



Módulo 6: Tecnología cloud y DevOps

Infraestructura como código (IaC)

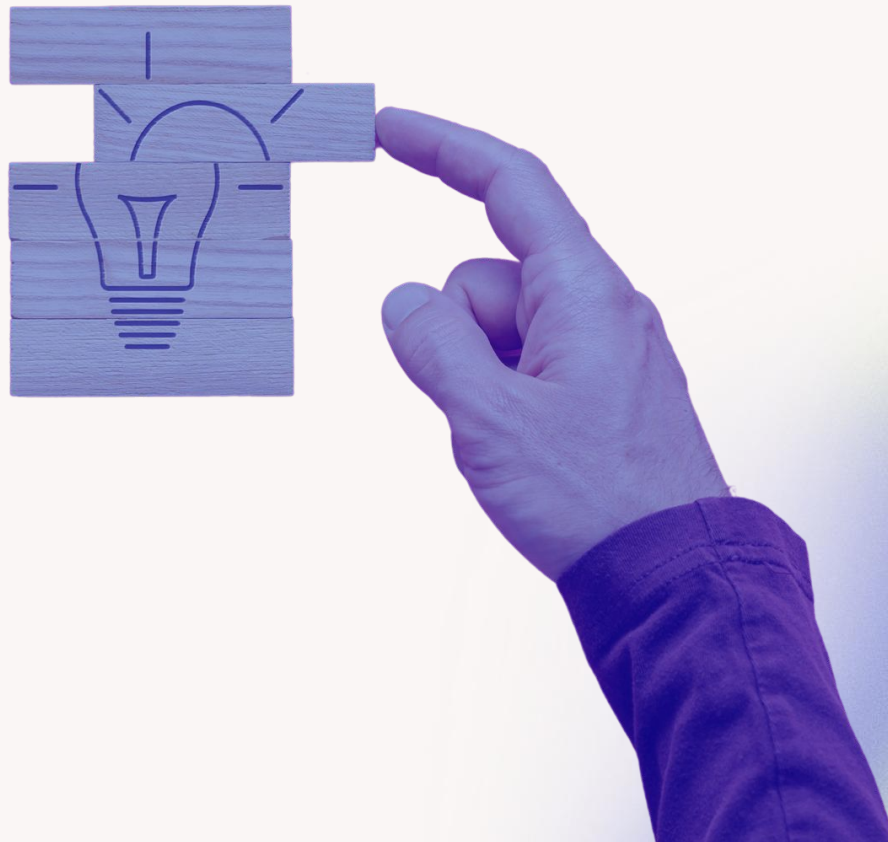


MÓDULO 6

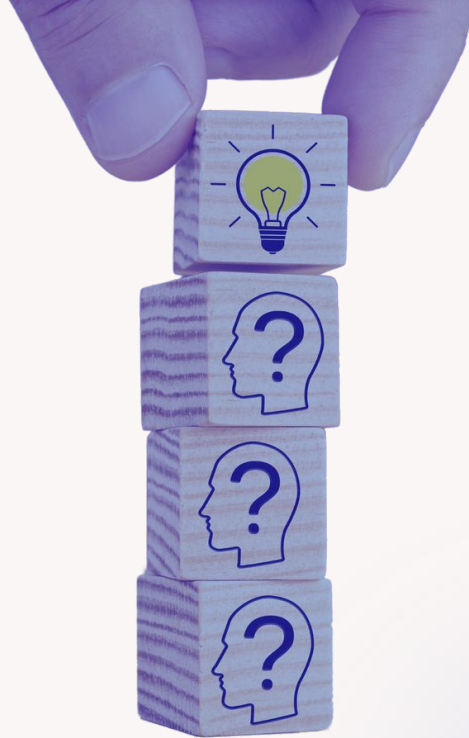
- Introducción a la computación en la nube
- Almacenamiento en cloud
- Servicios de cómputo en la nube
- Servicios de red en la nube
- Infraestructura como código (IaC)
- Servicios cloud de CI y CD
- Pipelines con AWS

Objetivos

Comprender los principios de Infraestructura como Código (IaC) y aprender a implementar soluciones con herramientas como Ansible, Terraform y CloudFormation para automatizar y gestionar infraestructuras en entornos DevOps.



¿Qué limitaciones presentan los métodos tradicionales de gestión de infraestructura?



Introducción a laC



Introducción a la Infraestructura como Código (IaC)

La Infraestructura como Código (IaC) es una metodología que permite definir y gestionar la infraestructura de TI (servidores, redes, bases de datos, etc.) mediante archivos de código legibles y versionables.

¿Qué es IaC?

IaC consiste en describir la infraestructura necesaria para un sistema mediante archivos de configuración, lo que permite desplegarla automáticamente sin intervención manual.

Principios fundamentales de IaC

- **Idempotencia:** Ejecutar el mismo script múltiples veces produce siempre el mismo estado del sistema.
- **Repetibilidad:** La infraestructura puede replicarse exactamente en distintos entornos (dev, staging, prod).
- **Versionamiento:** Toda infraestructura se puede rastrear, revisar y revertir con herramientas de control de versiones.
- **Automatización:** Se eliminan tareas manuales propensas a errores.
- **Declaratividad:** Se define *qué* se desea, no *cómo* llegar a ello (aunque algunas herramientas son imperativas).

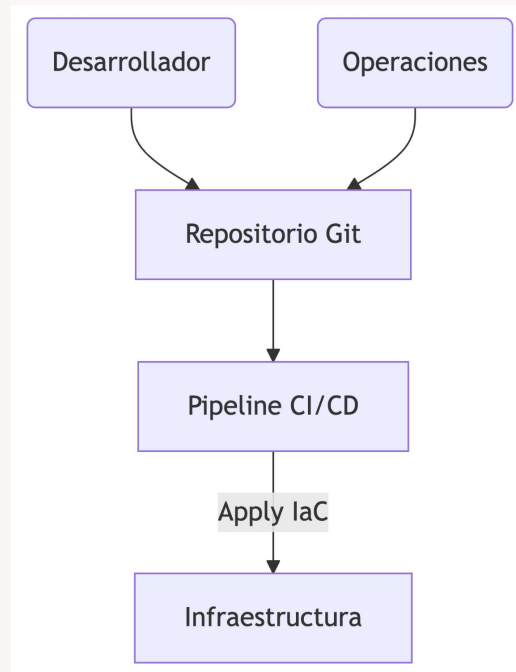
Comparación con métodos tradicionales

Característica	Gestión Manual	IaC
Documentación	Separada	Es el código mismo
Escalabilidad	Manual, lenta	Automática y rápida
Trazabilidad	Limitada	Control de versiones
Repetibilidad	Inconsistente	Garantizada
Tiempo de despliegue	Alto	Bajo

Importancia de IaC en DevOps

En DevOps, donde los despliegues son constantes, rápidos y automatizados, IaC permite:

- Automatizar entornos desde cero
- Integrar infraestructura en pipelines CI/CD
- Mejorar la colaboración entre Dev y Ops
- Garantizar coherencia entre ambientes



Beneficios y desafíos de IaC

Beneficios

- Infraestructura reproducible y consistente
- Reducción de errores humanos
- Ahorro de tiempo y costos
- Mejora en la trazabilidad y auditoría
- Escalabilidad automatizada

Desafíos

- Curva de aprendizaje de herramientas (Terraform, Ansible...)
- Complejidad en el manejo de estados
- Necesidad de controlar acceso seguro al código de IaC
- Pruebas automatizadas de infra

Ejercicio orientador: Nginx manual vs automatizado

📌 Comparación inicial de configuración de un servidor Nginx:

Manual (SSH):

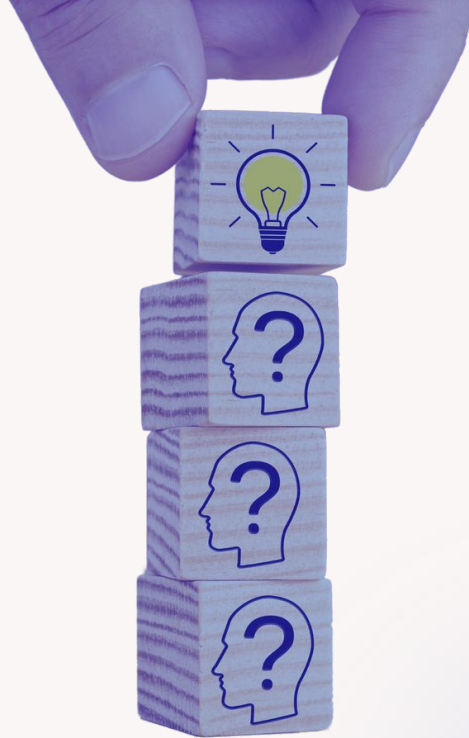
```
ssh usuario@ip-servidor
sudo apt update
sudo apt install nginx -y
sudo systemctl start nginx
```

Automatizado (con Ansible, más adelante):

```
- name: Instalar Nginx automáticamente
  hosts: webserver
  become: yes
  tasks:
    - name: Instala Nginx
      apt:
        name: nginx
        state: present
```

👉 Esto prepara el terreno para introducir Ansible y Terraform como casos prácticos.

¿Cuáles son los beneficios clave de aplicar Infraestructura como Código (IaC) en un entorno DevOps y qué desafíos podrían surgir al implementarla por primera vez en una organización?



Herramientas para implementar laC

Herramientas para implementar IaC

Las herramientas de Infraestructura como Código permiten definir, aprovisionar y administrar infraestructura a través de archivos de configuración automatizados. Entre las más populares están:

- **Ansible** (basado en YAML, agente-less)
- **Terraform** (declarativo, multi-cloud)
- **CloudFormation** (propietario de AWS)

Ansible

Ansible es una herramienta de automatización simple y poderosa, basada en archivos YAML, que no requiere agentes en los nodos gestionados.

✓ Ventajas:

- No necesita software adicional en los servidores (usa SSH)
- Simple de aprender (usa YAML)
- Reutilizable en múltiples entornos

📌 Ejemplo básico – Instalar Nginx en un servidor remoto:

```
# nginx.yml
- name: Instalación de Nginx
  hosts: servidores_web
  become: yes
  tasks:
    - name: Instalar Nginx
      apt:
        name: nginx
        state: present
```

🔧 Ejecución:

```
ansible-playbook -i hosts nginx.yml
```

Terraform

Terraform es una herramienta declarativa para crear y administrar infraestructura en múltiples proveedores cloud (AWS, GCP, Azure...).

✓ Ventajas:

- Soporte multi-cloud
- Mantiene un estado del entorno
- Gran comunidad y módulos reutilizables

📌 Ejemplo – Crear una instancia EC2 en AWS:

```
provider "aws" {  
    region = "us-east-1"  
}  
  
resource "aws_instance" "mi_instancia" {  
    ami           = "ami-0c55b159cbfafa1f0"  
    instance_type = "t2.micro"  
}
```

🔧 Comandos básicos:

```
terraform init  
terraform plan  
terraform apply
```


CloudFormation

CloudFormation es el servicio de AWS para definir infraestructura en plantillas JSON o YAML, integrándose nativamente con todos sus servicios.

✓ Ventajas:

- Totalmente integrado con AWS
- Control de dependencias y actualizaciones
- Compatible con la consola AWS y APIs

📌 Ejemplo – Crear una instancia EC2:

Resources:

EC2Instance:

Type: AWS::EC2::Instance

Properties:

InstanceType: t2.micro

ImageId: ami-0c55b159cbfafe1f0

🔧 Despliegue:

```
aws cloudformation deploy --template-file  
plantilla.yml --stack-name mi-stack
```

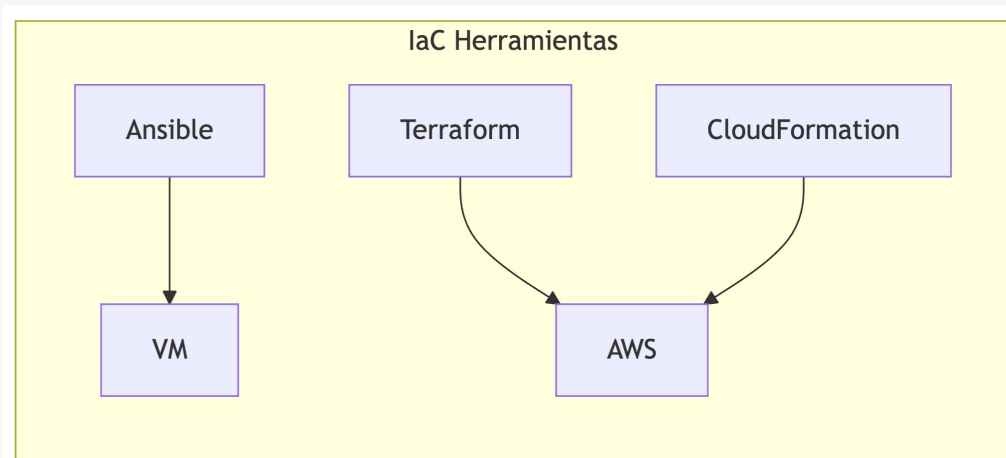
Comparativo entre herramientas IaC

Característica	Ansible	Terraform
Lenguaje	YAML	HCL
Multi-cloud	Sí	Sí
Declarativo	Parcialmente	Sí
Imperativo	Sí	No
Control de estado	No	Sí
Requiere agentes	No	No
Curva de aprendizaje	Baja	Media

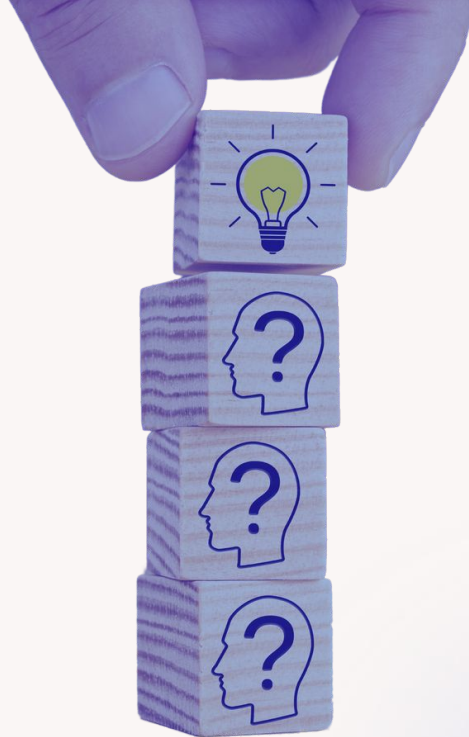
Ejercicio orientador

🎯 Objetivo: Comparar tres herramientas creando un mismo recurso: servidor Nginx

1. **Con Ansible:** Automatizar la instalación de Nginx en una máquina local/VM.
2. **Con Terraform:** Crear una instancia EC2 en AWS.
3. **Con CloudFormation:** Desplegar una plantilla con la misma funcionalidad.



¿Cuáles son los criterios más relevantes que se deben considerar al elegir entre Ansible, Terraform y CloudFormation?





aspasia
LA FORMACIÓN DE TU FUTURO

Ansible

¿Qué es Ansible y para qué se utiliza?

Ansible es una herramienta de automatización que permite gestionar configuraciones, despliegues y aprovisionamiento de infraestructura a través de **playbooks en YAML**, sin necesidad de instalar agentes en los sistemas remotos.

Usos comunes:

- Automatización de instalación de software
- Configuración de servidores
- Despliegue de aplicaciones
- Orquestación de flujos CI/CD

✓ Ventajas principales:

- Basado en SSH, sin agentes
- Fácil de leer y escribir (YAML)
- Idempotente: aplica solo los cambios necesarios
- Ampliamente usado en DevOps

Instalación y configuración de Ansible



Requisitos:

- Python \geq 3.6
- Acceso a terminal (Linux/macOS recomendado)
- Acceso SSH a máquinas remotas (opcional para pruebas locales)



Instalación en distintos entornos:

Linux / macOS (recomendado):

```
pip install ansible
```

Ubuntu (vía APT):

```
sudo apt update  
sudo apt install ansible -y
```

Windows: Usar **WSL (Ubuntu en Windows)** o una máquina virtual

Instalación y configuración de Ansible

 Verificación:

```
ansible --version
```

 **Estructura de carpetas sugerida para un proyecto:**

```
proyecto-ansible/  
├── inventory.ini  
├── nginx.yml  
└── roles/ (opcional)
```


Desarrollo e implementación de un plan de trabajo con Ansible


Objetivo del ejercicio:

Automatizar la instalación de **Nginx** en una máquina local o remota usando Ansible.

1. Crear archivo de inventario

 `inventory.ini`

```
[web]
localhost ansible_connection=local
```

 Puedes cambiar `localhost` por IPs si usas servidores remotos.

2. Crear el playbook

 `nginx.yml`

```
- name: Instalar y habilitar Nginx
  hosts: web
  become: yes
  tasks:
    - name: Instalar Nginx
      apt:
        name: nginx
        state: present
        update_cache: yes

    - name: Asegurar que Nginx esté activo
      service:
        name: nginx
        state: started
        enabled: yes
```

Desarrollo e implementación de un plan de trabajo con Ansible

3. Ejecutar el playbook

```
ansible-playbook -i inventory.ini nginx.yml
```



Resultado esperado:

- Nginx instalado
- Servicio activo y configurado para iniciar automáticamente

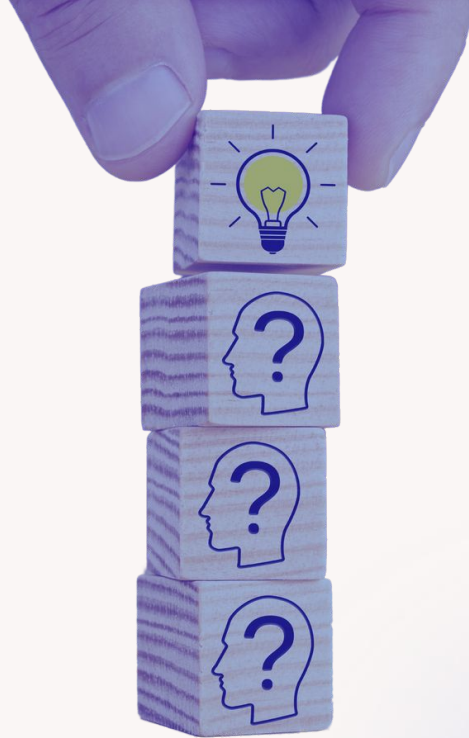
4. Verificar en el navegador

Abre:

```
http://localhost
```

Deberías ver la página por defecto de Nginx.

¿Qué ventajas ofrece
Ansible para la
automatización de
infraestructura?





aspasia
LA FORMACIÓN DE TU FUTURO

ITerraform

¿Qué es Terraform y para qué se utiliza?

Terraform es una herramienta de **Infraestructura como Código (IaC)** desarrollada por **HashiCorp**, que permite aprovisionar, gestionar y versionar infraestructura de manera declarativa.



Casos de uso:

- Desplegar infraestructura en la nube (AWS, Azure, GCP)
- Crear redes, instancias, bases de datos, balanceadores, etc.
- Automatizar entornos de desarrollo, staging y producción



Ventajas:

- Independencia de proveedor
- Control de versiones de infraestructura
- Reutilización mediante módulos
- Fácil integración con pipelines CI/CD

Instalación y configuración de Terraform



Requisitos:

- Sistema Linux/macOS/Windows
- Terraform \geq 1.x
- Acceso a un proveedor cloud (usaremos AWS en el ejercicio)



Instalación:

Linux/macOS (con Homebrew):

```
brew tap hashicorp/tap  
brew install hashicorp/tap/terraform
```

Manual (todos los SO): Descargar desde:


<https://www.terraform.io/downloads>

Extraer y mover el binario a `/usr/local/bin`

Verificación:

```
terraform version
```

Desarrollo e implementación de un plan de trabajo con Terraform

 Objetivo:

Crear una instancia EC2 en AWS usando Terraform (modo básico)

1. Crear un directorio de proyecto

```
mkdir terraform-aws-ec2 && cd terraform-aws-ec2
```

2. Crear archivo de configuración principal

 main.tf

```
provider "aws" {  
    region = "us-east-1"  
}  
  
resource "aws_instance" "ejemplo" {  
    ami           = "ami-0c94855ba95c71c99" #  
Ubuntu 18.04  
    instance_type = "t2.micro"  
    tags = {  
        Name = "mi-instancia-terraform"  
    }  
}
```


Desarrollo e implementación de un plan de trabajo con Terraform

Notas:

- Cambia el AMI según tu región
- Requiere credenciales de AWS configuradas (~/.aws/credentials)


3. Inicializar Terraform

```
terraform init
```

 Esto descarga los plugins necesarios del proveedor (en este caso, AWS).


4. Validar el plan de ejecución

```
terraform plan
```

 Verás los cambios que se aplicarán (infraestructura a crear).

5. Aplicar el plan

```
terraform apply
```

 Confirmar con **yes**.

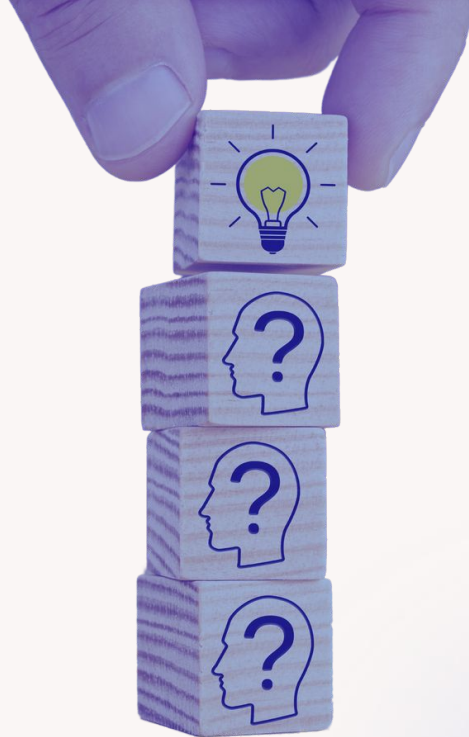
6. Verificar creación de recursos

Puedes ir a la consola de AWS → EC2 → Instancias y validar que está corriendo.

Ejecución de comandos Terraform

Comando	Descripción
terraform init	Inicializa un proyecto
terraform plan	Muestra los cambios que se aplicarán
terraform apply	Aplica los cambios a la infraestructura
terraform destroy	Elimina la infraestructura provisionada
terraform validate	Verifica errores de sintaxis
terraform fmt	Formatea archivos de configuración

¿Cómo facilita
Terraform la gestión
declarativa de
infraestructura en
entornos cloud?



CloudFormation



¿Qué es CloudFormation y para qué se utiliza?

AWS CloudFormation es un servicio de **Infraestructura como Código (IaC)** que permite describir y aprovisionar todos los recursos de AWS necesarios en un entorno de forma automática y reproducible mediante archivos en **YAML o JSON**.

Usos principales:

- Automatizar el despliegue de infraestructura
- Crear entornos consistentes para desarrollo, QA y producción
- Versionar infraestructura como código
- Reutilizar plantillas en múltiples cuentas o regiones

Configuración de CloudFormation



Requisitos:

- Tener una cuenta de AWS activa
- Configurar el CLI de AWS:

```
aws configure
```




Acceso:

Puedes usar:

- La **Consola de AWS** (GUI)
- La **CLI de AWS**
- AWS SDKs desde otros lenguajes

Desarrollo e implementación de un plan de trabajo con CloudFormation

 **Objetivo:**

Desplegar una instancia EC2 con CloudFormation usando una plantilla YAML.

 **Archivo plantilla:**
`ec2-instance.yaml`

```
AWSTemplateFormatVersion: '2010-09-09'  
Description: Crear una instancia EC2  
simple con CloudFormation
```

Resources:

 MiInstanciaEC2:

 Type: AWS::EC2::Instance

 Properties:

 InstanceType: t2.micro

 ImageId: ami-0c94855ba95c71c99 #

Ubuntu 18.04 LTS (ajustar a tu región)

 Tags:

 - Key: Name

 Value: InstanciaCF

Desarrollo e implementación de un plan de trabajo con CloudFormation



Crear el stack desde la CLI

```
aws cloudformation create-stack \  
  --stack-name MiStackEC2 \  
  --template-body file://ec2-instance.yaml \  
  --capabilities CAPABILITY_NAMED_IAM
```



Para eliminar el stack:

```
aws cloudformation delete-stack --stack-name  
MiStackEC2
```

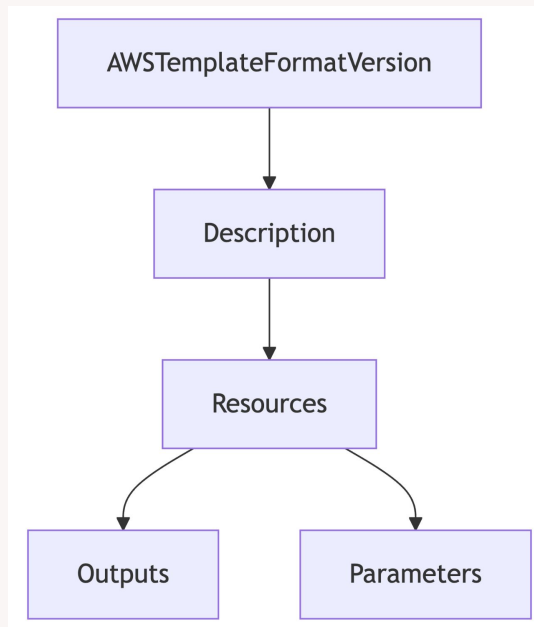


Para ver el estado del stack:

```
aws cloudformation describe-stacks  
--stack-name MiStackEC2
```

Estructura de una plantilla CloudFormation

- **Resources:** Obligatorio. Define lo que se desplegará (EC2, VPC, RDS, etc.)
- **Parameters:** Para entrada dinámica desde el usuario
- **Outputs:** Información devuelta por el stack (como IP o ID)
- **Mappings / Conditions / Metadata:** Opcionales y avanzados



Casos prácticos y demostraciones

Ejemplo 1: Stack de prueba desde consola


1. Ir a **AWS CloudFormation**
2. Crear un stack con nueva plantilla
3. Cargar archivo YAML ([ec2-instance.yaml](#))
4. Revisar parámetros y desplegar
5. Ver logs en "Eventos" y recursos en EC2

Ejemplo 2: Stack con parámetros personalizables

[ec2-parametrizado.yaml](#)

```
Parameters:
  InstanceType:
    Type: String
    Default: t2.micro

Resources:
  MiInstanciaEC2:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: !Ref InstanceType
      ImageId: ami-0c94855ba95c71c99
```

 Puedes cambiar el tipo de instancia en el despliegue.

Buenas prácticas con CloudFormation

- ✓ Versionar las plantillas con Git
- ✓ Usar plantillas parametrizadas y reutilizables
- ✓ Validar antes de aplicar:

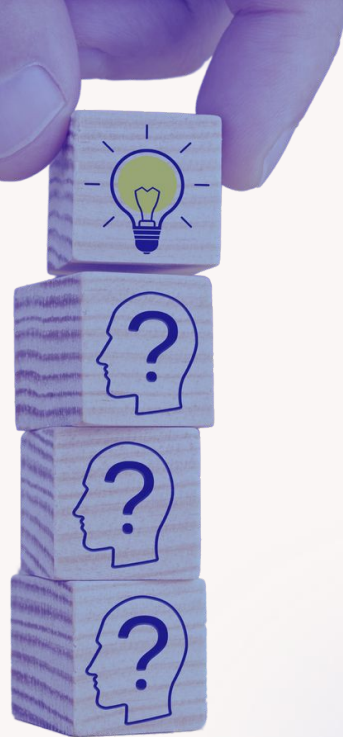
```
aws cloudformation validate-template  
--template-body file://ec2-instance.yaml
```

- ✓ Usar **StackSets** para desplegar en múltiples cuentas/regiones
- ✓ Aprovechar **Outputs** para integración con otros stacks



LA FORMACIÓN DE TU FUTURO

¿Qué ventajas ofrece
CloudFormation al
gestionar infraestructura
en AWS y cómo se
diferencia de otras
herramientas como
Terraform o Ansible?





Ejercicio Guiado: Provisión de Infraestructura en AWS con Terraform



Ejercicio Guiado: Provisión de Infraestructura en AWS con Terraform

Terraform es una herramienta declarativa de Infraestructura como Código (IaC) que permite definir, implementar y gestionar recursos de infraestructura en múltiples proveedores cloud, incluyendo AWS.

En este ejercicio aprenderás a usar Terraform para desplegar una VPC personalizada, una subred pública, un grupo de seguridad y una instancia EC2.

Objetivos

- Comprender los fundamentos de Terraform.
- Crear una infraestructura básica en AWS usando archivos `.tf`.
- Ejecutar Terraform para planificar, aplicar y destruir recursos.
- Identificar los beneficios de IaC en el ciclo DevOps.





Paso 1: Instalación de Terraform

Linux / macOS

```
# Linux
sudo apt update && sudo apt install -y unzip
wget https://releases.hashicorp.com/terraform/1.6.6/terraform_1.6.6_linux_amd64.zip
unzip terraform_1.6.6_linux_amd64.zip
sudo mv terraform /usr/local/bin/
terraform -v
```

Windows

Descargar desde: <https://developer.hashicorp.com/terraform/downloads>

Agregar la ruta del ejecutable a tu PATH.



Paso 2: Crear estructura de archivos

```
mkdir terraform-iac && cd terraform-iac  
touch main.tf variables.tf outputs.tf
```




Paso 3: Definir los recursos en Terraform

main.tf

```
provider "aws" {  
  region = "us-east-1"  
}  
  
resource "aws_vpc" "dev_vpc" {  
  cidr_block = "10.0.0.0/16"  
}  
  
resource "aws_subnet" "public_subnet" {  
  vpc_id      = aws_vpc.dev_vpc.id  
  cidr_block = "10.0.1.0/24"  
}  
  
resource "aws_internet_gateway" "gw" {  
  vpc_id = aws_vpc.dev_vpc.id  
}  
  
resource "aws_route_table" "public_rt" {  
  vpc_id = aws_vpc.dev_vpc.id  
  route {  
    cidr_block = "0.0.0.0/0"  
    gateway_id = aws_internet_gateway.gw.id  
  }  
}
```

```
resource "aws_route_table_association" "a" {  
  subnet_id      = aws_subnet.public_subnet.id  
  route_table_id = aws_route_table.public_rt.id  
}  
  
resource "aws_security_group" "dev_sg" {  
  name        = "dev_sg"  
  description = "Allow SSH and HTTP"  
  vpc_id      = aws_vpc.dev_vpc.id  
  ingress {  
    from_port = 22  
    to_port   = 22  
    protocol  = "tcp"  
    cidr_blocks = ["0.0.0.0/0"]  
  }  
  ingress {  
    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"]  
  }  
}
```

```
resource "aws_instance" "dev_ec2" {  
  ami           = "ami-0c101f26f147fa7fd" # Amazon Linux 2 (us-east-1)  
  instance_type = "t2.micro"  
  subnet_id     = aws_subnet.public_subnet.id  
  security_groups = [aws_security_group.dev_sg.name]  
  key_name      = var.key_name  
  tags = {  
    Name = "DevInstance"  
  }  
}
```



Paso 3: Definir los recursos en Terraform

variables.tf

```
variable "key_name" {  
    description = "Nombre de la clave SSH"  
    type        = string  
}
```

outputs.tf

```
output "public_ip" {  
    description = "IP pública de la  
    instancia"  
    value       =  
    aws_instance.dev_ec2.public_ip  
}
```



Paso 4: Inicializar y aplicar el proyecto

1. Inicializar Terraform:

```
terraform init
```

2. Verificar plan de ejecución:

```
terraform plan -var="key_name=clave-ec2"
```

3. Aplicar configuración:

```
terraform apply -var="key_name=clave-ec2"  
-auto-approve
```

4. Obtener la IP pública de la instancia:

```
terraform output
```

5. Conectarse vía SSH:

```
ssh -i clave-ec2.pem ec2-user@<ip_publica>
```



Paso 5: Destrucción de recursos

```
terraform destroy -var="key_name=clave-ec2" -auto-approve
```

Preguntas finales

- ¿Qué ventajas encuentras en definir tu infraestructura como código?
- ¿Qué desafíos enfrentaste usando Terraform?
- ¿Cómo versionarías este proyecto para un equipo de trabajo?

Entregable:

- Screenshot del **terraform apply** con recursos creados.
- Archivo **main.tf** con tu infraestructura personalizada.
- Dirección IP pública funcionando correctamente.
- Captura de conexión SSH a la instancia.



Resumen de lo aprendido

- **Fundamentos de IaC:** Automatiza la provisión de infraestructura usando código, promoviendo repetibilidad, versionamiento y reducción de errores.
- **Herramientas de IaC:** Comparativa de Ansible, Terraform y CloudFormation, cada una con enfoques distintos según el proveedor y tipo de automatización.
- **Implementación Práctica:** Desarrollo de planes de trabajo para desplegar recursos con Terraform y CloudFormation, y gestión de configuraciones.
- **Ventajas en DevOps:** IaC permite integración continua, consistencia en entornos, despliegue rápido y control centralizado, a pesar de desafíos.

Próxima clase...

Servicios cloud de CI y CD

