Gestió d'Infraestructures per al Processament de Dades

Infrastructure as Code

Remo Suppi.
Departament Arquitectura d'Ordinadors i Sistemes Operatius
UAB (Remo.Suppi@uab.cat)

Què veurem?

Com es defineix la infraestructura?

Quines opcions tinc?

Vagrant

Ansible

Terraform

Infrastructure as Code

Infraestructura com a codi (IaC) és una combinació d'estàndards, pràctiques, eines i processos per subministrar, configurar i gestionar la infraestructura informàtica mitjançant codi i altres fitxers llegibles per una màquina.

Per què? Durant molt de temps, la intervenció manual va ser l'única manera de gestionar la infraestructura informàtica. Els servidors s'havien de muntar en bastidors, s'havien d'instal·lar sistemes operatius i connectar i configurar les xarxes. En aquell moment, això no era un problema, ja que els cicles de desenvolupament eren tan llargs que els canvis en la infraestructura eren poc freqüents.

Problema: Les tasques fetes manualment provocaven un veritable desastre quan es treballava en equip, perquè ningú sabia com era la infraestructura gestionada, com es configuraven les màquines en funcionament i quins canvis es van fer. I quan es produïa un error al sistema, ningú podia dir quins canvis podrien haver fet caure el sistema. El treball de l'administrador del sistema en aquest cas es va convertir en una professió de risc (era el gurú del sistema o la pedra a la sabata).

Amb **l'evolució de les tecnologies com la virtualització i el núvol, combinades amb l'augment de DevOps i les pràctiques àgils,** van reduir dràsticament els cicles de desenvolupament de programari. Com a resultat, hi havia una demanda de **millors tècniques de gestió d'infraestructures**. Les organitzacions ja no podien permetre's el luxe d'esperar hores o dies perquè es despleguessin servidors.

La **infraestructura com a codi** és una manera d'elevar l'estàndard de la gestió de la infraestructura i el temps de desplegament. Mitjançant l'ús d'una combinació d'eines, llenguatges, protocols i processos, IaC pot crear i configurar elements d'infraestructura de manera segura en qüestió de segons.

Infrastructure as Code

En resum: IaC és el procés de gestió i subministrament dels recursos del centre de dades informàtics mitjançant fitxers de definició llegibles per màquina, en lloc de la configuració del maquinari físic o les eines de configuració interactives.

La infraestructura informàtica gestionada per aquest procés comprèn tant equips físics, com servidors nus, com màquines virtuals, i recursos de configuració associats. El codi dels fitxers de definició pot utilitzar scripts o definicions declaratives, en lloc de mantenir el codi mitjançant processos manuals, però IaC utilitza més sovint enfocaments declaratius.

On va sortir: 2006, el llançament de l'Elastic Compute Cloud d'Amazon Web Services i la versió 1.0 de Ruby on Rails van crear problemes d'escala generalitzats a l'empresa que abans només s'experimentaven a grans empreses multinacionals. Amb l'aparició de noves eines per gestionar aquest camp va néixer la idea d'IaC. La idea de modelar la infraestructura amb codi i, després, tenir la capacitat de dissenyar, implementar i desplegar la infraestructura d'aplicacions amb les millors pràctiques de programari conegudes va atreure tant als desenvolupadors de programari com als administradors d'infraestructures de TI.

Avantatges: cost, velocitat i risc.

Tipus d'enfocaments: declaratiu (funcional) i imperatiu (procedimental). La diferència entre l'enfocament declaratiu i imperatiu és essencialment "què" versus "com". L'enfocament declaratiu se centra en quina hauria de ser l'eventual configuració de destinació; l'imperatiu se centra en com s'ha de canviar la infraestructura per fer-ho.

Per exemple: `kubectl run nginx  --generator=run-pod/v1 -image=nginx` (imperatiu) i `kubectl apply -f app.yaml` (declaratiu)

L'enfocament declaratiu defineix l'estat desitjat i el sistema executa el que ha de passar per aconseguir aquest estat desitjat. L'imperatiu defineix ordres específiques que s'han d'executar en l'ordre adequat per acabar amb la conclusió desitjada.

Infrastructure as Code

Mètodes: Hi ha dos mètodes d'IaC: push i pull. La principal diferència és la manera en què se'ls indica als servidors com s'han de configurar. En el mètode push, el servidor que s'ha de configurar extreu la seva configuració del servidor de control. En el mètode push, el servidor de control envia la configuració al sistema de destinació.

Eines: Hi ha moltes eines que compleixen les capacitats d'automatització de la infraestructura i utilitzen IaC. A grans trets, qualsevol marc o eina que realitza canvis o configura la infraestructura de manera declarativa o imperativa basant-se en un enfocament programàtic es pot considerar IaC. Tradicionalment, s'utilitzaven eines d'automatització i gestió de configuració del servidor (cicle de vida) per aconseguir IaC però ara les empreses també utilitzen eines d'automatització de configuració contínua o marcs d'IaC autònoms, com ara PowerShell DSC de Microsoft o AWS CloudFormation.

Automatització de configuració contínua: Totes les eines d'aquest tipus (*continuous configuration automation* CCA) es poden considerar una extensió de l'IaC tradicional. Aprofiten IaC per canviar, configurar i automatitzar la infraestructura, i també proporcionen visibilitat, eficiència i flexibilitat en com es gestiona la infraestructura.

Tool	Released by	Method	Approach	Written in	Comments
Chef	Chef (2009)	Pull	Declarative and imperative	Ruby	-
Otter	Inedo (2015)	Push	Declarative and imperative	-	Windows-oriented
Puppet	Puppet (2005)	Push and Pull	Declarative and imperative	C++ & Clojure since 4.0, Ruby	-
SaltStack	SaltStack (2011)	Push and Pull	Declarative and imperative	Python	-
CFEngine	Northern.tech	Pull	Declarative	C	-
Terraform	HashiCorp (2014)	Push	Declarative and imperative	Go	-
Ansible / **Ansible Tower**	Red Hat (2012)	Push	Declarative and imperative	Python	-

Infrastructure as Code

Avantatges de la infraestructura com a codi: Una organització que utilitza IaC té aquests avantatges:

Velocitat: evitar la intervenció manual, els desplegaments d'infraestructures són ràpids i segurs.

Control font: es pot comprovar el codi abans per obtenir més transparència i rendició de comptes.

Documentació: el codi d'infraestructura serveix com a documentació viva de l'estat real de la infraestructura.

Coherència: es desplega una infraestructura única per a tot el projecte i que tothom està d'acord, evitant configuracions puntuals, de proves/desenvolupament/test/....

Agilitat: DevOps ha fet que el lliurament de programari sigui més eficient i IaC aporta agilitat a l'àmbit de la gestió d'infraestructures.

Reutilització: IaC facilita la creació de mòduls reutilitzables; per exemple, per replicar entorns de desenvolupament i producció.

Generalització: Tenint en compte aquestes idees la gent es va adonar que la majoria de les tasques de configuració i gestió de la infraestructura són realment molt habituals i ben definides: iniciar una màquina virtual amb característiques específiques, crear una regla de tallafoc, instal·lar paquets del sistema, copiar fitxers, iniciar un servei, etc.

I van dir: *"Per què no escrivim mòduls/funcions que realitzen aquestes tasques habituals per als sistemes amb els quals treballem? Els podríem provar, garantir que funcionessin a través dels nostres sistemes i que fossin* ***idempotents****"* (fet que significa que si executem un script diverses vegades, obtindrem els mateixos resultats).

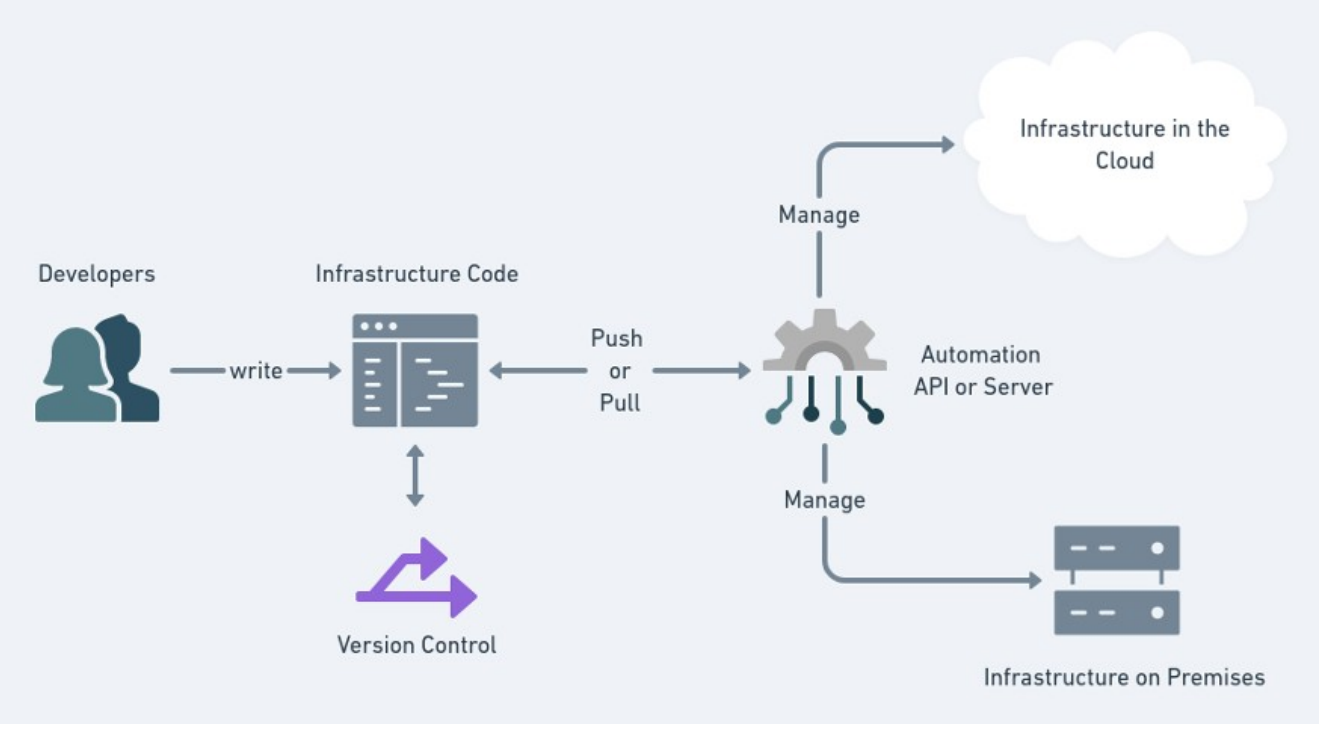
Resultat: en lloc d'escriure la nostra pròpia implementació dels scripts cada vegada que necessitem fer alguna cosa, podem fer servir aquests mòduls/funcions (abstraccions de nivell superior basades en els scripts) que realment funcionen.

Infrastructure as Code

Com funciona la infraestructura com a codi?

IaC es pot explicar des d'alt nivell en tres passos i visualitzar-la amb el diagrama següent.

1. Els desenvolupadors escriuen l'especificació d'infraestructura en un llenguatge específic.
2. Els fitxers resultants s'envien a un servidor master, a una API de gestió o a un dipòsit de codi.
3. La plataforma fa tots els passos necessaris per crear i configurar els recursos informàtics.



Infrastructure as Code

Per exemple, un script:

```
#!/bin/bash
sudo apt-get install ruby-full build-essential
```

A aquest playbook d'Ansible, ens proporciona una abstracció (idempotent) per fer la mateixa tasca:

```
- name: Configure Raddit App Instance
  hosts: all
  become: true
  tasks:
    - name: Install Ruby
      apt: "name={{ item }} state=present"
      with_items:
        - ruby-full
        - build-essential
```

Per què es necessita? L'enfocament de la **IaC** posa ordre en la tasca d'administració del sistema.

Ens permet controlar l'estat actual dels nostres sistemes gestionats. Com que ens assegurem que la configuració és idempotent, podem gestionar tot el cicle de vida del sistema gestionat mitjançant els mateixos scripts.

Si tornem a executar el *playbook*, podem confiar que si el sistema ja té els paquets adequats, no s'instal·larà res o si falten alguns paquets (per exemple, algun desenvolupador els va eliminar manualment) s'instal·laran. Gestionar tot el cicle de vida del sistema mitjançant els mateixos scripts significa que el codi de configuració del repositori de control d'origen reflecteix l'estat actual del sistema gestionat.

Ja hem vist desplegament similars? **Si, Docker-Compose.** Aquest és un exemple de com desplegar IaC per contenidors.

Infrastructure as Code

Tipus d'infraestructures com a codi: Hi ha quatre tipus principals de IaC

Scripting: escriure scripts és l'enfocament més directe de IaC. Els scripts ad-hoc són els millors per executar tasques senzilles, curtes o puntuals. No obstant això, per a configuracions complexes, és millor utilitzar una alternativa més especialitzada.

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. Alguns exemples d'aquestes eines són Cheff, Puppet i **Ansible**.

Eines de subministrament: les eines de subministrament se centren en la creació d'infraestructures. Mitjançant aquest tipus d'eines, els desenvolupadors poden definir components exactes de la infraestructura. En són exemples **Terraform**, AWS CloudFormation, OpenStack Heat, Pulumi.

Contenidors i eines de plantilles (*templating tools*): aquestes eines generen plantilles o imatges precarregades amb totes les biblioteques i 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. En són exemples **Docker** , rkt, **Vagrant** i Packer.

Comparació Eines:

Eina	Millor ús	Tipus de configuració	Proveïdor Cloud	Dificultat
Terraform	Multi-cloud	Declarativa (HCL)	Multi-cloud	Mitjana
AWS CloudFormation	AWS	Declarativa (YAML/JSON)	AWS exclusiu	Baixa
Ansible	Configuració de servidors	Procedimental (YAML)	Multi-cloud/on-prem	Baixa
Kubernetes	Orquestració de contenidors	Declarativa (YAML)	Multi-cloud/on-prem	Alta
Pulumi	Desplegament modern	Llenguatges de programació	Multi-cloud	Mitjana
Docker Compose	Entorns multi-contenidor	Declarativa (YAML)	Local/on-prem	Baixa

Comparació Eines:

Característica	Vagrant	Terraform	Ansible	Docker Compose
Objectiu principal	Entorns locals de desenvolupament	Multi-cloud i infraestructures	Configuració de sistemes	Contenidors locals
Provisionament	Manual o integrat amb Ansible	Declaratiu	Procedimental	Declaratiu
Multiplataforma	VirtualBox, VMware, Hyper-V	AWS, Azure, GCP	Multi-cloud i on-prem	Local
Facilitat d'ús	Alta (fitxers senzills)	Mitjana	Baixa	Alta
Escalabilitat	Limitada a màquines locals	Alta (infraestructura global)	Alta	Baixa
Usos destacats	Simulació de servidors locals	Desplegaments en producció	Gestió de configuracions	Desenvolupament amb contenidors

Comparació Eines: AWS CloudFormation

Descripció: Eina específica d'AWS per gestionar recursos mitjançant plantilles JSON o YAML.

Exemple d'ús:

Plantilla per crear un bucket S3:

```
Resources:
  MyBucket:
    Type: AWS::S3::Bucket
    Properties:
      BucketName: my-example-bucket
```

Funció: Defineix i desplega un bucket S3 amb el nom especificat.

Avantatges: Integració profunda amb AWS, Capacitats avançades per crear "stacks" d'infraestructura complexa.

Comparació Eines: Docker Compose

Descripció: Eina per definir i executar aplicacions multi-contenidor amb fitxers YAML.

Exemple d'ús: Aplicació amb un servidor web NGINX i una base de dades MySQL:

```
version: "2.1"
services:
  web:
    image: nginx
    ports:
      - "8080:80"
  db:
    image: mysql
    environment:
      MYSQL_ROOT_PASSWORD: example
```

Funció: Configura dos contenidors: un amb NGINX i un altre amb MySQL, connectats a la mateixa xarxa.

Avantatges: Fàcil configuració d’entorns de desenvolupament locals, compatible amb Kubernetes per producció.

Comparació Eines: Kubernetes

Descripció: Plataforma d'orquestració per gestionar contenidors en producció, com Docker. Gestiona la disponibilitat, escalabilitat i desplegaments d'aplicacions.

Exemple d'ús: Fitxer YAML per desplegar una aplicació:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: example-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: example
  template:
    metadata:
      labels:
        app: example
    spec:
      containers:
      - name: example-container
        image: nginx:1.21
        ports:
        - containerPort: 80
```

Funció: Crea un desplegament amb 3 rèpliques de l'aplicació basades en la imatge del servidor web NGINX.

Avantatges: facilita l'escalabilitat, monitoratge natiu amb Kubernetes Metrics Server.

Comparació Eines: Pulumi

Descripció: Alternativa moderna a Terraform, que permet utilitzar llenguatges de programació com Python, TypeScript o Go. Ideal per equips de desenvolupament amb experiència en aquests llenguatges.

Exemple d'ús amb Python: Crear una instància EC2 a AWS:

```
import pulumi
import pulumi_aws as aws
instance = aws.ec2.Instance("my-instance",
    ami="ami-12345678",
    instance_type="t2.micro",
    tags={
        "Name": "example-instance",
    })
```

Funció: Crea una màquina virtual EC2 amb AWS utilitzant Python.

Avantatges: Familiaritat amb llenguatges coneguts, Integració amb pipelines CI/CD.

Ansible (avui Red Hat)

Plataforma codi obert per configurar i administrar molt fàcilment sistemes IT, i desplegar aplicacions i programari en els elements de infraestructura. **I tot això utilitzant només SSH.**

Permet el *software provisioning*, gestió de configuracions i desplegament d'aplicacions en el que es coneix infraestructura com a codi. 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.

Moltes eines (com Chef o Puppet) necessiten un agent al host remot. En canvi, Ansible només necessita una connexió SSH i Python (2.4 o posterior) per dur a terme una acció en els nodes que s'han de configurar.

Ansible integra mòduls que treballen sobre **format JSON** i utilitza **YAML** per descriure configuracions reutilitzables. Els desenvolupadors són els mateixos que els del programari d'aprovisionament.

Té una arquitectura que distingeix entre el **controlador** i els **nodes**.

En el controlador s'inicia l'orquestració i és ell qui gestiona per **ssh** els nodes que coneix a través d'un inventari.

Ansible insereix mòduls als 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 fiabilitat. Com que cada mòdul pot ser escrit en qualsevol llenguatge estàndard (Python, Perl, Ruby, Bash, etc.), la corba d'aprenentatge és suau i permet estar operatiu ràpidament, fins i tot per a infraestructures complexes.

Ansible: definicions

Node de control: Qualsevol màquina amb Ansible instal·lat. Pot executar ordres i plabooks invocant l'ordre **ansible** o **ansible-playbook** des de qualsevol node de control (no es pot fer servir una màquina Windows com a node de control) i es poden tenir diversos nodes de control.

Nodes gestionats: Els dispositius de xarxa (i/o servidors) que es gestionen per Ansible. Els nodes gestionats també s'anomenen "amfitrions". Ansible no està instal·lat als nodes gestionats.

Inventari: Una llista de nodes gestionats. Un fitxer d'inventari també s'anomena de vegades "fitxer host". El vostre inventari pot especificar informació com l'adreça IP per a cada node gestionat. Un inventari també pot organitzar nodes gestionats, creant i nidificant 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. Podeu instal·lar i utilitzar col·leccions mitjançant Ansible Galaxy.

Mòduls: unitats de codi que Ansible executa. Cada mòdul té 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. Podeu invocar un únic mòdul amb una tasca o invocar diversos mòduls diferents en un playbook. A partir d'Ansible 2.10, els mòduls s'agrupen en col·leccions. Llista.

Tasques: Les unitats d'acció a Ansible. Podeu executar una sola tasca una vegada amb una ordre ad hoc.

Playbooks: Llistes ordenades de tasques, desades perquè pugueu 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. Sintaxis.

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

Ansible (avui Red Hat)

Instal·lació: **apt-get install ansible** verificar: **ansible --version**

Generar claus SSH: **ssh-keygen -t rsa** i copiar-les al client: **ssh-copy-id root@172.16.1.2** Fer el mateix pera tots els clients i verificar connexió sense passwd (recordar que en el clients */etc/ssh/sshd_config* posar **PermitRootLogin yes**

Generar inventari: **vi /etc/ansible/hosts** i afegim tots els hosts/clients (creat etiquetes o fent servir les que hi ha) en el nostre cas webservers (en Debian no es crea el directori /etc/ansible i s'ha de crear) :

```
[webservers]
b.gixpd.org
c.gixpd.org
```

També es pot afegir un directori que sigui **/etc/ansible/var_hosts** amb les configuracions del host, per exemple **mvb**:

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

Comprovem: **ansible -m ping webservers** **ansible -m ping mvb**

Executem ordres: **ansible -m command -a "df -h" webservers**

Ansible: instal·lació

Altres ordres:

```
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 aquest usuari (*testing)*, es podrà instal·lar el paquet **whois** i executar l’ordre:

mkpasswd –method=SHA-512 (o també amb openssl passwd -6 -stdin)

Com es pot observar, és molt simple de posar en marxa i molt potent; no presenta particularitats/complexitats d’instal·lació/configuració com altre paquets (p.ex. Puppet, Capistrano, Salt o Chef)

Atès que utilitza com a agent SSH (generalment disponible per defecte en tots els clients Linux), el seu desplegament és molt fàcil.

Ansible: playbook

Els **playbooks** són descripcions en text pla (en format YAML) que permeten organitzar tasques complexes mitjançant ítems i parells **key: values**

Dins d'un playbook podem trobar un o més grups de **hosts** (cadascun d'aquests és anomenat play) on es realitzaran determinades **tasques**. Per exemple: mkdir /etc/ansible/playbooks; 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
```

Generem 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>
```

Executem el playbook **ansible-playbook /etc/ansible/playbooks/apache.yml** i mirem la URL del 20.20.20.x

Exemples de playbooks: https://github.com/ansible/ansible-examples
Sintaxis: https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html

Ansible: eines i documentació

Hi ha una gran quantitat de tutorials (https://serversforhackers.com/c/an-ansible-tutorial) que mostren com fer playbooks més complexos i com gestionar diferents recursos. D'aquesta manera, l'administrador estarà en condicions d'implementar els seus propis playbooks d'una manera molt senzilla i amb un desplegament ràpid.

Ansible disposa d'una GUI anomenada Ansible Tower (comercial), basada en web, que permet totes les opcions de la CLI, a banda de característiques especials com ara monitorització dels nodes en temps real, interfície REST API per a Ansible, job scheduling, i d'una interfície gràfica per a la gestió de l'inventari i d'integració amb el cloud(per exemple, EC2, Rackspace, Azure).

Altervatives: **Rundeck** (http://rundeck.org) que permet executar playbooks d'Ansible (https://github.com/Batix/rundeck-ansible-plugin) o **Semaphore** (https://github.com/ansible-semaphore/semaphore) que és una UI per a AnsibleOpen Source

Vagrant

Aquesta eina és útil en entorns IT on es necessiti desenvolupament continu i té un paper diferent del de Docker però orientat 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.**

Pot proveir de màquines de diferents proveïdors (VirtualBox, Vmware, AWS, o altres) utilitzant scripts, Chef o Puppet per a instal·lar i configurar automàticament el programari de la VM.

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 de desenvolupadors tindrà el mateix entorn encara que treballin des d'uns altres SO o entorns, de manera que s'aconseguirà que tots els membres d'un equip estiguin executant codi en el mateix entorn i amb les mateixes dependències.

Amés, en l'àmbit IT permet tenir entorns d'un sol ús amb un flux de treball coherent per a 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) amb el mateix flux de treball.

Vagrant

S'ha de treballar sobre un **màquina/MV que tingui les extensions de virtualització** (i es pot instal·lar sobre Windows/macOS) i sobre aquesta instal·lar **VirtualBox i Vagrant** (p.ex. del repositori d'ubuntu)

Baixar i Instal·lar VirtualBox:

Depenent de la distribució de Linux es pot fer: **sudo apt install virtualbox**

O si no està actualitzat des del web del desenvolupador baixar el paquet i instal·lar-ho:

sudo dpkg -i virtualbox-6.1_xxx_amd64.deb

sudo apt install -f

apt install linux-headers-amd64 linux-headers-`uname -r`

/sbin/vboxconfig

Si esteu sobre una MV Virtualbox u OpenNebula recordeu a d'afegir Neested-Virtualization:

VBoxManage modifyvm vm-name --nested-hw-virt on

sudo apt vagrant (en Ubuntu 24.04 no està en el repositori i s'ha d'instal·lar de la web del desenvolupador)

A partit de aquí ja es pot carregar un **Box** de repositori de Hashicorp (https://portal.cloud.hashicorp.com/vagrant/discover) seleccionat virtualbox, per exemple:

vagrant init centos/7 *inicialitzarà/generarà un arxiu anomenat Vagrantfile amb les definicions de la VM*

vagrant up *descarregarà del repositori cloud de Vagrant una imatge d'Ubuntu 18.04 i l'engegarà.*

vagrant ssh *per a accedir-hi*

vagrant destroy *per eliminar la MV*

Vagrant

Veure les imatges preconfigurades que podrem carregar des del cloud simplement fent **vagrant box add nom**, per exemple **vagrant box add ubuntu/jammy64**

Quan s'executi l'ordre **up** veurem que ens pot donar una sèrie de missatges, entre els quals ens indicarà el port per a connectar-se a aquesta màquina virtual directament (per exemple, ssh vagrant@localhost -p 2222 i amb passwd vagrant) o simplement **vagrant ssh**

Per a utilitzar una VM com a base, podem modificar l'arxiu Vagrantfile i canviar el contingut com:

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

Vagrant

Si desitgem treballar amb dues màquines de manera simultània, haurem de modificar l'arxiu Vagrantfile amb el següent:

```
vagrant.configure("2") do |config|
  config.vm.define "centos" do |centos|
    centos.vm.box = "centos/7"
  end
  config.vm.define "ubu" do |ubu|
    ubu.vm.box = "ubuntu/precise32"
  end
end
```

Podrem veure que assigna un port SSH a cadascuna per a evitar col·lisions quan fem el vagrant up.

Des de la VM podríem instal·lar el programari com es fa habitualment, però per a evitar que cada persona faci el mateix hi ha una forma d'aprovisionament que s'executarà quan es faci el up de la VM.

La forma més fàcil es afegir:

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

I per afegir un port:

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

Vagrant

Per a treballar amb dues MV també es pot fer:

```
Vagrant.configure("2") do |config|
 # Màquina 1
 config.vm.define "web" do |web|
  web.vm.box = "ubuntu/bionic64"
  web.vm.network "private_network", type: "dhcp"
  web.vm.hostname = "web-server"
 end

 # Màquina 2
 config.vm.define "db" do |db|
  db.vm.box = "ubuntu/bionic64"
  db.vm.network "private_network", type: "dhcp"
  db.vm.hostname = "db-server"
 end
end
```

Vagrant et permet configurar diferents tipus de xarxes, incloent xarxes privades, públiques i de trànsit NAT.

```
config.vm.network "private_network", type: "dhcp"  # Xarxa privada amb DHCP
config.vm.network "public_network", bridge: "eth0"  # Xarxa pública amb bridge
```

Pots redirigir ports de la màquina virtual a la teva màquina amfitriona per tal de poder accedir als serveis locals.

```
config.vm.network "forwarded_port", guest: 80, host: 8080  # Redirigeix el port 80 a l'amfitrió al port 8080
```

Es pot compartir directoris entre la màquina host i la màquina virtual per tal de facilitar el treball amb fitxers:

```
config.vm.synced_folder "/path/to/host/folder", "/path/to/guest/folder"
```

Vagrant

Per automatitzar la configuració de la màquina virtual un cop iniciada, es pot utilitzar eines de aprovisionament com scripts shell, Ansible. Fent servir un script shell:

```
config.vm.provision "shell", path: "provision.sh"
```

Fent servir Ansible:

```
config.vm.provision "ansible" do |ansible|
ansible.playbook = "site.yml"
end
```

Una altra opció avançada és usar **Docker dins de Vagrant** per a contenidors. Això et permet utilitzar contenidors en lloc de màquines virtuals completes. Abans s'ha d'instal·lar el plugin de Docker:

```
Vagrant.configure("2") do |config|
 config.vm.network "forwarded_port", guest: 80, host: 8080
 config.vm.define "mydocker" do
  config.vm.provider "docker" do |d|
    d.image = "nginx"
    d.remains_running = true
   end
 end
End
```

Per posar-ho en marxa: **vagrant up –provider=docker**

Per executar una ordre: **vagrant docker-exec -it mydocker -- bash**

Per eliminar-ho: **vagrant destroy**

Vagrant

També es pot fer des de un script extern:

```
vagrant.configure("2") do |config|
  config.vm.define "ubu" do |ubu|
      ubu.vm.box = "ubuntu/bionic32"
      ubu.vm.provision: "shell", path: "init.sh"
  end
end
```

I després fer **vagrant reload --provision**.

Veurem com s'aprovisiona i carrega Apache i i ja ens podrem connectar al port 8080 del localhost

Ordres addicionals:
vagrant destroy
vagrant box list
vagrant box remove ubuntu/bionic64

És una eina molt potent que s'ha d'analitzar amb cura per a configurar les opcions necessàries per al nostre entorn, com per exemple aspectes del vagrant share o qüestions avançades del provisioning que aquí només hem tractat superficialment.

+info: https://learn.hashicorp.com/collections/vagrant/getting-started

Terraform: *«describe your complete infrastructure as code and build resources across providers»*

Eina desenvolupada por Hashicorp per IaC amb una filosofia de ***Write, Plan, Apply***. Es una eina *open-source* per desenvolupar IaC a través de una CLI i gestionar centenes de serveis *cloud*. Terraform codifica les API dels *cloud* en arxius de configuració declaratius que permeten un descripció fàcil i simple de la infraestructura a desplegar.

> Terraform permet construir, canviar i gestionar la infraestructura d'una manera segura i repetible. Els operadors i els 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 fitxer/s en lloc de configurar manualment els recursos en una interfície d'usuari. Un recurs és qualsevol infraestructura d'un entorn determinat, com ara una màquina virtual, un grup de seguretat, una interfície de xarxa, etc.

A un nivell alt, Terraform permet als operadors utilitzar HCL per crear fitxers que continguin definicions dels recursos desitjats en gairebé qualsevol proveïdor (AWS, GCP, GitHub, Docker, etc.) i automatitza la creació d'aquests recursos en el moment de fer la sol·licitud.

Terraform CLI treballa bàsicament amb 5 ordres principals: **init** (prepar l'entorn), **plan** (mostra el canvis requerit per la configuració), **apply** (crea o actualitza la infraestructura), **destroy** (elimina la infraestructura), **validate** (verifica la sintaxis).

Terraform: *Workflows*

Els passos a seguir seran:

- **Abast:** confirmar quins recursos cal crear per a un projecte determinat.
- **Autor:** creeu el fitxer de configuració a HCL basat en els paràmetres d'abast
- **Init:** executar **terraform init** al directori del projecte amb els fitxers de configuració. Això descarregarà els connectors de proveïdor correctes per al projecte.
- **Plan & Apply:** executar **terraform plan** per verificar el procés de creació i, a continuació **terraform apply** per crear recursos reals, així com un fitxer d'estat que compara els canvis futurs dels fitxers de configuració amb el que existeix realment al vostre entorn de desplegament.

Avantatges: Agnòstic en relació a la plataforma/cloud, State Management, Operator Confidence

1. En un CPD modern existeixen diversos entorns diferents per donar suport a les diverses aplicacions però això no afecta a Terraform ja que es pot gestionar un entorn heterogeni amb el mateix flux de treball creant un fitxer de configuració.

2. Terraform crea un fitxer d'estat quan s'inicialitza un projecte i s'utilitza aquest estat local per crear plans i fer canvis a la infraestructura i 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 compara aquests canvis amb el fitxer d'estat per determinar quins canvis resulten en un recurs nou o modificacions del recurs.

3. 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 realitzades no causin interrupcions al seu entorn.

Terraform: Instal·lació, aprovisionament & execució

Es pot fer directament de del repositori baixat un .zip, fent el **unzip** i movent aquest a /usr/bin o instal·lar el repositori i després amb un **apt update; apt install terraform**

Para provar desplegarem sobre Docker per tant en Ubuntu (ON) **apt install docker.io**

Crear **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
  }
}
```

Executar **terraform init** i després **terraform apply** i connectar-se el port 8000 per a veure la pàgina de Nginx. Es poden mirar el contenidors amb **docker ps**, l'estat amb **terraform show** i per terminar **terraform destroy**

Per què utilitzar Terraform amb Docker?

Terraform ens permet **definir i subministrar diversos recursos d'infraestructura de Docker**, com ara imatges, contenidors, volums i xarxes d'una manera declarativa i controlada per versions. Això vol dir que es pot **gestionar fàcilment la infraestructura de Docker com a codi**, de la mateixa manera que es gestiona el codi de l'aplicació, cosa que facilita la col·laboració i l'escalada de la infraestructura en diversos entorns. Això també significa que les configuracions de Docker són més reutilitzables i coherents.

També Terraform facilita **l'escalada automàtica de la infraestructura de Docker creant o suprimint contenidors en funció de les necessitats de càrrega de treball.** Es pot utilitzar Terraform per definir polítiques d'escala, com ara afegir més contenidors quan l'ús de la CPU arriba a un determinat llindar, i crearà o destruirà automàticament els contenidors en conseqüència.

Finalment Terraform ens permet **actualitzar fàcilment la infraestructura de Docker modificant els fitxers de configuració i aplicarà automàticament els canvis als contenidors en execució.** Això vol dir que es pot implementar actualitzacions de la infraestructura de manera ràpida i segura sense cap temps d'inactivitat.

Crear una **app amb python + flask:**

```
from flask import Flask
app = Flask(__name__)
@app.route("/")
def hello():
    return "Hello World"
```

Crear la imatge de Docker amb un Dockerfile:

```
FROM python:3.9
WORKDIR /app
COPY . /app
RUN pip install --upgrade pip
RUN pip install flask
CMD ["flask", "run", "--port", "8082", "—host=0.0.0.0"]
```

Generar l'imatge: **docker build --rm --tag flaskapp:1.0 .**

```
terraform {
  required_providers {
    docker = {
      source  = "kreuzwerker/docker"
      version = "~> 3.0.1"
    }
  }
}
provider "docker" {}
resource "docker_container" "flaskapp" {
  image   = "flaskapp:1.0"
  name    = "flaskapp"
  restart = "always"
  ports {
    internal = 8082
    external = 8082
  }
}
```

Executar:

terraform init

terraform plan

terraform apply

Verificar **docker ps**

Obrir el navegador: **localhost:8082**

Terraform: un exemple més general (AWS)

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 2.70"
    }
  }
}

provider "aws" {
  profile = "default"
  region  = "us-west-2"
}

resource "aws_instance" "example" {
  ami           = "ami-830c94e3"
  instance_type = "t2.micro"
}
```

main block config
provider

link a registry.terraform.io/hashicorp/aws

indica un tipus de recurs: imatge i característiques

Després **terraform init** (inicialització) **terraform fmt** (unifica format) **terraform validate** (verifica la configuració) i finalment **terraform apply** i **terraform show** per a veure l'estat. Per acabar **terraform destroy**

+info: https://www.terraform.io/docs/configuration/syntax.html

Providers: https://www.terraform.io/docs/providers/index.html

Terraform vs Ansible

Terraform	Ansible
Més adequat per aprovisionament d'infraestructura	Automatización d'IT
Excel-lent per a configurar infraestructures cloud	Millor eina per configurar servidors
Permet desplegar Load Balancers, VPC, ...	Permet desplegar apps sobre la infraestructura
Llenguatge Declaratiu	Llenguatge procedural
Definit un estat final, pot desenvolupar tot els passos per arribar	Se necessita indicar cada pas per tenir el resultat final
És considerat ideal per conservar un estat estable	Manté tots els components en execució reparant els errors en lloc de reemplaçar la infraestructura

https://blog.stackpath.com/infrastructure-as-code-explainer/
https://hackernoon.com/infrastructure-as-code-tutorial-e0353b530527
https://github.com/Artemmkin/infrastructure-as-code-tutorial
https://www.vagrantup.com/intro
https://intellipaat.com/blog/terraform-vs-ansible-difference/
https://blog.opennix.ru/posts/use_terraform_with_virtualbox/
https://stackoverflow.com/questions/39211000/experimenting-locally-with-terraform
http://openaccess.uoc.edu/webapps/o2/handle/10609/60685 (Mòdul 6: Introducció a Clúster, Cloud i DevOps.pdf)
https://docs.ansible.com/ansible/latest/user_guide/index.html
https://github.com/lerndevops/labs/tree/master/ansible/playbooks
https://serversforhackers.com/c/an-ansible-tutorial
https://www.vagrantup.com/intro
https://www.middlewareinventory.com/blog/vagrant-ansible-example/
https://intellipaat.com/blog/terraform-vs-ansible-difference/