020	10/01/2020
80	Presentacion	11/01/2020	15/01/2020
81	Video	16/01/2020	19/01/2020
78	Entrega TFM	20/01/2020	20/01/2020



3 Descripción de la entrega final

A continuación se explican con detalle los elementos que componen la entrega final: Infraestructura, flujos de Jenkins, blueprint de Lorax, playbooks y roles de Ansible.

3.1 Infraestructura

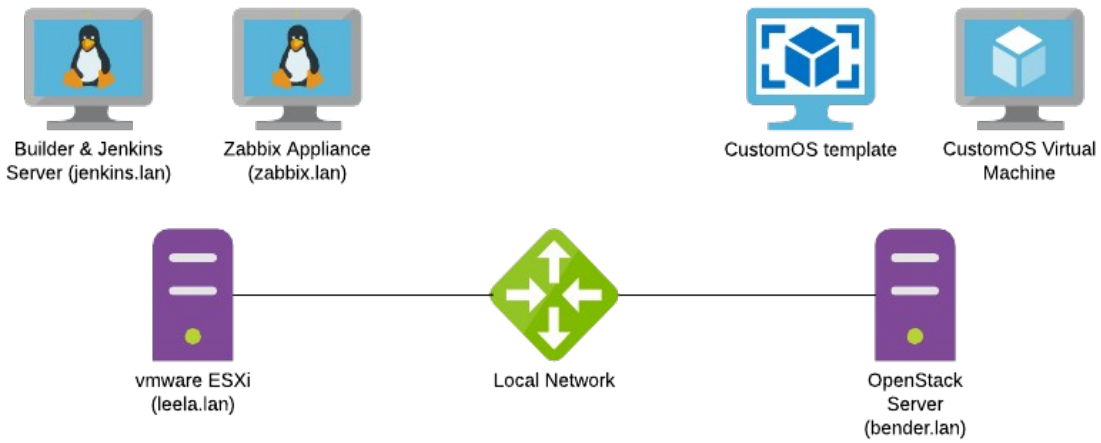


Figura 3: Infraestructura del proyecto

La infraestructura usada en la entrega final (Figura 3), está formada en primer lugar por un servidor Jenkins (jenkins.lan), que se encarga mediante un flujo de crear las imágenes de máquinas virtuales y de copiarlas. El otro elemento de la infraestructura, es el servidor OpenStack (bender.lan), al cual se conecta el flujo de Jenkins para copiar la imagen que crea y generar una máquina virtual a partir de ésta. A la hora de crear el servidor Jenkins, se ha usado una máquina virtual sobre un servidor vmware ESXi (leela.lan) con Fedora Server 31, instalado con las versiones 2.8 de Ansible, 31.9 de Lorax y 2.190.2 de Jenkins. En cuanto al servidor OpenStack, se ha instalado sobre CentOS 7 junto con un hipervisor KVM. Con el objetivo de ver la funcionalidad del agente Zabbix, incluido en la configuración de los servidores, se ha instalado en el servidor vmware la máquina virtual oficial (appliance), que incluye el servidor de monitorización preconfigurado (zabbix.lan).

Fedora es una distribución Linux con soporte comunitario y financiada por Red Hat, la cual usa las funcionalidades y novedades de ésta para integrarlas en su distribución comercial Red Hat Enterprise Linux (RHEL). Es la distribución elegida para el proyecto, dado que posee las últimas versiones de las herramientas usadas y contienen las últimas funcionalidades. Es probable que en entornos productivos, se consideren otras opciones a la hora de desplegar distribuciones Linux (como RHEL, Debian, Ubuntu o SLES), que primen más la estabilidad con ciclos de desarrollo

largos y menos novedades funcionales, frente a los ciclos de desarrollo cortos con últimas versiones de Fedora.

OpenStack es un software comunitario de código abierto para desplegar entornos de nube tanto privados como públicos. Dado que permite despliegues privados al contrario que otras soluciones como AWS, Azure o Google Cloud, exclusivamente públicas y no requiere licencia, ha sido elegido para éste proyecto. Con la intención de ahorrar tiempo en la instalación de Openstack, se ha usado Packstack, al ser Openstack una herramienta compleja a la hora de configurar y mantener. Packstack es una utilidad que permite desplegar rápidamente Openstack en un nodo como prueba de concepto o varios nodos en instalaciones más avanzadas. En el caso del proyecto, se ha escogido la instalación en un solo nodo. Por otra parte, tanto Openstack como Packstack se han obtenido usando los repositorios RDO, que permiten su instalación en sistemas basados en Red Hat como CentOS, Fedora o RHEL. RDO es un proyecto comunitario de código abierto financiado por Red Hat y basa sus fuentes en Red Hat OpenStack Platform (solución OpenStack comercial con soporte de Red Hat).

Vmware ESXi es un hipervisor comercial soportado por la empresa vmware y aunque no se usa directamente en la solución propuesta por el proyecto, la versión gratuita de éste es empleada para alojar los servidores Jenkins y Zabbix.

Zabbix es un programa de monitorización open-source para diversos componentes TI, incluyendo redes, servidores, máquinas virtuales y servicios de nube. Proporciona detalles como métricas de uso o consumo de recursos (carga de CPU, memoria, espacio en disco, etc).

3.2 Flujo Jenkins de creación de imágenes OpenStack

Jenkins es un servidor de automatización de código abierto, financiado por la compañía HashiCorp. Aunque su principal uso es automatizar tareas de construcción, prueba y despliegue o entrega de software; puede usarse como herramienta de creación, prueba, orquestación y despliegue de infraestructura como código. La segunda utilidad es la que aplica a este proyecto. Jenkins funciona mediante flujos de tareas agrupadas en fases, que pueden ser codificadas en ficheros Jenkinsfile.

El flujo Jenkins de creación de imágenes OpenStack (image-build), se encarga de orquestar la creación y validación automática de una imagen OpenStack para poder instanciar con ella posteriormente máquinas virtuales.

La imagen es creada a partir de un blueprint (customos) de Lorax. Primeramente, el flujo, descarga un fichero blueprint desde GitHub, con el que inicia la creación de la imagen. Mediante los playbooks vmbuild y copyimg de Ansible, copia la imagen y crea una máquina virtual temporal. En la máquina virtual se aplican pruebas automáticas, usando la herramienta pytest, para validar la configuración del blueprint. Si el resultado de los tests es satisfactorio, la imagen permanecerá disponible en OpenStack.

El código del flujo, se puede encontrar en el repositorio publico de GitHub iac_jenkins-pipe-image-build y la versión que se corresponde con la entrega final es la 0.9.1 (que se encuentra promocionada en la rama master) (Figura 4). Dentro del repositorio se puede encontrar el fichero **Jenkinsfile,** el cual es descargado y ejecutado por el proyecto **IaC,** configurado en el servidor Jenkins (Figura 5).

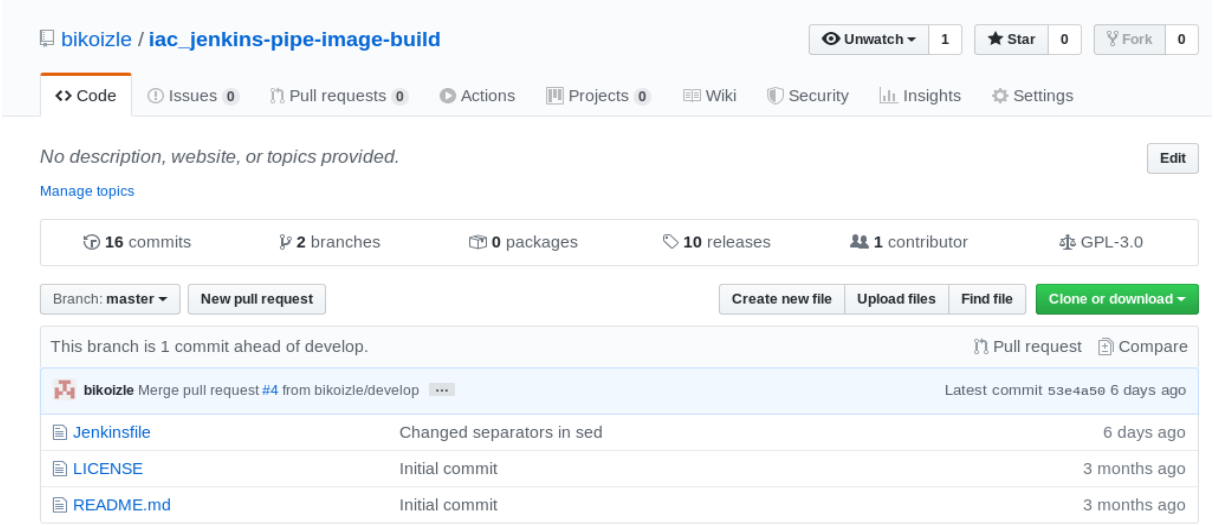


Figura 4: Repositorio image-build en GitHub

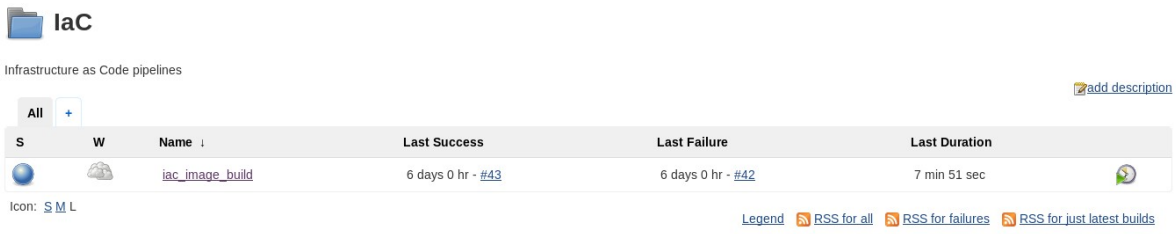


Figura 5: Proyecto IaC en el servidor Jenkins con la pipeline image-build configurada

Dentro de el flujo o *pipeline image-build*, se han creado varias fases o *stages* (Figura 6), la cuales se describen a continuación:

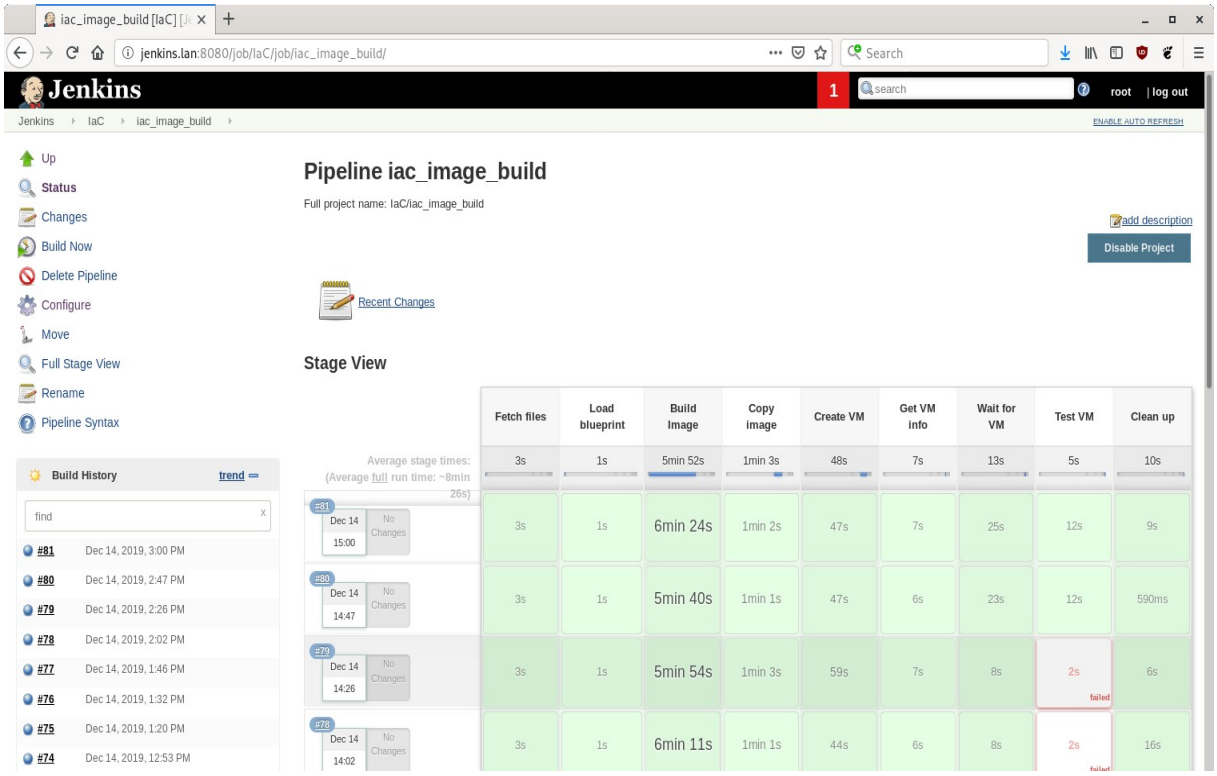


Figura 6: Fases del flujo

- **Fetch files:** Es la fase inicial y se encarga de clonar los repositorios de GitHub que alojan el fichero blueprint de Lorax (en formato toml) y los playbooks de Ansible (en formato yml). Los archivos de cada repositorio se descargan en una carpeta específica, dentro del directorio de trabajo del flujo, para ser ejecutados posteriormente. Tanto las URLs de los repositorios, como sus versiones a clonar son especificadas en las variables del flujo.

- **Load blueprint:** Una vez descargados los archivos, se carga el fichero blueprint **customos.toml** en la configuración del servicio Lorax con el cliente composer-cli. Además, antes de cargarse, se inyecta la contraseña final del usuario *customos* en el fichero, con esto se evita que esté almacenada en los repositorios públicos.

- **Build image:** Después de cargar la configuración en el servicio Lorax, se inicia la creación de la template de máquina virtual con el cliente composer-cli y se comprueba el estado de creación. Una vez terminada, se guarda el ID de la template para posteriormente copiar el fichero de imagen generado.

- **Copy image:** Tras generar el archivo de imagen qcow2 con Lorax, compatible con OpenStack, se copia al servidor OpenStack usando el playbook de Ansible **copyimg.yml.** Las variables con las credenciales y detalles del proyecto donde copiar la imagen, son pasadas como parámetros al

playbook. El nombre final de la imagen incluye la marca de tiempo en la que se ha creado (fecha y hora).

- Create VM: Una vez copiada la imagen con el nombre de **customos,** se ejecuta el playbook **vmbuild.yml** con los parámetros de creación, para crear la máquina virtual en el servidor de OpenStack. Una máquina virtual con el nombre **customos_test**, a partir de la imagen **customos**, es creada en el servidor OpenStack.

- Get VM info: Para que el flujo pueda acceder a la máquina virtual y realizar las pruebas automáticas, se utiliza el playbook de Ansible **getvminfo**. Dicho playbook, además de los datos de conexión con el servidor OpenStack, recibe el nombre de la máquina virtual y genera un fichero json con información de su hardware virtual. El fichero json puede ser interpretado por el flujo de Jenkins y cargado en una variable para acceder a cada dato. Con lo anterior, se puede obtener la IP que OpenStack ha asignado a la máquina virtual y acceder a ella. El acceso se realizará directamente desde el servidor Jenkins como usuario root dado que su clave ssh ha sido insertada previamente al construir la template con la que se genera.

- Wait for VM: En ésta fase se espera, usando el comando ansible con el módulo **ping**, a que la máquina termine de arrancar y esté disponible. El módulo ping de Ansible comprueba el acceso ssh y que exista python en la máquina, no realiza el comando usual ping de diagnostico de red que usa tramas icmp únicamente.

- Test VM: Cuando la máquina se encuentra accesible, se ejecutan los scripts de pruebas automatizadas de pytest, incluidas la carpeta tests del repositorio del **blueprint**. Adicionalmente, antes de ejecutar los scripts de pruebas, su sintaxis es comprobada con la herramienta **flake8**. Si tanto la comprobación de sintaxis como las pruebas automáticas fallan, la fase terminará con error y la imagen generada será borrada del servidor OpenStack en la fase “Clean up”. En caso contrario la imagen se mantendrá y estará lista para ser usada.

- Clean up: Es la última fase y se ejecuta siempre para eliminar la máquina virtual temporal y el entorno de Ansible generado. En el caso de que la fase de Test VM haya fallado, borrará la imagen creada también. Para realizar estas acciones se usan los playbooks **vmdelete** e **imgdelete**.

En todas las fases, se usan credenciales almacenadas en la sección **Credentials** del proyecto **IaC** en Jenkins y son referidas en el código mediante un ID (Figura 7).

Figura 7: Credenciales en Jenkins

Al finalizar la ejecución del flujo, si se entra en la consola de OpenStack, se puede ver la template **CustomOS** creada con la marca de tiempo en el nombre (Figura 8). Durante la ejecución del flujo, también se puede observar la creación de la máquina virtual temporal **customos_test** (Figura 9).

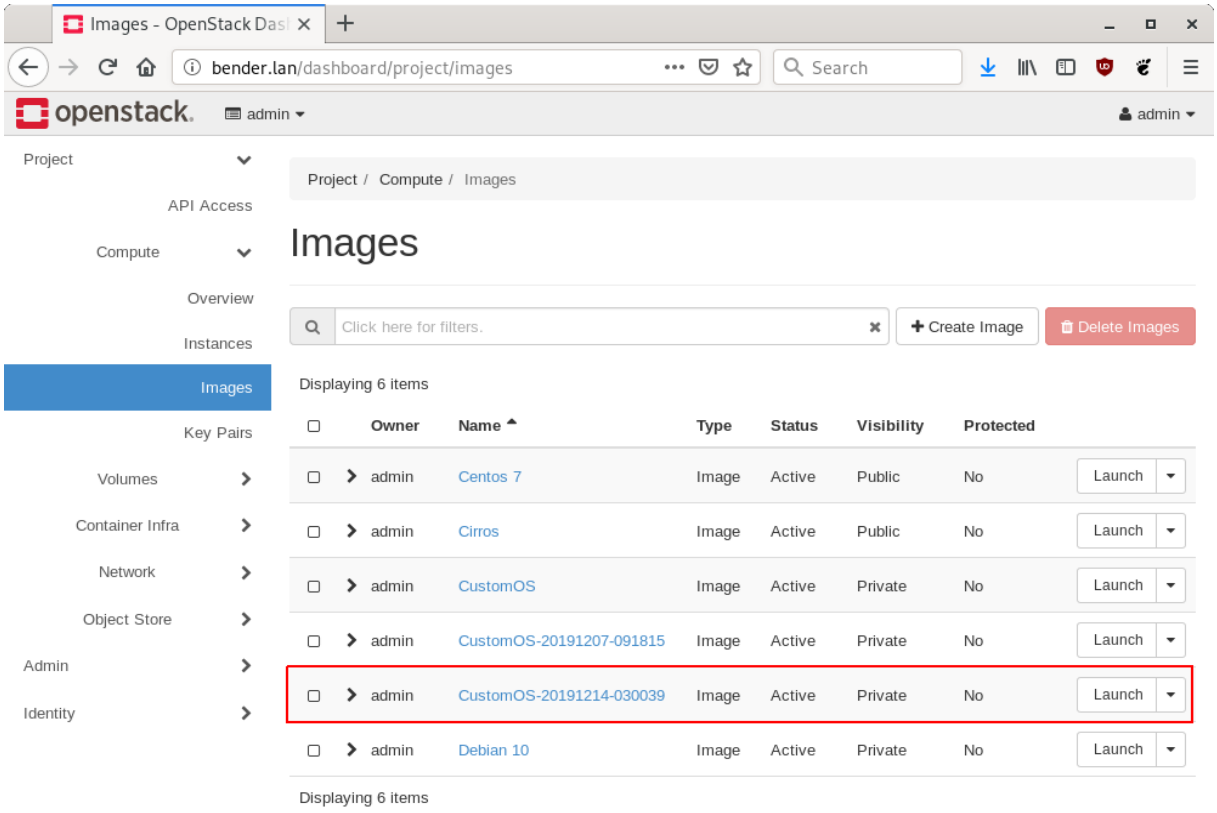


Figura 8: Template en OpenStack

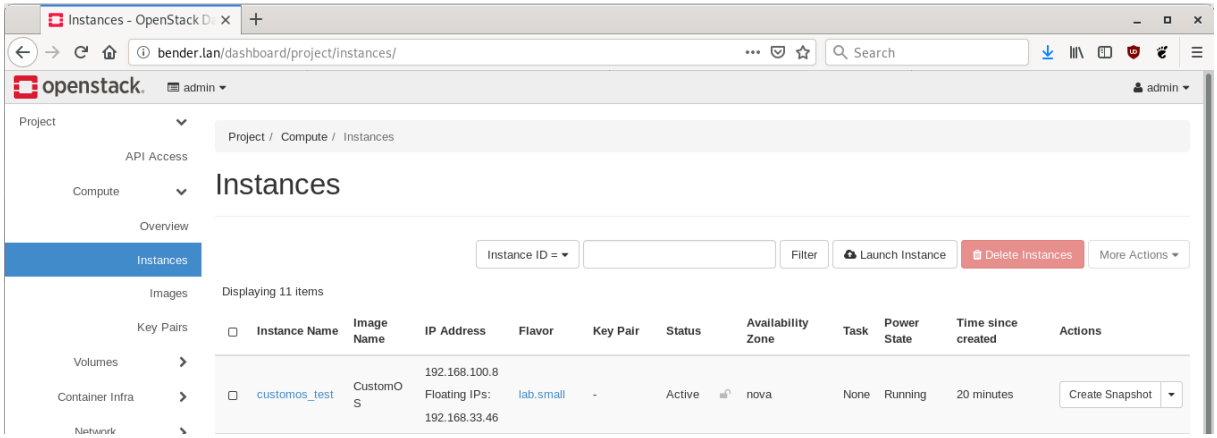


Figura 9: Máquina virtual customos_test en OpenStack

Usando un cliente ssh y la dirección IP asignada, se puede entrar en la máquina virtual de prueba customos_test, con Fedora Server 31 instalado, mediante el usuario customos y la contraseña configurada en Jenkins (Figura 10), mientras dure la ejecución del flujo.

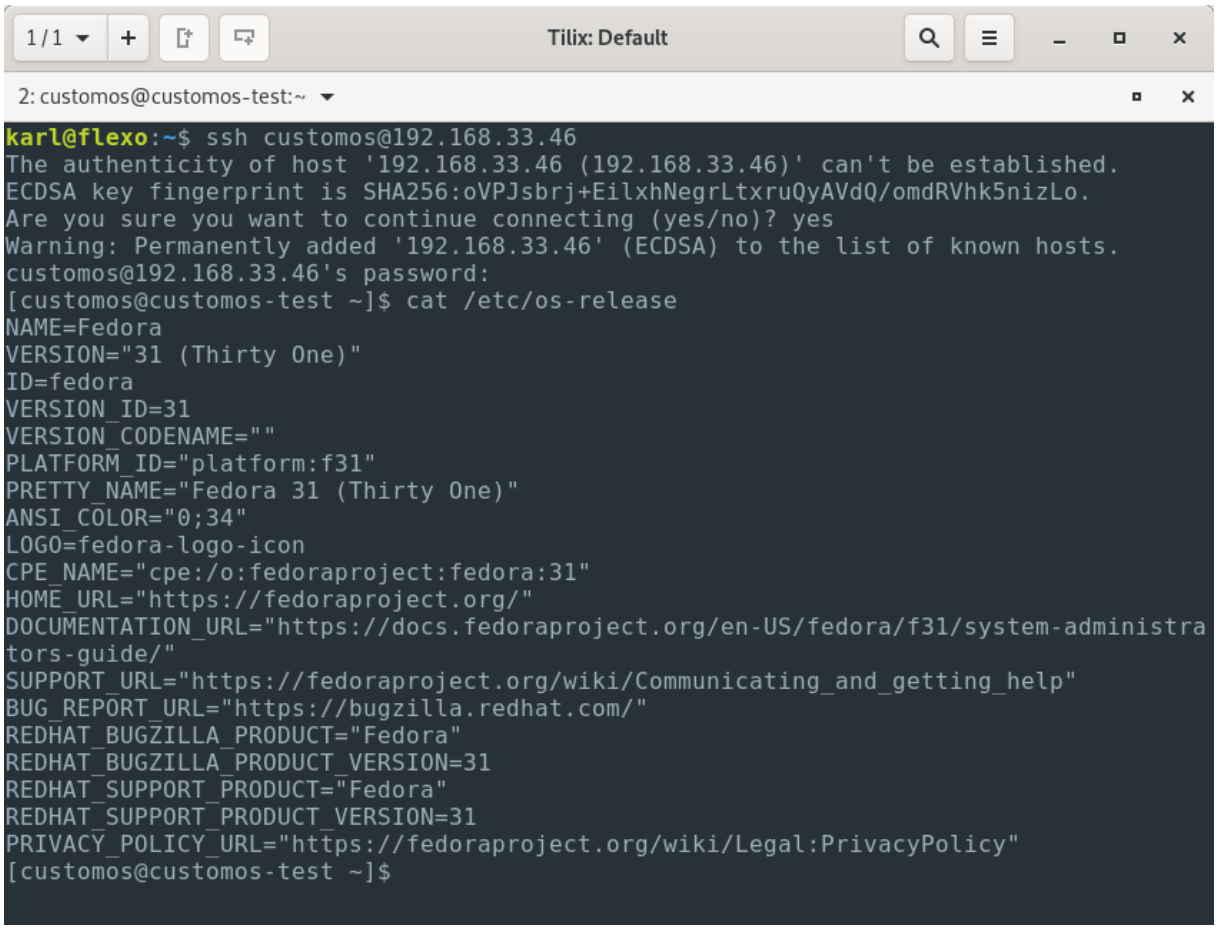


Figura 10: Consola remota en la máquina de prueba customos_test

3.3 Blueprint de Lorax

Para configurar la imagen de máquina virtual de OpenStack, se usa Lorax composer, el cual es un servidor API, que automatiza la creación de imágenes de disco de distribuciones tipo Red Hat (Fedora, CentOS, RHEL, etc). Dicho servidor, usa Blueprints, que son ficheros de texto en formato toml, para configurar el sistema operativo de la imagen (paquetes, usuarios, servicios, etc). La forma de comunicarse con el servidor API, es mediante el cliente composer-cli, que es el usado por el flujo de Jenkins para construir la imagen. Lorax composer permite construir imágenes de disco para varios tipos de hipervisores (como vmware o lvm), entornos cloud (como AWS, OpenStack o Google cloud) e imágenes iso o img.

El blueprint del proyecto, **customos.toml**, se encuentra alojado en el repositorio de GitHub iac_lorax-blueprint-customos y la versión usada en la última entrega es la 0.6.7 (promocionada en la rama master). La configuración del blueprint, incluye la instalación de los siguientes paquetes: openssh-server (para acceso a la máquina), cloud-init (servicio de configuración de OpenStack), cloud-utils-growpart (Utilidad de OpenStack para extender particiones de máquinas virtuales, requerido para que las máquinas tengan el tamaño de disco escogido), clamav-* (paquetes del antivirus open source ClamAV) y zabbix-agent (agente zabbix).

También, el blueprint incluye la creación del usuario customos, que se configura dentro del grupo wheel para poder tener privilegios root a través de sudo en la máquina. La causa de lo anterior, es que el login de root está desactivado por defecto a través de ssh en Fedora 31 y es más seguro además, que la cuenta root no tenga tampoco login directo configurado. El valor de contraseña del usuario permanece como **insecure** y es sustituido en el flujo de Jenkins por la contraseña real. Por último, los servicios de firewall y de cloud-init se activan (éste último para que OpenStack pueda configurar las máquinas virtuales que se generen), mientras que los de los agentes de ClamAV y Zabbix se desactivan para ser configurados posteriormente (Figura 11).

```
53  [[packages]]
54  name = "clamav-server-systemd"
55  version = "*"
56
57  [[customizations.user]]
58  name = "customos"
59  password = "insecure"
60  groups = ["users", "wheel"]
61  uid = 3000
62  gid = 3000
63
64  [[customizations.sshkey]]
65  user = "root"
66  key = "ROOT_KEY"
67
68  [customizations.services]
69  enabled = ["sshd", "firewalld", "cloud-init"]
70  disabled = ["zabbix-agent", "clamd@scan"]
```

Figura 11: Extracto del código blueprint

3.4 Playbooks de Ansible

Ansible es una herramienta de gestión de configuración de sistemas automatizada. Permite configurar desde maquinas simples con sistemas operativos Unix/Linux o Windows a grandes entornos cloud como Azure, Google Cloud, AWS u OpenStack. La configuración de los sistemas abarca desde la configuración básica del sistema operativo hasta el despliegue y configuración de aplicaciones en diferentes entornos. El funcionamiento de Ansible, es con acceso sin agente a los sistemas, mediante usuario y protocolos de acceso remotos como ssh (Unix/Linux), WinRM (Windows) o HTTP/REST (Aplicaciones). Usa el lenguaje yaml para escribir la configuración de los sistemas en playbooks (ficheros de texto en formato yaml) y llamar a módulos de configuración escritos en lenguaje Python. Todo lo anterior, unido a que es de código abierto, hace que sea una herramienta muy versátil y que cada vez sea más usada, ampliando los módulos de configuración a cualquier entorno.

Los playbooks copyimg (para copiar una imagen de máquina virtual a OpenStack), vmbuild (para crear una máquina virtual a partir de una imagen en OpenStack), getvminfo (para obtener información de una máquina virtual), imgdelete (para borrar una imagen) y vmdelete (para borrar una máquina virtual) usan los **módulos** de Ansible para OpenStack. Todos los anteriores poseen variables por defecto en el archivo **vars/main.yml,** que contienen la URL del servidor OpenStack, proyecto, dominio. Para los playbooks de imagen, además, nombre de la imagen a crear, copiar usar o borrar. En el caso de los playbooks de máquina virtual se añade, la red y el tipo para la creación y el nombre de la máquina virtual. Las variables de usuario y contraseña de acceso a OpenStack se guardan en el archivo **vars/vault.yml** con valores de ejemplo**.** Los valores de las variables reales son pasadas por el flujo de Jenkins, y en el caso del archivo **vault.yml**, el flujo pasa al playbook la ruta del fichero **vault** (situado en el servidor Jenkins) con el usuario y la contraseña verdaderos.

Los repositorios de los playbooks son:

iac_ansible-playbook-copyimg para **copyimg,** iac_ansible-playbook-vmbuild para **vmbuild**, iac_ansible-playbook-getvminfo para **getvminfo**, iac_ansible-playbook-imgdelete para **imgdelete** e iac_ansible-playbook-vmdelete para **vmdelete**.

En la última entrega, se han usado las siguientes versiones, promocionadas en la rama master: copyimg 0.2.1, vmbuild 0.2.1, getvminfo 0.1.3, imgdelete 0.1.0 y vmdelete 0.1.0.

3.5 Roles de Ansible

Un rol de Ansible es una forma de automatizar la carga de ficheros de variables, tareas, eventos y dependencias, mediante una estructura definida de archivos. Agrupando la configuración de un servidor mediante roles, facilita su distribución. Los roles pueden ser importados por playbooks de Ansible o como dependencias por otros roles.

En la última entrega se han incluido los siguientes roles para configurar diferentes servidores: Servidor básico, servidor Apache, servidor MariaDB y servidor Wordpress. Todos los roles anteriores emplean la estructura que crea por defecto la utilidad ansible-galaxy, la cual permite la descarga de roles desde repositorios git o desde el repositorio principal de ansible-galaxy. A la estructura inicial se le ha agregado un archivo de flujo de Jenkins, **Jenkinsfile**, para poder probar el funcionamiento del rol con scripts de pruebas automatizadas pytest (situados en la carpeta **tests**) (Figura 13). Además, si el rol incluye dependencias de otros roles en el fichero **meta/main** (Figura 12), éstos con su versión y repositorio de git pueden añadirse en el archivo oculto **.requirements** para que el flujo de prueba de Jenkins pueda descargarlos automáticamente con ansible-galaxy (Figura 14).

```yaml
dependencies:
  - iac_ansible-role-server
  # List your role dependencies here, one per line. Be sure to remove the '[]' above,
  # if you add dependencies to this list.
```

Figura 12: Dependencias de roles en el archivo meta/main.yml

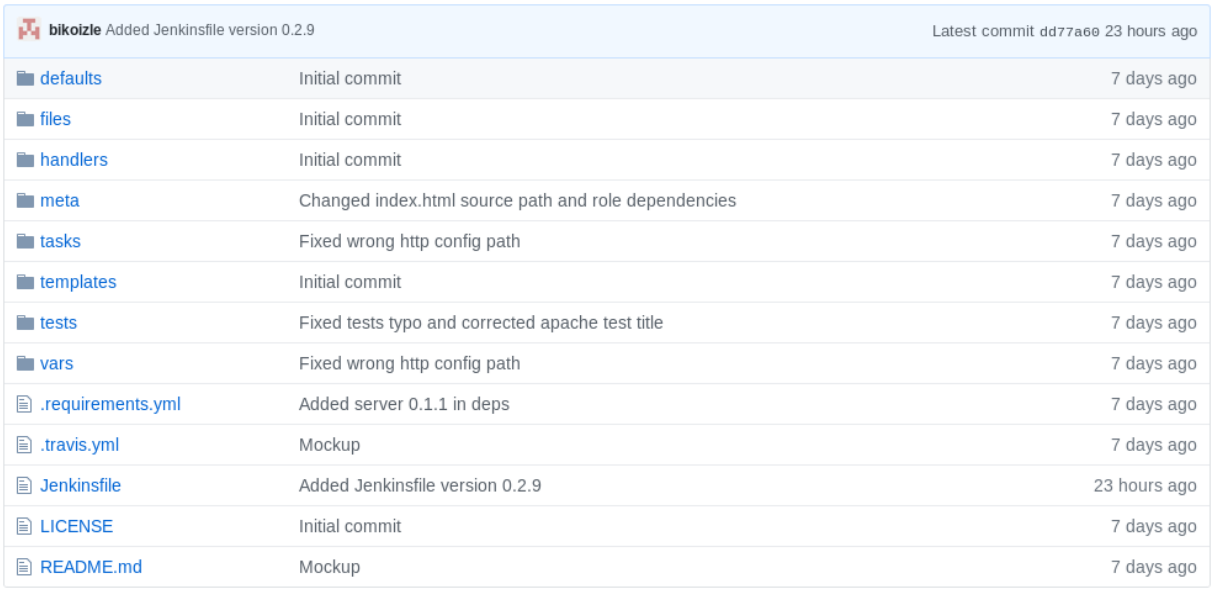


Figura 13: Estructura de rol con flujo de Jenkins y archivo .requirements

```
- src: https://github.com/bikoizle/iac_ansible-role-server.git
  version: "0.1.1"
```

Figura 14: Repositorio y versión del rol para descargar en el archivo .requirements

Un archivo del flujo de Jenkins de prueba de roles, junto con un rol sin tareas y scripts con tests de ejemplo, se puede encontrar en el repositorio iac_jenkins-pipe-role-test (versión 0.2.9 en la rama master para éste prototipo). El archivo Jenkinsfile de dicho repositorio puede copiarse a un repositorio git de cualquier rol que respete la estructura descrita (Figura 14).

El flujo de prueba de roles, crea una máquina virtual temporal, a partir de una imagen existente en OpenStack (indicada en la variable OS_IMAGE_NAME), ejecuta el rol para configurarla, realiza las pruebas automáticas y destruye la máquina virtual. Posee fases similares al de construcción de template, pero se diferencia en no copiar ni crear una imagen y de incluir las fases “Run Playbook” y “Test Playbook” (Figura 15).

La fase “Run Playbook” usa los archivos tests/inventory y tests/playbook.yml creados en la estructura genérica del rol para incluir la dirección IP de la máquina virtual temporal y ejecutar la configuración del rol. Seguidamente, la fase “Run Playbook” comprueba con flake8 los scripts .py

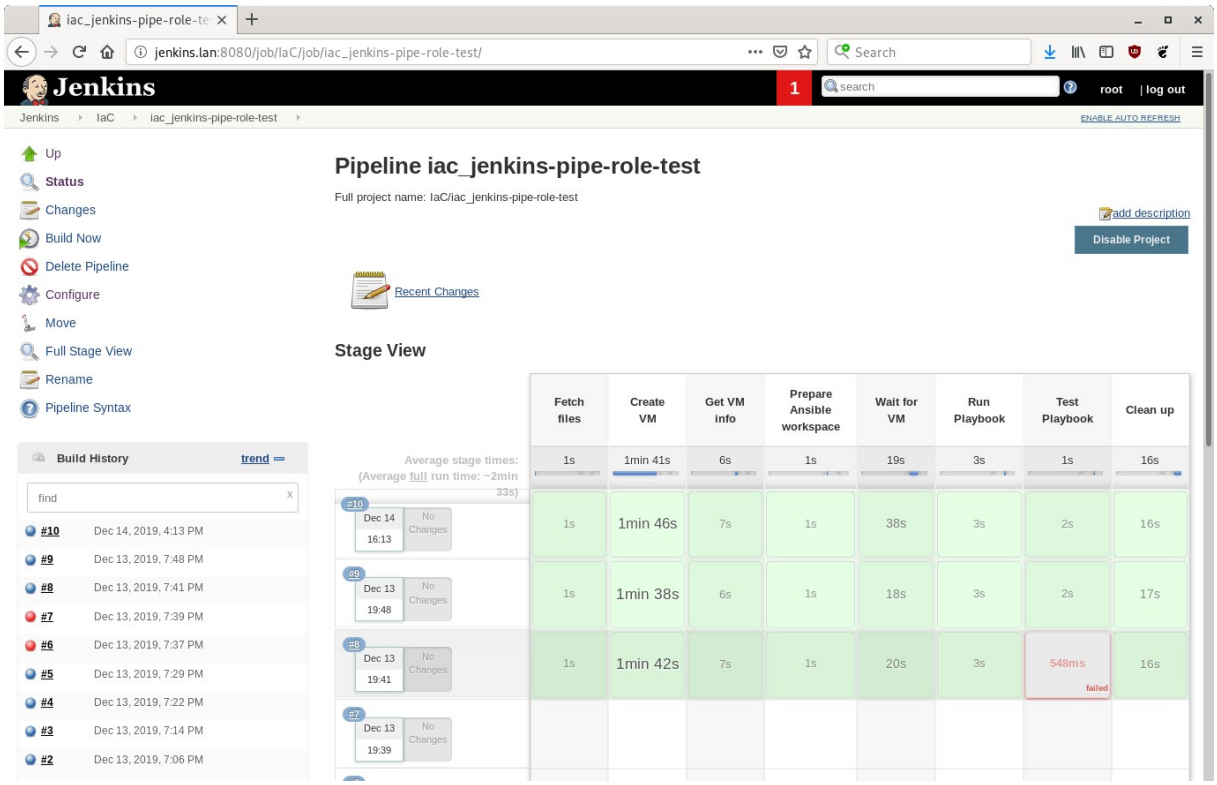


Figura 15: Fases del flujo de prueba de roles

con las pruebas de pytest que existan en la carpeta **tests** y los ejecuta en la máquina. Tanto si las pruebas fallan como si son satisfactorias, la máquina virtual es borrada junto con la carpeta que contiene el entorno temporal de Ansible por la fase “Clean up”.

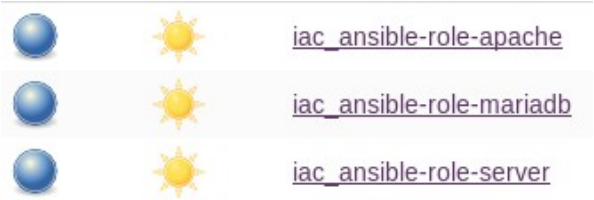


Figura 16: Flujos de prueba de los roles del proyecto en Jenkins

3.5.1 Rol de Servidor

Realiza la configuración básica del sistema operativo, basado en Fedora 31, CustomOS. Configura el estado de SELinux a permissive y desactiva el servicio de Firewall (teniendo en cuenta que estamos en un entorno de pruebas, ésta configuración puede no resultar segura para entornos productivos). Aunque los paquetes de los agentes de ClamAV y Zabbix vienen instalados por defecto en las imágenes, para ahorrar tiempo de despliegue, el rol tiene como tarea instalarlos, copiar las plantillas de los ficheros de configuración correspondientes para cada uno y activar sus servicios. La plantilla del fichero de configuración de Zabbix recibe la variable de servidor de monitorización con el valor de **zabbix.lan** y la variable de metadato con el valor **OpenStack**, para que la máquina sea monitorizada correctamente. En el caso de las plantillas de los ficheros de configuración de ClamAV, se tiene un modulo de servicio extra, clam-freshclam.service, que configura la actualización automática de la base de datos de virus y los parámetros que activan el servicio de búsqueda de virus.

Los tests automáticos se pueden encontrar en el archivo tests/test_default.py dentro del rol y comprueban lo siguiente:

- Desactivación del servicio de firewalld

- Estado de SELinux configurado a Permissive

- Servicios de ClamAV arrancados y activados en el arranque.

- Servicios de Zabbix arrancados y activados en el arranque.

El repositorio del rol en GitHub es **iac_ansible-role-server** y la versión correspondiente al prototipo en la rama master es la 0.2.3

3.5.2 Rol de servidor web apache

Éste rol configura un servidor web apache httpd con una página web simple de prueba (Figura 17). Las tareas que realiza son las de instalar los paquetes del servidor, copiar la template con la configuración personalizada, crear el directorio raíz (‘root’) de la página web, copiar el fichero html de la página y reiniciar y activar en el arranque el servicio httpd. Además incluye como dependencia el rol de servidor, por lo que se garantiza que al aplicarlo sobre cualquier máquina también se va a configurar el sistema con los parámetros básicos.

El test automático se puede encontrar en el archivo tests/test_default.py dentro del rol y comprueba que la web está disponible en la url http://localhost.

El repositorio del rol en GitHub es **iac_ansible-role-apache** y la versión correspondiente a la última entrega en la rama master es la 0.3.7

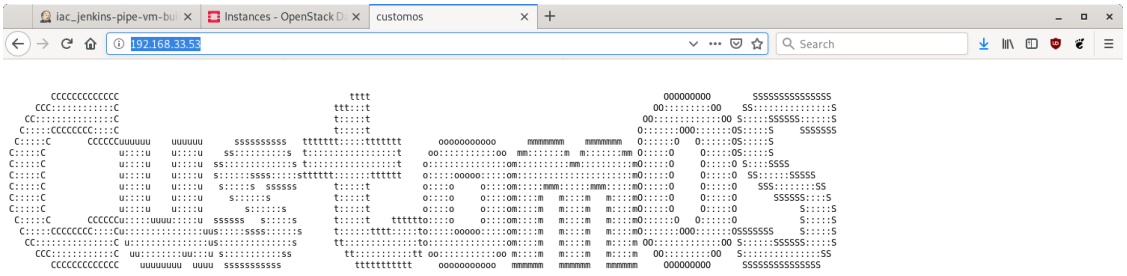


Figura 17: Página web de prueba del servidor Apache httpd

3.5.3 Rol de servidor de bases de datos MariaDB

Un vez aplicado éste rol, queda configurado un servidor de bases de datos MariaDB. Para ello, ejecuta las tareas de: instalación del paquete de Mariadb junto con el modulo PyMySQL de python (para poder ejecutar los módulos mysql de Ansible), arrancar y activar el servicio Mariadb y aplicar las medidas de seguridad básicas del servidor de bases de datos. El rol, recibe como parámetro la contraseña de root (usuario de la base de datos, no confundir con el del sistema), la cual tiene como valor por defecto “insecure”. Al igual que el rol de servidor web, se incluye como dependencia el rol de servidor.

Los tests automáticos se pueden encontrar en el archivo tests/test_default.py dentro del rol y comprueban que el usuario root está configurado con la contraseña “insecure” y no existe el usuario de pruebas “test”.

El repositorio del rol en GitHub es **iac_ansible-role-mariadb** y la versión correspondiente a la última entrega en la rama master es la 0.1.9

3.5.4 Rol de servidor de aplicación Wordpress

Aprovechando como dependencias todos los roles anteriores, el último rol del proyecto crea un servidor de aplicación Wordpress. Dado que las tareas de configuración de los servicios Web y de Bases de datos son realizadas por las dependencias, las tareas de éste rol se centran únicamente en configurar la aplicación. En primer lugar, instala los paquetes del lenguaje php y su módulo para mysql (valido para MariaDB). Una vez php está instalado, configura una base de datos y un usuario que usará la aplicación para conectarse. Tanto la contraseña del usuario para Wordpress como de administración de la DB se pueden especificar como parametros del rol, estando con valor ‘insecure’ por defecto. Tras dejar preparadas las dependencias, descarga la última versión de Wordpress de su web (http://wordpress.org/latest.tar.gz), extrae los archivos y los copia al directorio /var/www/html/customapp (el cual es configurado previamente por el rol de apache).

Como resultado de la ejecución del rol, al acceder a la url del servidor aparece la página de la configuración inicial de Wordpress (Figura 23).

El test automático se puede encontrar en el archivo tests/test_default.py dentro del rol y comprueba que la página de Wordpress está disponible en la url http://localhost.

El repositorio del rol en GitHub es **iac_ansible-role-wordpress** y la versión correspondiente a la última entrega en la rama master es la 0.1.4

3.6 Flujo de creación de máquina virtual

Con éste flujo de Jenkins, se pueden crear de forma automatizada máquinas virtuales, configuradas como servidor básico, servidor web, servidor de bases de datos o servidor de aplicación Wordpress. La creación se hace a partir de una de las imágenes generadas de CustomOS y se permite ajustar el hardware virtual de la máquina.

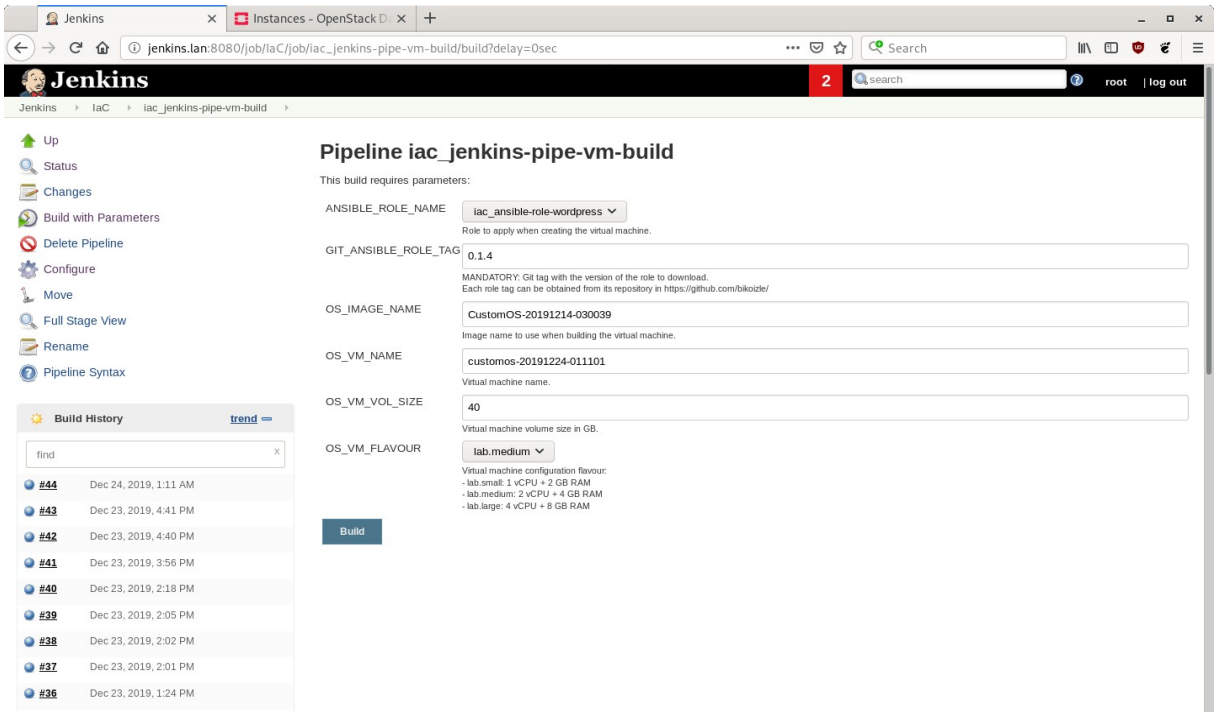


Figura 18: Opciones de creación de máquina virtual

Dado que el flujo está configurado para recibir parámetros, al iniciarlo se mostrará un formulario en el que se incluyen las opciones de creación de la máquina virtual (Figura 18).

El formulario de parámetros, contiene los siguientes campos a rellenar obligatoriamente:

ANSIBLE_ROLE_NAME: Rol que se aplicará para configurar la máquina, las opciones a elegir son iac_ansible-role-server (rol de servidor básico), iac_ansible-role-apache (rol de servidor web), iac_ansible-role-mariadb (rol de servidor de bases de datos) e iac_ansible-role-wordpress (rol de servidor Wordpress) (en el ejemplo iac_ansible-role-wordpress).

GIT_ANSIBLE_ROLE_TAG: Tag del repositorio de git del rol a aplicar, que coincide con la versión del rol (en el ejemplo 0.1.4).

OS_IMAGE_NAME: Nombre de la imagen que se usará para construir la máquina virtual (en el ejemplo: CustomOS-20191214-030039)

OS_VM_NAME: Nombre de la nueva máquina virtual (en el ejemplo: customos-20191224-011101).

OS_VM_VOL_SIZE: Tamaño del disco de la máquina virtual en Gigabytes (en el ejemplo: 40 GB).

OS_VM_FLAVOUR: Perfil de hardware virtual de OpenStack (“flavour”), las opciones a elegir son lab.small (1 vCPU + 2 GB RAM), lab.medium (2 vCPU + 4 GB RAM) y lab.large: (4 vCPU + 8 GB RAM) (en el ejemplo: lab.medium).

Las fases de éste flujo están basadas en el flujo de pruebas de roles, con la diferencia de que la fase de ejecución de playbook se ha renombrado a “Configure VM” y la fase de prueba de rol desaparece (Figura 19). Además, la fase de “Clean up” solo borra el entorno de ejecución Ansible, dejando la máquina virtual arrancada.

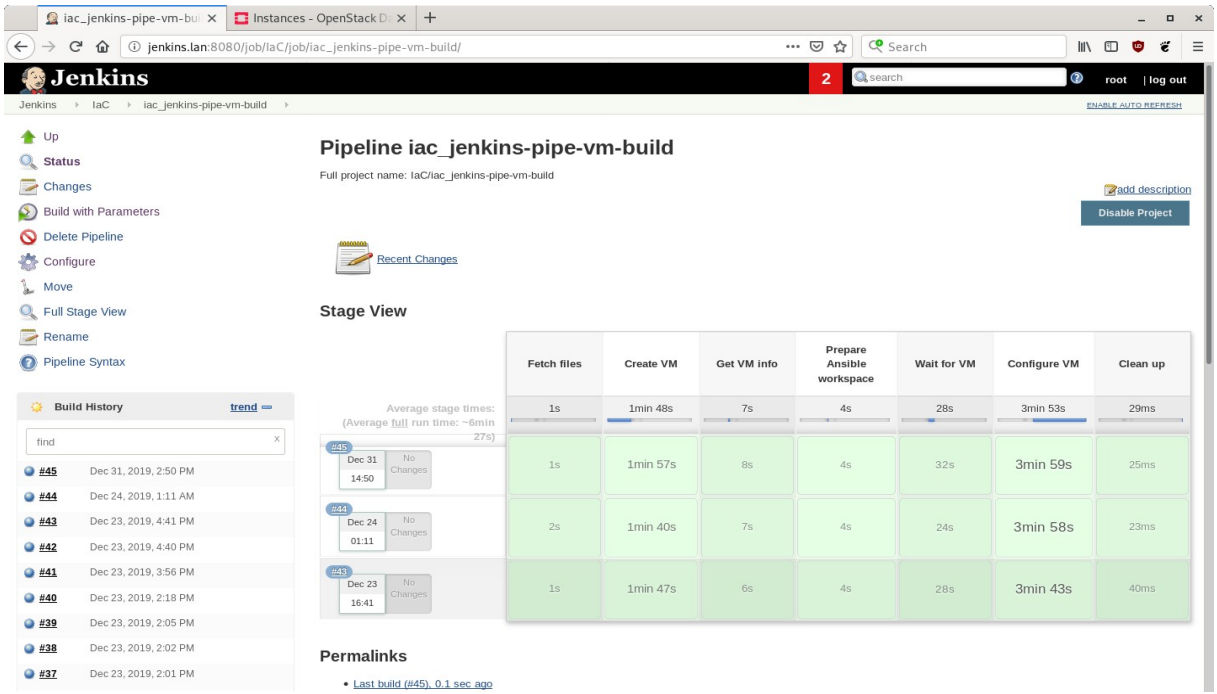


Figura 19: Fases del flujo de creación de máquina virtual

Una vez ejecutado el flujo, la máquina virtual aparecerá en la consola de OpenStack, pudiendo comprobar que se le ha aplicado el perfil de hardware virtual seleccionado y que se le ha asignado la dirección IP 192.168.33.53 en la red local (Figura 20).

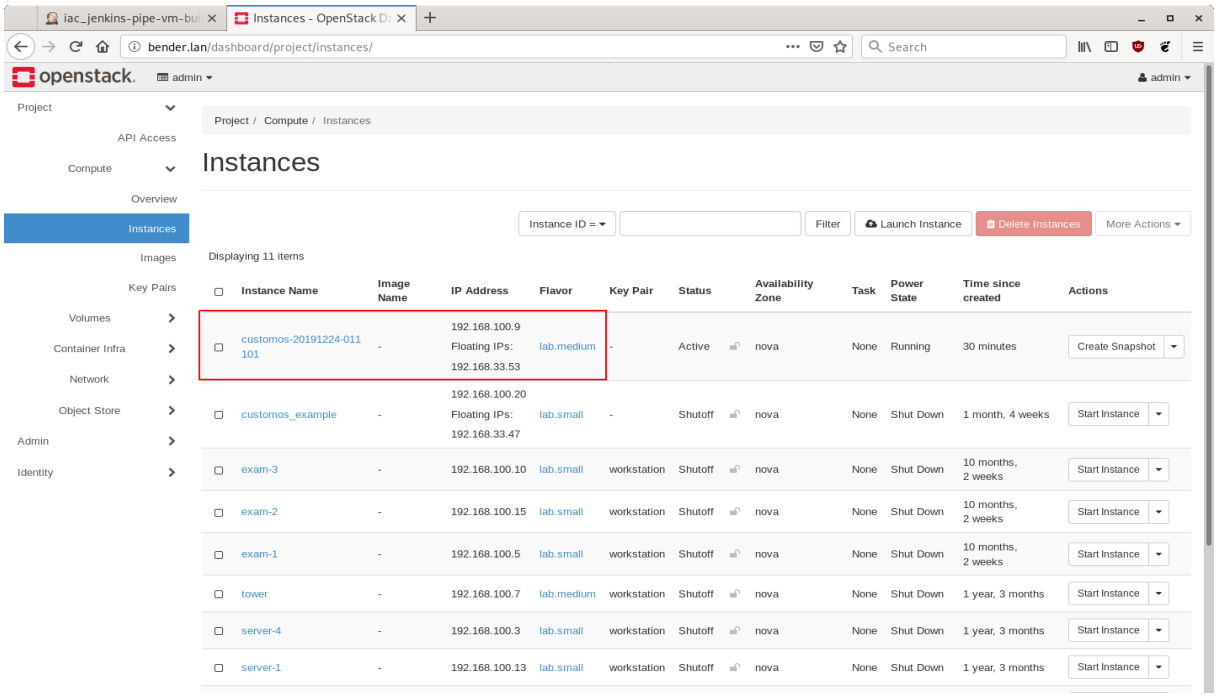


Figura 20: Máquina virtual creada en OpenStack

Usando un cliente ssh y el usuario customos es posible entrar en la máquina y ejecutar sudo para conseguir acceso root. Si se ejecuta el comando lsblk se puede ver que el espacio del disco es de 40 GB y que las particiones del sistema están dimensionadas acorde éste con el comando df -h (Figura 21).

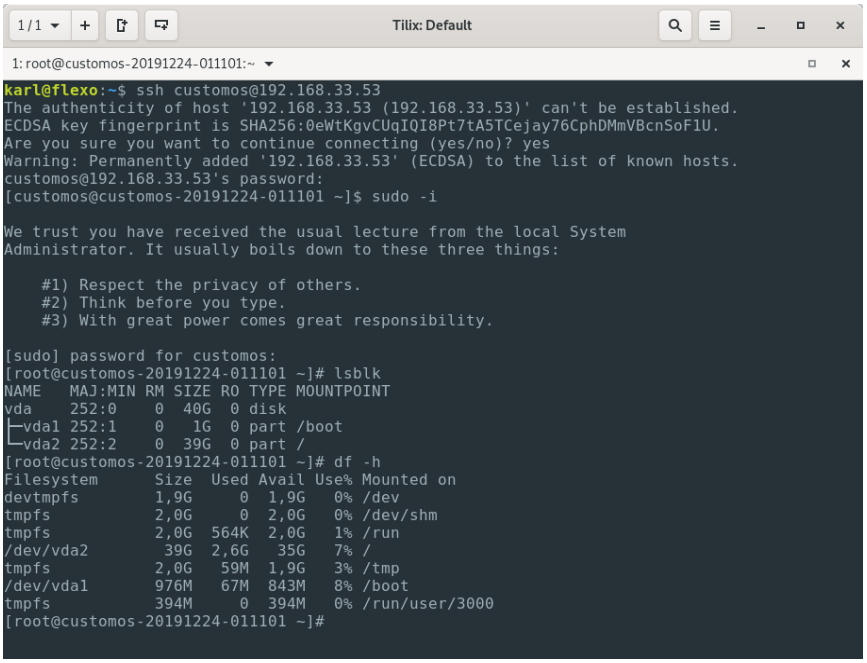


Figura 21: Acceso a la máquina virtual por ssh

Con systemctl se puede comprobar que el servicio principal de ClamAV (clamd@scan) está arrancado (Figura 22) y en el caso del servicio extra clam-freshclam las actualizaciones de antivirus se han descargado (Figura 23).

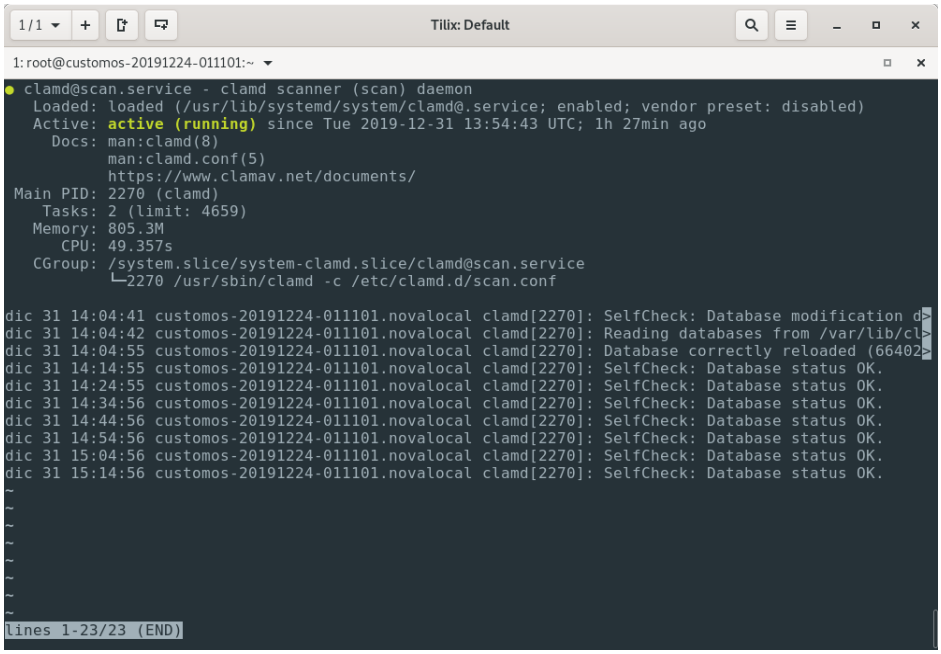


Figura 22: Servicio clamd@scan ejecutado y activado

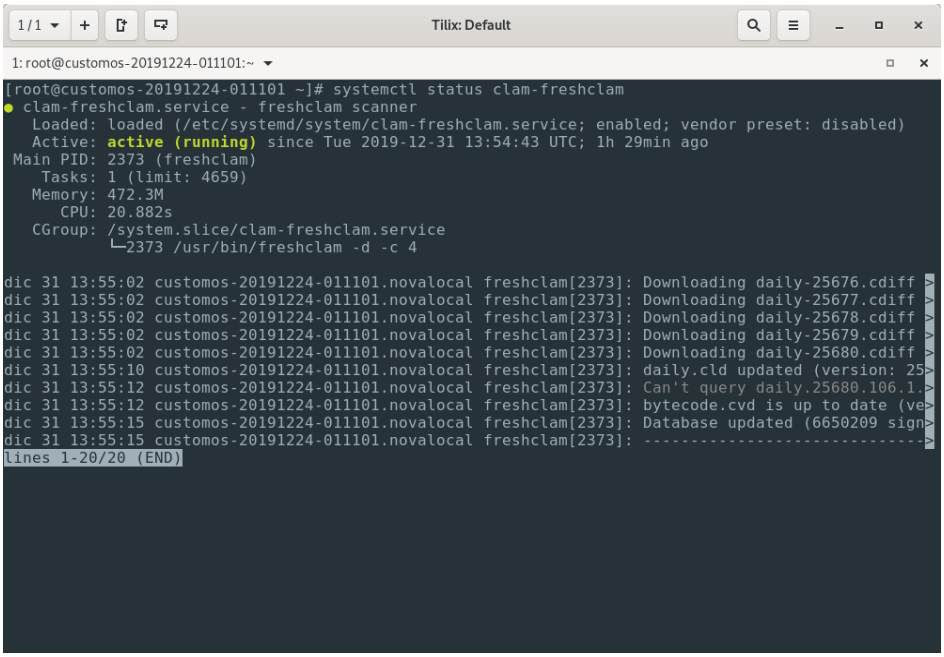


Figura 23: Servicio extra clam-freshclam descargando bases de datos de antivirus

De igual forma, el servicio del agente Zabbix se encuentra arrancado y esperando la conexión con el servidor (Figura 24).

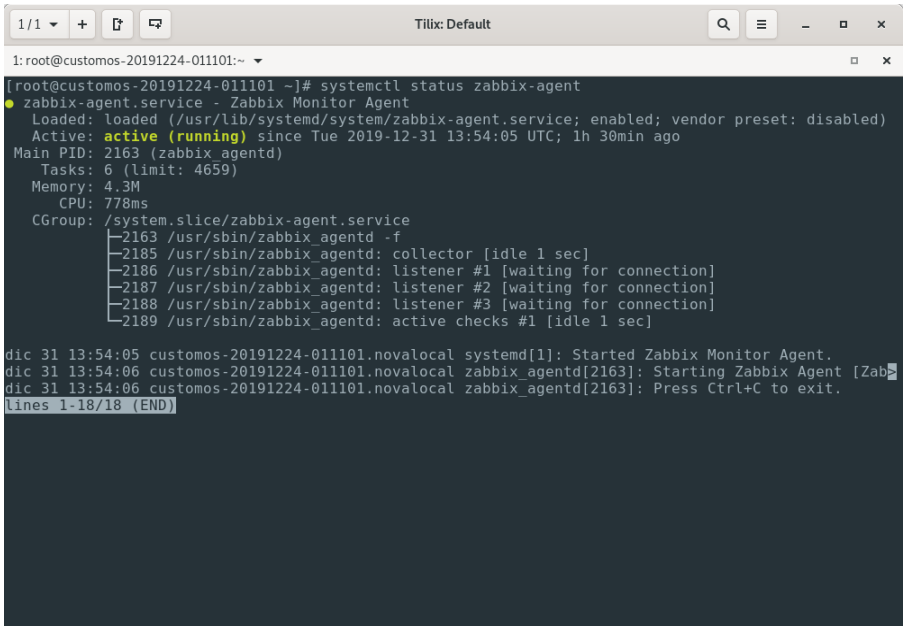


Figura 24: Agente Zabbix ejecutado y activo

El estado de SELinux es Permissive y el servicio de firewall, firewalld, está parado y desactivado (Figura 25).

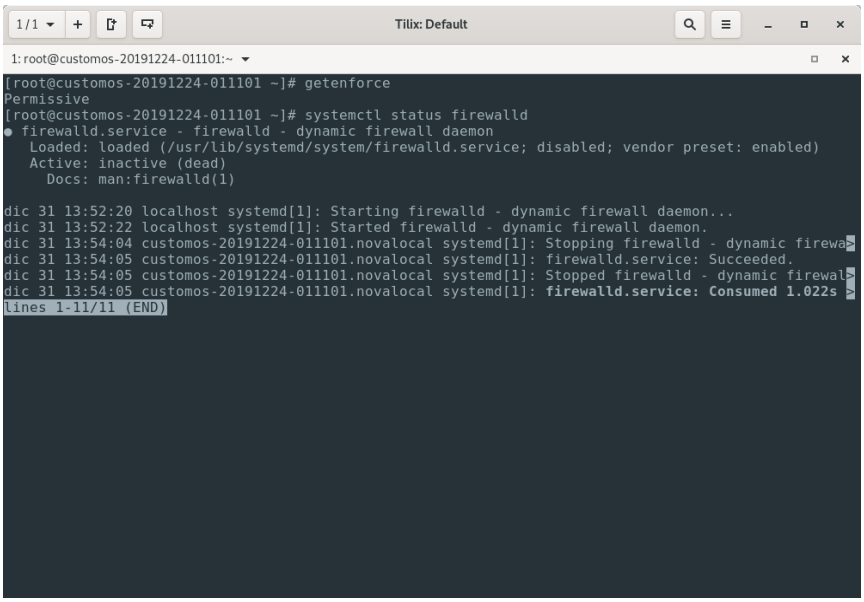


Figura 25: SELinux con estado Permissive y firewalld parado y desactivado.

Por último, abriendo un navegador web a la dirección http://192.168.33.53/ aparece la página inicial de configuración de Wordpress (Figura 26).

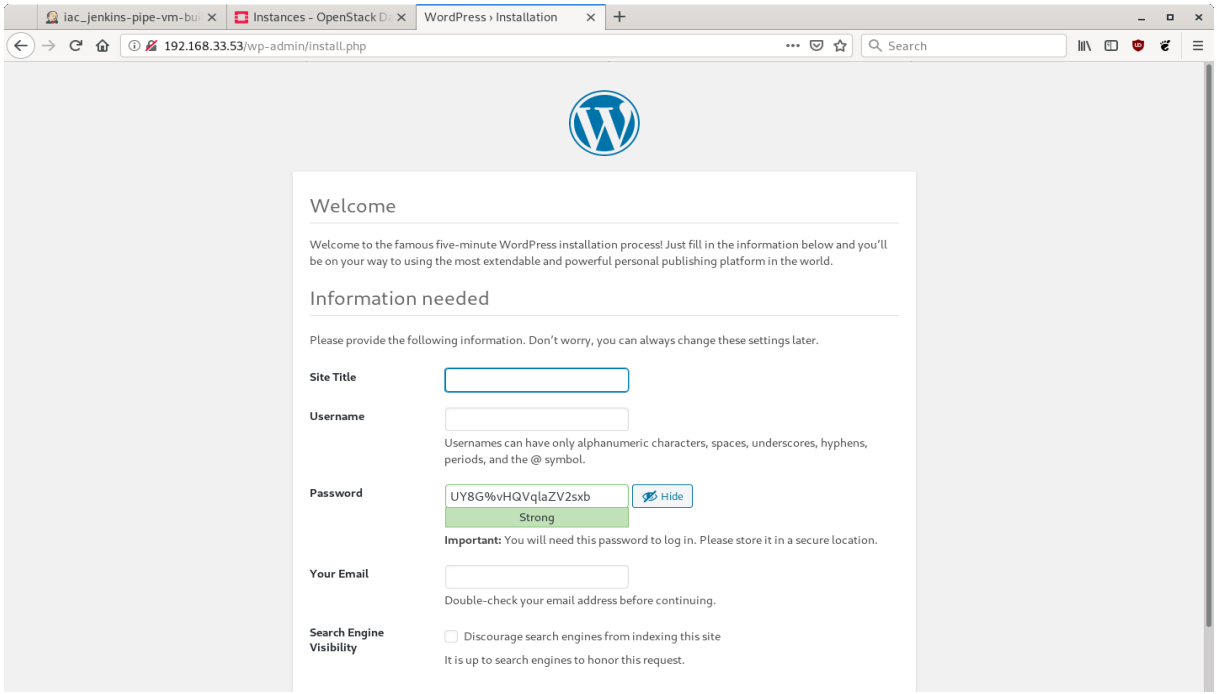


Figura 26: Página inicial de configuración de Wordpress en la máquina virtual

El flujo se puede encontrar en el repositorio iac_jenkins-pipe-vm-build y la versión usada en la última entrega es la 0.3.5, disponible en la rama master.

4 Conclusiones finales

Los entregables de éste proyecto podrían ser usados como referencia para crear un entorno productivo, ya fuera basado en OpenStack como nube interna o en nubes públicas como AWS o Google Cloud Platform, gracias a la modularidad de cada componente. Además, todos los componentes y programas usados son de código abierto, por lo que su utilización no requiere inversión (sin tener en cuenta los costes de la infraestructura física interna o externa). Como contrapartida, al ser software libre, el soporte recae en la comunidad, por lo que para garantizar la resolución rápida de fallos en entornos productivos, se debería contratar algún tipo de soporte externo o adoptar versiones comerciales de las soluciones empleadas.

Teniendo en cuenta que durante el proyecto no se ha requerido la participación de ningún recurso adicional, ni de activos provistos por terceros, ni de dependencias con otros proyectos; los riesgos que podían afectar los tiempos de entrega han sido mínimos.

Finalmente, existen mejoras que podrían hacer evolucionar el proyecto, como adaptar los flujos de Jenkins para su uso con Molecule, a la hora de probar los roles de Ansible. Molecule permite realizar mayor número de pruebas a los roles de Ansible y también utiliza pytest. Otra mejora podría ser el uso de módulos avanzados de Ansible en los playbooks para crear una infraestructura de aplicación entera en OpenStack, conocida como **stack**, que incluye elementos de red virtuales y los servidores que forman el aplicativo. Llendo aún más lejos, los playbooks de Ansible se podrían adaptar para usarse con Kubernetes, los cuales son similares a las stacks y soportadas por la mayoría de soluciones de nube.

5 Referencias

Ansible documentation: https://docs.ansible.com/

Apache HTTPD Server documentation: https://httpd.apache.org/docs/

ClamAV documentation: https://www.clamav.net/documents

Extreme Programming: https://en.wikipedia.org/wiki/Extreme_programming

Fedora User documentation: https://docs.fedoraproject.org/

Infrastructure as Code - Managing Servers in the Cloud - Kief Morris - ISBN: 9781491924358

Jenkins User documentation: https://jenkins.io/doc/

Kubernetes: https://kubernetes.io/

Lorax documentation: https://weldr.io/lorax/

MariaDB documentation: https://mariadb.org/documentation/

Molecule: https://molecule.readthedocs.io/

OpenStack documentation: https://docs.openstack.org/

Packstack - Create a proof of concept cloud: https://www.rdoproject.org/install/packstack/

Wordpress support: https://wordpress.org/support/

Zabbix manuals: https://www.zabbix.com/manuals