HowTo-ASIX-VM_Deployment_and_Provisioning

Integrants: Marc Fornés i Rubén Rodríguez.

Índex:

Temes a tractar:

VM Deployment and Provisioning tools * Cloud Managment Tools * Packer * Vagrant * Ansible * Terraform

Concepte Cloud Managment Tools:

Són eines per a que els administradors puguin gestionar productes i serveis del cloud en diferents plataformes Cloud. Permet per tant, el concepte d'orquestració de tots els recusos. Per recursos podem entendre desde accés d'usuaris (permisos), dades (bbdd), aplicacions (soft), serveis (API), màquines virtuals (servers/hosts). Permet tenir tots aquests recursos monitoritzats i poder aplicar de forma ràpida i automatizada l'escalada de recursos (infraestructura) en cas de necessitat produïda per canvis. Normalment es comuniquen amb les API dels proveïdors de CLOUD per controlar y modificar recursos.

Concepte IaC i utilitats i característiques de cada eina IaC:

IaC (Infrastructure as Code):

Són eines/softwares de 'Development Operations' on a la pràctica s'utiltizen fitxers tipus scripts per configurar l'estructura de forma automatitzada, és a dir, quines màquines virtuals/dockers necessitarem segons els recursos (memòria de disc, ampla de banda, RAM, CPU..). A destacar que gràcies a la 'IaC' es molt fàcil escalar recursos (i realitzar canvis) segons les necessitats dels serveis oferts i garanteix aconseguir sempre el mateix entorn/resultat.

Les eines 'DevOps' del tipus IaC tenen 2 estils:

- *Estil procedimental:* Pas a pas (secuencialment), anem especificant en el codi que fer consecutivament i d'on agafar els repositoris/recursos. (cas Ansible)

- *Estil declaratiu:* Vull aquest resultat final, buscat la vida. (cas Terraform)

Ansible:

És principalement un gestor de configuració. Hem de veure Ansible com un instal · lador/configurador de software remot a màquines o servidor ja existents o UP (aixecats). Hem de tenir present que Ansible té repetibilitat, això vol dir que si executem el playbook sempre obtindrem el mateix resultat, de forma que si l'executem 2 vegades, tindrem (P.E) dos instalacions iguals.

PROS: Multipltaforma, open-source, free.

CONTR: És lent i té dependencies de serveis/paquets de 3ers (tema mutabilitat).

Packer by HashiCorp:

Packer es una eina 'Open Source' que utilitza els formats '.json' (format obert utilitzat como alternativa a XML per la trasnferència de dades estructurades entre un servidor Web i una aplicación Web) i '.hcl' (format per proporcionar funcionalitats automatitzades de documentació) per automatitzar la creació d'imatges.

Packer agafa una imatge (en el nostre cas una AMI (Amazon Machine Image)) habitualment amb un S.O, crea una instància (Virtual Machine) amb aquesta imatge, instal · la el S.O en la màquina objectiu, obre un servidor SSH en la instància creada i vía SSH, Ansible instal · la les dependències que constin en el 'playbook' en la instància. Finalment Packer crea una nova imatge amb tot el conjunt (S.O + software instal · lat per Ansible) que anomenem 'template'.

Packer és molt similar a Ansible però queda limitat a VM. Els seus templates d'imatges sempre tenen la mateixa versió que definim en el 'Packer Build'. Permet tenir doncs, una infraestructura inmutable (és a dir, que un cop es crea o s'instal · la alguna cosa, aquest mai serà modificat).

Aquí sota definim un exemple de com treballa Packer:

- **Pas 1:** Tenim una VM amb diversos softwares i volem substituir 'Apache' per 'NGINX'.
- **Pas 2:** Executem Packer partint d'una template 'python_apache_mysql' canviant el 'playbook' d'Ansible fent que desinstal · li 'Apache' i instal · li 'NGINX', llavors tindrem 2 VM, la inicial i la desitjada.
- **Pas 3:** Un cop hem comprovat que tot està bé, donem de baixa la inicial i donem d'alta la desitjada.

PROS: Multipltaforma, inmutabilitat.

CONTR: Només per a VM.

Vagrant by HashiCorp:

Gestiona únicament màquines virtuals (VM) i en un volum petit. S'utilitza principalment per a desenvolupadors de codi que busquin replicar escenaris de

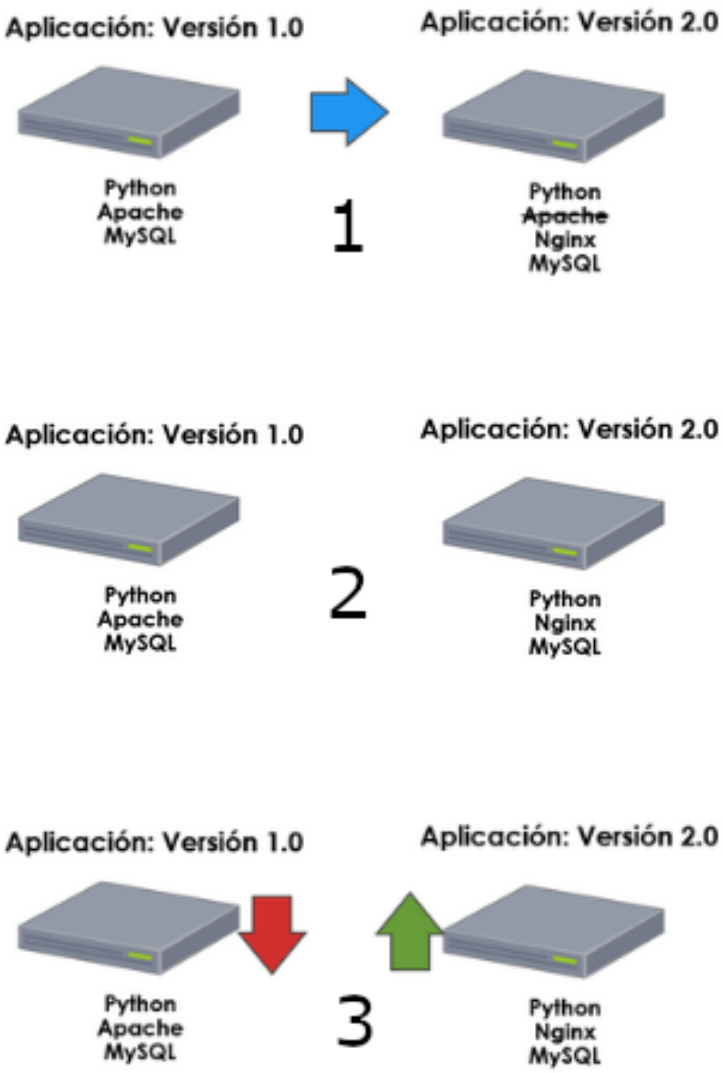

Figure 1: Imatge Packer

desenvolupament/test. Així executant Vagrant tots els desenvolupadors d'un mateix equip tindran el mateix entorn. (mateixes versions de soft)

Terraform by HashiCorp:

Es un 'IaC' que construeix/aixeca/desmonta/escala/replica i gestiona infraestructura (per infraestructura entenem conjunt de màquines i xarxes).

Podem gestionar diverses subestructures de diversos servidors Cloud només loggejats a Terraform per tant podem mantenir la nostra infraestructura obiant quín és el servidor Cloud on son els recursos (hosts/servidors/serveis/xarxes). Utilitza fitxers tipus *.tf* programats amb llenguatge 'HCL' on definim l'escenari desitjat. També tenim el 'TF CLI' (CLI de Terraform) on podem executar el 'terraform plan', què compararà l'escenari/estat actual amb el desitjat i comprovarà les operacions a realitzar o els canvis. Això és molt útil si només volem saber quins canvis és produïràn respecte a la infraestructura existent.

El pas 'terraform apply' serà on Terraform mitjançant les API (Application Programming Interface) i els token API, es communicarà amb el(s) proveïdors per executar les ordres per conseguir l'escenari desitjat.

PROS: Es open-source, gratutita i multiplatforma. Es adecuat per fer canvis en infraestructures grans, amb múltiples components i servidors de Cloud.

Diferències entre Ansible, Terraform i Vagrant:

Passos a seguir en Terraform per aixercar l'infraestrucutra del nostre projecte i alguns conceptes de Terraform i AWS:

Passos a realitzar:

- Creació d'una 'VPC'.
- Afegir dos 'Public Subnet', dos 'Private Subent' (cada parell en una regió d'una mateixa zona) i el GW.
- Afegir 'Route Table' i la ruta per defecte de la 'Private Subnet' i de la 'Public Subnet' + vinculació d'aquestes).
- Afegir les interficies NAT per poder sortir a internet.
- Creació i vinculació a una interficie del 'ALB' una Elastic IP (pública).
- Creació d'un servei 'ALB' (Application Load Balancer de proxy invers).
- Creació d'un 'SecurityGroup' per la 'Private Subnet' i la 'Public Subnet'.
- Configuració de l'AutoScaling i creació d'un "target group" i vinculació amb el ALB.
- Configuració de les alarmes del 'CloudWatch' (més del 60 % CPU) i accions (aixecar una altre instancia de la AMI).

Alguns conceptes sobre la sintàxi de Terraform:

- **resource** –> Accedir a un recurs del proveïdor.
- **var.XXX** –> Referenciem una variable del fitxer '.tfvars'.

- **${var.XXX}** -p 6000 –> Fem interpolació entre text dins d'una variable i text constant.
- **data.** –> Accedir a dades d'algun fitxer. Similar a un 'simbolic link'.
- **${ }** –> Una forma de cridar una variable.
- [] –> Llista que espera més d'un resultat.
- **file** –> Referència a les dades dins d'un fitxer.
- **tag** –> Com es dirà a AWS el nostre recurs.
- **$(element(var.llista, count.index))** –> Dins d'una llista agafa tots els valors.

Ordres més importants de Terraform:

- **terraform init** –> Inicialitza refrescant/baixant tots els fitxers de configuració dins del directori de treball (proveïdors, dependències, pluggins etc). *IMPORTANT!!:* Executar quan s'afegeixi/canviï un proveïdor.
- **terraform plan -auto-approve** –> Veiem els canvis que és faràn sense necessitat de dir 'yes'.
- **terraform apply -auto-approve** –> Executem el plà de terraform sense necessitat de dir 'yes'.
- **terraform fmt** –> Identar correctament el fitxer.
- **terraform destroy** –> Desmontem tot el que hem fet.
- **terraform state show aws_instance.nginx_instance** –> Mostra informació del 'resource' escollit.
- **terraform refresh** –> Llegeix la configuració actual de tots els objectes remots gestionats i actualitza l'estat de Terraform perquè coincideixi.

Alguns conceptes i sigles d'AWS:

- **arn**: Els noms de recursos d'Amazon (ARN) identifiquen de forma exclusiva els recursos d'AWS.
- **tag**: Argument que permet identificiar els recursos configurats pel client a AWS.
- **SNS**: Simple Notification Service.
- **ASG**: Auto Scaling Group.

Explicació dels fitxers que utilitzarem:

- **main.tf**: Fitxer principal on ficarem les coses més importants per el nostre treball.
- **IGW_vpc_subnets.tf**: Fitxer on configurarem tot el que tingui que veure amb la xarxa (Internet GW, NAT GW, VPC, Elastic IP, Subnets, Route Tables).
- **cloud_watch.tf**: Fitxer on crearem el CloudWatch el tipus d'alarma (per controlar la CPU) i polítiques d'escalada i de baixada.
- **datasources.tf**: Fitxer on posarem quina AMI utilitzarà la nostra instància.
- **outputs.tf**: Fitxer que ens servirà d'ajuda per quan fem 'terraform plan' i 'terraform apply', ens mostri per pantalla el valor d'algun resource.

- **providers.tf**: Fitxer on especificarem quin és el nostre proveïdor (AWS), les nostres credencials...
- **security_groups.tf**: Fitxer on especificarem els 'SecurityGroups' que tindràn els diferents recursos.
- **terraform.tfstate**: Fitxer que guarda l'estat actual de l'arquitecura si hem fet un 'terraform plan' o un 'terraform apply'.
- **terraform.tfvars**: Fitxer on declarem variables que cridarem en diferents fitxers.
- **userdata[2].tpl**: Script provisional (això ho farem amb Ansible) que executa codi bash (perquè la nostra imatge és Ubuntu).

Configuració de l'arquitectura AWS amb Terraform:

Per començar, obrirem el 'Visual Studio Code', i instal · larem l'extensió necessaria per poder començar a treballar:

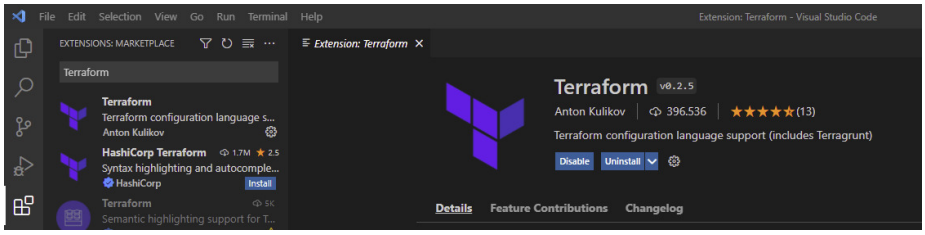

Figure 2: Terraform Extension

A continuació, editarem el següent fitxer ('$home/.aws/credentials') on crearem un perfil (li podem posar el nom que vulguem) i li ficarem la nostra 'aws_access_key_id' i 'aws_secret_access_key':

```
1
2 # Amazon Web Services Credentials File used by AWS CLI, SDKs, and tools
3 # This file was created by the AWS Toolkit for Visual Studio Code extension.
4 #
5 # Your AWS credentials are represented by access keys associated with IAM users.
6 # For information about how to create and manage AWS access keys for a user, see:
7 # https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html
8 #
9 # This credential file can store multiple access keys by placing each one in a
10 # named "profile". For information about how to change the access keys in a
11 # profile or to add a new profile with a different access key, see:
12 # https://docs.aws.amazon.com/cli/latest/userguide/cli-config-files.html
13 #
14 [     :IAM]
15 # The access key and secret key pair identify your account and grant access to AWS.
16 aws_access_key_id =
17 # Treat your secret key like a password. Never share your secret key with anyone. Do
18 # not post it in online forums, or store it in a source control system. If your secret
19 # key is ever disclosed, immediately use IAM to delete the access key and secret key
20 # and create a new key pair. Then, update this file with the replacement key details.
21 aws_secret_access_key =
```

Figure 3: AWS credentials

Un cop fet tot el mencionat, començarem amb la creació i edició dels fitxers explicats anteriorment.

IMPORTANT!!: Hi ha algún error en el codi i no funciona correctament el desplegament (no fa la funció descrita).

Fitxer 'main.tf':

```
variable Azones { description = "zones disponibles" }
variable Azone { description = "zona única a assignada" }
variable Azone2 { description = "zona única b assignada" }

resource "aws_lb" "loadbalancer" {
  name                       = "terraform-alb"
  load_balancer_type         = "application"
  subnets                    = [aws_subnet.public_subnet.id,aws_subnet.public_subnet_2.id]
  security_groups            = [aws_security_group.permetre_http_s.id]
  enable_deletion_protection = false
  tags                       = {Name = "terraform-alb"}
}

resource "aws_lb_listener" "web_http" {
  load_balancer_arn = aws_lb.loadbalancer.arn
  port              = "80"
  protocol          = "HTTP"
   default_action {

    type             = "forward"
    target_group_arn = aws_lb_target_group.test1.arn
  }
}

resource "aws_lb_target_group_attachment" "register_instance_to_group" {
  target_group_arn = aws_lb_target_group.test1.arn
  target_id        = aws_instance.nginx1.id
  port             = 80
}

resource "aws_lb_target_group_attachment" "register_instance_to_group2" {
  target_group_arn = aws_lb_target_group.test1.arn
  target_id        = aws_instance.nginx3.id
  port             = 80
}

resource "aws_lb_target_group" "test1" {
  name     = "targetgrouplb1"
  port     = 80
```

```
  protocol = "HTTP"
  vpc_id   = aws_vpc.main_vpc.id

  health_check {
    healthy_threshold   = 2
    unhealthy_threshold = 2
    interval            = 30
  }
}

variable ssh_pub {
  description = "Ruta on estan la clau pública en local i que es propagarà per a cada instan
}

resource "aws_key_pair" "key_pair_server_web" {
  key_name   = "server_web_key"
  public_key = file(var.ssh_pub)
}

resource "aws_instance" "nginx1" {
  instance_type = "t3.micro"
  ami           = data.aws_ami.server_web.id
  key_name      =  "server_web_key"
  user_data     = file("userdata.tpl")
  monitoring    = true
  root_block_device {
    volume_size = 10
  }

  tags = {
    Name = "nginx-node1"
  }

  network_interface {
    network_interface_id = aws_network_interface.loadbalancer.id
    device_index         = 0
  }
}

resource "aws_instance" "nginx3" {
  instance_type = "t3.micro"
  ami           = data.aws_ami.server_web.id
  key_name      = "server_web_key"
  user_data     = file("userdata2.tpl")
  monitoring    = true
```

```
  root_block_device {
    volume_size = 10
  }

  tags = {
    Name = "nginx-node3"
  }

  network_interface {
    network_interface_id = aws_network_interface.loadbalancer.id
    device_index         = 0
  }
}
```

Fitxer 'IGW_vpc_subnets.tf':

```
resource "aws_internet_gateway" "main_internet_gateway" {
  vpc_id = aws_vpc.main_vpc.id
  tags   = {Name = "principal-igw"}
}

#  VPC

resource "aws_vpc" "main_vpc" {
  cidr_block = "10.0.0.0/16"
  tags       = {Name = "principal"}
}

# aws_main_route_table_association = Provides a resource for managing the main routing table

resource "aws_main_route_table_association" "default" {
  count          = "1"
  vpc_id         = aws_vpc.main_vpc.id
  route_table_id = aws_route_table.main_public_rt[count.index].id
}

# SUBNET PUBLICA 1
resource "aws_subnet" "public_subnet" {
  vpc_id                  = aws_vpc.main_vpc.id
  cidr_block              = "10.0.0.0/24"
  map_public_ip_on_launch = true
  availability_zone       = var.Azone
  tags                    = {Name = "principal-public"}
  depends_on              = [aws_internet_gateway.main_internet_gateway]
}

resource "aws_route_table" "main_public_rt" {
```

```
  count  = "1"
  vpc_id = aws_vpc.main_vpc.id
  tags   = {Name = "principal_public_rt"}
}

resource "aws_route" "default_route_pu" {
  count                  = "1"
  route_table_id         = aws_route_table.main_public_rt[count.index].id
  destination_cidr_block = "0.0.0.0/0"
  gateway_id             = aws_internet_gateway.main_internet_gateway.id
}

resource "aws_route_table_association" "main_public_assoc" {
  count          = "1"
  subnet_id      = aws_subnet.public_subnet.id
  route_table_id = aws_route_table.main_public_rt[count.index].id
}

# SUBNET PUBLICA 2
resource "aws_subnet" "public_subnet_2" {
  vpc_id                  = aws_vpc.main_vpc.id
  cidr_block              = "10.0.1.0/24"
  map_public_ip_on_launch = true
  availability_zone       = var.Azone2
  tags                    = {Name = "principal-public2"}
  depends_on              = [aws_internet_gateway.main_internet_gateway]
}

resource "aws_route" "route" {
  route_table_id         = "${aws_vpc.main_vpc.main_route_table_id}"
  destination_cidr_block = "0.0.0.0/0"
  gateway_id             = aws_internet_gateway.main_internet_gateway.id
}

resource "aws_route_table_association" "main_public_assoc2" {
  count          = "1"
  subnet_id      = aws_subnet.public_subnet_2.id
  route_table_id = aws_route_table.main_public_rt[count.index].id
}

# Network Interface 'LB'
resource "aws_network_interface" "loadbalancer"{
  private_ips     = ["10.0.0.8"]
  subnet_id       = aws_subnet.public_subnet.id
  security_groups = [aws_security_group.permetre_http_s.id]
}
```

```
# Elastic IP
resource "aws_eip" "elastic" {
 associate_with_private_ip = "10.0.0.8"
 vpc                       = true
 depends_on                = [aws_internet_gateway.main_internet_gateway]
}
```

Fitxer 'cloud_watch.tf':

```
resource "aws_autoscaling_policy" "web_policy_up" {
  name                      = "web_policy_up"
  adjustment_type           = "ChangeInCapacity"
  estimated_instance_warmup = 120
  autoscaling_group_name    = aws_autoscaling_group.group_autoscal.name
  policy_type               = "TargetTrackingScaling"

  target_tracking_configuration {
    predefined_metric_specification {
      predefined_metric_type = "ASGAverageCPUUtilization"
    }

    target_value = 60.0
  }
}

resource "aws_cloudwatch_metric_alarm" "web_server_ALARM_up" {
  alarm_name          = "web_server_CPU_alarm_up"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods  = "2"
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = "120"
  statistic           = "Average"
  threshold           = "60"

  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.group_autoscal.name
  }

  alarm_description = "This metric monitors ec2 cpu utilization"
  alarm_actions     = ["${aws_autoscaling_policy.web_policy_up.arn}"]
  actions_enabled = true
}

resource "aws_autoscaling_policy" "web_policy_down" {
  name                   = "web_policy_down"
```

```
  scaling_adjustment     = -1
  adjustment_type        = "ChangeInCapacity"
  cooldown               = 120
  autoscaling_group_name = aws_autoscaling_group.group_autoscal.name
}

resource "aws_cloudwatch_metric_alarm" "web_server_ALARM_down" {
  alarm_name          = "web_server_CPU_alarm_down"
  comparison_operator = "LessThanOrEqualToThreshold"
  evaluation_periods  = "2"
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = "120"
  statistic           = "Average"
  threshold           = "10"

  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.group_autoscal.name
  }

  alarm_description = "This metric monitors ec2 cpu utilization"
  alarm_actions     = ["${aws_autoscaling_policy.web_policy_down.arn}"]
}
```

Fitxer 'datasources.tf':

```
data "aws_ami" "server_web" {
    most_recent = true
    owners      = ["099720109477"]
    filter {
        name   = "name"
        values = ["ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-*"]
    }

    filter {
      name   = "virtualization-type"
      values = ["hvm"]
    }
}
```

Fitxer 'outputs.tf':

```
output "elastic_IP" {
  count = "1"
  value = aws_eip.loadbalancer[count.index].public_ip
}
```

Fitxer 'providers.tf':

```
terraform {
  required_providers {
    aws = {
      source = "hashicorp/aws"
    }
  }
}

provider "aws" {
  region                   = "us-west-2"
  shared_credentials_files = ["~/.aws/credentials"]
  profile                  = "fornesIAM"
}
```

Fitxer 'security_groups.tf':

```
resource "aws_security_group" "permetre_http_s" {
  name        = "permetre_trafic_http_s"
  description = "Permetre trafic HTTP i HTTPS inbound (entrada a la pública)"
  vpc_id      = aws_vpc.main_vpc.id

  ingress {
    description      = "SSH"
    from_port        = 22
    to_port          = 22
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
  }

  ingress {
    description      = "HTTP from VPC"
    from_port        = 80
    to_port          = 80
    protocol         = "tcp"
    cidr_blocks      = ["0.0.0.0/0"]
  }

  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    cidr_blocks      = ["0.0.0.0/0"]
    ipv6_cidr_blocks = ["::/0"]
  }

  tags = {
    Name = "permetre_http_s"
```

```
  }
}
```

Fitxer 'terraform.tfstate':

```
{
  "version": 4,
  "terraform_version": "1.1.9",
  "serial": 1351,
  "lineage": "f7f56707-39fb-5de8-9713-233ce0b29099",
  "outputs": {},
  "resources": []
}
```

Fitxer 'terraform.tfvars':

```
ssh_pub = "/var/tmp/projecte_asix2/ssh_keys/id_ed25519.pub"
Azones  = ["us-west-2a", "us-west-2b"]
Azone   = "us-west-2a"
Azone2  = "us-west-2b"
```

Fitxer 'userdata.tpl':

```
#!/bin/bash

sudo apt-get update -y
sudo apt install nginx -y
systemctl start nginx
systemctl enable nginx
```

Fitxer 'userdata2.tpl':

```
#!/bin/bash

sudo apt-get update -y
sudo apt install nginx -y
systemctl start nginx
systemctl enable nginx

echo "<h1>hello world</h1>" > /var/www/html/index.html
```

Utilització de Packer per crear la imatge i Ansible per instal · lar el software:

Per què Packer i Ansible?

En el nostre cas tenim un AutoScaling i en el cas de necessitar fer actualizacions en el codi de la aplicació (codi del servei de venta d'entrades), és necessari tornar a generar la imatge perquè l'AutoScaling inicï novament les instàncies amb la versió més recent de l'aplicació.

Primer de tot instal · lem Packer i Ansible:

LINK DE LES PÀGINES OFICIALS:

- **Packer:** https://learn.hashicorp.com/tutorials/packer/get-started-install-cli
- **Ansible:** https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html

A continuació, creem els directoris de treball per ficar els fitxers de Packer i Ansible. L'estructura serà la següent:

mkdir packer | cd packer/ | mkdir ansible

Explicació del fitxer:

- **'packer { ... ':**
  - *required_plugins:* Instal · la els pluggins del proveïdor espceficat segons la versió de Packer.
- **source ... :** Especifiquem la imatge orígen, el Hardware i tot allò per definir la instància que crearem.
- **build:**
  - *sources:* A partir d'una imatge crea una instància PROVISIONAL.
  - *provisioner "ansible":* Especifiquem quina eina utilitzarem per la instal · lació del Software (en el nostre cas 'Ansible').
  - *provisioner "file":* Especifiquem si volem copiar algún fitxer a la màquina destí.

Dins del directori 'packer' crearem el següent fitxer (**ubuntu_aws_nginx.pkr.hcl**) i posarem el següent:

```
packer {
  required_plugins {
    amazon = {
      version = ">= 1.0.9"
      source  = "github.com/hashicorp/amazon"
    }
  }
}

source "amazon-ebs" "ubuntu" {
  secret_key = "XXXXXXXXXXXXXXXXXXXXX"
  access_key = "XXXXXXXXXXXXXXXXXXXXX"
  ami_name        = "nginx-ami-ubuntu-{{timestamp}}"
  ami_description = "Nginx Web Server"
  instance_type   = "t3.micro"
  region          = "us-west-2"
  source_ami      = "ami-0ee8244746ec5d6d4"
```

```
  ssh_username    = "ubuntu"
  vpc_id = "vpc-092fb7ce2a9e2f011"
  subnet_id = "subnet-0ff1c6f3fd4a4b43d"
  associate_public_ip_address = "true"
  tags = {
    Name = "Nginx"
    Os   = "Ubuntu 20.04"
  }
}

build {
  name = "nginx-ami"
  sources = [
    "source.amazon-ebs.ubuntu"
  ]

  provisioner "ansible" {
    playbook_file   = "ansible/playbook_ubuntu.yml"
  }

  provisioner "file" {
    source      = "config/webapp.conf"
    destination = "/tmp/webapp.conf"
  }
}
```

Explicació del fitxer:

- **'tasks':**
  - *name (1):* Instal · lem NGINX.
  - *name (2):* Inicialitzem NGINX.
  - *name (3):* Remplacem la documentació de 'root' per la nostra.
- **handlers:** Especifiquem una tasca que quan li arribi una notificació (en el nostre cas, quan s'hagin copiat els fitxers), reinicï NGINX.

Dins del directori 'ansible' crearem el 'playbook' (**playbook_ubuntu.yml**) amb la instal · lació d'NGINX:

```
---
- name: 'Install nginx'
  hosts: default

  tasks:
    - name: Install nginx
      apt:
        name: nginx
        state: present
      become: yes
```

```
    - name: start nginx
      service:
        name: nginx
        state: started
        enabled: yes
      become: yes

    - name: Update default document root
      become: yes
      replace:
        path: "/etc/nginx/nginx.conf"
        replace: "root         /usr/share/nginx/html/app;"
        regexp: "root         /usr/share/nginx/html;"
      notify: restart nginx

  handlers:
    - name: restart nginx
      service:
        name: nginx
        state: restarted
      become: yes
```

Un cop fets els fitxers, els pasos previs següents:

- Entrem en el nostre compte d'AWS i creem una VPC (amb subnet) 'default' associada al nostre usuari.

Ara executem les següents comandes a la línea de comandes:

- Baixem els pluggins que requerixi el fitxer de Packer:

```
packer init
```

- Validem la sintàxi del fitxer 'Packer', sigui '.json' o '.hcl':

```
packer validate ubuntu_aws_nginx.pck.hcl
```

- Compilem i contruim la imatge a 'AWS':

```
packer build ubuntu_nginx.pck.hcl
```

Validacions fetes a la interfície gràfica d'AWS i a la línea de comandes:

- VPC Packer:
- AMI generada correctament:
- Instància creada per Packer:
- AMI amb NGINX:

CONCEPTES A TENIR EN COMPTE:

Figure 4: VPC Packer

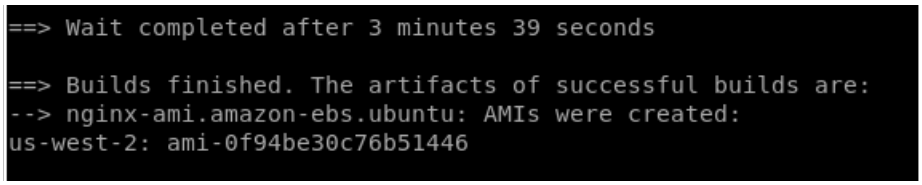

Figure 5: AMI generada correctament

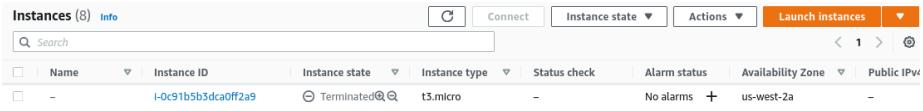

Figure 6: Instància creada per Packer

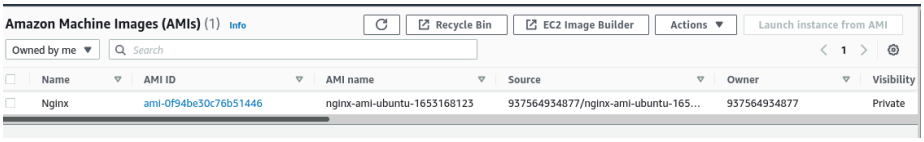

Figure 7: AMI amb NGINX

provisioner “ansible-local” **VS** provisioner “ansible”:

Type: ansible -> CMIIW Run ansible on machine running the packer (**aka local machine**)

Type: ansible-local -> CMIIW Run ansible on machine created by packer (**aka remote machine**)

Conclusions finals:

Un cop finalitzat el projecte ens hem donat compte de la gran magnitud i utilitat de les eines que segueixen la filosofia ‘IaC’, així com de la virtualització de màquines, que de fet, van de la mà. Es molt important començar a entendre que hi ha moltes eines que realitzen les mateixes funcions però que n’hi ha unes que destaquen entre les altres. Per exemple, Terraform és una eina molt potent per orquestrar infraestructures ja que es mutliproveidor i multiplataforma, a més de seguir un estil declaratiu (li dius el resultat però no com ho ha d’obtenir), que vol dir que nosaltres especifiquem que volem i el software ja es busca la vida per tenir el resultat final.

En el cas d’Ansible (un altre eina d’automatització), destaca per instal · lar software en S.O de màquines “vives” de forma local o remota, tant en màquines virtuals com reals a més a més, és tracta d’una eina *client-less*, que vol dir que no necesita cap software client per funcionar.

D’altra banda, tenim Packer, l’eina què ens garanteix inmutabilitat gràcies a les seves templates. La inmutabilitat ens garanteix gràcies a què totes les imatges que creem amb la plantilla seràn exactament iguals ja que especifiquem dins les versions del S.O, Software, etc.

Respecte al nostre projecte en concret, per falta de temps no hem pogut arribar fins on ens hagués agradat.

Encara això, hem pogut despelgar solucions parcials a Terraform utilitzant com a proveïdor AWS, com ara; desplegar un sistema de balanceig de càrrega (ALB) contra servidors NGINX en màquines diferents, i per altre banda, hem pogut fer l’autoescalat horitzontal de varies màquines ubicades a diferents CPDs (regions) on definim polítiques (regles) per auto-escalar i executar alarmes del servei CloudWatch. També hem pogut generar una imatge customitzada del servidor NGINX emprant Packer amb Ansible en local i guardant la imatge resultant de forma autòmatica a AWS.

Bibliografía:

Documentació oficial utilitzada:

- https://www.awsacademy.com
- https://docs.aws.amazon.com/
- https://www.terraform.io/

- https://www.packer.io/
- https://www.ansible.com/
- https://www.blazemeter.com/
- https://obsproject.com/es
- https://shotcut.org/

Recomanacions pròpies que ens han sigut de molta utilitat:

- https://youtu.be/iRaai1IBlB0
- https://youtu.be/SLB_c_ayRMo
- https://morethancertified.com/
- https://www.freecodecamp.org/
- https://uvinum.engineering/versionando-los-templates-con-packer-11c9206ae26a
- https://github.com/quickbooks2018/Terraform-Classic-Modules/tree/master/modules
- https://docs.gitlab.com/ee/user/infrastructure/iac/