Infraestructura como código (IaC). Monitorización y seguridad

PID_00286211

Remo Suppi Boldrito

Tiempo mínimo de dedicación recomendado: 2 horas

Universitat Oberta de Catalunya

Remo Suppi Boldrito

Ingeniero de Telecomunicaciones. Doctor en Informática por la Universidad Autónoma de Barcelona (UAB). Profesor del Departamento de Arquitectura de Computadores y Sistemas Operativos en la UAB.

El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por el profesor: Josep Jorba Esteve

Primera edición: febrero 2022

Av. Tibidabo, 39-43, 08035 Barcelona
Autoría: Remo Suppi Boldrito
Producción: FUOC

Índice

Objetivos

Los objetivos principales de este módulo son los siguientes:

1. Conocer los diferentes aspectos y herramientas que se aplican al concepto y metodología de IaC.

2. Estudiar y desarrollar conocimientos sobre la monitorización de infraestructuras.

3. Analizar los principales aspectos de seguridad tanto en entornos locales como de red.

4. Desplegar pruebas de concepto y ejemplos de uso con las principales herramientas para cada objetivo propuesto anteriormente.

El estudiante deberá centrar su atención en los siguientes conceptos fundamentales de este módulo:

- Infraestructura como código.
- Monitorización.
- Seguridad.
- Casos de uso y herramientas.

Se recomienda analizar y realizar pruebas de instalación y configuración de las principales herramientas indicadas.

1. ¿Qué es la infraestructura como código?

En este módulo se analizarán los principales conceptos de la infraestructura como código (IaC) con la lectura de los capítulos 1 y 2 del libro ***Infrastructure as Code***,[1] **segunda edición (Kief Morris, O'Reilly)**. Los conceptos principales se centran en definir:

[(1)]Se puede encontrar este libro en el repositorio de la UOC en el siguiente enlace: https://bit.ly/3Ho3Dt8

- Las cuatro métricas principales de la entrega de software y el rendimiento operativo: plazo de entrega, frecuencia de implementación, porcentaje de fallos y tiempo medio de restauración (MTTR).

- Las tres prácticas básicas para infraestructura como código que permiten lograr aportar valor a estas: definir todo como código, probar y entregar de forma continua mientras se desarrolla y construir piezas pequeñas y simples que se puedan cambiar independientemente.

Esto se complementará con los principios de la infraestructura de la era de la nube del capítulo 2, en el que se muestran las diferencias entre la infraestructura estática tradicional y la infraestructura dinámica moderna en cuanto a conceptos como confiabilidad, reproducibilidad y repetibilidad, entre otros.

El estudiante podrá ampliar sus conceptos ya obtenidos en otras asignaturas o módulos haciendo una revisión del capítulo 3, en el que se describen los elementos de la infraestructura.

2. Definiendo todo como código

De la lectura del capítulo 4 el estudiante verá cuáles son los lenguajes y entornos más utilizados para la gestión de la infraestructura (Puppet, Chef, Ansible y Saltstack, entre otros) para servidores virtualizados y *cloud* IaaS o herramientas y entornos llamados *stack-oriented tools* como Terraform y CloudFormation, que han significado una simplificación en IaC. No menos importantes son las aportaciones de nuevas tendencias que utilizan lenguajes de propósitos generales para definir la infraestructura, como Pulumi y AWS CDK (*Cloud Development Kit*), que soportan Typescript, Python, y Java, como apuesta para solucionar las limitaciones de los lenguajes declarativos utilizados por otras herramientas.

El estudiante podrá complementar estos conceptos con los capítulos 5, 6 y 7, que le permitirán tener una visión completa de los aspectos más importantes de IaC:

- cómo se desarrolla un *stack* como código,
- cómo se construye un entorno con *stacks* y
- cómo se configuran estos *stacks*.

3. Casos de uso: Ansible, Vagrant y Terraform

Para sentar los principales aspectos de IAC, se realizarán tres pruebas de concepto centradas en las tres categorías principales de una de las clasificaciones más utilizadas:

1) Herramientas de gestión de configuraciones: también conocidas como *configuración como código*, son herramientas especializadas diseñadas para gestionar software. Normalmente, se centran en instalar y configurar servidores.

Ejemplos de herramientas de gestión de configuraciones

Cheff, Puppet y **Ansible**.

2) Contenedores y herramientas de plantillas (*templating tools*): estas herramientas generan plantillas o imágenes precargadas con todas las bibliotecas y los componentes necesarios para ejecutar una aplicación. Las cargas de trabajo en contenedores son fáciles de distribuir y tienen un gasto general muy inferior al que se ejecuta en un servidor.

Ejemplos de contenedores y herramientas de plantillas

Docker, rkt, **Vagrant** y Packer.

3) Herramientas de provisión y despliegue: las herramientas de provisión se centran en la creación de infraestructuras. Mediante este tipo de herramientas, los desarrolladores pueden definir componentes exactos de la infraestructura.

Ejemplos de herramientas de provisión y despliegue

Son ejemplos **Terraform**, AWS CloudFormation y OpenStack Heat.

3.1. Ansible

Es una plataforma código abierto (ahora propiedad de Red Hat) para configurar y administrar muy fácilmente sistemas IT, desplegar aplicaciones y software en los elementos de infraestructura y con una particularidad muy atractiva: solo requiere SSH para hacer el despliegue.

Esta herramienta permite el software *provisioning*, gestión de configuraciones y despliegue de aplicaciones en lo que se conoce como *infraestructura como código*, e incluye su propio lenguaje declarativo para describir la configuración del sistema. Ansible fue escrito por M. DeHaan (desarrollador de Cobbler –*Linux provisioning server*–) y adquirido por Red Hat en 2015.

La diferencia de Ansible y su popularidad sobre otras herramientas similares (como Chef o Puppet), que necesitan un agente en el *host* remoto y por lo que los servidores donde será desplegado el *stack* deben acondicionarse antes para este fin, se debe a que Ansible solo necesita una conexión SSH y Python (2.4 o posterior) para llevar a cabo una acción en los nodos que deben configurarse, y por lo general todos los servidores ya disponen de este software en su instalación básica.

Ansible integra módulos que trabajan en formato JSON y utiliza YAML para describir configuraciones reutilizables, por lo que permite que los desarrolladores puedan describir los patrones para aprovisionar el software.

Ansible tiene una arquitectura que distingue entre el controlador y los nodos, donde el primero inicia la orquestación y es quien gestiona por SSH los nodos que conoce a través de un inventario.

Ansible inserta módulos en los nodos (mediante SSH) que se guardan temporalmente y se comunican con el controlador mediante el protocolo JSON sobre una salida estándar.

Su diseño está basado en una arquitectura minimalista (sin imponer dependencias adicionales), consistente, segura y de alta confiabilidad. Dado que cada módulo puede ser escrito en cualquier lenguaje estándar (Python, Perl, Ruby, Bash, etc.), permite a cualquier programador o responsable de aprovisionamiento estar operativo rápidamente, incluso para infraestructuras complejas.

Entre las definiciones de este entorno se encuentran las siguientes:

- **Nodo de control**: cualquier máquina con Ansible instalado que puede ejecutar comandos y *playbooks* invocando `ansible` o `ansible-playbook`, que actúa como nodo de control (es importante tener en cuenta que no se puede utilizar una máquina Windows como nodo de control). Se pueden tener varios nodos de control.

- **Nodos gestionados**: los dispositivos de red (o servidores) que se gestionan por Ansible. Los nodos gestionados también se llaman *anfitriones*. Ansible no está instalado en los nodos gestionados.

- **Inventario**: una lista de nodos gestionados. Un archivo de inventario también se denomina a veces *archivo host*. El inventario puede especificar información como la dirección IP para cada nodo gestionado o puede organizar nodos gestionados, creando y nidificando grupos para facilitar el escalado.

- **Colecciones**: las colecciones son un formato de distribución para el contenido de Ansible que puede incluir *playbooks*, roles, módulos y conectores. Se pueden instalar y utilizar colecciones mediante el entorno Ansible Galaxy.

- **Módulos**: unidades de código que Ansible puede ejecutar para un uso particular, desde la administración de usuarios en un tipo específico de base de datos hasta la gestión de interfaces VLAN en un tipo específico de dispositivo de red. Puede invocar un único módulo con una tarea o invocar varios módulos diferentes en un *playbook*. A partir de Ansible 2.10, los

Enlace recomendado

Podéis profundizar en la sintaxis de los *playbooks* en el siguiente enlace: https://bit.ly/3JY7sXK.

módulos se agrupan en colecciones; en el enlace recomendado se puede ver una lista de todos estos.

- **Tareas**: las unidades de acción en Ansible. Puede ejecutar una sola tarea una vez con un comando *ad hoc*.

- ***Playbooks***: listas ordenadas de tareas, guardadas para que pueda ejecutar estas tareas en este orden repetidamente. Los *playbooks* pueden incluir variables y tareas, están escritos en YAML y son fáciles de leer, escribir, compartir y entender.

La instalación de Ansible se puede hacer desde los repositorios de las distribuciones (se encuentra disponible en la mayor parte de ellas) o también a través del comando `pip` para la versión de Python correspondiente.

Enlace recomendado

Podéis consultar más información en la Installation Guide.

Después de instalado, se deben generar las llaves PKI para que SSH se pueda conectar a los nodos gestionados por llave pública y privada y realizar el inventario, como por ejemplo:

1) Generar llaves público-privadas y copiarlas a los nodos gestionados mediante: `ssh-keygen`; para copiarlas, `ssh-copy-id root@IP_nodo_gestionado` haciendo lo mismo para todos los nodos. Hay que verificar la conexión y recordar que en los clientes se debe habilitar en */etc/ssh/sshd_config* que el usuario *root* se pueda conectar con **PermitRootLogin yes**.

2) Generar el inventario editando el archivo */etc/ansible/hosts* agregando todos los *hosts*/clientes (creando etiquetas o utilizando las que ya se han indicado, por ejemplo, en este caso *webservers*).

```
[webservers]
mva
mvb
mvc
```

3) También se puede agregar un directorio */etc/ansible/var_hosts* con las definiciones y configuraciones del *host*, por ejemplo, **mvb**:

```
ansible_ssh_host: 172.16.1.2
ansible_ssh_port: 22
ansible_ssh_user: root
```

4) Con ello, ya se puede comprobar haciendo:

```
ansible -m ping webservers       ansible -m ping mvb
ansible -m command -a "df -h" webservers
ansible -m command -a "free -mt" webservers
ansible -m command -a  "uptime" webservers
```

```
ansible -m command -a "arch" webservers
ansible -m shell -a "hostname" webservers
ansible webservers -m copy -a "src=/etc/hosts dest=/tmp/hosts"
ansible all -m user -a 'password=$6$n... n2Mn1 name=testing'
```

Para generar el *passwd* para el usuario *testing* del último comando, se podrá instalar el paquete *whois* y ejecutar `mkpasswd –method=SHA-512` o también generarlo con `openssl`.

Como se puede observar, es muy simple de poner en marcha y muy potente para desplegar infraestructura software, ya que no presenta particularidades ni complejidades de instalación/configuración como otros paquetes (por ejemplo, Puppet, Capistrano, Salt o Chef).

Generar un *playbook* resulta muy fácil, ya que son en texto plano (en formato YAML) y se permite organizar tareas complejas mediante ítems con formato **key: values.**

Dentro de un *playbook* podemos encontrar uno o más grupos de ***hosts*** (cada uno se denomina *play*) donde se realizarán las tareas. Por ejemplo, para crear uno hacemos: `mkdir /etc/ansible/playbooks`; y dentro de este directorio se crea un archivo con `vi /etc/ansible/playbooks/apache.yml`.

```
---
- hosts : webservers
  tasks:

  - name: install apache2
    apt:
      name: apache2
      update_cache: yes
      state: latest
  - name: copy index.html
    copy:
      src: /tmp/index.html
      dest: /var/www/html/
  - name:  restart apache2
    service:
      name: apache2
      state: started
```

Luego se genera el archivo `vi /tmp/index.html`.

```
<!DOCTYPE html>
<html lang="en">
<head><meta charset="utf-8"/></head>
<body>
```

```
    <h1>Apache was started in this host via Ansible </h1><br>
    <h2>Ansible is Easy !! </h2>
</body></html>
```

Ejecutando `ansible-playbook /etc/ansible/playbooks/apache.yml`, posteriormente, en un navegador, se podrá ver el resultado del despliegue en todas las máquinas declaradas en el investario.

Enlaces recomendados

Deberéis complementar vuestros conocimientos analizando y ejecutando otros *playbooks* desde el conjunto de ejemplos propuestos por Ansible y prestar atención a su sintaxis.

3.2. Vagrant

Esta herramienta es útil en entornos IT en los que se necesite desarrollo continuo y desempeña un papel diferente al de otras herramientas (por ejemplo, Docker), pero se encuentra orientada hacia los mismos objetivos: proporcionar entornos fáciles de configurar, reproducibles y portátiles con un único flujo de trabajo, que ayudará a maximizar la productividad y la flexibilidad en el desarrollo de aplicaciones/servicios.

El aprovisionamiento de máquinas puede ser desde distintos proveedores (VirtualBox, Vmware, AWS u otros) utilizando *scripts*, Chef o Puppet para instalar y configurar automáticamente el software de las máquinas virtuales.

Para los desarrolladores, Vagrant aislará las dependencias y configuraciones dentro de un único entorno disponible, consistente, sin sacrificar ninguna de las herramientas que el desarrollador utiliza habitualmente y teniendo en cuenta un archivo llamado *Vagrantfile*. El resto de los desarrolladores tendrán el mismo entorno, aunque trabajen desde otros SO (pueden ser Linux, Windows o MacOS) o entornos, por lo que se conseguirá que todos los miembros de un equipo estén ejecutando código en el mismo entorno y con las mismas dependencias.

Además, en el ámbito IT permite tener entornos desechables con un flujo de trabajo coherente para desarrollar y probar *scripts* de administración de la infraestructura, ya que rápidamente se pueden hacer pruebas en un entorno de virtualización local, como VirtualBox o Vmware. Luego, con la misma configuración, puede probar estos *scripts* en el *cloud* (por ejemplo, AWS o Rackspace) utilizando el mismo flujo de trabajo.

Su instalación puede ser de los repositorios del sistema operativo o simplemente descargar e instalar el paquete desde el desarrollador. Su puesta en marcha es sumamente fácil y se dispone de guías para rápidamente poner en marcha un *Box* de entre una gran cantidad disponible y listos para funcionar.

Se pueden hacer fácilmente pruebas sobre cualquier sistema operativo de los habituales, ya que solamente se debe tener instalado VirtualBox. Para cargar un *box* se puede hacer lo siguiente:

- `vagrant init centos/7`: inicializará/generará un archivo **Vagrantfile** con las definiciones de la VM.
- `vagrant up`: descargará del repositorio del Vagrant una imagen de Centos/7 y la pondrá en marcha.
- `vagrant ssh`: para acceder a la MV.
- `vagrant destroy`: para eliminar la MV.

Se pueden ver las imágenes preconfiguradas y precargarlas desde el *cloud* simplemente haciendo `vagrant box add nombre` (por ejemplo, podemos usar `vagrant box add ubuntu/jammy64`). Cuando se ejecute el `up` se observarán en la consola una serie de mensajes entre los que se indicará el puerto para conectarse a la MV (por ejemplo, `ssh vagrant@localhost -p 2222` y la contraseña vagrant). También se puede hacer simplemente `vagrant ssh`.

Si se desea cambiar, por ejemplo, la versión o configuración de la MV, se puede modificar el archivo *Vagrantfile* y cambiar el contenido:

```
vagrant.configure("2") do |config|
     config.vm.box = "ubuntu/precise32"
end
```

Dentro del Vagrantfile se pueden agregar configuraciones específicas, como, por ejemplo, para instalar software:

```
  config.vm.provision  "shell", inline: <<-SHELL
   apt-get update
   apt-get install -y apache2
  SHELL
```

O para agregar un puerto:

```
config.vm.network  "forwarded_port", guest: 80, host: 8080
```

Enlaces recomendados

Deberéis analizar la diferente sintaxis que se puede utilizar en Vagrantfile y otros aspectos interesantes de esta herramienta como son el aprovisionamiento, red, *folders*, *cloud-init* y *plugins*, entre otros.

3.3. Terraform

La premisa de esta herramienta es *«describe your complete infrastructure as code and build resources across providers»*. Es una herramienta desarrollada por Hashicorp para IaC teniendo en mente la filosofía de *write, plan* y *apply*.

> Es una herramienta *open source* para desarrollar IaC a través de una línea de comandos (CLI) y gestionar centenares de servicios *cloud*. Terraform codifica las API de los *cloud* en archivos de configuración declarativos que permiten una descripción fácil y simple de la infraestructura que se quiere desplegar, al tiempo que permite construir, cambiar y gestionar la infraestructura de una manera segura y repetible.

Los operadores y equipos de infraestructuras pueden utilizar Terraform para gestionar entornos con un lenguaje de configuración llamado *HashiCorp Configuration Language* (HCL) para despliegues automatizados y legibles. Como ya se ha explicado, la IaC como concepto es el proceso de gestión de la infraestructura en un archivo o más en lugar de configurar manualmente los recursos en una interfaz de usuario; es un recurso de cualquier infraestructura de un determinado entorno, como una máquina virtual, un grupo de seguridad, una interfaz de red, etc.

Con un alto nivel de abstracción, Terraform permite a los operadores utilizar HCL para crear archivos que contengan definiciones de los recursos deseados en casi cualquier proveedor (AWS, GCP, GitHub, Docker, etc.), automatiza la creación de estos recursos en el momento de realizar la solicitud y trabaja básicamente con cinco órdenes principales:

1) ***init*** (prepara el entorno)

2) ***plan*** (muestra los cambios requeridos para la configuración)

3) ***apply*** (crea o actualiza la infraestructura)

4) ***destroy*** (elimina la infraestructura)

5) ***validate*** (verifica la sintaxis)

Los pasos que se deben seguir (*workflow*) serán los siguientes:

1) Alcance: confirmar qué recursos es necesario crear para un proyecto determinado.

2) Autor: crear el archivo de configuración en HCL basado en los parámetros de alcance

3) *Init*: ejecutar `terraform init` en el directorio del proyecto con los archivos de configuración. Esto descargará los conectores de proveedor correctos para el proyecto.

4) *Plan & Apply*: se ejecuta `terraform plan` para verificar el proceso de creación y, a continuación, `terraform apply` para crear recursos reales, así como un archivo de estado que compara los cambios futuros de los archivos de configuración con el que existe realmente en su entorno de despliegue.

Entre las principales ventajas que se pueden mencionar es que es agnóstico en relación con la plataforma/*cloud*, gestiona de modo simple el estado de la infraestructura (*state management*), mejora la confianza del operador (*operator confidence*) y está diseñado para escalar y simplificar las mejores prácticas de uso de forma transparente.

En un CPD moderno pueden existir diferentes entornos para dar soporte a las diversas aplicaciones, pero esto no afecta a Terraform, ya que se puede gestionar un entorno heterogéneo con el mismo flujo de trabajo creando un archivo de configuración.

Terraform crea un archivo de estado cuando se inicia un proyecto y se utiliza este estado local para crear planes y realizar cambios en la infraestructura; antes de cualquier operación, se hace una actualización del estado con la infraestructura real. Si se realiza un cambio o se añade un recurso a una configuración, se comparan estos cambios con el archivo de estado para determinar qué cambios resultan en un nuevo recurso o modificaciones del recurso. El flujo de trabajo integrado tiene como objetivo infundir confianza a los usuarios promoviendo operaciones fácilmente repetibles y una fase de planificación para permitir a los usuarios asegurarse de que las acciones realizadas no causen interrupciones en su entorno.

Veremos ahora una prueba muy simple para instalar Terraform y desplegar un recurso (en este ejemplo se utilizará Docker como proveedor, pero podría ser cualquier otro). Para su instalación se puede hacer directamente del repositorio haciendo *unzip* y moviéndolo al directorio adecuado (por ejemplo, */usr/bin*) o instalarlo del repositorio de la distribución (por ejemplo, con `apt update`; `apt install terraform`). Para desplegar un contenedor con Nginx, se debe crear el *main.tf*:

```
terraform {
   required_providers {
    docker = {
      source = "terraform-providers/docker"
      }
```

```
    }
}


provider "docker" {}


resource "docker_image" "nginx" {
  name = "nginx:latest"
  keep_locally = false
}


resource "docker_container" "nginx" {
  image = docker_image.nginx.latest
  name  = "tutorial"
  ports {
    internal = 80
    external = 8000
    }
}
```

Luego se deberá ejecutar `terraform init` y después `terraform apply` y, finalmente, ya se podrá conectar al puerto 8000 del navegador para ver la página inicial de Nginx. Se pueden mirar los contenedores con `docker ps`, y el estado con `terraform show`. Para terminar, se puede ejecutar el comando `terraform destroy`.

Enlaces recomendados

Deberéis complementar vuestro estudio con los diferentes apartados de los distintos aspectos de Terraform CLI, Modules y Provision.

4. Monitorización y seguridad

Dos aspectos relevantes para una infraestructura son su monitorización y la seguridad. Sobre estos conceptos, en este apartado deberéis seguir el material publicado en Administración avanzada del sistema operativoGNU/Linux, en los módulos «Sintonización, optimización y alta disponibilidad» y «Administración de seguridad» respectivamente.

Para **monitorización**, deberéis centraros en la introducción y, a continuación, en los aspectos básicos de sintonización, optimización y alta disponibilidad para analizar estos conceptos y adquirir los conocimientos necesarios sobre los principales aspectos del tema. Posteriormente, deberéis analizar y ejecutar pruebas de concepto y configuraciones de diferentes herramientas, como Nagios, Ganglia, Munin, Monit, SNMP + MRTG y Cacti, entre otras.

Para la monitorización de contenedores, deberéis analizar cAdvisor (*container advisor*), que proporciona información sobre el uso de recursos y las características de rendimiento de sus contenedores en ejecución. Está basado en un *daemon* que recopila, agrega, procesa y exporta información sobre contenedores en ejecución manteniendo para cada contenedor los parámetros de aislamiento de recursos, uso histórico de recursos, histogramas de uso histórico completo de recursos y estadísticas de red exportando estos datos por contenedor y en toda la máquina.

cAdvisor tiene soporte nativo para contenedores Docker y debería admitir casi cualquier otro tipo de contenedor listo para usar que se base en lmctfy, permitiendo esta abstracción que los contenedores estén intrínsecamente anidados de forma jerárquica.

Como complemento de la monitorización y gestión de contenedores, deberéis analizar Portainer.io (en su versión *Community*). Portainer es una interfaz de usuario de administración (liviana) que le permite administrar fácilmente diferentes entornos Docker (*hosts* Docker o clústeres *Swarm*) y está diseñada para ser tan simple de implementar como de usar. Consiste en un único contenedor que puede ejecutarse en cualquier motor Docker (se puede implementar como contenedor de Linux o contenedor nativo de Windows, y también es compatible con otras plataformas) y así administrar y monitorizar todos los recursos Docker (contenedores, imágenes, volúmenes y redes, entre otros). Su arquitectura se basa en dos elementos (Portainer Server y Portainer Agent), donde ambos se ejecutan como contenedores en una infraestructura en contenedores existente. El agente de Portainer debe implementarse en cada nodo del clúster que se va a monitorizar y este informará a Portainer Server de las evoluciones de contenedores bajo su ámbito para centralizar las medidas y el uso de recursos. Su instalación en sumamente simple, ya que solo se debe

Enlace recomendado

Para instalar un contenedor *standalone* sobre Linux, podéis ver el siguiente enlace: https://docs.portainer.io/v/ce-2.9/start/install/server/docker/linux.

instalar un contenedor *standalone* sobre Linux (podéis ver otros ejemplos en la misma documentación). Se puede ver una demo del Portainer en funcionamiento en http://demo.portainer.io/ (hay que utilizar *username* = admin y *password* = tryportainer).

Para los aspectos de **seguridad**, el estudiante se deberá concentrar en el módulo 4, en el que se analizan en primer lugar los principales conceptos (tipos y métodos de los ataques, técnicas utilizadas en los ataques, contramedidas y seguridad del sistema), para luego centrarse en la seguridad local (*bootloaders*, contraseñas y *shadows*, *bits sticky* y *setuid*, módulos PAM, alteraciones del sistema, limitación de recursos por *cgroups* y *chroot*, protección de ejecutables mediante *Hardening-Wrapper*, *SELinux*). Por último, el estudiante deberá analizar las herramientas de seguridad más comunes para la detección de vulnerabilidades e intrusiones (OpenVAS, Denyhosts, Fail2ban, Snort, Suricata), protección mediante filtrado (TCP wrappers y cortafuegos -iptables/nftables-), así como el análisis de registros.

Bibliografía

Jorba, J.; Suppi, R. (2016). Administración avanzada del sistema operativo GNU/Linux, módulos 4 y 5. http://openaccess.uoc.edu/webapps/o2/handle/10609/60686

Morris, K. (2020). *Infrastructure as Code* (2.ª ed.). O'Reilly Media.

Enlaces (accedidos por última vez en diciembre de 2021)

Ansible. «Ansible concepts». https://docs.ansible.com/ansible/latest/user_guide/basic_concepts.html#basic-

Ansible. «Automation controller». https://www.ansible.com/products/controller

GitHub. «cAdvisor». https://github.com/google/cadvisor

Github. «Rundeck ansible plugin». https://github.com/Batix/rundeck-ansible-plugin

Github. «Semaphore». https://github.com/ansible-semaphore/semaphore

Hashi Corp. «Introduction to Terraform». www.terraform.io/intro/index.html

Hashi Corp. «Terraform CLI Documentation». www.terraform.io/docs/cli/index.html

Hashi Corp. «Get started - Docker». https://learn.hashicorp.com/collections/terraform/docker-get-started

Hashi Corp. «Vagrant Documentation». www.vagrantup.com/docs

Portainer.io. «Portainer features». www.portainer.io/features

Rundeck. «Rundeck open source». https://www.rundeck.com/open-source

Servers for Hackers (2014). «An Ansible tutorial». https://serversforhackers.com/c/an-ansible-tutorial

The HomeLab Wiki. «Portainer.io». https://thehomelab.wiki/books/docker/page/portainer-git