Infraestructura com a codi (IaC). Monitoratge i seguretat

PID_00286204

Remo Suppi Boldrito

Temps mínim de dedicació recomanat: 2 hores

Universitat Oberta de Catalunya

Remo Suppi Boldrito

Enginyer de Telecomunicacions. Doctor en Informàtica per la Universitat Autònoma de Barcelona (UAB). Professor del Departament d'Arquitectura de Computadors i Sistemes Operatius en la UAB.

L'encàrrec i la creació d'aquest recurs d'aprenentatge UOC han estat coordinats pel professor: Josep Jorba Esteve

Primera edició: febrer 2022

Av. Tibidabo, 39-43, 08035 Barcelona
Autoria: Remo Suppi Boldrito
Producció: FUOC

Índex

Objectius

Els objectius principals d'aquest mòdul són els següents:

1. Conèixer els diferents aspectes i eines que s'apliquen al concepte i a la metodologia d'IaC.
2. Estudiar i desenvolupar coneixements sobre el monitoratge d'infraestructures.
3. Analitzar els principals aspectes de seguretat tant en entorns locals com de xarxa.
4. Desplegar proves de concepte i exemples d'ús amb les eines principals per a cada objectiu proposat anteriorment.

L'estudiant haurà de centrar l'atenció en els conceptes fonamentals següents d'aquest mòdul:

- Infraestructura com a codi.
- Monitoratge.
- Seguretat.
- Casos d'ús i eines.

Es recomana analitzar i fer proves d'instal·lació i configuració de les principals eines indicades.

1. Què és la infraestructura com a codi?

En aquest mòdul s'analitzaran els conceptes principals de la infraestructura com a codi (IaC) amb la lectura dels capítols 1 i 2 del llibre ***Infrastructure as Code***,[1] **segona edició (Kief Morris, O'Reilly)**. Els conceptes principals se centren a definir:

(1)Es pot trobar aquest llibre en el repositori de la UOC en l'enllaç següent: https://bit.ly/3vZFXZ8

- Les quatre mètriques principals del lliurament de programari i el rendiment operatiu: termini de lliurament, freqüència d'implementació, percentatge de fallades i temps mitjà de restauració (MTTR).

- Les tres pràctiques bàsiques per a infraestructura com a codi que permeten aconseguir aportar valor a aquestes: definir-ho tot com a codi, provar i lliurar de manera contínua mentre es desenvolupa i construir peces petites i simples que es puguin canviar independentment.

Això es complementarà amb els principis de la infraestructura de l'era del núvol del capítol 2, en el qual es mostren les diferències entre la infraestructura estàtica tradicional i la infraestructura dinàmica moderna quant a conceptes com *confiabilitat*, *reproductibilitat* i *repetibilitat*, entre altres.

L'estudiant podrà ampliar els conceptes que ja ha adquirit en altres assignatures o mòduls fent una revisió del capítol 3, en el qual es descriuen els elements de la infraestructura.

2. Definint-ho tot com a codi

En la lectura del capítol 4, l'estudiant veurà quins són els llenguatges i entorns més utilitzats per a la gestió de la infraestructura (Puppet, Chef, Ansible i Saltstack, entre altres) per a servidors virtualitzats i *cloud* IaaS o eines i entorns anomenats *stack-oriented tools* com Terraform i CloudFormation, que han significat una simplificació en IaC. No menys importants són les aportacions de noves tendències que utilitzen llenguatges de propòsits generals per definir la infraestructura, com Pulumi i AWS CDK (*Cloud Development Kit*), que suporten Typescript, Python i Java, com a aposta per solucionar les limitacions dels llenguatges declaratius utilitzats per altres eines.

L'estudiant podrà complementar aquests conceptes amb els capítols 5, 6 i 7, que li permetran tenir una visió completa dels aspectes més importants d'IaC:

- com es desenvolupa un *stack* com a codi,
- com es construeix un entorn amb *stacks* i
- com es configuren aquests *stacks*.

3. Casos d'ús: Ansible, Vagrant i Terraform

Per assentar els aspectes principals d'IAC, es realitzaran tres proves de concepte centrades en les tres categories principals d'una de les classificacions més utilitzades:

1) Eines de gestió de configuracions: també conegudes com a *configuració com a codi*, són eines especialitzades dissenyades per gestionar programari. Normalment, se centren a instal·lar i configurar servidors.

Exemples d'eines de gestió de configuracions

Cheff, Puppet i **Ansible**.

2) Contenidors i eines de plantilles (*templating tools*): aquestes eines generen plantilles o imatges precarregades amb totes les biblioteques i els components necessaris per executar una aplicació. Les càrregues de treball en contenidors són fàcils de distribuir i tenen una despesa general molt inferior a la que s'executa en un servidor.

Exemples de contenidors i eines de plantilles

Docker, rkt, **Vagrant** i Packer.

3) Eines de provisió i desplegament: les eines de provisió se centren en la creació d'infraestructures. Mitjançant aquesta mena d'eines, els desenvolupadors poden definir components exactes de la infraestructura.

Exemples d'eines de provisió i desplegament

Són exemples **Terraform**, AWS CloudFormation i OpenStack Heat.

3.1. Ansible

És una plataforma de codi obert (ara propietat de Red Hat) per configurar i administrar molt fàcilment sistemes IT, desplegar aplicacions i programari en els elements d'infraestructura i amb una particularitat molt atractiva: només requereix SSH per fer el desplegament.

Aquesta eina permet el programari *provisioning*, gestió de configuracions i desplegament d'aplicacions en el que es coneix com a *infraestructura com a codi*, i inclou el seu propi llenguatge declaratiu per descriure la configuració del sistema. Ansible va ser escrit per M. DeHaan (desenvolupador de Cobbler –*Linux provisioning server*–) i adquirit per Red Hat el 2015.

La diferència d'Ansible i la seva popularitat sobre altres eines similars (com Chef o Puppet), que necessiten un agent en el *host* remot, per la qual cosa els servidors on serà desplegat l'*stack* han de condicionar-se abans per a aquest fi, es deu al fet que Ansible només necessita una connexió SSH i Python (2.4 o posterior) per dur a terme una acció en els nodes que han de configurar-se, i en general tots els servidors ja disposen d'aquest programari en la seva instal·lació bàsica.

Ansible integra mòduls que treballen en format JSON i utilitza YAML per descriure configuracions reutilitzables, per la qual cosa permet que els desenvolupadors descriguin els patrons per aprovisionar el programari.

Ansible té una arquitectura que distingeix entre el controlador i els nodes, en què el primer inicia l'orquestració i és qui gestiona per SSH els nodes que coneix per mitjà d'un inventari.

Ansible insereix mòduls en els nodes (mitjançant SSH) que es desen temporalment i es comuniquen amb el controlador mitjançant el protocol JSON sobre una sortida estàndard.

El seu disseny està basat en una arquitectura minimalista (sense imposar dependències addicionals), consistent, segura i d'alta confiabilitat. Atès que cada mòdul es pot escriure en qualsevol llenguatge estàndard (Python, Perl, Ruby, Bash, etc.), permet a qualsevol programador o responsable d'aprovisionament estar operatiu ràpidament, fins i tot per a infraestructures complexes.

Entre les definicions d'aquest entorn es troben les següents:

- **Node de control**: qualsevol màquina amb Ansible instal·lat que pot executar ordres i *playbooks* cridant `ansible` o `ansible-playbook`, que actua com a node de control (és important tenir en compte que no es pot utilitzar una màquina Windows com a node de control). Es poden tenir diversos nodes de control.

- **Nodes gestionats**: els dispositius de xarxa (o servidors) que es gestionen per Ansible. Els nodes gestionats també es diuen *amfitrions*. Ansible no està instal·lat en els nodes gestionats.

- **Inventari**: una llista de nodes gestionats. Un arxiu d'inventari també es denomina a vegades *arxiu host*. L'inventari pot especificar informació com l'adreça IP per a cada node gestionat o pot organitzar nodes gestionats, creant i imbricant grups per facilitar l'escalat.

- **Col·leccions**: les col·leccions són un format de distribució per al contingut d'Ansible que pot incloure *playbooks*, rols, mòduls i connectors. Es poden instal·lar i utilitzar col·leccions mitjançant l'entorn Ansible Galaxy.

- **Mòduls**: unitats de codi que Ansible pot executar per a un ús particular, des de l'administració d'usuaris en un tipus específic de base de dades fins a la gestió d'interfícies VLAN en un tipus específic de dispositiu de xarxa. Pot cridar un únic mòdul amb una tasca o cridar diversos mòduls diferents en un *playbook*. A partir d'Ansible 2.10, els mòduls s'agrupen en col·leccions. En l'enllaç recomanat es pot veure una llista de tots aquests.

Enllaç recomanat

Podeu aprofundir en la sintaxi dels *playbooks* en l'enllaç següent: https://bit.ly/3JY7sXK.

- **Tasques**: les unitats d'acció a Ansible. Pot executar una sola tasca una vegada amb una ordre *ad hoc*.

- ***Playbooks***: llistes ordenades de tasques, desades perquè pugui executar aquestes tasques en aquest ordre repetidament. Els *playbooks* poden incloure variables i tasques, estan escrits en YAML i són fàcils de llegir, escriure, compartir i entendre.

La instal·lació d'Ansible es pot fer des dels repositoris de les distribucions (es troba disponible en la majoria d'aquestes) o també per mitjà de l'ordre `pip` per a la versió de Python corresponent.

Enllaç recomanat

Podeu consultar més informació en la Installation Guide.

Un cop instal·lat, s'han de generar les claus PKI perquè SSH es pugui connectar als nodes gestionats per clau pública i privada i fer l'inventari, com per exemple:

1) Generar claus publicoprivades i copiar-les en els nodes gestionats mitjançant `ssh-keygen`; per copiar-les, `ssh-copy-id root@ip_node_gestionat` fent el mateix per a tots els nodes. Cal verificar la connexió i recordar que en els clients s'ha d'habilitar en */etc/ssh/sshd_config* que l'usuari *root* es pugui connectar amb **PermitRootLogin yes**.

2) Generar l'inventari editant l'arxiu */etc/ansible/hosts* agregant tots els *hosts*/clients (creant etiquetes o utilitzant les que ja s'han indicat, per exemple, en aquest cas *webservers*).

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

3) També es pot agregar un directori */etc/ansible/var_hosts* amb les definicions i configuracions del *host*, per exemple, **mvb**:

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

4) Amb això, ja es pot comprovar fent:

```
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'
```

Per generar el *passwd* per a l'usuari *testing* de l'última ordre, es podrà instal·lar el paquet *whois* i executar `mkpasswd –method=SHA-512` o també generar-lo amb `openssl`.

Com es pot observar, és molt simple de posar en marxa i molt potent per desplegar la infraestructura programari, ja que no presenta particularitats ni complexitats d'instal·lació/configuració com altres paquets (per exemple, Puppet, Capistrano, Salt o Chef).

Generar un *playbook* resulta molt fàcil, ja que són en text net (en format YAML) i es permet organitzar tasques complexes mitjançant ítems amb format **key: values**.

Dins d'un *playbook* podem trobar un o més grups de ***hosts*** (cadascun es denomina *play*) on es faran les tasques. Per exemple, per crear-ne un fem servir: `mkdir /etc/ansible/playbooks`; i dins d'aquest directori es crea un arxiu amb `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
```

Després es genera l'arxiu `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>
```

Executant `ansible-playbook /etc/ansible/playbooks/apache.yml`, posteriorment, en un navegador, es podrà veure el resultat del desplegament en totes les màquines declarades en l'inventari.

Enllaços recomanats

Haureu de complementar els vostres coneixements analitzant i executant altres *playbooks* des del conjunt d'exemples proposats per Ansible i prestar atenció a la seva sintaxi.

3.2. Vagrant

Aquesta eina és útil en entorns IT en els quals es necessiti desenvolupament continu i exerceix un paper diferent del d'altres eines (per exemple, Docker), però es troba orientada cap als mateixos objectius: proporcionar entorns fàcils de configurar, reproduïbles i portàtils amb un únic flux de treball, que ajudarà a maximitzar la productivitat i la flexibilitat en el desenvolupament d'aplicacions/serveis.

Les màquines es poden obtenir de diferents proveïdors (VirtualBox, Vmware, AWS o altres) utilitzant *scripts*, Chef o Puppet per instal·lar i configurar automàticament el programari de les màquines virtuals.

Per als desenvolupadors, Vagrant aïllarà les dependències i configuracions dins d'un únic entorn disponible, consistent, sense sacrificar cap de les eines que el desenvolupador utilitza habitualment i tenint en compte un arxiu anomenat *Vagrantfile*. La resta dels desenvolupadors tindran el mateix entorn, encara que treballin des d'uns altres SO (poden ser Linux, Windows o MacOS) o entorns, per la qual cosa s'aconseguirà que tots els membres d'un equip estiguin executant codi en el mateix entorn i amb les mateixes dependències.

A més, en l'àmbit IT permet tenir entorns d'un sol ús amb un flux de treball coherent per desenvolupar i provar *scripts* d'administració de la infraestructura, ja que ràpidament es poden fer proves en un entorn de virtualització local, com VirtualBox o Vmware. Després, amb la mateixa configuració, pot provar aquests *scripts* en el *cloud* (per exemple, AWS o Rackspace) utilitzant el mateix flux de treball.

La seva instal·lació pot ser dels repositoris del sistema operatiu o simplement s'ha de descarregar i instal·lar el paquet des del desenvolupador. La posada en marxa és summament fàcil i es disposa de guies per posar en marxa ràpidament un *Box* d'entre una gran quantitat disponible, i a punt per funcionar.

Es poden fer fàcilment proves sobre qualsevol sistema operatiu dels habituals, ja que solament s'ha de tenir instal·lat VirtualBox. Per carregar un *box* es pot fer el següent:

- `vagrant init centos/7`: inicialitzarà/generarà un arxiu **Vagrantfile** amb les definicions de la VM.

- `vagrant up`: descarregarà del repositori del Vagrant una imatge de Centos/7 i la posarà en marxa.

- `vagrant ssh`: per accedir a la MV.

- `vagrant destroy`: per eliminar la MV.

Es poden veure les imatges preconfigurades i precarregar-les des del *cloud* simplement fent `vagrant box add nom` (per exemple, podem usar `vagrant box add ubuntu/jammy64`). Quan s'executi l'`up` s'observaran en la consola una sèrie de missatges entre els quals s'indicarà el port per connectar-se a la MV (per exemple, `ssh vagrant@localhost -p 2222` i la contrasenya vagrant). També es pot fer simplement `vagrant ssh`.

Si es vol canviar, per exemple, la versió o configuració de la MV, es pot modificar l'arxiu *Vagrantfile* i canviar el contingut:

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

Dins del Vagrantfile es poden agregar configuracions específiques, com, per exemple, per instal·lar programari:

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

O per agregar un port:

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

Enllaços recomanats

Haureu d'analitzar la diferent sintaxi que es pot utilitzar en Vagrantfile i altres aspectes interessants d'aquesta eina com són l'aprovisionament, red, *folders*, *cloud-init* i *plugins*, entre altres.

3.3. Terraform

La premissa d'aquesta eina és *«describe your complete infrastructure as code and build resources across providers»*. És una eina desenvolupada per Hashicorp per a IaC tenint al cap la filosofia de *write, plan* i *apply*.

És una eina *open source* per desenvolupar IaC mitjançant una línia d'ordres (CLI) i gestionar centenars de serveis *cloud*. Terraform codifica les API dels *cloud* en arxius de configuració declaratius que permeten una descripció fàcil i simple de la infraestructura que es vol desplegar, al mateix temps que permet construir, canviar i gestionar la infraestructura d'una manera segura i repetible.

Els operadors i equips d'infraestructures poden utilitzar Terraform per gestionar entorns amb un llenguatge de configuració anomenat *HashiCorp Configuration Language* (HCL) per a desplegaments automatitzats i llegibles. Com ja s'ha explicat, la IaC com a concepte és el procés de gestió de la infraestructura en un arxiu o més en lloc de configurar manualment els recursos en una interfície d'usuari; és un recurs de qualsevol infraestructura d'un determinat entorn, com una màquina virtual, un grup de seguretat, una interfície de xarxa, etc.

Amb un alt nivell d'abstracció, Terraform permet als operadors utilitzar HCL per crear arxius que continguin definicions dels recursos desitjats en gairebé qualsevol proveïdor (AWS, GCP, GitHub, Docker, etc.), automatitza la creació d'aquests recursos en el moment de fer la sol·licitud i treballa bàsicament amb cinc ordres principals:

1) *init* (prepara l'entorn)

2) *plan* (mostra els canvis que requereix la configuració)

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

4) *destroy* (elimina la infraestructura)

5) *validate* (verifica la sintaxi)

Els passos que s'han de seguir (*workflow*) seran els següents:

1) Abast: confirmar quins recursos és necessari crear per a un projecte determinat.

2) Autor: crear l'arxiu de configuració en HCL basat en els paràmetres d'abast.

3) *Init*: executar `terraform init` en el directori del projecte amb els arxius de configuració. Això descarregarà els connectors de proveïdor correctes per al projecte.

4) ***Plan & Apply***: s'executa `terraform plan` per verificar el procés de creació i, a continuació, `terraform apply` per crear recursos reals, així com un arxiu d'estat que compara els canvis futurs dels arxius de configuració amb el que hi ha realment en el seu entorn de desplegament.

Entre els avantatges principals que es poden esmentar és que és agnòstic en relació amb la plataforma/*cloud*, gestiona de manera simple l'estat de la infraestructura (*state management*), millora la confiança de l'operador (*operator confidence*) i està dissenyat per escalar i simplificar les millors pràctiques d'ús de manera transparent.

En un CPD modern hi poden haver diferents entorns per donar suport a les diverses aplicacions, però això no afecta Terraform, ja que es pot gestionar un entorn heterogeni amb el mateix flux de treball creant un arxiu de configuració.

Terraform crea un arxiu d'estat quan s'inicia un projecte i s'utilitza aquest estat local per crear plans i fer canvis en la infraestructura; abans de qualsevol operació, es fa una actualització de l'estat amb la infraestructura real. Si es fa un canvi o s'afegeix un recurs a una configuració, es comparen aquests canvis amb l'arxiu d'estat per determinar quins canvis resulten en un nou recurs o modificacions del recurs. El flux de treball integrat té com a objectiu infondre confiança als usuaris promovent operacions fàcilment repetibles i una fase de planificació per permetre als usuaris assegurar-se que les accions no causin interrupcions en el seu entorn.

Veurem ara una prova molt simple per instal·lar Terraform i desplegar un recurs (en aquest exemple s'utilitzarà Docker com a proveïdor, però podria ser qualsevol altre). La instal·lació es pot fer directament del repositori fent *unzip* i movent-lo al directori adequat (per exemple, */usr/bin*) o també es pot instal·lar del repositori de la distribució (per exemple, amb `apt update`; `apt install terraform`). Per desplegar un contenidor amb Nginx, s'ha de 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
    }
}
```

Després s'haurà d'executar `terraform init` i després `terraform apply` i, finalment, ja es podrà connectar al port 8000 del navegador per veure la pàgina inicial de Nginx. Es poden mirar els contenidors amb `docker pg`, i l'estat amb `terraform show`. Per acabar, es pot executar l'ordre `terraform destroy`.

Enllaços recomanats

Haureu de complementar el vostre estudi amb els diferents apartats dels diferents aspectes de Terraform CLI, Modules i Provision.

4. Monitoratge i seguretat

Dos aspectes rellevants per a una infraestructura són el monitoratge i la seguretat. Sobre aquests conceptes, en aquest apartat haureu de seguir el material publicat a Administració avançada del sistema operatiu GNU/Linux, en els mòduls «Sintonització, optimització i alta disponibilitat» i «Administració de seguretat» respectivament.

Per al **monitoratge**, haureu de centrar-vos en la introducció i, a continuació, en els aspectes bàsics de sintonització, optimització i alta disponibilitat per analitzar aquests conceptes i adquirir els coneixements necessaris sobre els principals aspectes del tema. Posteriorment, haureu d'analitzar i executar proves de concepte i configuracions de diferents eines, com Nagios, Ganglia, Munin, Monit, SNMP + MRTG i Cacti, entre altres.

Per al monitoratge de contenidors, haureu d'analitzar cAdvisor (*contenidor advisor*), que proporciona informació sobre l'ús de recursos i les característiques de rendiment dels seus contenidors en execució. Està basat en un *daemon* que recopila, agrega, processa i exporta informació sobre contenidors en execució mantenint per a cada contenidor els paràmetres d'aïllament de recursos, ús històric de recursos, histogrames d'ús històric complet de recursos i estadístiques de xarxa exportant aquestes dades per contenidor i en tota la màquina.

cAdvisor té suport natiu per a contenidors Docker i hauria d'admetre gairebé qualsevol altre tipus de contenidor a punt per usar que es basi en lmctfy, permetent aquesta abstracció que els contenidors estiguin intrínsecament imbricats de manera jeràrquica.

Com a complement del monitoratge i gestió de contenidors, haureu d'analitzar Portainer.io (en la seva versió *Community*). Portainer és una interfície d'usuari d'administració (lleugera) que li permet administrar fàcilment diferents entorns Docker (*hosts* Docker o clústers *Swarm*) i està dissenyada per ser tan simple d'implementar com d'usar. Consisteix en un únic contenidor que pot executar-se en qualsevol motor Docker (es pot implementar com a contenidor de Linux o contenidor natiu de Windows, i també és compatible amb altres plataformes) i així administrar i monitorar tots els recursos Docker (contenidors, imatges, volums i xarxes, entre altres). La seva arquitectura es basa en dos elements (Portainer Server i Portainer Agent), els quals s'executen com a contenidors en una infraestructura en contenidors existents. L'agent de Portainer ha d'implementar-se en cada node del clúster que es farà el monitoratge i aquest informarà Portainer Server de les evolucions de contenidors sota el seu àmbit per centralitzar les mesures i l'ús de recursos. La seva instal·lació és summament simple, ja que només s'ha d'instal·lar un contenidor *standalone*

Enllaç recomanat

Per instal·lar un contenidor *standalone* sobre Linux, podeu veure l'enllaç següent: https://docs.portainer.io/v/ce-2.9/start/install/server/docker/linux.

sobre Linux (podeu veure altres exemples en la mateixa documentació). Es pot veure una demo del Portainer en funcionament a http://demo.portainer.io/ (cal utilitzar *username* = admin i *password* = tryportainer).

Per als aspectes de **seguretat**, l'estudiant s'haurà de concentrar en el mòdul 4, en el qual s'analitzen, en primer lloc, els conceptes principals (tipus i mètodes dels atacs, tècniques utilitzades en els atacs, contramesures i seguretat del sistema), per després centrar-se en la seguretat local (*bootloaders*, contrasenyes i *shadows*, *bits sticky* i *setuid*, mòduls PAM, alteracions del sistema, limitació de recursos per *cgroups* i *chroot*, protecció d'executables mitjançant *Hardening-Wrapper*, *SELinux*). Finalment, l'estudiant haurà d'analitzar les eines de seguretat més comunes per a la detecció de vulnerabilitats i intrusions (OpenVAS, Denyhosts, Fail2ban, Snort, Suricata), protecció mitjançant filtratge (TCP *wrappers* i tallafocs -iptables/nftables-) així com l'anàlisi de registres.

Bibliografia

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* (2a. ed.). O'Reilly Media.

Enllaços (data de l'últim accés: desembre 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