TIC

TALENTO TECH

Misión 3

ARQUITECTURA EN LA NUBE

Innovador

# Tema 1: Infraestructure as Code

iTraining Escenarios de innovación y tecnología | UNIVERSIDAD DE ANTIOQUIA | Universidad de Caldas | ubicua Technology

*Campista, llegó el momento de retar tus conocimientos y que los pongas aprueba a través de los diferentes recursos que encontraras en este espacio como son: conceptos, ejemplos, herramientas, actividades prácticas y retos, los cuales te ayudaran alcanzar los objetivos trazados en el nivel innovador.*

# Infrastructure as Code

Infrastructure as Code (IaC) es un enfoque clave en la gestión moderna de infraestructuras en la nube. En esta lección, aprenderás los conceptos básicos de IaC, explorarás sus ventajas y desventajas, y te familiarizarás con las herramientas principales utilizadas en la industria, como AWS CloudFormation y Terraform.

## 1. Concepto Básico de IaC

**Definición:**

IaC se refiere a la práctica de gestionar y aprovisionar infraestructuras a través de código, en lugar de realizar configuraciones manuales. Esto se logra mediante scripts que definen el estado deseado de la infraestructura, permitiendo automatizar la creación, actualización y eliminación de recursos.

**Importancia:**

IaC es fundamental para asegurar la consistencia, la reproducibilidad y el control de versiones en entornos de TI complejos, especialmente en la nube.

## 2. Desventajas de IaC

### Reproducibilidad

Permite replicar configuraciones de infraestructura de manera exacta en diferentes entornos, como desarrollo, pruebas y producción.

### Control de versiones

Facilita el seguimiento de cambios en la infraestructura a través de sistemas de control de versiones como Git, lo que permite revertir a estados anteriores en caso de errores.

### Escalabilidad

IaC permite gestionar grandes infraestructuras de manera eficiente, automatizando procesos que serían complejos y propensos a errores si se hicieran manualmente.

### Automatización

Reduce la posibilidad de errores humanos al automatizar tareas repetitivas y complejas.

## 3. Ventajas de IaC

### Curva de Aprendizaje

Implementar IaC requiere conocimientos técnicos avanzados y una comprensión profunda de las herramientas y lenguajes de configuración utilizados.

### Complejidad en la Gestión de Cambios

A medida que la infraestructura crece, la gestión de cambios y actualizaciones puede volverse más complicada, especialmente en entornos con múltiples dependencias.

### Dependencia de herramientas

Infrastructure as Code permite gestionar y provisionar la infraestructura de manera automatizada y eficiente. Al tratar la infraestructura como código, se simplifican las

tareas repetitivas, se mejora la coherencia en la configuración y se facilita la colaboración, lo que optimiza la administración de entornos en la nube.

## 4. Herramientas Principales de IaC

En esta sección, exploramos dos de las herramientas más populares para implementar IaC:

### AWS CloudFormation

**Descripción:**
AWS CloudFormation es un servicio de Amazon Web Services (AWS) que permite modelar y configurar los recursos de AWS de manera automática a través de templates definidos en JSON o YAML.

**Funcionalidad:**
CloudFormation te permite aprovisionar y gestionar recursos de AWS como instancias EC2, buckets S3, y VPCs, asegurando que todos los recursos se creen y configuren de acuerdo con un estado definido en un template.

**Casos de Uso:**
Ideal para organizaciones que están profundamente integradas con AWS y necesitan automatizar la creación de infraestructuras dentro de ese ecosistema.

### Terraform

**Descripción:**
Terraform es una herramienta de código abierto desarrollada por HashiCorp, que permite crear, modificar y versionar la infraestructura de manera eficiente.

**Funcionalidad:**
A diferencia de CloudFormation, Terraform no se limita a un solo proveedor de nube; es compatible con múltiples proveedores, como AWS, Azure, Google Cloud, y más. Utiliza su propio lenguaje de configuración, HCL (HashiCorp Configuration Language).

**Casos de Uso:**

Ideal para entornos multicloud o híbridos, donde se necesita gestionar infraestructuras en múltiples plataformas.

## 5. Herramientas Principales de IaC


**Alcance:**

CloudFormation es específico para AWS, mientras que Terraform soporta múltiples proveedores de nube.

**Lenguaje:**

CloudFormation utiliza JSON o YAML, mientras que Terraform usa HCL, un lenguaje de configuración más simple y declarativo.

**Flexibilidad:**

Terraform ofrece mayor flexibilidad en entornos multicloud, permitiendo gestionar recursos en diferentes proveedores desde un único archivo de configuración.

**Integración:**

CloudFormation está profundamente integrado con los servicios específicos de AWS, lo que permite automatizar configuraciones muy detalladas en ese entorno.

## 6. Resumen de la Lección

- **IaC Automatiza la Infraestructura:** Permite la gestión eficiente y reproducible de infraestructuras a través de código.
- **Herramientas Clave:** AWS CloudFormation y Terraform son dos herramientas principales que facilitan la implementación de IaC, cada una con sus ventajas y limitaciones.
- **Elección de Herramienta:** La selección entre CloudFormation y Terraform depende del entorno de despliegue y de las necesidades específicas del proyecto.


*Campista al finalizar esta lección, serás capaz de entender los conceptos básicos de AWS CloudFormation, cómo se estructura una plantilla, el uso de pseudofunciones, y cómo crear y gestionar stacks y Change Sets en AWS.*

## 1. ¿Qué es AWS CloudFormation?

AWS CloudFormation es un servicio que te permite modelar y configurar los recursos de AWS para tu aplicación utilizando Infrastructure as Code (IaC). Mediante un archivo de plantilla, que generalmente está escrito en JSON o YAML,

puedes definir todos los recursos de infraestructura que necesitas, y CloudFormation se encarga de aprovisionarlos y configurarlos automáticamente.

### Ventajas de usar AWS CloudFormation:

- **Automatización del despliegue:** Facilita la creación y gestión de toda la infraestructura en la nube de manera automática.
- **Consistencia y control de versiones:** Asegura que las configuraciones sean consistentes en diferentes entornos y permite revertir cambios fácilmente.
- **Replicación de entornos:** Simplifica la replicación de infraestructuras para diferentes fases del ciclo de vida de la aplicación, como desarrollo, pruebas y producción.
- **Reducción de errores humanos:** Minimiza el riesgo de errores manuales en la configuración de recursos, mejorando la seguridad y eficiencia.

## 2. Estructura de una Plantilla de CloudFormation

Una plantilla de CloudFormation está compuesta por varios componentes clave que definen cómo se configurarán los recursos.

Aquí están los elementos más importantes:

### a. Resources:

Es la sección más crucial de una plantilla, ya que define los recursos que se implementarán en la infraestructura, como instancias EC2, VPCs, RDS, etc.

**Ejemplo:**

```
Resources:
  MyEC2Instance:
    Type: 'AWS::EC2::Instance'
    Properties:
      InstanceType: 't2.micro'
      KeyName: 'my-key'
      ImageId: 'ami-0abcdef12345abcde'
```

**b. Parameters:**

Permiten personalizar la plantilla en el momento del despliegue, haciendo que la plantilla sea reutilizable y adaptable a diferentes entornos.

**Ejemplo:**

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

**c. Mapping**

Facilitan la personalización de la plantilla para diferentes regiones, entornos o tipos de instancias.

**Ejemplo:**

```
Mappings:
  RegionMap:
    us-east-1:
      AMI: 'ami-0abcdef12345abcde'
    us-west-2:
      AMI: 'ami-0fedcba54321edcba'
```

### d. Outputs

Proporcionan información de salida después de que el stack ha sido desplegado, como IDs de recursos o URLs.

**Ejemplo:**

```
Outputs:
  InstanceId:
    Description: "InstanceId of the newly created EC2 instance"
    Value: !Ref MyEC2Instance
```

### e. Conditions

Permiten la creación de recursos de manera condicional, basándose en parámetros o configuraciones específicas.

**Ejemplo:**

```
Conditions:
  CreateProdResources: !Equals [!Ref EnvironmentType, 'prod']
```

### f. Metadata

Almacena información adicional sobre los recursos que no es usada directamente por AWS, pero puede ser útil para documentar o gestionar la plantilla.

**Ejemplo:**

```
Metadata:
  AWS::CloudFormation::Interface:
    ParameterGroups:
      - Label:
          default: "Network Configuration"
        Parameters:
          - VPC
          - Subnet
```

## 3. Pseudofunciones en CloudFormation

Las pseudofunciones en CloudFormation te permiten realizar operaciones dinámicas en la plantilla, como obtener atributos de recursos, realizar sustituciones en cadenas, o aplicar lógica condicional.

**Algunas Pseudofunciones importantes:**

Fn::GetAtt:

Obtiene atributos de un recurso.

**Ejemplo:**

!GetAtt MyEC2Instance.PublicIp

Fn::Sub:

Permite realizar sustituciones dentro de cadenas utilizando variables.

**Ejemplo:**

!Sub 'arn:aws:s3:::${BucketName}/*'

Fn::Join:

Une una lista de valores en una sola cadena.

**Ejemplo:**

!Join ['-', ['mybucket', !Ref 'AWS::Region']]

Fn::If:

Ejecuta condiciones en tiempo de ejecución.

**Ejemplo:**

!If [CreateProdResources, !Ref 'ProdSubnet', !Ref 'DevSubnet']

Fn::ImportValue:

Importa valores exportados por otros stacks.

**Ejemplo:**

!ImportValue 'SharedSecurityGroup'

Fn::Select:

Selecciona un elemento específico de una lista.

**Ejemplo:**

!Select [1, ['apples', 'bananas', 'grapes']]

## 4. Creación y Despliegue de Stacks con CloudFormation

Un Stack en CloudFormation es un conjunto de recursos definidos en una plantilla. Para crear un stack, debes seguir estos pasos:

1. Crear un stack desde la consola de AWS:
    - Selecciona "Create Stack" y sube tu plantilla.
    - Configura los parámetros y opciones avanzadas según lo requerido.
2. Definir la infraestructura en la plantilla:
    - Asegúrate de que todos los recursos necesarios estén definidos en la plantilla.
3. Desplegar el stack:
    - Revisa la configuración y despliega el stack. CloudFormation se encargará de crear y configurar todos los recursos automáticamente.
4. Gestionar actualizaciones mediante Change Set:
    - Antes de realizar cambios en un stack existente, es recomendable crear un Change Set para previsualizar los cambios.

## 5. Profundización en Change Sets

Change Sets te permiten previsualizar cómo los cambios en tu plantilla afectarán a los recursos en un stack existente antes de aplicarlos.

### Proceso de uso de Change Sets:

**Crear un Change Set:**

CloudFormation genera un Change Set mostrando los cambios que se realizarán en los recursos existentes o los nuevos que se añadirán.

**Aplicar un Change Set:**

Una vez revisado, puedes aplicar el Change Set para actualizar los recursos.

**Actualizar recursos:**

Permite actualizar recursos sin necesidad de recrearlos desde cero, minimizando el impacto en el servicio.

# Introducción a Terraform

## 1. Introducción a Infrastructure as Code

- **Concepto de IaC:** Infrastructure as Code es un enfoque para gestionar y aprovisionar la infraestructura de TI mediante código en lugar de procesos manuales. Este método permite una mayor consistencia, repetibilidad, y automatización en la gestión de la infraestructura.
- **Importancia de IaC:** Facilita la colaboración entre equipos, reduce errores humanos, y permite la implementación rápida y consistente de entornos de desarrollo, pruebas y producción.

## 2. Qué es Terraform

- **Definición:** Terraform es una herramienta de Infrastructure as Code (IaC) desarrollada por HashiCorp. Permite a los usuarios definir y aprovisionar

infraestructura a través de archivos de configuración escritos en HashiCorp Configuration Language (HCL).

- **Ventajas de Terraform:**
  - **Independencia del proveedor:** Terraform soporta múltiples proveedores de nube, incluyendo AWS, Azure, Google Cloud, y muchos más.
  - **Automatización declarativa:** Terraform permite describir la infraestructura deseada, y se encarga de aplicar los cambios necesarios para alcanzar ese estado.
  - **Colaboración:** Gestiona el estado de la infraestructura a través de archivos de estado, facilitando el trabajo en equipo.
- **Desventajas de Terraform:**
  - **Curva de aprendizaje:** Puede ser complejo para principiantes.
  - **Gestión de estado:** El manejo de archivos de estado en equipos grandes puede ser desafiante.
  - **Complejidad:** Puede requerir una gestión cuidadosa de configuraciones para evitar problemas.

### 3. Proveedores de Terraform:

- **Concepto de Proveedor:** Un proveedor en Terraform es el plugin que permite a Terraform interactuar con APIs externas, como las de servicios de nube, herramientas de monitorización, y otros sistemas.
- **Proveedores comunes:**
  - **AWS (Amazon Web Services):** Permite gestionar recursos en la nube de AWS.
  - **Azure (Microsoft Azure):** Permite gestionar recursos en la nube de Microsoft.
  - **GCP (Google Cloud Platform):** Para la gestión de recursos en Google Cloud.
  - **Kubernetes:** Para la gestión de clústeres y recursos de Kubernetes.
  - **GitHub:** Para gestionar repositorios y otros recursos en GitHub.
  - **VMware vSphere, Datadog, Cloudflare**, entre otros.

## 4. Ejemplos de Configuración de Proveedores y Recursos

**1. Configuración de un Proveedor AWS:**

```
provider "aws" {
  region     = "us-west-2"
  access_key = "your-access-key"
  secret_key = "your-secret-key"
}
```

**2. Creación de una Instancia EC2 en AWS:**

**Creación de una Instancia EC2 en AWS:**

```
resource "aws_instance" "example" {
  ami           = "ami-123456"
  instance_type = "t2.micro"
}
```

**3. Creación de un Grupo de Recursos en Azure:**

**Creación de un Grupo de Recursos en Azure:**

```
resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = "East US"
}
```

## 4. Creación de una Red Virtual en Azure:

**Creación de una Red Virtual en Azure:**

```
resource "azurerm_virtual_network" "example" {
  name                = "example-network"
  address_space       = ["10.0.0.0/16"]
  location            = azurerm_resource_group.example.location
  resource_group_name = azurerm_resource_group.example.name
}
```

## 5. Ciclo de una configuración de Terraform:

- **terraform init:** Inicializa un nuevo entorno de Terraform descargando los plugins necesarios para los proveedores especificados.
- **terraform plan:** Crea un plan de ejecución que muestra los cambios que Terraform realizará en la infraestructura.
- **terraform apply:** Aplica los cambios especificados en el plan y aprovisiona los recursos.
- **terraform destroy:** Elimina todos los recursos gestionados por Terraform, devolviendo la infraestructura al estado previo.

## 6. Comparativa Terraform vs AWS Cloudformation

- **Independencia del Proveedor:**
  - **Terraform:** Soporta múltiples proveedores de nube y es independiente de ellos.
  - **CloudFormation:** Específico para AWS.
- **Lenguaje de Configuración:**
  - **Terraform:** Utiliza HCL (HashiCorp Configuration Language).
  - **CloudFormation:** Utiliza JSON o YAML.
- **Gestión Multi-cloud:**
  - **Terraform:** Puede gestionar infraestructura en múltiples nubes desde una única configuración.
  - **CloudFormation:** Está limitado a recursos y servicios de AWS.

## 7. Ejemplo de creación de un Bucket S3 en AWS con Terraform

```
provider "aws" {
  region = "us-west-2"
}

resource "aws_s3_bucket" "example_bucket" {
  bucket = "my-unique-bucket-name"
  acl    = "private"

  tags = {
    Name        = "My Example Bucket"
    Environment = "Dev"
  }
}
```


*Campista: recuerda revisar el PDF "Guía Terraform".*

# Actividad práctica:

**Modificación de un grupo de seguridad:**

Puedes usar un Change Set para agregar nuevas reglas a un grupo de seguridad sin afectar las instancias en ejecución.

**Resumen y Consideraciones Finales**

- **CloudFormation** es una herramienta poderosa para la automatización y gestión de infraestructura en AWS.
- Las **plantillas** son la base de CloudFormation, y están compuestas por varias secciones que definen todos los aspectos de la infraestructura.
- Las **pseudofunciones** permiten operaciones dinámicas dentro de las plantillas, facilitando la flexibilidad y reutilización.

Los **Change Sets** proporcionan una manera segura de gestionar actualizaciones de recursos, evitando cambios imprevistos o interrupciones de servicio.

# Despliegue de servicios de AWS con Cloudformation

*El objetivo de esta práctica es aprender a desplegar diferentes servicios en AWS utilizando CloudFormation. Los estudiantes crearán una plantilla que definirá ydesplegará una instancia EC2 y un bucket S3, permitiéndoles adquirir experiencia en la creación y gestión de infraestructura como código (IaC).*

**¡Campista: recuerda revisar este material de apoyo!**