Exercice 3 — Déploiement Nginx avec Terraform et GitHub

Énoncé

Mettre en place un projet qui déploie un **service Nginx non-root** sur OpenShift via **Terraform**, avec gestion de versions dans **GitHub**. Deux versions :

- **Version 1** : page d'accueil « Version 1 ».
- **Version 2** : modification via variable Terraform → page « Version 2 modifiée ».
- Retour à V1 possible en utilisant GitHub (rollback).

Correction

1. Initialisation du projet

```
mkdir terraform-nginx
cd terraform-nginx
```

2. Fichier `main.tf` (V1)

```
terraform {
  required_providers {
    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = "2.27.0"
    }
  }
}

provider "kubernetes" {
  config_path = "~/.kube/config"
}

variable "page_content" {
  type    = string
  default = "<h1>Version 1 - Nginx via Terraform</h1>"
}

resource "kubernetes_config_map" "nginx_index" {
  metadata { name = "nginx-index" }
  data = { "index.html" = var.page_content }
}
```

```
resource "kubernetes_deployment" "nginx" {
  metadata { name = "nginx-app" }
  spec {
    replicas = 1
    selector { match_labels = { app = "nginx-app" } }
    template {
      metadata { labels = { app = "nginx-app" } }
      spec {
        container {
          name  = "nginx"
          image = "registry.access.redhat.com/ubi9/nginx-120"
          port { container_port = 8080 }
          volume_mount {
            mount_path = "/usr/share/nginx/html/index.html"
            sub_path   = "index.html"
            name       = "html"
          }
        }
        volume { name = "html" config_map { name =
kubernetes_config_map.nginx_index.metadata[0].name } }
      }
    }
  }
}

resource "kubernetes_service" "nginx" {
  metadata { name = "nginx-service" }
  spec { selector = { app = "nginx-app" } port { port = 8080 target_port =
8080 } }
}
```

3. Déploiement Version 1

```
terraform init
terraform apply -auto-approve
oc expose service nginx-service
$ROUTE = oc get route nginx-service -o jsonpath="{.spec.host}"
$URL = "http://$ROUTE"
$URL
```

➡️Navigateur : « Version 1 - Nginx via Terraform ».

4. Sauvegarde dans GitHub (V1)

```
git init
git add .
git commit -m "V1 - Terraform Nginx Version 1"
git branch -M main
git remote add origin https://github.com/<USERNAME>/terraform-nginx.git
git push -u origin main
```

5. Passage à Version 2 (via variable)

Modifier `main.tf` :

```
variable "page_content" {
  type    = string
  default = "<h1>Version 2 - Nginx modifié via variable Terraform</h1>"
}
```

Appliquer et pousser :

```
git checkout -b feature-v2
git add .
git commit -m "V2 - Terraform mise à jour via variable"
git push origin feature-v2
terraform apply -auto-approve
$URL
```

➡️Navigateur : « Version 2 - Nginx modifié via variable Terraform ».

6. Rollback vers Version 1

Basculer sur la branche principale :

```
git checkout main
git pull origin main
terraform apply -auto-approve
$URL
```

➡️Retour à « Version 1 ».

Exercice 4 — Application Flask avec Terraform et GitHub

Énoncé

Déployer une application Flask simple via Terraform, avec gestion des versions dans **GitHub**. Deux versions :

- **Version 1** : route `/` affichant « Flask V1 ».
- **Version 2** : modification avec ajout de la route `/status`.

- Retour à V1 via GitHub.

Correction

1. Initialisation

```
mkdir terraform-flask
cd terraform-flask
```

2. Fichier `main.tf` (V1)

```
variable "flask_code" {
  type    = string
  default = <<EOT
from flask import Flask
app = Flask(__name__)
@app.route('/')
def home():
    return '<h1>Flask V1</h1>'
app.run(host='0.0.0.0', port=8080)
EOT
}

resource "kubernetes_config_map" "flask_code" {
  metadata { name = "flask-code" }
  data = { "app.py" = var.flask_code }
}

resource "kubernetes_deployment" "flask" {
  metadata { name = "flask-app" }
  spec {
    replicas = 1
    selector { match_labels = { app = "flask-app" } }
    template {
      metadata { labels = { app = "flask-app" } }
      spec {
        container {
          name  = "flask"
          image = "registry.access.redhat.com/ubi9/python-39"
          command = ["python3", "/app/app.py"]
          volume_mount { mount_path = "/app/app.py" sub_path = "app.py" name
= "code" }
          port { container_port = 8080 }
        }
        volume { name = "code" config_map { name =
kubernetes_config_map.flask_code.metadata[0].name } }
      }
    }
  }
}

resource "kubernetes_service" "flask" {
  metadata { name = "flask-service" }
  spec { selector = { app = "flask-app" } port { port = 8080 target_port =
8080 } }
}
```

3. Déploiement Version 1

```
terraform init
terraform apply -auto-approve
oc expose service flask-service
$ROUTE = oc get route flask-service -o jsonpath="{.spec.host}"
$URL = "http://$ROUTE"
$URL
```

➡️Navigateur : « Flask V1 ».

4. GitHub (V1)

```
git init
git add .
git commit -m "V1 - Terraform Flask"
git branch -M main
git remote add origin https://github.com/<USERNAME>/terraform-flask.git
git push -u origin main
```

5. Passage à Version 2

Modifier la variable :

```
default = <<EOT
from flask import Flask, jsonify
app = Flask(__name__)
@app.route('/')
def home():
    return '<h1>Flask V2</h1>'
@app.route('/status')
def status():
    return jsonify(status='running', version='V2')
app.run(host='0.0.0.0', port=8080)
EOT
```

Appliquer :

```
git checkout -b feature-v2
git add .
git commit -m "V2 - Flask modifié avec /status"
git push origin feature-v2
terraform apply -auto-approve
$URL
```

➡️Navigateur : `/` → « Flask V2 », `/status` → `{ "status": "running", "version": "V2" }`.

6. Rollback vers V1

Revenir sur la branche principale :

```
git checkout main
git pull origin main
terraform apply -auto-approve
$URL
```

➡️Retour à « Flask V1 ».

Exercice 5 — Application Node.js avec Terraform et GitHub

Énoncé

Déployer une application Node.js simple avec Terraform et GitHub. Utiliser des **branches** pour gérer deux versions :

- **Version 1 (`main`)** : route `/` affichant « Node.js V1 ».
- **Version 2 (`feature-v2`)** : ajout d'une route `/about`.
- Retour à V1 possible en changeant de branche.

Correction

1. Initialisation

```
mkdir terraform-node
cd terraform-node
```

2. Fichier `main.tf` (V1)

```
variable "node_code" {
  type    = string
  default = <<EOT
const http = require('http');
http.createServer((req,res)=>{
  if(req.url === '/') res.end('<h1>Node.js V1</h1>');
  else {res.statusCode=404; res.end('Not Found');}
}).listen(8080);
EOT
}

resource "kubernetes_config_map" "node_code" {
  metadata { name = "node-code" }
  data = { "server.js" = var.node_code }
}

resource "kubernetes_deployment" "node" {
  metadata { name = "node-app" }
  spec {
    replicas = 1
    selector { match_labels = { app = "node-app" } }
    template {
      metadata { labels = { app = "node-app" } }
      spec {
        container {
          name  = "node"
          image = "registry.access.redhat.com/ubi9/nodejs-18"
          command = ["node", "/app/server.js"]
          volume_mount { mount_path =
"/app/server.js" sub_path = "server.js" name = "code" }
          port { container_port = 8080 }
        }
        volume { name = "code" config_map { name =
kubernetes_config_map.node_code.metadata[0].name } }
      }
    }
  }
}

resource "kubernetes_service" "node" {
  metadata { name = "node-service" }
  spec { selector = { app = "node-app" } port { port = 8080 target_port =
8080 } }
}
```

3. Déploiement V1

```
terraform init
terraform apply -auto-approve
oc expose service node-service
$ROUTE = oc get route node-service -o jsonpath="{.spec.host}"
$URL = "http://$ROUTE"
$URL
```

➡️Navigateur : « Node.js V1 ».

4. GitHub (V1)

```
git init
git add .
git commit -m "V1 - Terraform Node.js"
git branch -M main
git remote add origin https://github.com/<USERNAME>/terraform-node.git
git push -u origin main
```

5. Passage à Version 2 (branche `feature-v2`)

```
git checkout -b feature-v2
```

Modifier la variable :

```
default = <<EOT
const http = require('http');
http.createServer((req,res)=>{
  if(req.url === '/') res.end('<h1>Node.js V2</h1>');
  else if(req.url === '/about') res.end('<p>About Node.js V2</p>');
  else {res.statusCode=404; res.end('Not Found');}
}).listen(8080);
EOT
```

Déployer :

```
git add .
git commit -m "V2 - Node.js avec /about"
git push origin feature-v2
terraform apply -auto-approve
$URL
```

➡️Navigateur : `/` → « Node.js V2 », `/about` → « About Node.js V2 ».

6. Rollback vers V1

Revenir sur la branche principale :

```
git checkout main
git pull origin main
terraform apply -auto-approve
$URL
```

➡️Retour à « Node.js V1 ».

Exercice 6 — Application Apache avec Terraform et GitHub

Énoncé

Déployer une application Apache avec Terraform et gérer deux versions grâce à **GitHub** et l'utilisation des **branches** :

- **Version 1 (`main`)** : page HTML verte.
- **Version 2 (`feature-v2`)** : page HTML bleue.
- Rollback vers V1 en basculant de branche.

Correction

1. Initialisation

```
mkdir terraform-apache
cd terraform-apache
```

2. Fichier `main.tf` (V1)

```
variable "page_content" {
  type    = string
  default = "<html><body style='background-color:lightgreen'><h1>Apache V1</
h1></body></html>"
}

resource "kubernetes_config_map" "apache_index" {
  metadata { name = "apache-index" }
  data = { "index.html" = var.page_content }
}

resource "kubernetes_deployment" "apache" {
  metadata { name = "apache-app" }
  spec {
    replicas = 1
    selector { match_labels = { app = "apache-app" } }
    template {
      metadata { labels = { app = "apache-app" } }
      spec {
        container {
          name  = "apache"
          image = "registry.access.redhat.com/ubi9/httpd-24"
          ports { container_port = 8080 }
          volume_mount { mount_path = "/var/www/html/index.html" sub_path =
"index.html" name = "html" }
        }
        volume { name = "html" config_map { name =
kubernetes_config_map.apache_index.metadata[0].name } }
      }
    }
  }
}

resource "kubernetes_service" "apache" {
  metadata { name = "apache-service" }
  spec { selector = { app = "apache-app" } port { port = 8080 target_port =
8080 } }
}
```

3. Déploiement V1

```
terraform init
terraform apply -auto-approve
oc expose service apache-service
$ROUTE = oc get route apache-service -o jsonpath="{.spec.host}"
$URL = "http://$ROUTE"
$URL
```

➡️Navigateur : « Apache V1 » (fond vert).

4. GitHub (V1)

```
git init
git add .
git commit -m "V1 - Terraform Apache"
git branch -M main
git remote add origin https://github.com/<USERNAME>/terraform-apache.git
git push -u origin main
```

5. Passage à Version 2 (branche `feature-v2`)

```
git checkout -b feature-v2
```

Modifier la variable :

```
default = "<html><body style='background-color:lightblue'><h1>Apache V2</
h1></body></html>"
```

Appliquer et pousser :

```
git add .
git commit -m "V2 - Apache page bleue"
git push origin feature-v2
terraform apply -auto-approve
$URL
```

➡️Navigateur : page bleue (V2).

6. Rollback vers V1

Basculer sur la branche principale :

```
git checkout main
git pull origin main
terraform apply -auto-approve
$URL
```

➡️Retour à Apache V1 (fond vert).

Conclusion

Ces quatre exercices (Nginx, Flask, Node.js, Apache) montrent : - L'utilisation de **Terraform** pour déployer des applications sur OpenShift. - L'usage de **variables Terraform** pour gérer les contenus ou routes. - La gestion de **versions** grâce aux **branches GitHub** (main / feature-v2). - La capacité de faire un **rollback rapide** en basculant de branche.