

Automating Infrastructure using Terraform

Excercise description:

Nowadays, infrastructure automation is critical. We tend to put the most emphasis on software development processes, but infrastructure deployment strategy is just as important. Infrastructure automation not only aids disaster recovery, but it also facilitates testing and development. Your organization is adopting the DevOps methodology and in order to automate provisioning of infrastructure there's a need to setup a centralised server for Jenkins.

Terraform is a tool that allows you to provision various infrastructure components. Ansible is a platform for managing configurations and deploying applications. It means you'll use Terraform to build a virtual machine, for example, and then use Ansible to instal the necessary applications on that machine.

Considering the Organizational requirement you are asked to automate the infrastructure using Terraform first and install other required automation tools in it.

Tools required: Terraform, AWS account with security credentials, Keypair

Expected Deliverables:

- Launch an EC2 instance using Terraform
- Connect to the instance
- Install Jenkins, Java and Python in the instance

Author: Francisco Pedraza Vázquez

1. Create AWS credential Keys

First we conect to AWS, and search for IAM services.

IAM , or identity access management services is one of the many services provided by AWS in which we can manage perms, create users and generate key-access pairs.

The screenshot shows the AWS Services dashboard. At the top, there is a dark header with the text "Servicios" on the left and "Ver los 16 resultados ▶" on the right. Below the header, there is a list of services. The "IAM" service is highlighted with a red square icon, a blue "IAM" label, and a yellow star icon. Below the icon, the text "Administrar el acceso a los recursos de AWS" is displayed. Underneath this, there is a section titled "Características principales" with links for "Grupos", "Usuarios", "Roles", "Políticas", and "Analizador de acceso".

Click on Users

Identity and Access Management (IAM)

Users (1) Información

Nombre de usuario: corestack-e40d5 | Ruta: / | Grupo: - | Última actividad: - | MFA: - | Antigüedad de la clave: 1082 días | Último inicio de sesión: - | ID de clave de acceso: - | Antigüedad de la clave: -

Acceso denegado | Acceso denegado | Acceso denegado

Crear usuario | Eliminar | Detalle

Create the new User with administrator perms

Crear usuario

ID de clave de acceso | Antigüedad de la clave

Detalles del usuario

Nombre de usuario: terraform

Proporcione acceso de usuario a la consola de administración de AWS: opcional

Si está creando acceso mediante programación a través de claves de acceso o credenciales específicas de servicios para AWS CodeCommit o Amazon Keyspaces, puede generarlos después de crear este usuario de IAM.

Más información

Siguiente

Establecer permisos

Agregue un usuario a un grupo existente o cree uno nuevo. El uso de grupos es una práctica recomendada para administrar los permisos de usuario según las funciones laborales. [Más información](#)

Opciones de permisos

- Agregar usuario al grupo: Agregue el usuario a un grupo existente o cree uno nuevo. Le recomendamos que utilice grupos para administrar los permisos de usuario según las funciones laborales.
- Copiar permisos: Copie todas las suscripciones a grupos, las políticas administradas adjuntas y las políticas insertadas de un usuario existente.
- Adjuntar políticas directamente: Adjunte una política administrada a un usuario de manera directa. Como práctica recomendada, le sugerimos, en cambio, adjuntar políticas a un grupo. A continuación, agregue el usuario al grupo adecuado.

Políticas de permisos (1/1124)

Elija una o varias políticas para asociarlas al nuevo usuario.

Nombre de la política	Tipo	Entidades asociadas
<input checked="" type="checkbox"/> AccessAnalyzerServiceRolePolicy	Administrada por AWS	0
<input checked="" type="checkbox"/> AdministratorAccess	Administrada por AWS: función de trabajo	2
<input type="checkbox"/> AdministratorAccess	Administrada por AWS	0

Usuarios (2) Información
Un usuario de IAM es una identidad con credenciales válidas a largo plazo que se utiliza para interactuar con AWS en una cuenta.

<input type="checkbox"/>	Nombre de usuario	Ruta	Grupos	Última actividad	MFA	Antigüedad de	Último inicio de sesión	ID de clave de acceso	Antigüedad de la
<input type="checkbox"/>	corestack-e40d5	/	2	Acceso denegado	-	⚠ 1082 días	-	Acceso denegado	Acceso denegado
<input type="checkbox"/>	terraform	/	0	○	-	-	-	-	-

Permisos | Grupos | Etiquetas | **Credenciales de seguridad** | Access Advisor

Inicio de sesión en la consola

Enlace de inicio de sesión en la consola
🔗 <https://simplelearncourses.signin.aws.amazon.com/console>

Contraseña de la consola
No habilitado

Habilitar el acceso a la consola

Autenticación multifactor (MFA) (0)
Utilice MFA para aumentar la seguridad del entorno de AWS. Para iniciar sesión con MFA, se requiere un código de autenticación de un dispositivo MFA. Cada usuario puede tener un máximo de 8 dispositivos MFA asignados. [Más información](#)

Eliminar | Volver a sincronizar | Asignar dispositivo MFA

Tipo de dispositivo	Identificador	Certificaciones	Creada el
No hay dispositivos MFA. Asigne un dispositivo MFA para mejorar la seguridad del entorno de AWS.			

Asignar dispositivo MFA

Claves de acceso (0)
Utilice las claves de acceso para enviar llamadas mediante programación a AWS desde AWS CLI, Herramientas de AWS para PowerShell, AWS SDK o llamadas directas a la API de AWS. Puede tener un máximo de dos claves de acceso (activas o inactivas) a la vez. [Más información](#)

Crear clave de acceso

No hay claves de acceso. Como práctica recomendada, evite el uso de credenciales a largo plazo, como las claves de acceso. En su lugar, utilice herramientas que proporcionen credenciales a corto plazo. [Más información](#)

Crear clave de acceso

We enter **Security Credentials** and create a new **CLI** set of credentials

Caso de uso

Interfaz de línea de comandos (CLI)
Tiene previsto utilizar esta clave de acceso para permitir que la AWS CLI obtenga acceso a su cuenta de AWS.

Código local
Tiene previsto utilizar esta clave de acceso para habilitar el código de aplicación en un entorno de desarrollo local para obtener acceso a su cuenta de AWS.

Aplicación ejecutada en un servicio de computación de AWS
Tiene previsto utilizar esta clave de acceso para permitir que el código de aplicación que se ejecuta en un servicio de computación de AWS como Amazon EC2, Amazon ECS o AWS Lambda obtenga acceso a su cuenta de AWS.

Servicio de terceros
Tiene previsto utilizar esta clave de acceso para habilitar el acceso a una aplicación o servicio de terceros que supervise o administre sus recursos de AWS.

Aplicación ejecutada fuera de AWS
Tiene previsto utilizar esta clave de acceso para habilitar una aplicación que se ejecute en un host local o para utilizar un cliente de AWS local o un complemento de AWS de terceros.

Otros
Su caso de uso no aparece aquí.

Alternativas recomendadas

- Utilice [AWS CloudShell](#), una CLI basada en navegador, para ejecutar comandos. [Más información](#)
- Utilice la [AWS CLI V2](#) y habilite la autenticación a través de un usuario en el Centro de identidades de IAM. [Más información](#)

Confirmación

Entiendo la recomendación anterior y deseo proceder a la creación de una clave de acceso.

Cancelar | Siguiente

We write down the **secret key** and **access key** provided

Clave de acceso

Si pierde u olvida la clave de acceso secreta, no podrá recuperarla. En su lugar, cree una nueva clave de acceso y deje inactiva la antigua.

Clave de acceso

Clave de acceso secreta



[Mostrar](#)

2. Install Terraform/ Update Terraform

Since we count this scenario as a new set up, we should not have **terraform** installed.

We check terraform instalation status.

```
fpedrazav02gmai@ip-172-31-27-37:~/Desktop$ terraform --version
Terraform v1.1.6
on linux_amd64
```

```
Your version of Terraform is out of date! The latest version
is 1.5.6. You can update by downloading from https://www.terraform.io/downloads.
html
```

We can use a simple bash script as root to install and update our terraform instalation provided the user has the correct perms in the sudoers file.

```
 wget https://releases.hashicorp.com/terraform/1.5.6/terraform_1.5.6_linux_386.zip
 unzip terraform_1.5.6_linux_386.zip
 mv terraform /usr/local/bin
```

Check installation.

```
fpedrazav02gmai@ip-172-31-27-37:~/Desktop$ terraform --version
Terraform v1.5.6
on linux_386
```

3. Gather AWS needed data.

We connect to AWS and gather the next data.

Ami ID: ami-051f7e7f6c2f40dc1

Region: N.Virginia

Instance type: t2.micro

Private Key: -

Access Key: -

4. Write Terraform

Since we have all the data available, we can now start coding our **terraform** code.

1) Initialize Terraform and download cloud providers dependencies

Prior to crafting a whole script we can add a simple *providers* block to download any cloud provider module needed.

```
provider "aws" {
  region      = "us-east-1"
  access_key  = "my-access-key"
  secret_key  = "my-secret-key"
}
```

Then we run `terraform init` to start and download this said modules.

```
fpedrazav02gmai@ip-172-31-27-37:~/Desktop/terraform$ terraform init
```

Initializing the backend...

Initializing provider plugins...

- Finding latest version of hashicorp/aws...
- Installing hashicorp/aws v5.15.0...
- Installed hashicorp/aws v5.15.0 (signed by HashiCorp)

Terraform has created a lock file `.terraform.lock.hcl` to record the provider selections it made above. Include this file in your version control repository so that Terraform can guarantee to make the same selections by default when you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work.

If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary.

We can then write our security rules as well as inbound and outbound traffic in our code.

```
resource "aws_security_group" "SecurityFwRules" {
  name        = "allow_tls"
  description = "Allow TLS inbound traffic"
}

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

```
ingress {
    description = "Http connection from VPC"
    from_port   = 8080
    to_port     = 8080
    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"]
}
```

With this simple code we are allowing both *ssh* on port 22 to connect to the VM (ingress) and the traffic to the port 8080 via *http*. Which we will use to host Jenkins web portal.

Also, we allow outbound traffic to any IP via any protocol.

After this we can generate a new key pair. In the portal, if launch the **key pairs** section, and generate one with the next settings.

Create key pair

Key pair name

Key pairs allow you to connect to your instance securely.

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

 RSA

RSA encrypted private and public key pair

 ED25519

ED25519 encrypted private and public key pair

Private key file format

 .pem

For use with OpenSSH

 .ppk

For use with PuTTY



When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. [Learn more](#)

[Cancel](#)[Create key pair](#)

Next, we add the `aws_instance` resource block in order to connect the key and what type of instance we want to create.

```
resource "aws_instance" "AWS-TERRA-LAUNCH" {
  ami           = "ami-051f7e7f6c2f40dc1"
  instance_type = "t2.micro"
  tags = {
    Name = "Project1"
  }
  key_name = "fpedrazavNewKPair"
}
```

In this block we can add the tag `user_data` which lets us execute commands on the created machine. We will use this to insert a bash script to install the required dependencies.

```
user_data = <<-EOF
```

```

#!/bin/bash

sudo yum install git -y

sudo amazon-linux-extras install java-openjdk11 -y

sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo

sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io-2023.key

sudo yum install jenkins -y

sudo systemctl start jenkins

sudo apt install python3

EOF

```

Finally, with the resource `aws_network_interface_sg_attachment` we match our security group with our instance.

```

resource "aws_network_interface_sg_attachment" "sg_attachment1" {
    security_group_id = aws_security_group.SecurityFwRules.id
    network_interface_id = aws_instance.AWS-TERRA-LAUNCH.primary_network_interface_id
}

```

Once we finish our `terraform` main file we can apply it via CLI with `terraform apply`.

```

Plan: 3 to add, 0 to change, 0 to destroy.

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

aws_instance.AWS-TERRA-LAUNCH: Creating...
aws_security_group.SecurityFwRules: Creating...
aws_security_group.SecurityFwRules: Creation complete after 2s [id=sg-00305623a8bab3faf]
aws_instance.AWS-TERRA-LAUNCH: Still creating... [10s elapsed]
aws_instance.AWS-TERRA-LAUNCH: Still creating... [20s elapsed]
aws_instance.AWS-TERRA-LAUNCH: Still creating... [30s elapsed]
aws_instance.AWS-TERRA-LAUNCH: Creation complete after 32s [id=i-0e77caaceclea23a9]
aws_network_interface_sg_attachment.sg_attachment1: Creating...
aws_network_interface_sg_attachment.sg_attachment1: Creation complete after 1s [id=sg-00305623a8bab3faf_eni-066039a3ebcc024f1]

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

```

We see how our 3 resources are created as well as the interfaces attached.

If we check on AWS.

We can see Project1 running.

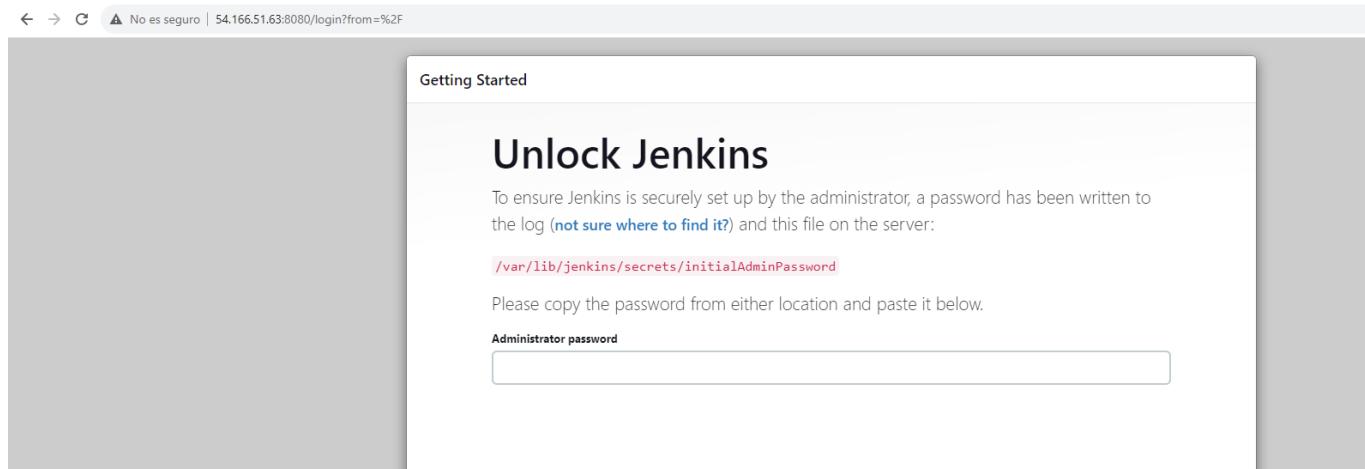
	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
	Project1	i-0e77caaceclea23a9	Running	t2.micro	2/2 checks passed	No alarms	+ us-east-1b

If we take a peek inside to check if everything is installed we can see it has all dependencies

```
https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-172-31-42-16 ~]$ git --version
git version 2.40.1
[ec2-user@ip-172-31-42-16 ~]$ python --version
Python 2.7.18
[ec2-user@ip-172-31-42-16 ~]$ systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
  Loaded: loaded (/usr/lib/systemd/system/jenkins.service; disabled; vendor preset: disabled)
    Active: activating (start) since Fri 2023-09-01 22:26:32 UTC; 48s ago
      Main PID: 4241 (java)
        CGroup: /system.slice/jenkins.service
           └─4241 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=%C/jenkins/war --httpPort=8080

Sep 01 22:26:49 ip-172-31-42-16.ec2.internal jenkins[4241]: ****
Sep 01 22:26:49 ip-172-31-42-16.ec2.internal jenkins[4241]: ****
Sep 01 22:26:49 ip-172-31-42-16.ec2.internal jenkins[4241]: **** Jenkins initial setup is required. An admin user has been created and a password generated.
Sep 01 22:26:49 ip-172-31-42-16.ec2.internal jenkins[4241]: Please use the following password to proceed to installation:
Sep 01 22:26:49 ip-172-31-42-16.ec2.internal jenkins[4241]: 2acacd2d96bd044df9c6ba5e6fe80bec
Sep 01 22:26:49 ip-172-31-42-16.ec2.internal jenkins[4241]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Sep 01 22:26:49 ip-172-31-42-16.ec2.internal jenkins[4241]: ****
[ec2-user@ip-172-31-42-16 ~]$ java --version
openjdk 11.0.20 2023-07-18 LTS
OpenJDK Runtime Environment (Red Hat 11.0.20+0.8-1.amzn2.0.1) (build 11.0.20+8-LTS)
OpenJDK 64-Bit Server VM (Red Hat 11.0.20+0.8-1.amzn2.0.1) (build 11.0.20+8-LTS, mixed mode, sharing)
[ec2-user@ip-172-31-42-16 ~]$
```

We can also see Jenkins web portal is installed and running in port 8080.



Finally the Instance IP settings are:

Details	Security	Networking	Storage	Status checks	Monitoring	Tags
Instance: i-01f8c9214f1561ecc (Project1)						
Details Security Networking Storage Status checks Monitoring Tags						
Instance summary Info		Public IPv4 address 54.166.51.63 open address			Private IPv4 addresses 172.31.42.16	
Instance ID i-01f8c9214f1561ecc (Project1)		Instance state Running			Public IPv4 DNS ec2-54-166-51-63.compute-1.amazonaws.com open address	
IPv6 address -		Private IP DNS name (IPv4 only) ip-172-31-42-16.ec2.internal				
Hostname type IP name: ip-172-31-42-16.ec2.internal						

[Check on GitHub](#)