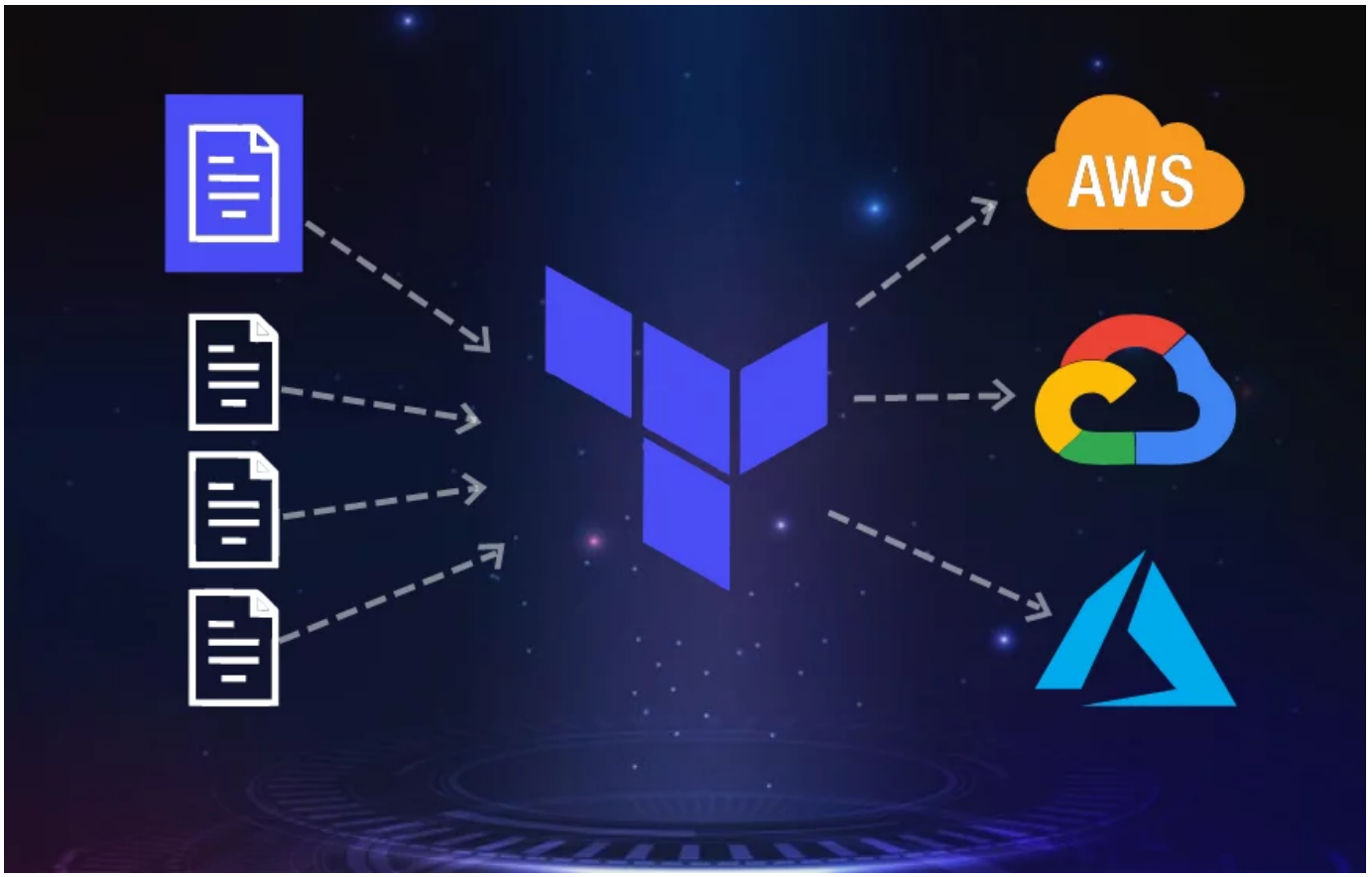
Infraestructures al Cloud amb Terraform



Marc Saez Rama
Projecte M14
CFGS ASIX 2
Institut Joan d'Àustria

Índex

Què és Terraform?

Terraform és una eina d'infraestructures com a codi Open Source. Amb Terraform podem definir una infraestructura i aixecar-la amb tres simples comandes: **terraform init**, **terraform plan** i **terraform apply**.
Les avantatges que ens dona tenir la nostra infraestructura definida en codi son moltes. Per exemple la Alta Disponibilitat (HA), qualsevol error, caiguda, pèrdua de la informació… no serà més que un petit problema, ja que amb les tres anteriors comandes tindrem de nou la infraestructura aixecada. Un altre avantatge és lo fàcil que és entendre com està construïda una infraestructura, ja que només llegint el codi pots saber amb què comptem i com està dissenyat.

```
# Lee el archivo y le pasa las siguientes variables
data "template_file" "cloud-init-config" {
  template = file("./config/cloud-init.yaml")
  vars = {
    docker_version = var.docker_version
    username       = local.node_username
  }
}
```

Apart, Terraform és compatible amb molts proveïdors de Cloud com Amazon Web Service, Google Cloud, Microsoft Azure, Cloudflare,... Encara que cada proveïdor té els seus propis recursos, per tant un codi escrit per aixecar un servidor web a AWS no servirà per qualsevol altre encara que serán molt semblants i igual de fàcil d'entendre.

Com funciona? El que fas tú es definir una infraestructura amb recursos de Terraform, un cop definida Terraform s'encarrega de descarregar els proveïdors necessaris que la crearan al Cloud. Llavors Terraform guardarà a un fitxer *terraform.tfstate* la infraestructura definida per el próxim cop que es facin canvis saber que ha canviat i que no.

Terraform com equip

Quan es treballa sol és fàcil controlar el teu propi codi i saber quins canvis han hagut pero en la majoria de casos treballarem amb altres persones, aquestes faran canvis al codi que nosaltres no sabrem i això farà dues coses, o que el codi deixi de funcionar o que destruïm coses que s'han creat perquè al nostre codi no las han escrit llavors Terraform detectarà que ja no les volem i les destruirà. Ja que el codi de Terraform és un mirall a la infraestructura del Cloud.
Hi han moltes solucions a tot això, però com no hi ha una millor que un altre he escollit la millor segons el meu criteri:
Tenir el arxiu *terraform.tfstate* a un bucket compartit S3 de AWS i controlar els canvis amb una base de dades DynamoDB perquè no es facin dos *terraform apply* a l'hora. Tot això, juntament amb GitHub, on tindré el codi penjat a un repositori que podràn accedir els meus "companys".

1ª part codi Terraform

En aquesta part crearé el bucket S3 i la DynamoDB pero primer hem d'entendre com funcionen els recursos a Terraform:

```
resource "nom_del_recurs" "nom_variable" {
  parametre = "valor"
}
```

El primer que hem de fer es definir el tipos de recurs (resource, data, provider, variable).

- Resource: tots els recursos que definirem per crear qualsevol cosa.
- Data: es fan servir per extreure dades de recursos ja existents per fer-los servir per proporcionar dades a altres recursos. També se li poden passar variables.
- Provider: és on especifiquem el proveïdor que necessitem i on li pasem les credencial per accedir-hi.
- Variable: simplement variables com a qualsevol altre llenguatge.

Després va el **nom del recurs**, Terraform sol ficar com a prefix el nom del proveïdor de Cloud (aws_), seguit del nom del recurs, exemple: *aws_s3_bucket*.
Seguidament va el **"nom variable"** que és el nom únic dintre del nostre codi que li donarem al nostre recurs per si després l'hem de cridar que Terraform sàpiga a quin recurs estàs fent referència.
Per últim van els paràmetres de cada recurs, que son únics per cada recurs i funcionen amb clau i valor, exemple: foo = "bar".

***
És important saber que Terraform per conectarse a AWS si no especifiquem res agafa les claus guardades al directori ***~/.aws/credential*** que tinguem per defecte.
***

Bucket S3 i DynamoDB

Primer de tot es crear el recurs que crea el S3, *aws_s3_bucket*:

```
resource "aws_s3_bucket" "terraform_state" {
  bucket = "terraform-state-msaez"
}
```

Aquest recurs crearà un bucket S3 a la nostre compte de AWS, i l'únic paràmetre obligatori és *bucket* on hem d'especificar el nom del bucket (important saber que Amazon demana que el nom del bucket sigui únic en tota la regió on estem).

El següent és un tema de seguretat, habilitar el versionament al bucket. Aquest recurs de AWS fa que quan un arxiu es sobreescriu es guardin les version, així si per algun cas hem de retrocedir a una versió anterior ,ja sigui perquè hem fet algun canvi malament o per qualsevol motiu, podrem retrocedir.

```
resource "aws_s3_bucket_versioning" "enabled" {
  bucket = aws_s3_bucket.terraform_state.id
  versioning_configuration {
    status = "Enabled"
  }
}
```

Al paràmetre *bucket* estem cridant a la id del recurso anterior, i al *versioning_configuration* li especifiquem que s'habiliti.

Per tenir més seguretat farem que el S3 estigui encriptat amb el següent recurs:

```
resource "aws_s3_bucket_server_side_encryption_configuration" "default" {
  bucket = aws_s3_bucket.terraform_state.id

  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
  }
}
```

Un altre cop cridem al *bucket* i li fiquem una regla, *apply_server_side_encryption_by_default* amb el paràmetre *sse_algorithm* on li diem l'algoritme d'encriptació (*AES256*)

Per últim, és important que només poguem accedir nosaltres al S3, ja que no hi ha cap necessitat de compartir el bucket a ningú que no estigui a la nostre compte. Llavors denegarem l'accés de l'exterior amb el següent recurs:

```
resource "aws_s3_bucket_public_access_block" "public_access" {
  bucket                  = aws_s3_bucket.terraform_state.id
  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}
```

Llavors només ens queda crear la DynamoDB perquè no es pugin fer dos *terraform apply* a l'hora, per això necessitem el recurs *aws_dynamodb_table* que ens permet controlar això.

```
resource "aws_dynamodb_table" "terraform_locks" {
  name         = "terraform-locks"
  billing_mode = "PAY_PER_REQUEST"
  hash_key     = "LockID"

  attribute {
    name = "LockID"
    type = "S"
  }
}
```

El primer paràmetre necessari és *name*, el nom de la taula. Després *billing_mode*, el mode de facturació que l'establirem en *PAY_PER_REQUEST* perquè per l'ús que el farem servir és el més econòmic. Per últim cal especificar el *hash_key* que és el nom que tindrà la clau de la taula, i, ademés cal dir de quin tipus serà a *attribute*, on indiquem el nom de la clau i el tipus "*S*", string.

Un cop fet això només caldrà executar Terraform amb un *init* i un *apply* es crearàn els recursos a la nostre conta de AWS.

```
aws_dynamodb_table.terraform_locks: Creating...
aws_s3_bucket.terraform_state: Creating...
aws_s3_bucket.terraform_state: Creation complete after 6s [id=terraform-state-msaez]
aws_s3_bucket_versioning.enabled: Creating...
aws_s3_bucket_public_access_block.public_access: Creating...
aws_s3_bucket_server_side_encryption_configuration.default: Creating...
aws_s3_bucket_server_side_encryption_configuration.default: Creation complete after 2s [id=terraform-state-msaez]
aws_s3_bucket_public_access_block.public_access: Creation complete after 2s [id=terraform-state-msaez]
aws_dynamodb_table.terraform_locks: Creation complete after 9s [id=terraform-locks]
aws_s3_bucket_versioning.enabled: Creation complete after 3s [id=terraform-state-msaez]

Apply complete! Resources: 5 added, 0 changed, 0 destroyed.
```

Però perquè Terraform fagi servir aquests recursos cal una cosa més, a qualsevol arxiu del directori on tenim tots els arxius de Terraform amb extensió *.tf* cal escriure el següent, únicament a un arxiu:

```
terraform {
  backend "s3" {
    bucket         = "terraform-state-msaez"
    key            = "global/s3/terraform.tfstate"
    region         = "us-east-1"
    profile        = "insti"
    dynamodb_table = "terraform-locks"
    encrypt        = true
  }
```

Aquí especificarem a Terraform d'on agafarà el *terraform.tfstate* i on el guardarà. Amb el paràmetre *backend* li diem que està a un *s3* i dins d'aquest paràmetre li diem el nom del *bucket* i on el guardarà/localitzarà amb *key*. Després, *dynamodb_table* el nom de la taula

anteriorment creada i *encrypt* perquè el guardi encriptat. Els paràmetres *region* i *profile* els hi específico jo perquè no agafi els per defecte.

```
Initializing the backend...
Backend configuration changed!

Terraform has detected that the configuration specified for the backend
has changed. Terraform will now check for existing state in the backends.

Acquiring state lock. This may take a few moments...
Acquiring state lock. This may take a few moments...
Do you want to copy existing state to the new backend?
  Pre-existing state was found while migrating the previous "s3" backend to the
  newly configured "s3" backend. No existing state was found in the newly
  configured "s3" backend. Do you want to copy this state to the new "s3"
  backend? Enter "yes" to copy and "no" to start with an empty state.

  Enter a value:
```

EC2 simple

Tot lo anterior es important per treballar en equip, però ara començarem amb un petit exemple de com crear una instància a AWS amb tot el que necesita.
Primer comencem amb la EC2:

```
resource "aws_instance" "hosting" {
  ami           =  data.aws_ami.ubuntu
  instance_type = var.instance_type
  associate_public_ip_address = true
  tags = {
    "Name" = "msaez"
  }
  vpc_security_group_ids = [aws_security_group.hosting.id]
  key_name = aws_key_pair.marc.key_name
}
```

Per començar cridem al recurs *aws_instance*, i li fiquem el nom que volguem (*hosting*). El primer paràmetre en tenir en compte és la *ami*, aquí hem d'escriure la id de la ami de Amazon que vulguem per la instància, el que passa és que a Amazon depenent de la regió on estem la id de cada ami canvia, i també va canviant durant el temps perquè s'actualitza. Llavors perquè el nostre codi funcioni el que fem es cridar al recurs *data* següent:

```
data "aws_ami" "ubuntu" {
  most_recent = true
  owners      = ["099720109477"] # Canonical

  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-bionic-18.04-amd64-server-*"]
  }

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

El recurs *data "aws_ami"* el que farà es buscar la id de la ami que li especifiquem als paràmetres, llavors ens servirà a qualsevol regió i en qualsevol moment. El primer paràmetre, *most_recent*, com indica el nom busca la més recent. Després l'indiquem la id del propietari (*owners*), que en el nostre cas és la id de Canonical ja que volem una imatge Ubuntu. Després li passem filtres per que busqui una versió en específic. En El nostre cas una Ubuntu Server Bionic 18.04 amd64, i després un altre filtre que és el tipos de virtualització, *hvm*, que aquest és el tipus que fa una virtualització completa, com si tinguéssim de veritat una màquina amb Ubuntu Bionic 18.04.

Com hem vist, així és filtra per obtenir la id correcte. I com podem veure a la imatge del recurs *hosting*, es crida simplement cridant al tipos de recurs i al seu nom únic (*data.aws_ami.ubuntu*).

Doncs amb el següent paràmetre, *instance_type*, fem el mateix. Cridem a la variable *instance_type* que podem tenir guardada en el mateix arxiu o en un altre del mateix repositori:

```
# VARS
variable "instance_type" { default = "t3a.medium" }
variable "docker_version" {default = "19.03"}
locals {
  node_username = "ubuntu"
}
variable "node_username" {}
```

L'anterior imatge és el contingut d'un fitxer *variables.tf* guardat en el mateix repositori que l'altre fitxer principal (*main.tf*). Només ens hem de centrar en la primera variable les altres son per fer servir més endavant.

Els següents paràmetres, *associate_public_ip_address* i *tags*, serveixen per associar una ip pública i els tags que li ficarem a la instància per identificar-la en aquest cas, el tag *Name* és el nom que tindrà la EC2, però podem ficar els tags que volguem.

```
associate_public_ip_address = true
tags = {
  "Name" = "msaez"
}
```

Els dos últims paràmetres son importants, ademés necessiten la creació d'un altre recurs per crear-los. Començarem amb *key_name*.

SSH Key

Aquest paràmetre és per passar el nom de la clau SSH pública per la qual ens connectarem. Pero per poder passar-li el nom primer necessitem passar-li la clau pública, i ho farem amb el recurs *aws_key_pair*.

```
resource "aws_key_pair" "marc" {
  key_name   = "marc-ssh_publickey"
  public_key = "ssh-rsa AAAAB3NzaC1yc2
}
```

Aquest recurs és molt senzill de crear pero el necessitarem bastant. Primer li passem al paràmetre *key_name* el nom amb el que es guardarà a AWS i després a *public_key* la clau SSH pública nostre per més tard poder entrar a la EC2 per SSH.
Un cop fet ja li podrem passar a l'altre recurs el nom de la clau pública cridant al recurs (*aws_key_pair.marc.key_name*) o directament escrivint el nom que li hem ficat (*marc-ssh_publickey*).

Seguim amb el Security Group

Security Groups

Aquest recurs s'indiquen les regles de sortida i entrada que tindràn tots els recursos de AWS que perteneixin al mateix grup. Per crear-lo hem de cridar al recurs *aws_security_group:*

```
resource "aws_security_group" "hosting" {
  name = "msaezsg-hosting"
  egress {
      cidr_blocks      = [ "0.0.0.0/0" ]
      description      = "Permitir salir a todo"
      from_port        = 0
      ipv6_cidr_blocks = []
      prefix_list_ids  = []
      protocol         = "-1"
      security_groups  = []
      self             = false
      to_port          = 0
    }

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

Primer cal ficar-li un nom amb el paràmetre *name*. I després especificar les regles de sortida (*egress*) i las d'entrada (*ingress*). Comencem per les de sortida.

Com nosaltres volem que la EC2 pugui accedir a qualsevol lloc li indicarem que el *cidr_blocks* que son el rang d'ips a las qual aplicará la regla sigui *0.0.0.0/0* que vol dir qualsevol IP, desde el port 0 (*from_port*) al port 0 (*to_port*), qualsevol port. Amb el protocol -1, que significa qualsevol protocol. Els demés paràmetres no són necessaris en el nostre cas.

Amb les regles d'entrada hem de ser més cuidadosos. Com nosaltres lo únic que volem és accedir per SSH, només obrirem el port 22 (por per defecte del SSH) del la següent manera:

```
ingress             = [
  {
    cidr_blocks      = [ "0.0.0.0/0" ]
    description      = "SSH connection"
    from_port        = 22
    protocol         = "tcp"
    security_groups  = []
    to_port          = 22
    ipv6_cidr_blocks = []
    prefix_list_ids  = []
    self             = false
 },
```

Si volguessim obrir qualsevol altre port, hauriem de fer el mateix adaptant-lo a la nostre necessitat.

Un cop creat l'anterior recurs només cal indicar-li al recurs de la EC2 que se li apliqui aquest grup de seguretat.

```
vpc_security_group_ids = [aws_security_group.hosting.id]
```

2ª part codi Terraform

Sabent fer algo bàsic con una EC2 a AWS, Terraform és pot complicar molt més fins el punt de crear tota una infraestructura de una empresa amb ell (VPC, Subnets, Load Balancer, VPN,...). El que faré jo, serà crear una instancia EC2 a la qual li pasarem un arxiu de configuració *cloud-init* que farà que quan es creï es configuri el que jo li volgui aplicar, un servei d'orquestració amb interfície gràfica (Rancher). I també, un servidor web que només li haurem de passar les pàgines web per aixecar una web.

Rancher a una EC2

Per poder crear Rancher necessitem la EC2 amb totes les seves necessitats com:

- Un Security Group amb els ports següents oberts:
  - SSH port 22 per connectar-nos remotament
  - HTTP port 80 per veure la interfície gràfica
  - HTTPS port 443 si ho volem fer de manera segura (SSL/TLS)
- La clau privada i pública SSH
- L'arxiu de configuració *cloud-init* per configurar la instal·lació de Rancher
- El recurs *aws_instance* on li passem tot lo creat

Security Group

Com ja ho hem vist abans, no hem centraré en com he creat aquest security group. Només mostraré com és:

```
# Security group
resource "aws_security_group" "rancher_sg_allowall" {
  name        = "rancher_sg"
  description = "Rancher"

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

  ingress {
    from_port   = "443"
    to_port     = "443"
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = ""
  }

  ingress {
    from_port   = "80"
    to_port     = "80"
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = ""
```

SSH Keys

Les claus SSH per la EC2 es poden pasar de moltes maneres, una manera és com l'hem vist anteriorment (simplement passant-li la pública a AWS amb el recurs aws_ssh_key_pair), però també es poden crear parells de claus SSH amb Terraform amb un recurs local anomenat *tls_private_key*.

```
resource "tls_private_key" "global_key" {
  algorithm = "RSA"
  rsa_bits = 2048
}
```

Aquest recurs genera un parell de claus i els podem guardar a on vulguem amb el recurs *local_file* de la següent manera:

```
resource "local_file" "ssh_private_key_pem" {
  filename = "keys/id_rsa"
  sensitive_content = tls_private_key.global_key.private_key_pem
  file_permission = "0600"
}

resource "local_file" "ssh_public_key_openssh" {
  filename = "keys/id_rsa.pub"
  content = tls_private_key.global_key.public_key_openssh
}
```

Així ja tindrem aquest parell de claus que podem fer servir per accedir a la EC2 passant-li a AWS com hem fet anteriorment:

```
# SSH Key pair
resource "aws_key_pair" "rancher_key_pair" {
  key_name = "rancher-key"
  public_key = tls_private_key.global_key.public_key_openssh
}
```

Cloud-init

Un arxiu cloud-init simplement és un yaml que li passem diferents paràmetres perquè a l'hora d'aixecar la EC2 s'autoconfigurin. Nosaltres el combinarem amb el recurs *data template_file* per poder passar-li variables.

```
#cloud-config
ssh_authorized_keys:
  - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQDpJGWF4vUpSyZgsT6/uylFwecdaL4six8YOgy2PmPnJCmwwbI6jUuPy8LsWfc7paRsc976/
  8IxUGOxlxQ7EucMrz5NrfPhjUv6DDztEKNTFg2moSCdReNuBNqjmdSd1Z68uVmMFqSMfbLfds7c+Kib3UF/
  VreKDS5nKs6gjLlQhjwbWtBW8WAAX2c0kqr6UbiGhYPDF2eiGLjPa6donhYxdaFZwN0cIVMWY48WX51Ptd9qJTKvtE10ZOgkmKy3LZK56+V0IJ81UC/
  DWEmrS0nHU2lolY8vsYrDSftf+krpvhKbx76FulPV8ZGRyxbOaNU/OhifyjZ6WTpAOmdOyy9AgjTNvx8/UIdOQkKeaHMuxwRWT7FIECziM+TqCLChjrv
  +RpZvIPuLKLLoX/AS1FsZbd6zNN9sHfNTQMTRqfEjE+5OIlteK8VuMMM5t1QyZeusb8/Ouz0YDA5vMuxTUFxucwP7jm+FFcU4jZjgPsxz2WUPedqJgeUluuXmh/
  xpFFE= austria@austria-Lenovo-V14-ADA

# Esta parte escribe el archivo rancher.service el el siguiente path
write_files:
  - path: /etc/systemd/system/rancher.service
    permissions: 0644
    owner: root
    content: |
      [Unit]
      Description=rancher-server como servicio
      Requires=docker.service
      After=docker.service

      [Service]
      Restart=on-failure
      RestartSec=10
      ExecStart=/usr/bin/docker run --name %p --rm --privileged -p 80:80 -p 443:443 -e CATTLE_BOOTSTRAP_PASSWORD=admin -v /opt/
      rancher:/var/lib/rancher rancher/rancher:stable
      ExecStop=-/usr/bin/docker stop -t 2 %p

      [Install]
      WantedBy=multi-user.target
runcmd:
  - export DEBIAN_FRONTEND=noninteractive
  - curl -sL https://releases.rancher.com/install-docker/${docker_version}.sh | sh
  - sudo usermod -aG docker ${username}
  - sudo systemctl start rancher
```

Anem per parts, primer és important saber que aquest arxiu ha de portar a l'inici del codi **#cloud-config**.

A continuació he incluit la meva clau principal SSH pública apart de la que ja passem anteriorment amb Terraform, *ssh_authorized_keys* simplement escriu a l'arxiu *authorized_keys* la clau pública que li especifiquem.

Després va *write_files*. El que fa això, és escriure el contingut que escrivim en *content* al directori que especifiquem a *path*. Donant-li els permisos que vulguem (*permissions*) i al propietari que diguem (*owner*).
El contingut de l'arxiu és simplement un servei que el que fa es iniciar un contenidor docker a on tindrem tot lo necessari per ficar en marxa Rancher (ports mapejats, volum mapejat i variable d'entorn especificada).

Per últim, *runcmd* el que fa és executar per ordre les comandes que escrivim:

- ***export DEBIAN_FRONTEND=noninteractive*:** perquè no necessiti interacció si ha d'actualitzar el sistema o fer un update.
- ***curl -sL https://{url} | sh:*** perquè instal·li docker a la versió que li especifiquem a la variable *docker_version* que veurem més endavant.
- ***sudo usermod -aG docker ${username}*:** per afegir l'usuari que especifiquem a la variable *username* al grup docker per no tenir la necessitat de ser superusuaris per executar docker.
- ***sudo systemctl start rancher*:** iniciar el servei de rancher creat anteriorment.

Aquest arxiu el guardarem com a *cloud-init.yaml* i el llegirem amb terraform amb el recurs *data template_file*

```
data "template_file" "cloud-init-config" {
  template = file("./config/cloud-init.yaml")
  vars = {
    docker_version = var.docker_version
    username       = var.username
  }
}
```

El paràmetre *template* li passarem la funció *file()* que buscarà a la ruta relativa l'arxiu que vulguem, i amb el paràmetre *vars* substituirà el que trobi dins de l'arxiu amb ${variable} per la variable que li diguem.

EC2

Un cop creats tots els "prerequisits", ja podem ajuntar-ho tot i crear la EC2 al complet.

```
resource "aws_instance" "rancher" {
  ami = data.aws_ami.ubuntu.id
  instance_type = var.instance_type
  key_name = aws_key_pair.rancher_key_pair.key_name
  subnet_id = "subnet-00f67cae2874a3204"
  security_groups = [aws_security_group.rancher_sg_allowall.id]
  user_data_base64 = base64encode(data.template_file.cloud-init-config.rendered)
  ebs_optimized = true
  iam_instance_profile = "LabInstanceProfile"
  root_block_device {
    volume_type = "gp2"
    volume_size = 16
    encrypted = true
  }

  provisioner "remote-exec" {
    inline = [
      "echo 'Waiting for cloud-init to complete...'",
      "cloud-init status --wait > /dev/null",
      "echo 'Completed cloud-init!'",
      "echo 'Rancher Server ready!'",
    ]
    connection {
      type = "ssh"
      host = self.public_ip
      user = var.username
      private_key = tls_private_key.global_key.private_key_pem
    }
  }

  tags = {
    Name = "rancher-server"
  }
}
```

Paràmetres importants a saber nous que podem veure son els següents:

- ***subnet_id*:** id de la subnet on volem la EC2
- ***user_data_base64*:** li passem el recurs *data* renderitzat que hem llegit el *cloud-init*, amb la funció *base64encode()* perquè per la màquina és més fàcil llegir-ho passat a base64
- ***ebs_optimitzed i root_block_device*** són paràmetres que li paso per millorar la eficiencia de la EC2, però no són obligatoris.
- ***iam_instance_profile*:** és un perfil que li paso a la EC2 perquè tingui permisos per fer algunes accions a AWS, però pel que estem fent no és necessari. Millor ignorar.

Com es pot veure, dins del recurs anterior li passo un altre recurs anomenat *provisioner remote-exec*. És important saber el que fa aquest recurs.
Simplement al paràmetre *connection* escriurem el tipus de connexió (ssh), el host (ip pública d'ella mateixa), l'usuari (ubuntu) i la clau privada ssh amb la que es connectarà. Llavors podrem passar-li comandes a la màquina per el paràmetre *inline* a l'hora que executem Terraform. Les comandes que li passem son simplement per saber si l'arxiu *cloud-init* s'ha acabat d'executar, quan ho hagi fet ens avisarà.

Outputs

Per no haver d'anar a la interfície gràfica d'Amazon es pot cridar a un recurs *output* perquè et doni la ip pública de la EC2 que li diguis:

```
output "ip_addr" {
  value       = aws_instance.rancher.public_ip
}

output "http_link" {
  value       = "http://${aws_instance.rancher.public_ip}"
  description = "HTTP Link Address"
}
```

Aquests recursos, al acabar de crear-ho tot, et donaràn la IP pública de la instancia.

Demo

```
Plan: 6 to add, 0 to change, 0 to destroy.

Changes to Outputs:
  + http_link = (known after apply)
  + ip_addr   = (known after apply)
```

```
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes

tls_private_key.global_key: Creating...
tls_private_key.global_key: Creation complete after 0s [id=8c39a441d7d741094c133a852d4d12c26595b58e]
local_file.ssh_public_key_openssh: Creating...
local_file.ssh_private_key_pem: Creating...
local_file.ssh_public_key_openssh: Creation complete after 0s [id=d07e33cd88c7ea290616db8bafdc05d1964de8ba]
local_file.ssh_private_key_pem: Creation complete after 0s [id=deff81d9770332231514ae4dfa14ab6d7e66bd99]
aws_key_pair.rancher_key_pair: Creating...
aws_key_pair.rancher_key_pair: Creation complete after 3s [id=rancher-key]
aws_security_group.rancher_sg_allowall: Creating...
```

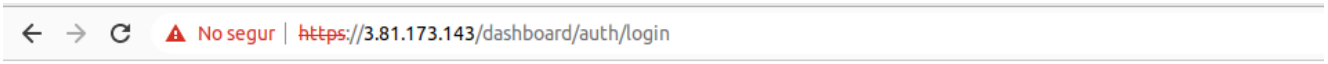
```
aws_security_group.rancher_sg_allowall: Creating...
aws_security_group.rancher_sg_allowall: Creation complete after 4s [id=sg-0e01c46c35dcd020c]
aws_instance.rancher: Creating...
aws_instance.rancher: Still creating... [10s elapsed]
aws_instance.rancher: Provisioning with 'remote-exec'...
aws_instance.rancher (remote-exec): Connecting to remote host via SSH...
aws_instance.rancher (remote-exec):   Host: 3.81.173.143
aws_instance.rancher (remote-exec):   User: ubuntu
aws_instance.rancher (remote-exec):   Password: false
aws_instance.rancher (remote-exec):   Private key: true
aws_instance.rancher (remote-exec):   Certificate: false
aws_instance.rancher (remote-exec):   SSH Agent: true
aws_instance.rancher (remote-exec):   Checking Host Key: false
aws_instance.rancher (remote-exec):   Target Platform: unix
```

```
aws_instance.rancher (remote-exec): Connecting to remote host via SSH...
aws_instance.rancher (remote-exec):   Host: 3.81.173.143
aws_instance.rancher (remote-exec):   User: ubuntu
aws_instance.rancher (remote-exec):   Password: false
aws_instance.rancher (remote-exec):   Private key: true
aws_instance.rancher (remote-exec):   Certificate: false
aws_instance.rancher (remote-exec):   SSH Agent: true
aws_instance.rancher (remote-exec):   Checking Host Key: false
aws_instance.rancher (remote-exec):   Target Platform: unix
aws_instance.rancher: Still creating... [40s elapsed]
aws_instance.rancher (remote-exec): Connected!
aws_instance.rancher (remote-exec): Waiting for cloud-init to complete...
aws_instance.rancher: Still creating... [50s elapsed]
aws_instance.rancher: Still creating... [1m0s elapsed]
```

```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

http_link = "http://3.81.173.143"
ip_addr = "3.81.173.143"
```



Howdy!

Welcome to Rancher

It looks like this is your first time visiting Rancher; if you pre-set your own bootstrap password, enter it here. Otherwise a random one has been generated for you. To find it:

For a "docker run" installation:

- Find your container ID with `docker ps`, then run:
- `docker logs container-id 2>&1 | grep "Bootstrap Password:"`

For a Helm installation, run:

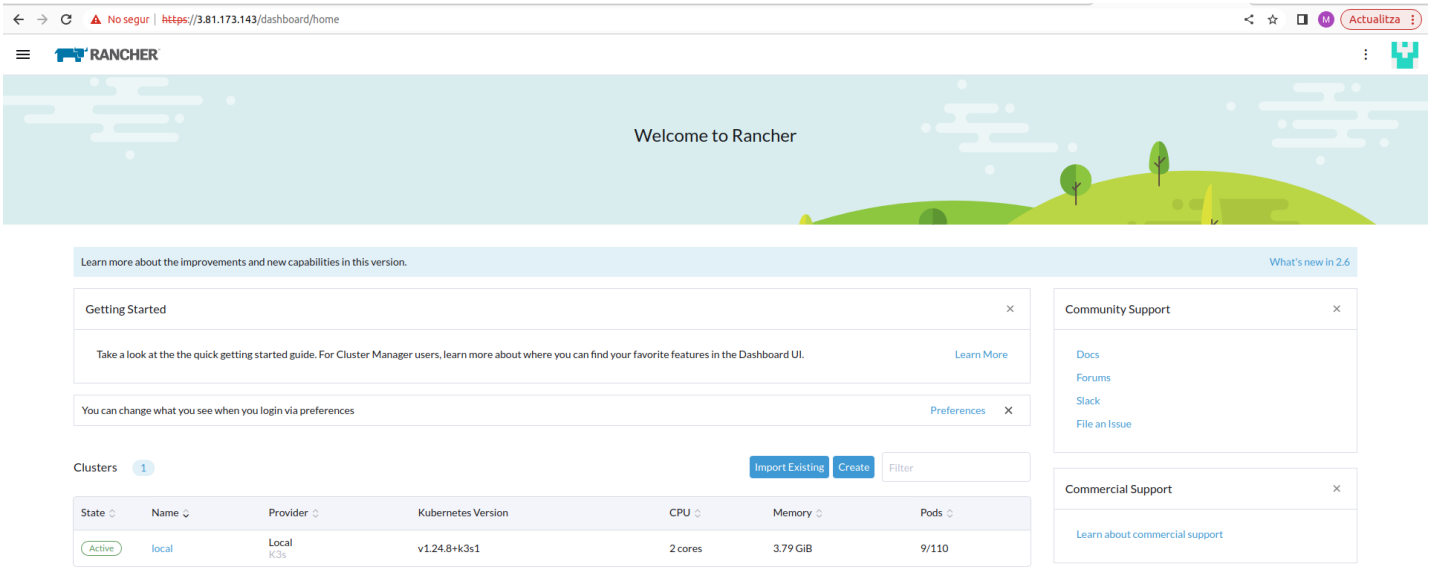
```
kubectl get secret --namespace cattle-system bootstrap-secret -o go-
template='{{.data.bootstrapPassword|base64decode}}{{"\n"}}'
```

Password

Show

Log in with Local User

English


No segur | https://3.81.173.143/dashboard/home
Actualitza
RANCHER
Welcome to Rancher
Learn more about the improvements and new capabilities in this version.
What's new in 2.6
Getting Started
Take a look at the the quick getting started guide. For Cluster Manager users, learn more about where you can find your favorite features in the Dashboard UI.
Learn More
You can change what you see when you login via preferences
Preferences
Community Support
Docs
Forums
Slack
File an Issue
Clusters 1
Import Existing
Create
Filter
State
Name
Provider
Kubernetes Version
CPU
Memory
Pods
Active
local
Local
K3s
v1.24.8+k3s1
2 cores
3.79 GiB
9/110
Commercial Support
Learn about commercial support

Servidor Web

La base del Servidor Web necessita cassi el mateix que Rancher, ports oberts per accedir per SSH, HTTP i HTTPS, passar-li les claus SSH, els outputs per saber la IP pública i el arxiu cloud-init per configurar el servidor web (apache). Pero aquí afegirem una segona part que la farem amb el recurs *null_resource* amb el qual li passarem al servidor cada pàgina web.
De la primera part que es crear el servidor web os explicaré només l'arxiu *cloud-init*, ja que lo demés no cambia molt.

main.tf

```
# EC2 hosting
resource "aws_instance" "hosting" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = var.ec2_type
  #key_name      = aws_key_pair.marc.key_name
  vpc_security_group_ids = [aws_security_group.hosting.id]
  associate_public_ip_address = true
  user_data_base64 = base64encode(data.template_file.cloud-init-config.rendered)
  tags = {
    "Name" = "${var.name}hosting"
  }
}
```

data.tf

```
# Ubuntu Server 22.04 AMI (Latest)
data "aws_ami" "ubuntu" {
    most_recent = true

    filter {
        name   = "name"
        values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-*"]
    }

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

    owners = ["099720109477"] # Canonical
}

# cloud-init.yaml
data "template_file" "cloud-init-config" {
  template = file("./config/cloud-init.yaml")
  vars = {
    ssh_msaez    = var.ssh_key_msaez
  }
}
```

variable.tf

```
#SSH
variable "ssh_key_msaez"    { default = "ssh-rsa AAAAB
# OTHERS
variable "name"        { default = "msaez-" }

# EC2
variable "ec2_type"        { default = "t2.micro" }
```

output.tf

```
output "ip_ec2" {
  value = "http://${aws_instance.hosting.public_ip}"
}
```

Cloud-init

```
#cloud-config

package_upgrade: true

packages:
  - apache2

ssh_authorized_keys:
  - ${ssh_msaez}

write_files:
  - path: /etc/apache2/sites-available/web1.conf
    content: |
      <VirtualHost *:80>
        ServerName saez912.com
        ServerAlias www.saez912.com
        DocumentRoot /var/www/web1
      </VirtualHost>
  - path: /etc/apache2/sites-available/web2.conf
    content: |
      <VirtualHost *:80>
        ServerName marcsaez.com
        ServerAlias www.marcsaez.com
        DocumentRoot /var/www/web2
      </VirtualHost>

runcmd:
  - a2dissite 000-default.conf
  - a2ensite web1.conf
  - a2ensite web2.conf
  - systemctl restart apache2
  - sudo mkdir /var/www/web1
  - sudo mkdir /var/www/web2
```

A *package_upgrade* li diem que s'actualitzi el sistema i a *packages* que s'instal·li *apache2*.
Al paràmetre *write_files* configurem els arxius .conf on li indicarem el path de on guardarem les webs i els noms que tindran per quan la busquin per DNS redirigeixi a cada una.
Per últim a *runcmd* deshabilitem la web que instal·la per defecte apache2 i activem les que hem configurat, i creem els directoris on les guardarem.

Un cop fet això ja tindrem el servidor web per ficar dos webs que volguessim, pero falta passar les webs. Això ho farem a continuació.

Null resource

Aquest recurs de terraform ens permet connectar-nos a qualsevol màquina que puguem accedir i passar-li arxius i/o comandes. El farem servir per passar les webs.

```
# WEB 1
resource "null_resource" "web1" {
  triggers = {
    instance_id = data.terraform_remote_state.main.outputs.ip_ec2
  }
  connection {
    type     = "ssh"
    user     = "ubuntu"
    host     = "${data.terraform_remote_state.main.outputs.ip_ec2}"
    private_key = "${file("~/.ssh/id_rsa")}"
  }
  provisioner "file" {
    source = "./webs/index.html"
    destination = "/home/ubuntu/index1.html"
  }
  provisioner "remote-exec" {
    inline = [
      "cat ~/index1.html",
      "sleep 60",
      "sudo cp ~/index1.html /var/www/web1/index.html"
    ]
  }
}
```

Anem per passos.
El primer és el paràmetre *triggers* on a podem escriure a on volem apuntar, en el nostre cas seria la IP pública de la EC2 on tenim l'Apache2. Jo l'extrec cridant a un recurs data que llegeix el *terraform.tfstate* on es guarda la IP pero és més senzill escriure la IP.

Després és important especificar la connexió, en el nostre cas com hem habilitat l'SSH o farem així, i només cal especificar cara paràmetre com ja hem fet anteriorment.

A continuació, amb el paràmetre *provisioner "file"* escriurem el path de la web que volem passar i com a destí el path on volem que es guardi en la EC2. Jo la guardo al home per tenir una copia a la mateixa màquina i no passar-la directament a */var/web/*.

Per últim, *remote-exec* serveix per escriure comandes a la màquina mitjançant Terraform, llavors li passarem les comandes necessàries per copiar la web al seu directori. Jo li fico un *sleep 60* perquè com ho faig tot de cop, a vegades no s'ha instal·lat apache quan estic passant-li ja la web.

Demo

```
Apply complete! Resources: 2 added, 0 changed, 0 destroyed.

Outputs:

ip_ec2 = "http://18.204.37.180"
```

```
  + resource "null_resource" "web2s" {
      + id       = (known after apply)
      + triggers = {
          + "instance_id" = "http://18.204.37.180"
        }
    }

Plan: 2 to add, 0 to change, 0 to destroy.
null_resource.web1: Creating...
null_resource.web1: Provisioning with 'file'...
null_resource.web1: Still creating... [10s elapsed]
null_resource.web1: Still creating... [20s elapsed]
null_resource.web1: Still creating... [30s elapsed]
null_resource.web1: Still creating... [40s elapsed]
null_resource.web1: Still creating... [50s elapsed]
null_resource.web1: Still creating... [1m0s elapsed]
```

```
null_resource.web2s (remote-exec):   Certificate: false
null_resource.web2s (remote-exec):   SSH Agent: true
null_resource.web2s (remote-exec):   Checking Host Key: false
null_resource.web2s (remote-exec):   Target Platform: unix
null_resource.web2s (remote-exec): Connected!
null_resource.web2s (remote-exec): total 68
null_resource.web2s (remote-exec): -rw-r--r-- 1 ubuntu ubuntu  3734 May 15 14:04 README.md
null_resource.web2s (remote-exec): -rw-r--r-- 1 ubuntu ubuntu  9427 May 15 14:04 about.html
null_resource.web2s (remote-exec): drwxr-xr-x 7 ubuntu ubuntu  4096 May 15 14:04 assets
null_resource.web2s (remote-exec): -rw-r--r-- 1 ubuntu ubuntu 11796 May 15 14:04 blog-list.html
null_resource.web2s (remote-exec): -rw-r--r-- 1 ubuntu ubuntu 14427 May 15 14:04 blog-post.html
null_resource.web2s (remote-exec): -rw-r--r-- 1 ubuntu ubuntu  1150 May 15 14:04 favicon.ico
null_resource.web2s (remote-exec): -rw-r--r-- 1 ubuntu ubuntu 12554 May 15 14:04 index.html
null_resource.web2s: Creation complete after 17s [id=8825996940981407011]
```

Normativa legal pàgines web

Lleis a tenir en compte per la nostre pàgina web:

Llei de Serveis de la Societat de la Informació i Comerç Electrònic (LSSI-CE):

Aquesta llei regula els serveis de la societat de la informació i el comerç electrònic a Espanya. Estableix els requisits legals que han de complir les pàgines web, com la informació que s'ha de proporcionar als usuaris, els drets dels consumidors, la publicitat, etc.

Llei Orgànica de Protecció de Dades (LOPD):

Aquesta llei regula la protecció de les dades personals a Espanya. Quan una pàgina web recopila i tracta dades personals dels usuaris, ha de complir amb les disposicions de la LOPD, com ara obtenir el consentiment de l'usuari per al tractament de les seves dades, garantir la seva seguretat, informar sobre la finalitat del tractament, etc.

Reglament General de Protecció de Dades (RGPD):

Aquesta normativa de la Unió Europea també és aplicable a les pàgines web espanyoles. El RGPD estableix normes estrictes sobre la protecció de les dades personals dels ciutadans de la Unió Europea. Això inclou informar sobre les pràctiques de privacitat, obtenir el consentiment explícit per al tractament de les dades, proporcionar drets als usuaris per controlar les seves dades, etc.

Llei de Serveis de la Societat de la Informació (LSSI):

Aquesta llei complementa la LSSI-CE i regula altres aspectes com ara els serveis de la societat de la informació en l'àmbit de l'administració electrònica, els contractes electrònics, les signatures digitals, etc.

Llei de Propietat Intel·lectual (LPI):

Aquesta llei regula els drets de propietat intel·lectual a Espanya, incloent-hi els drets d'autor i altres drets relacionats amb la propietat intel·lectual. Quan es publiquen continguts protegits per drets d'autor a una pàgina web, és necessari respectar els drets dels titulars dels mateixos.

Webgrafia

Recursos Terraform:
https://registry.terraform.io/providers/hashicorp/aws/latest/docs
https://developer.hashicorp.com/terraform/language/resources/syntax

Apache:
https://ubuntu.com/tutorials/install-and-configure-apache#1-overview

Normatives:
https://protecciondatos-lopd.com/empresas/requisitos-legales-pagina-web/#Requisitos_legales_para_una_pagina_web
https://webysocialmedia.com/legislacion-aplicable-las-paginas-web-espanolas/

Bibliografia

O'Reilly (oreilly.com):
Terraform Up and Running, 3rd Edition by Yevgeniy Brikman