Construcción de Compiladores - Laboratorio Guiado No. 4 — Uso de ANTLR para Parsear Terraform y Administrar Droplets de DigitalOcean

1 Introducción

En este laboratorio, deberán usar ANTLR para parsear un archivo de Terraform y mapear el resultado a llamadas de API REST con Python para administrar Droplets de DigitalOcean. Además, utilizarán Docker para configurar y ejecutar su entorno para hacerlo más fácil.

Pero antes de eso, deberán conocer primero las diferencias entre usar Terraform para crear un Droplet en DigitalOcean y usar las API Rest de DigitalOcean para crear el droplet. Para ello realice lo siguiente:

- Lea el archivo titulado **"Construcción de Compiladores - Laboratorio Guiado No. 4 - Terraform"** y realice las actividades dentro de esta guía.
- Lea el archivo titulado **"Construcción de Compiladores - Laboratorio Guiado No. 4 - Bash"** y realice las actividades dentro de esta guía.
- Para ambas guías, deberá de tomar capturas de pantalla de las operaciones y resultados y adjuntarlos al reporte de este laboratorio.
- Luego, realice las actividades dentro de esta guía. Se proporciona código base que usted puede utilizar, además puede utilizar herramientas de GenAI o cualquier material de apoyo para realizar las actividades. La idea principal es que usted experimente con sus propias manos el proceso de crear un parser "pseudo-compilador" para un lenguaje de programación de DSL que pueda analizar y luego transformar a otro conjunto de operaciones resultantes, con el fin que usted vea un caso de uso real que podemos dar a los conceptos de compiladores en la vida real.
- Consulte todos los archivos proporcionados en Canvas.
- No realice operaciones indebidas con el API TOKEN de Digital Ocean que usaremos para estas actividades, de lo contrario se penalizará sobre la entrega por no seguir instrucciones.
- **Deberá realizar este laboratorio en grupos de 5 personas**.

2 Paso 1: Definiendo la Gramática de ANTLR

Primero, creen un archivo llamado `TerraformSubset.g4` con el siguiente contenido:

```
grammar TerraformSubset;

terraform: (provider | resource | variable | output)* EOF;

provider: 'provider' IDENTIFIER '{' providerBody '}';
providerBody: (keyValuePair)*;

resource: 'resource' IDENTIFIER IDENTIFIER '{' resourceBody '}';
resourceBody: (keyValuePair)*;

variable: 'variable' IDENTIFIER '{' variableBody '}';
variableBody: (keyValuePair)*;

output: 'output' IDENTIFIER '{' outputBody '}';
outputBody: (keyValuePair)*;

keyValuePair: IDENTIFIER '=' STRING;

IDENTIFIER: [a-zA-Z_][a-zA-Z_0-9]*;
STRING: '"' ('\\' . | ~["\\])* '"';

WS: [ \t\r\n]+ -> skip;
```

3 Paso 2: Generando el Parser y Lexer de ANTLR

Para generar el parser y lexer desde la gramática de ANTLR, ejecuten el siguiente comando:

```
antlr4 -Dlanguage=Python3 TerraformSubset.g4
```

Este comando generará varios archivos Python que podrán usar para parsear archivos de Terraform (simples, nada fancy).

4 Paso 3: Creando el Código en Python para Parsear Terraform y Realizar Llamadas a la API

Primero, instalen los paquetes necesarios en Python:

```
pip install antlr4-python3-runtime requests
```

Luego, creen un archivo de Python llamado `terraform_parser.py` con el siguiente contenido (esto es base, usted debe modificar):

```
import sys
import requests
from antlr4 import *
from TerraformSubsetLexer import TerraformSubsetLexer
from TerraformSubsetParser import TerraformSubsetParser
from TerraformSubsetListener import TerraformSubsetListener
```

```
API_TOKEN = "API_TOKEN_DE_DIGITALOCEAN"

class TerraformListener(TerraformSubsetListener):
    def __init__(self):
        self.provider = {}
        self.resources = []

    def enterProvider(self, ctx:TerraformSubsetParser.ProviderContext)
       :
        provider_name = ctx.IDENTIFIER().getText()
        self.provider[provider_name] = {}
        for kv in ctx.providerBody().keyValuePair():
            key = kv.IDENTIFIER().getText()
            value = kv.STRING().getText().strip('"')
            self.provider[provider_name][key] = value

    def enterResource(self, ctx:TerraformSubsetParser.ResourceContext)
       :
        resource_type = ctx.IDENTIFIER(0).getText()
        resource_name = ctx.IDENTIFIER(1).getText()
        resource = {"type": resource_type, "name": resource_name, "
           attributes": {}}
        for kv in ctx.resourceBody().keyValuePair():
            key = kv.IDENTIFIER().getText()
            value = kv.STRING().getText().strip('"')
            resource["attributes"][key] = value
        self.resources.append(resource)

    def getProvider(self):
        return self.provider

    def getResources(self):
        return self.resources

def create_droplet(api_token, resource):
    url = "https://api.digitalocean.com/v2/droplets"
    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {api_token}"
    }
    data = {
        "name": resource["attributes"]["name"],
        "region": resource["attributes"]["region"],
        "size": resource["attributes"]["size"],
        "image": resource["attributes"]["image"]
    }
    response = requests.post(url, headers=headers, json=data)
    if response.status_code == 202:
        droplet = response.json()["droplet"]
        print(f"Droplet creado con ID: {droplet['id']}")
        return droplet['id']
    else:
```

```
        print(f"Error al crear el droplet: {response.text}")
        return None

def delete_droplet(api_token, droplet_id):
    url = f"https://api.digitalocean.com/v2/droplets/{droplet_id}"
    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {api_token}"
    }
    response = requests.delete(url, headers=headers)
    if response.status_code == 204:
        print(f"Droplet con ID {droplet_id} ha sido destruido")
    else:
        print(f"Error al destruir el droplet: {response.text}")

def main(argv):
    input_stream = FileStream(argv[1])
    lexer = TerraformSubsetLexer(input_stream)
    stream = CommonTokenStream(lexer)
    parser = TerraformSubsetParser(stream)
    tree = parser.terraform()

    listener = TerraformListener()
    walker = ParseTreeWalker()
    walker.walk(listener, tree)

    provider = listener.getProvider()
    resources = listener.getResources()

    if provider and resources:
        for resource in resources:
            if resource["type"] == "droplet":
                droplet_id = create_droplet(API_TOKEN, resource)
                if droplet_id:
                    delete_droplet(API_TOKEN, droplet_id)

if __name__ == '__main__':
    main(sys.argv)
```

En la ejecución de su programa, se debe de mostrar en pantalla o debe guardar algún archivo, el ID del droplet creado para validar que su "compilador" está creando droplets y luego destruyéndolos.

También modifique el código para que utilice el API TOKEN de DigitalOcean que se proveerá en el archivo de Terraform y no el valor hardcodeado.

5 Paso 4: Configuración de Docker

Cree un archivo `Dockerfile` para configurar un contenedor que pueda ejecutar el script de Python:

```
FROM python:3.9-slim

WORKDIR /app
```

```
COPY . /app

RUN pip install --no-cache-dir antlr4-python3-runtime requests

CMD ["bash"]
```

Cree un archivo `docker-compose.yml`:

```
version: '3.7'

services:
  terraform-parser:
    build: .
    environment:
      - API_TOKEN=API_TOKEN
    volumes:
      - .:/app
    working_dir: /app
```

6 Paso 5: Ejecutando los Scripts

1. Construyendo la imagen de Docker:

```
docker-compose build
```

2. Ejecutando el script de Python para parsear el archivo de Terraform y crear el droplet:

```
docker-compose run terraform-parser python terraform_parser.py main.tf
```

El archivo `main.tf` que deberá parsear deberá ser más simple que los usados antes con Terraform, utilice este:

```
provider "digitalocean" {
  token = var.digitalocean_token
}

resource "digitalocean_droplet" "web" {
  image  = "ubuntu-20-04-x64"
  name   = "example-droplet"
  region = "nyc1"
  size   = "s-1vcpu-1gb"
}

variable "digitalocean_token" {
  description = "The DigitalOcean API token."
  type        = string
  sensitive   = true
  default     = "API_TOKEN"
}
```