

# RAPPORT MINI PROJET CLOUD

## Introduction :

Ce projet consistait à déployer une application web complète (React + Node.js + MySQL) sur Microsoft Azure en utilisant Terraform pour l'Infrastructure as Code (IaC). L'objectif après la mise en place de notre infrastructure cloud sera d'automatiser la création d'une machine virtuelle et d'un stockage cloud avec Azure.

## Etape 1 :

Pour configurer le provider cloud Azure on crée déjà un dossier pour notre projet ensuite on créera un fichier provider.tf dans lequel on définit le provider azure avec les fonctionnalités de base d'Azure. Puis l'on va créer notre fichier main.tf où l'on mettra nos instructions pour la création de l'infrastructure. Nos différents fichiers contiendront :

### - main.tf - Ressources principales

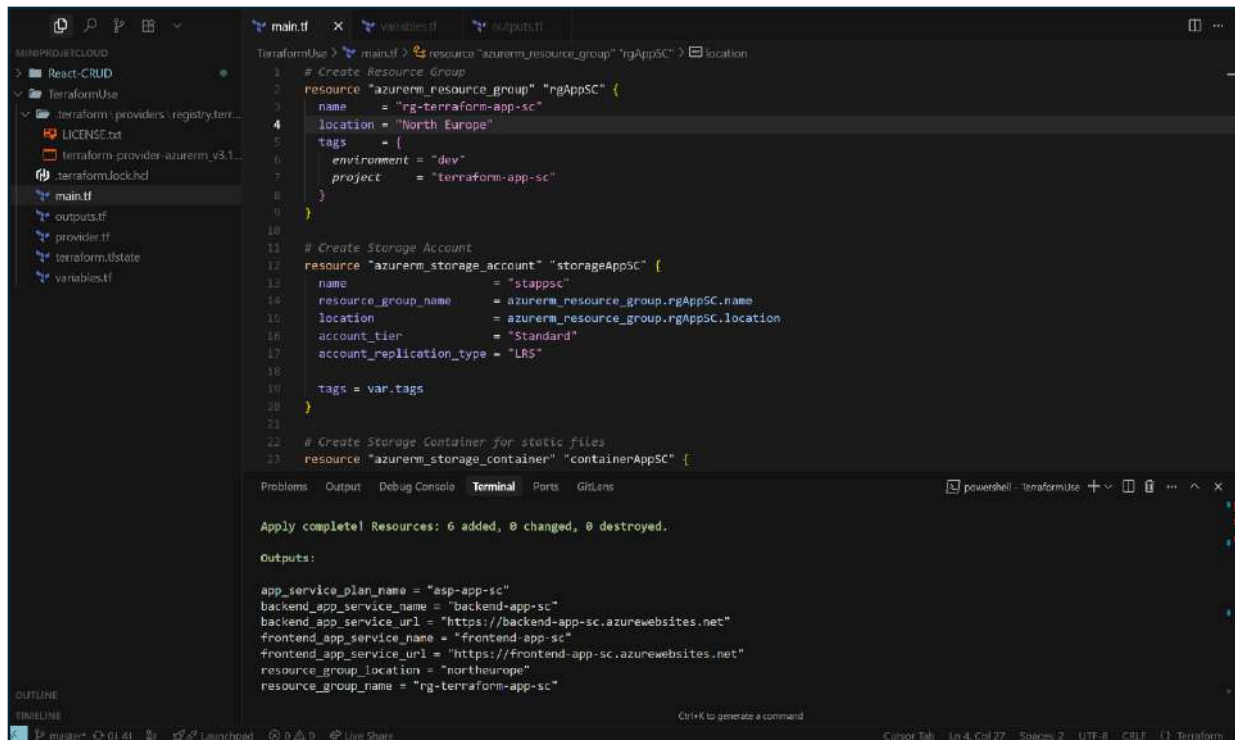
- **Resource Group** : Groupe de ressources principal
- **Storage Account** : Compte de stockage avec container pour fichiers statiques
- **App Service Plan** : Plan d'hébergement Linux (B1)
- **Frontend App Service** : Service web pour le frontend React
- **Backend App Service** : Service web pour le backend Node.js

### - variables.tf - Variables de configuration

- **resource\_group\_name** : Nom du groupe de ressources
- **location** : Région Azure (par défaut "East US")
- **environment** : Environnement (dev, staging, prod)
- **project\_name** : Nom du projet
- **tags** : Tags communs pour toutes les ressources

### - outputs.tf - Sorties importantes

- URLs des applications frontend et backend
- Informations du storage account
- Noms des ressources créées



Enfin on se connecte à azure avec commande `az login` exécuter dans un terminal puis on tape les lignes de commandes suivante de Terraforms :

- **terraform init** (pour initialiser Terraform)
- **terraform plan** (pour planifier le déploiement)
- **terraform apply** (pour déployer nos ressources)

La configuration est prête et on déploiera une infrastructure complète pour notre application react-crud sur Azure.

## Etape2 :

### A – Creation de la VM :

Avant de réaliser cette étape il faut qu'on passe par la création de quatre sous-tâches en ressource.

#### 1.Création clef SSH

Dans cette étapes on va tout d'abord crée notre réseau virtuel, puis notre **ip publique** pour la futur Virtual Machine puis enfin crée notre **VM** pour terminer mais avant tout il nous faut d'abord générer une **clef SSH**.

On commence donc par générer et vérifier une clé ssh avec ces lignes de commandes dans un terminal :

- `ssh-keygen -t rsa -b 4096 -C "votre-email@example.com".`
- `ls -la ~/.ssh/`

On aura donc `id_rsa` (clé privée) et `id_rsa.pub` (clé publique).

## 2.Création du Réseau Virtuel et des Sous-Réseaux

Ensuite sur main.tf on va se rendre dans la documentation officielle de terraform pour utiliser le code qui correspond à la configuration du Virtual Network et du Subnet puis l'insérer en l'adaptant à notre besoin.

```
# Create Virtual Network
resource "azurerm_virtual_network" "vnetAppSC" {
  name                = "vnet-app-sc"
  address_space       = ["10.0.0.0/16"]
  location             = azurerm_resource_group.rgAppSC.location
  resource_group_name = azurerm_resource_group.rgAppSC.name

  tags = var.tags
}

# Create Subnet
resource "azurerm_subnet" "subnetAppSC" {
  name                = "subnet-app-sc"
  resource_group_name = azurerm_resource_group.rgAppSC.name
  virtual_network_name = azurerm_virtual_network.vnetAppSC.name
  address_prefixes     = ["10.0.1.0/24"]

  service_endpoints = ["Microsoft.Storage"]
}

# Create Network Security Group and rules
resource "azurerm_network_security_group" "nsgAppSC" {
  name                = "nsg-app-sc"
  location             = azurerm_resource_group.rgAppSC.location
  resource_group_name = azurerm_resource_group.rgAppSC.name

  security_rule {
    name                = "SSH"
    priority             = 100
    direction           = "Inbound"
    access              = "Allow"
    protocol             = "Tcp"
    source_port_range    = "*"
    destination_port_range = "22"
    source_address_prefix = "*"
    destination_address_prefix = "*"
  }

  security_rule {
    name                = "HTTP"
    priority             = 110
    direction           = "Inbound"
    access              = "Allow"
    protocol             = "Tcp"
    source_port_range    = "*"
    destination_port_range = "80"
    source_address_prefix = "*"
    destination_address_prefix = "*"
  }
}
```

En effet avant d'installer la VM il nous faut au préalable configurer le réseau et le subnet car sans réseau, une VM n'a pas d'adresse IP ni de moyen de communication avec l'extérieur ou d'autres ressources. En somme :

- Le **VNet** agit comme un réseau privé virtuel dans le cloud.
- Le **subnet** divise ce VNet en segments logiques, permettant d'organiser et d'isoler les ressources.

De plus cela nous évitera de devoir tout reconfigurer plus tard si on ajoute des VMs ou services supplémentaires.

### 3.Création du NGS

Ensuite on fait pareil pour le **network security group and rules** on récupère le code de la documentation officiel de Terraform puis on l'insère pour créer un pare-feu au niveau du réseau. En effet un NGS permet de contrôler le trafic réseau vers et depuis les ressources Azures comme :

- machine virtuelles (VM)
- interface réseau (NIC)
- sous-réseau (subnet)

Ainsi toutes les ressources dans ce subnet hériteront des règles ce qui est idéal pour sécuriser un groupe de VM.

On définit donc les règles NSG pour notre réseau avec ces configurations :

```

aformUse > main.tf > resource "azurerm_network_interface" "nicAppSC" > ip_configuration > public_ip_address_id
resource "azurerm_network_security_group" "nsgAppSC" {
  security_rule {
    destination_address_prefix = "*"
  }

  security_rule {
    name                = "AllowAnyCustom5100-5300Inbound"
    priority             = 400
    direction            = "Inbound"
    access               = "Allow"
    protocol              = "Tcp"
    source_port_range     = "*"
    destination_port_ranges = ["5100-5300"]
    source_address_prefix = "*"
    destination_address_prefix = "*"
  }

  security_rule {
    name                = "Backend-Node"
    priority             = 500
    direction            = "Inbound"
    access               = "Allow"
    protocol              = "Tcp"
    source_port_range     = "*"
    destination_port_range = "5170"
    source_address_prefix = "*"
    destination_address_prefix = "*"
  }

  security_rule {
    name                = "Front-React"
    priority             = 600
    direction            = "Inbound"
    access               = "Allow"
    protocol              = "Tcp"
    source_port_range     = "*"
    destination_port_range = "5193"
    source_address_prefix = "*"
    destination_address_prefix = "*"
  }

  tags = var.tags
}

```

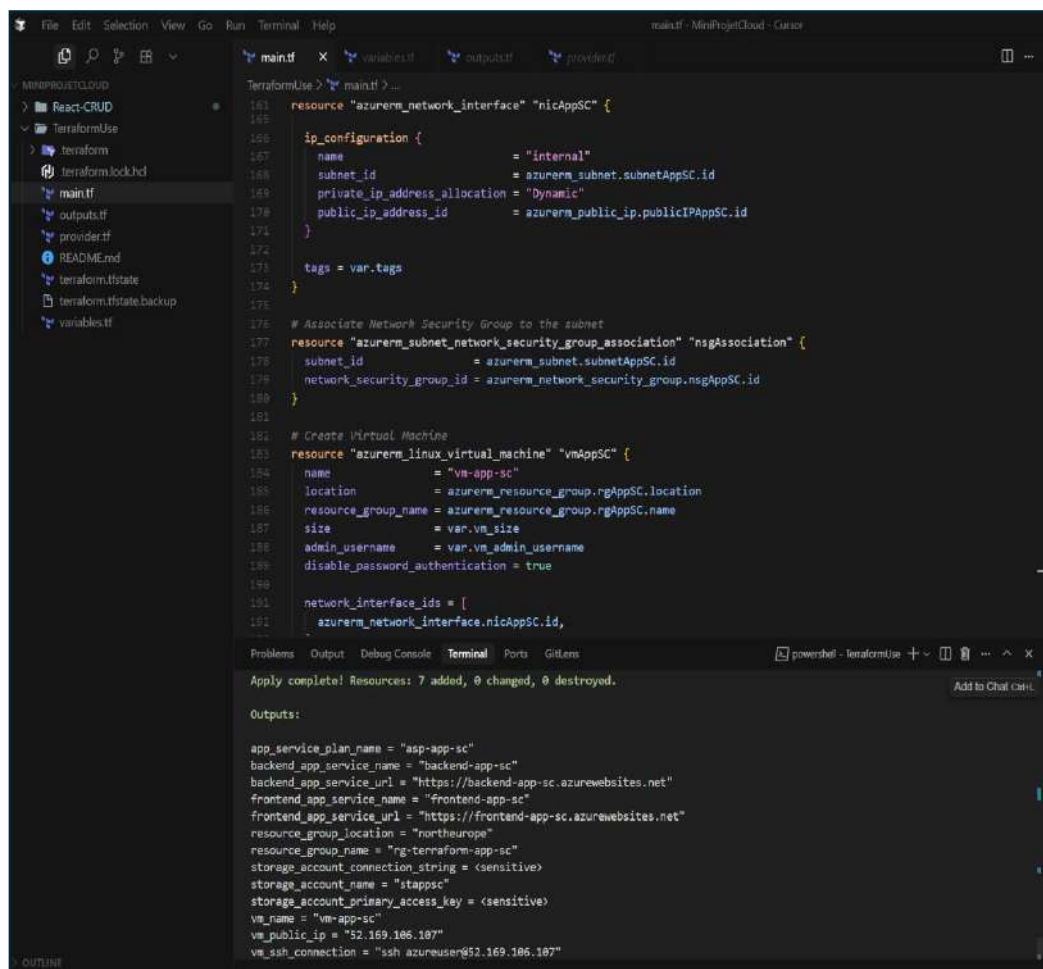
## 4. Création de l'Interface Réseau (NIC-Network Interface Card)

Une **NIC** est une carte réseau virtuelle que vous **attachez à une VM**. Elle permet à la VM de **communiquer** avec le reste du réseau ce qui est essentiel avant d'installer une VM. Elle contiendra donc :

- Une **adresse IP privée** (obligatoire, pour la communication dans le VNet)
- Optionnellement une **adresse IP publique**
- Un **NSG** (optionnel) pour sécuriser le trafic
- Un lien avec un **sous-réseau (subnet)** dans un **Virtual Network (VNet)**

Une IP publique est nécessaire si on veut que votre VM soit **accessible depuis Internet**, par exemple :

- Connexion SSH (port 22)
- Accès RDP (port 3389)
- Hébergement d'un site web
- Accès via une API publique



```
main.tf x variables.tf outputs.tf provider.tf
TerraformUse> main.tf > ...
161 resource "azurerm_network_interface" "nicAppSC" {
162
163   ip_configuration {
164     name                 = "internal"
165     subnet_id            = azurerm_subnet.subnetAppSC.id
166     private_ip_address_allocation = "Dynamic"
167     public_ip_address_id    = azurerm_public_ip.publicIPAppSC.id
168   }
169   tags = var.tags
170 }
171
172 # Associate Network Security Group to the subnet
173 resource "azurerm_subnet_network_security_group_association" "nsgAssociation" {
174   subnet_id            = azurerm_subnet.subnetAppSC.id
175   network_security_group_id = azurerm_network_security_group.nsgAppSC.id
176 }
177
178 # Create Virtual Machine
179 resource "azurerm_linux_virtual_machine" "vmAppSC" {
180   name                = "vm-app-sc"
181   location            = azurerm_resource_group.rgAppSC.location
182   resource_group_name = azurerm_resource_group.rgAppSC.name
183   size               = var.vm_size
184   admin_username     = var.vm_admin_username
185   disable_password_authentication = true
186
187   network_interface_ids = [
188     azurerm_network_interface.nicAppSC.id,
189   ]
190 }
191
192 Apply complete! Resources: 7 added, 0 changed, 0 destroyed.
193
194 Outputs:
195
196 app_service_plan_name = "asp-app-sc"
197 backend_app_service_name = "backend-app-sc"
198 backend_app_service_url = "https://backend-app-sc.azurewebsites.net"
199 frontend_app_service_name = "frontend-app-sc"
200 frontend_app_service_url = "https://frontend-app-sc.azurewebsites.net"
201 resource_group_location = "northeurope"
202 resource_group_name = "rg-terraform-app-sc"
203 storage_account_connection_string = <sensitive>
204 storage_account_name = "stappsc"
205 storage_account_primary_access_key = <sensitive>
206 vm_name = "vm-app-sc"
207 vm_public_ip = "52.169.106.107"
208 vm_ssh_connection = "ssh azureuser@52.169.106.107"
```

## 5. Associer un NGS à un subnet :

Après avoir identifié le code depuis le site dédié à terraform on peut associer un NGS à un subnet afin que toutes les ressources tel que notre future VM héritent automatiquement des règles du NGS au niveau de leur **trafic réseau entrant et sortant**. On peut donc facilement contrôler le trafic réseau de tout un groupe de ressources en une seule configuration très pratique pour l'automatisation.

## 6. Création de la VM :

Avec le code de création de la VM en main qu'on insère dans notre main.tf on l'adapte à notre projet puis termine en intégrant nos données pour la connections ssh. Dans ce code on va configurer les paramètres de base de vm, définir l'OS et inclure nos données sur la clef ssh.

```
# Create Virtual Machine
resource "azurerm_linux_virtual_machine" "vmAppSC" {
  name                = "vm-app-sc"
  location            = azurerm_resource_group.rgAppSC.location
  resource_group_name = azurerm_resource_group.rgAppSC.name
  size               = var.vm_size
  admin_username     = var.vm_admin_username
  disable_password_authentication = true

  network_interface_ids = [
    azurerm_network_interface.nicAppSC.id,
  ]

  admin_ssh_key {
    username   = var.vm_admin_username
    public_key = file(var.ssh_public_key_path) # Assurez-vous d'avoir une clé SSH
  }

  os_disk {
    caching              = "ReadWrite"
    storage_account_type = "Premium_LRS"
  }

  source_image_reference {
    publisher = "Canonical"
    offer     = "0001-com-ubuntu-server-focal"
    sku       = "20_04-lts-gen2"
    version   = "latest"
  }

  tags = var.tags
}
```

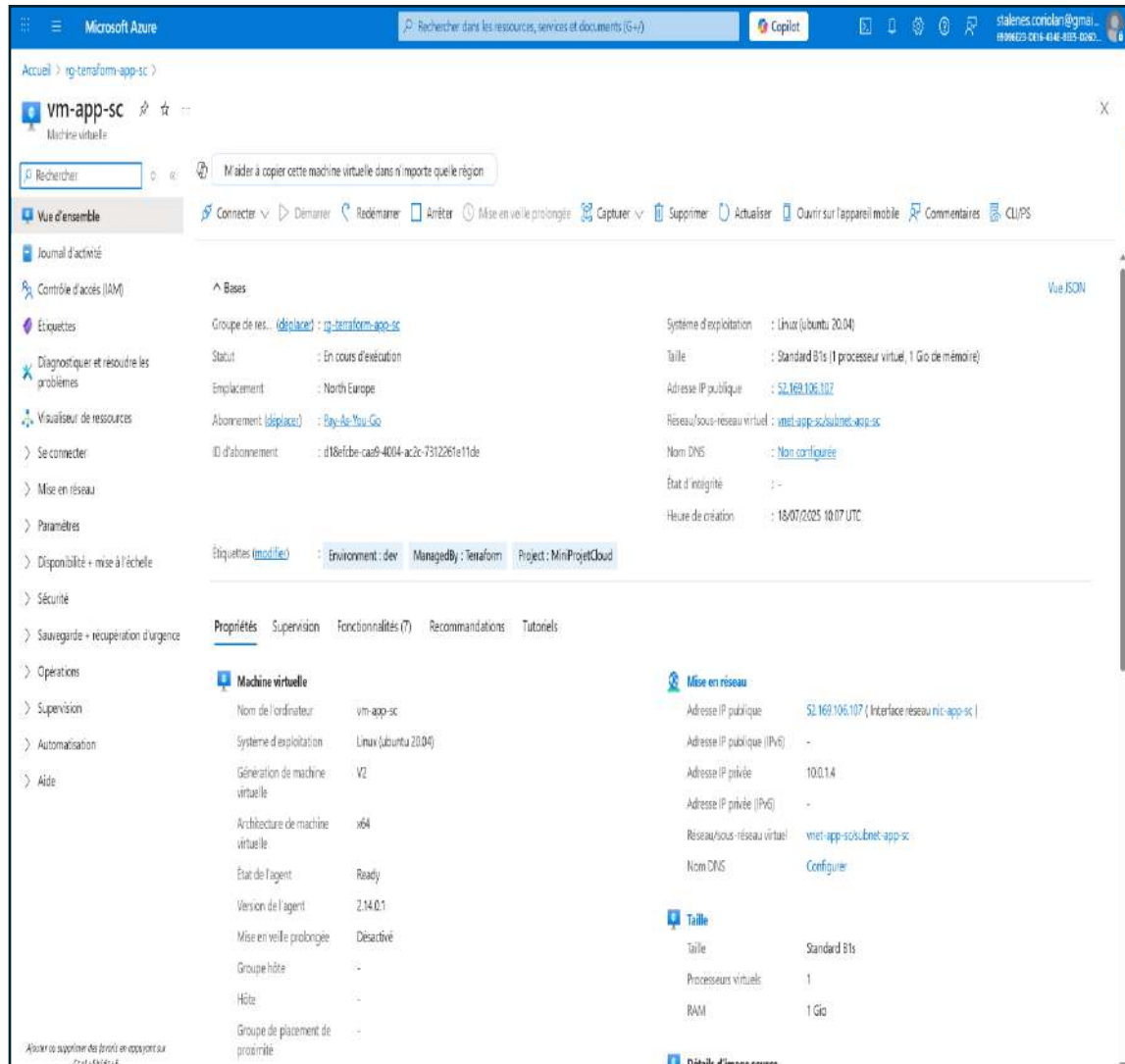
On déploie avec les commande terraform init -> terraform plan -> terraform apply.  
Ensuite dans notre fichier outputs.tf on ajoute :

- vm\_public\_ip : Adresse IP publique de la VM
- vm\_ssh\_connection : Commande SSH pour se connecter
- vm\_name : Nom de la VM

Le déploiement fait on peut finalement se connecter à la VM via :

ssh [azureuser@52.169.106.107](mailto:azureuser@52.169.106.107)

L'infrastructure est maintenant complète avec App Services + VM Ubuntu accessible publiquement.



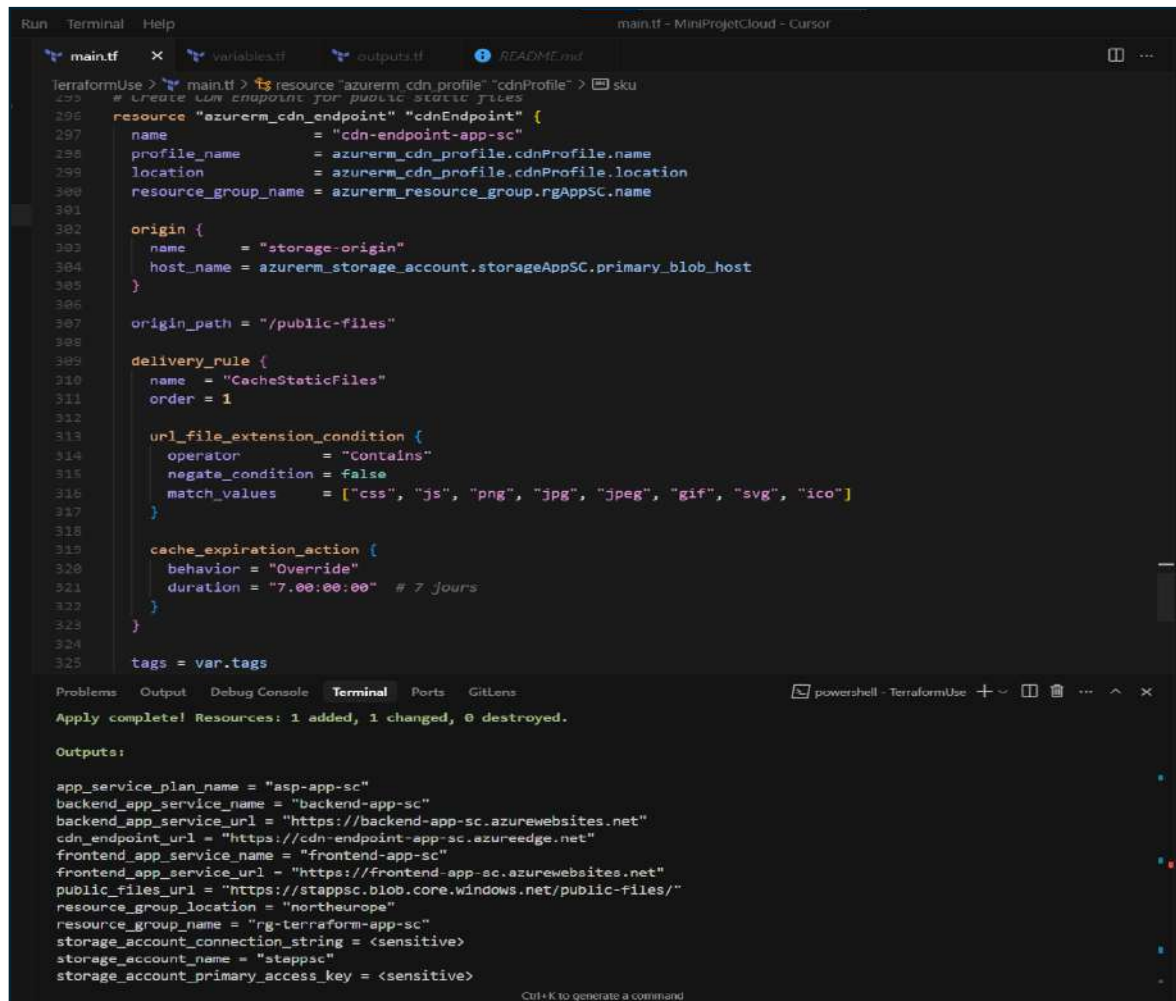
## **B - Création du stockage cloud avec Blob :**

Pour cette tâche on va utiliser plusieurs ressources pour créer un stockage cloud avec Blob. On va donc rechercher le code de nos ressources utile pour le projet et les insérer dans notre fichier main.tf de terraform.



## 1. Ressource Azurerem CDN Endpoint

Pour accélérer la livraison de fichier statique on va utiliser un Content Delivery Network (CDN) et permettre un accès global rapide via une URL publique. Grâce au CDN Endpoint de Azure on pourra donc distribuer des fichiers statiques (image, vidéo, JS...) stockés dans un Azure Storage Account (blob container).



```
main.tf
# Create CDN endpoint for public static files
resource "azurerem_cdn_endpoint" "cdnEndpoint" {
  name = "cdn-endpoint-app-sc"
  profile_name = azurerem_cdn_profile.cdnProfile.name
  location = azurerem_cdn_profile.cdnProfile.location
  resource_group_name = azurerem_resource_group.rgAppSC.name

  origin {
    name = "storage-origin"
    host_name = azurerem_storage_account.storageAppSC.primary_blob_host
  }

  origin_path = "/public-files"

  delivery_rule {
    name = "CacheStaticFiles"
    order = 1

    url_file_extension_condition {
      operator = "contains"
      negate_condition = false
      match_values = ["css", "js", "png", "jpg", "jpeg", "gif", "svg", "ico"]
    }

    cache_expiration_action {
      behavior = "Override"
      duration = "7.00:00:00" # 7 jours
    }
  }
}

tags = var.tags
```

```
Apply complete! Resources: 1 added, 1 changed, 0 destroyed.

Outputs:
app_service_plan_name = "asp-app-sc"
backend_app_service_name = "backend-app-sc"
backend_app_service_url = "https://backend-app-sc.azurewebsites.net"
cdn_endpoint_url = "https://cdn-endpoint-app-sc.azureedge.net"
frontend_app_service_name = "frontend-app-sc"
frontend_app_service_url = "https://frontend-app-sc.azurewebsites.net"
public_files_url = "https://stappsc.blob.core.windows.net/public-files/"
resource_group_location = "northeurope"
resource_group_name = "rg-terraform-app-sc"
storage_account_connection_string = <sensitive>
storage_account_name = "stappsc"
storage_account_primary_access_key = <sensitive>
```

Nos fichiers seront accessibles via url : « **<https://stappsc.blobcore.windows.net>** »

## 2. Ressource Storage Container

Maintenant pour **héberger** des **fichiers publics** (image, pdf, js, vidéos...) dans Azure Blog Storage, on a créé un Storage Container configuré pour autoriser l'accès public en lecture. Notre url pour l'accès des fichiers publiquement est :

« **<https://stappsc.blobcore.windows.net/public-files/>** »



Ensuite pour **uploader des fichiers privés** par les utilisateurs on a créé une ressource pour les fichiers privée nommée « **PrivateFiles** ». Ainsi on pourra uploades des fichiers privés via le backend(**serveur API**) ou directement depuis le navigateur en générant une url temporaire.

Pour stocker des **logs d'application** dans **Azure Blob Storage**, on crée un **Storage Container privé** configuré pour des **écritures fréquentes**, une **rétenion longue**, et un **accès sécurisé**.

L'objectif sera de :

- Centraliser les **logs applicatifs** (ex: fichiers .log, .json, .txt)
- Sécuriser l'accès (lecture/écriture par l'application uniquement)
- Activer des **règles de rétention (lifecycle policy)** pour supprimer les anciens logs

```
Run Terminal Help main.tf - MiniProjetCloud - Cursor
main.tf x variables.tf outputs.tf README.md
TerraformUse > main.tf > resource "azurecdn_profile" "cdnProfile" > sku
14 resource "azurerm_storage_account" "storageAppSC" {
20 }
21
22 # Create Storage Container for public static files (CSS, JS, images)
23 resource "azurerm_storage_container" "publicFiles" {
24     name                = "public-files"
25     storage_account_name = azurerm_storage_account.storageAppSC.name
26     container_access_type = "blob" # Accès public en lecture seule
27 }
28
29 # Create Storage Container for private user uploads
30 resource "azurerm_storage_container" "privateFiles" {
31     name                = "private-files"
32     storage_account_name = azurerm_storage_account.storageAppSC.name
33     container_access_type = "private" # Accès privé uniquement
34 }
35
36 # Create Storage Container for application Logs
37 resource "azurerm_storage_container" "appLogs" {
38     name                = "app-logs"
39     storage_account_name = azurerm_storage_account.storageAppSC.name
40     container_access_type = "private" # Accès privé uniquement
41 }
42
43 # Create Storage Container for backup files
44 resource "azurerm_storage_container" "backupFiles" {
45     name                = "backup-files"
46     storage_account_name = azurerm_storage_account.storageAppSC.name
47     container_access_type = "private" # Accès privé uniquement
48 }
49
```

## C – Déployer une base de donnée :

Avant déploiement de notre base de données on va créer un serveur avec notre ressource azure disponible. La sécurisation se fait de manière progressive, en commençant par allouer les ressources nécessaires à la création d'un pare-feu et en terminant par la mise en place d'une règle réseau afin de protéger l'accès à la base de données.

### 1. Serveur Mysql :

Créer un MySQL Server sur Azure nous permettra d'héberger une base de données relationnelle dans le cloud avec haute disponibilité.

```
# Create MySQL Server
resource "azurerm_mysql_flexible_server" "mysqlAppSC" {
  name = "mysql-app-sc"
  resource_group_name = azurerm_resource_group.rgAppSC.name
  location = azurerm_resource_group.rgAppSC.location

  administrator_login = var.mysql_admin_username
  administrator_password = var.mysql_admin_password

  sku_name = var.mysql_sku_name
  version = var.mysql_version

  storage {
    size_gb = var.mysql_storage_size_gb
  }

  backup_retention_days = var.mysql_backup_retention_days

  # Ignorer les changements de zone (non modifiable après création)
  lifecycle {
    ignore_changes = [
      zone
    ]
  }

  tags = var.tags
}
```

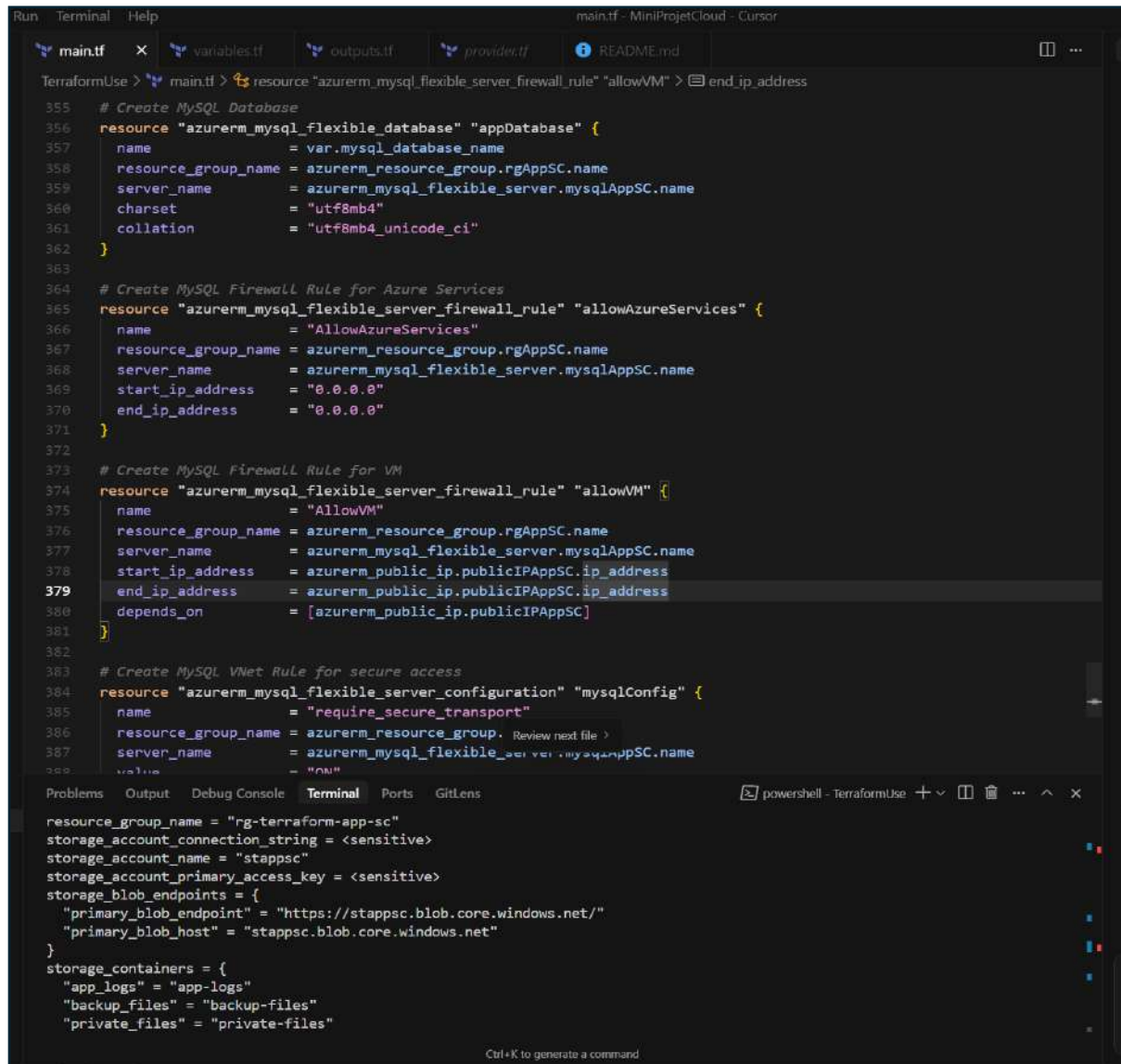
### 2. Creation de la BDD Mysql :

Pour créer notre ressource correspondant à la bdd MySQL on prend le bloc de code qui nous intéresse via le site de terraform puis on l'insère dans le main.tf.

Dans ce bloc on va créer notre base de données **var.mysql\_database\_name** sur notre serveur Azure **Database for MySQL Flexible Server** existant.

### 3. Creation des pare-feux pour le serveur Mysql et la VM:

Une fois nos ressources pour le serveur cloud configuré et la base de données créée avec Azure pour Mysql on peut commencer à sécuriser notre serveur dédié avec un pare-feu.



```
main.tf | MiniProjetCloud - Cursor
TerraformUse > main.tf > resource "azurerm_mysql_flexible_server_firewall_rule" "allowVM" > end_ip_address

355 # Create MySQL Database
356 resource "azurerm_mysql_flexible_database" "appDatabase" {
357   name                = var.mysql_database_name
358   resource_group_name = azurerm_resource_group.rgAppSC.name
359   server_name         = azurerm_mysql_flexible_server.mysqlAppSC.name
360   charset             = "utf8mb4"
361   collation           = "utf8mb4_unicode_ci"
362 }
363
364 # Create MySQL Firewall Rule for Azure Services
365 resource "azurerm_mysql_flexible_server_firewall_rule" "allowAzureServices" {
366   name                = "AllowAzureServices"
367   resource_group_name = azurerm_resource_group.rgAppSC.name
368   server_name         = azurerm_mysql_flexible_server.mysqlAppSC.name
369   start_ip_address    = "0.0.0.0"
370   end_ip_address      = "0.0.0.0"
371 }
372
373 # Create MySQL Firewall Rule for VM
374 resource "azurerm_mysql_flexible_server_firewall_rule" "allowVM" {
375   name                = "AllowVM"
376   resource_group_name = azurerm_resource_group.rgAppSC.name
377   server_name         = azurerm_mysql_flexible_server.mysqlAppSC.name
378   start_ip_address    = azurerm_public_ip.publicIPAppSC.ip_address
379   end_ip_address      = azurerm_public_ip.publicIPAppSC.ip_address
380   depends_on          = [azurerm_public_ip.publicIPAppSC]
381 }
382
383 # Create MySQL VNet Rule for secure access
384 resource "azurerm_mysql_flexible_server_configuration" "mysqlConfig" {
385   name                = "require_secure_transport"
386   resource_group_name = azurerm_resource_group.rgAppSC.name
387   server_name         = azurerm_mysql_flexible_server.mysqlAppSC.name
388   value              = "ON"
389 }

Problems | Output | Debug Console | Terminal | Ports | GitLens
resource_group_name = "rg-terraform-app-sc"
storage_account_connection_string = <sensitive>
storage_account_name = "stappsc"
storage_account_primary_access_key = <sensitive>
storage_blob_endpoints = {
  "primary_blob_endpoint" = "https://stappsc.blob.core.windows.net/"
  "primary_blob_host" = "stappsc.blob.core.windows.net"
}
storage_containers = {
  "app_logs" = "app-logs"
  "backup_files" = "backup-files"
  "private_files" = "private-files"
}
```

On crée une règle pare-feu sur MySQL Flexible Server Azure pour permettre aux **services Azure** de se connecter à ta base de données.

Dans Azure « 0.0.0.0. » comme IP de début et de fin permettent uniquement aux IP internes d’Azure d’accéder à notre serveur.

De même pour la VM on crée aussi une règle de pare-feu sur Azure MySQL Serveur pour permettre un accès unique à notre VM spécifié via son adresse IP publique.

## D – Déployer un backend :

Pour le backend de notre application on a utilisé Node.JS, MySQL et puisque l'on a déjà crée la ressource dans la VM on va procéder ainsi pour le déploiement :

1)

```
main.tf deploy-to-vm.sh X variables.tf terraform.tfvars outputs.tf README.m
React-CRUD > backend > deploy-to-vm.sh
You: 21 minutes ago | 1 author (you)
1 #!/bin/bash
2
3 # Script de déploiement sur VM Azure
4 # Utilisation: ./deploy-to-vm.sh
5
6 echo "Déploiement sur VM Azure (52.169.106.107)"
7
8 # Variables
9 VM_IP="52.169.106.107"
10 VM_USER="azureuser"
11
12 # Vérifier la connectivité SSH
13 echo "Vérification de la connectivité SSH..."
14 if ! ssh -o BatchMode=yes -o ConnectTimeout=5 $VM_USER@$VM_IP exit 2>/dev/null; then
15     echo "Impossible de se connecter à la VM Azure"
16     echo "Vérifiez que vous avez accès SSH à $VM_USER@$VM_IP"
17     exit 1
18 fi
19
20 echo "Connexion SSH établie"
21
22 # Installer Node.js sur la VM
23 echo "Installation de Node.js sur la VM..."
24 ssh $VM_USER@$VM_IP << 'EOF'
25 # Vérifier si Node.js est déjà installé
26 if command -v node &> /dev/null; then
27     echo "Node.js déjà installé: $(node --version)"
28 else
29     echo "Installation de Node.js..."
30     curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
31     sudo apt-get install -y nodejs
32     echo "Node.js installé: $(node --version)"
33 fi
34
35 # Installer MySQL client
36 if ! command -v mysql &> /dev/null; then
37     echo "Installation du client MySQL..."
38     sudo apt-get install -y mysql-client-core-8.0
39 fi
```

2)

```
CRUD > backend > deploy-to-vm.sh
EOF
# Créer le répertoire backend et le fichier .env
echo "Configuration de l'environnement backend..."
ssh $VM_USER@$VM_IP << 'EOF'
# Créer le répertoire backend
mkdir -p ~/backend
cd ~/backend
# Créer le fichier .env
cat > .env << 'ENVEOF'
PORT=5178
NODE_ENV=development
DB_HOST=mysql-app-sc.mysql.database.azure.com
DB_PORT=3306
DB_NAME=appdb
DB_USER=mysqladmin
DB_PASSWORD=P@ssw0rd123!
FRONTEND_URL=http://52.169.106.107:5193
DB_SSL=true
WEBSITE_NODE_DEFAULT_VERSION=18-lts
ENVEOF
echo "Fichier .env créé"
# Créer un package.json minimal
cat > package.json << 'PKGEOF'
{
  "name": "backend-azure",
  "version": "1.0.0",
  "description": "Backend pour VM Azure",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "node server.js"
  },
  "dependencies": {
    "express": "^4.18.2",
    "cors": "^2.8.5",
    "helmet": "^6.0.1",
    "express-rate-limit": "^6.7.0",
    "morgan": "^1.10.0",
    "dotenv": "^16.0.3",
    "sequelize": "^6.29.0",

```

3)

```
    "mysql2": "^3.2.0",
    "multer": "^1.4.5-lts.1"
  }
}
PKGEOF
echo "Package.json créé"
EOF
# Copier les fichiers du projet
echo "Copie des fichiers du projet..."
scp -r React-CRUD/backend/* $VM_USER@$VM_IP:~/backend/
# Installer les dépendances et démarrer le serveur
echo "Installation des dépendances..."
ssh $VM_USER@$VM_IP << 'EOF'
cd ~/backend
npm install
```

Avec la commande « **scp** React-CRUD/backend/deploy-to-azure.sh  
azureuser@52.169.106.107:~/deploy-to-azure.sh »

L'application sera donc facilement déployable avec le script **deploy-to-vm.sh** en VM.

4)

```
output "vm_applications_urls" {
  description = "URLs des applications sur la VM"
  value = {
    backend_health = "http://${azurerm_public_ip.publicIPAppSC.ip_address}:5170/health"
    backend_api    = "http://${azurerm_public_ip.publicIPAppSC.ip_address}:5170/api/products"
    frontend       = "http://${azurerm_public_ip.publicIPAppSC.ip_address}:5193"
    ssh_access     = "ssh ${var.vm_admin_username}@${azurerm_public_ip.publicIPAppSC.ip_address}"
  }
}
```

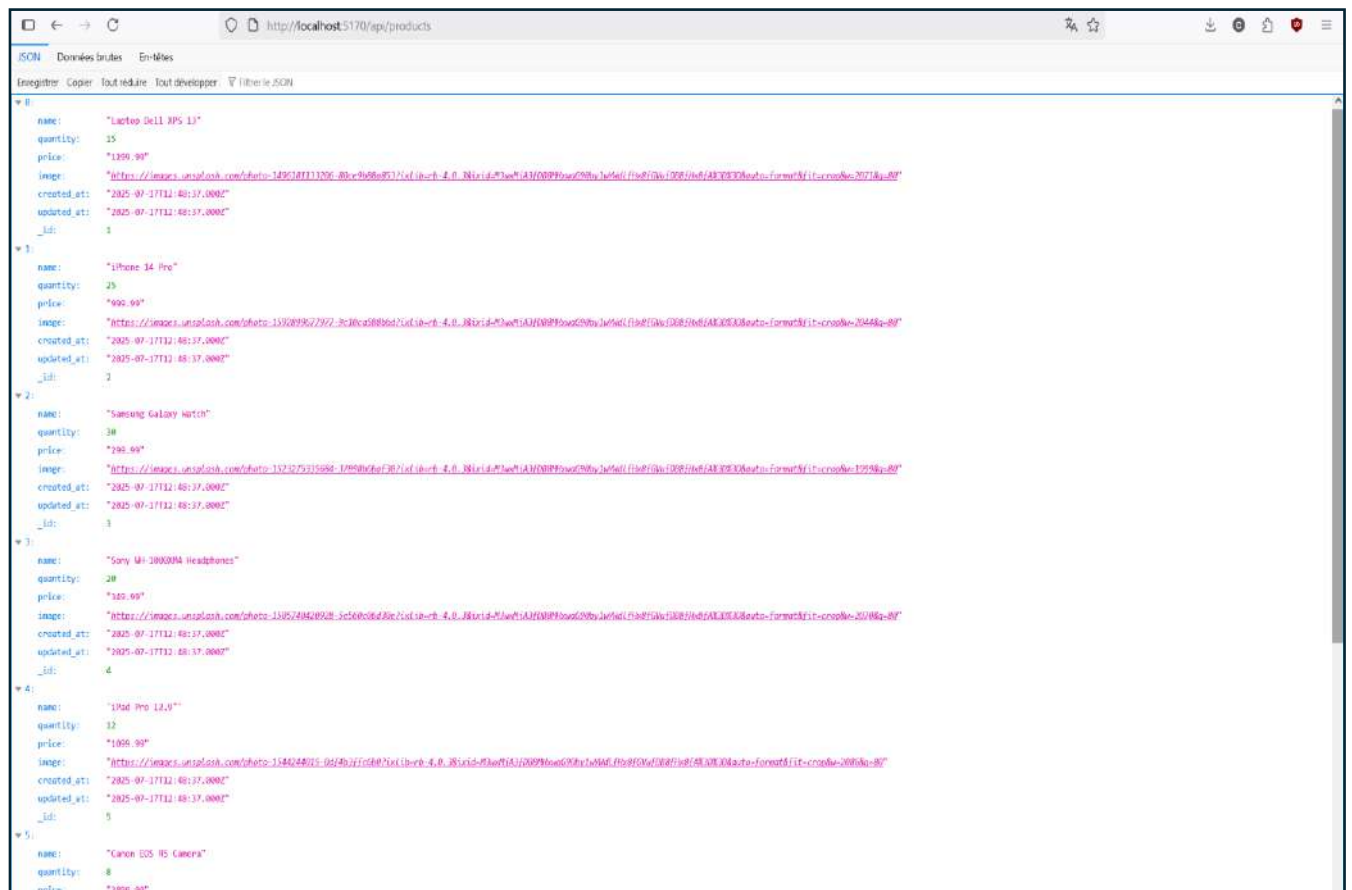
Connection via la VM avec : ssh azureuser@52.169.106.107

## Etape3 :

### A – Implementation du CRUD :

Concernant le CRUD étant donné qu'il y a eu des soucis dans l'attribution des ports on affiche ce qu'on devrait avoir dans la VM, on simulera via notre localhost le déploiement de l'application ainsi que son fonctionnement avec la base de donnée.

Tout d'abord on va se connecter au backend avec via le port 5170 :





Dans le code toute nos routes sont aussi bien définies, on le voit avec les méthodes :



```
vars    outputs.tf    JS productController.js X    README.md TerraformUse

- CRUD > backend > controllers > JS productController.js > ...

// Create new product
export const createProduct = asyncHandler(async (req, res) => {
  const { name, quantity, price } = req.body;

  // Use uploaded image or default placeholder
  const image = req.imageUrl || '/uploads/default-product.png';

  const product = await Product.create({
    name,
    quantity,
    price,
    image
  });

  // Return product data directly for frontend compatibility
  res.status(201).json(product);
});

// Update product
export const updateProduct = asyncHandler(async (req, res) => {
  const { id } = req.params;
  const updateData = req.body;

  const product = await Product.findByPk(id);

  if (!product) {
    throw new ApiError('Product not found', 404);
  }

  // Store product reference for middleware
  req.product = product;

  // Use uploaded image if available
  if (req.imageUrl) {
    updateData.image = req.imageUrl;
  }

  await product.update(updateData);

  // Return updated product data directly for frontend compatibility
  res.status(200).json(product);
});

// Delete product
```

Ensuite on va voir les routes disponibles sur le port 5170 afin qu'on puisse exécuter nos requête api.



```

t-CRUD > backend > routes > productRoutes.js > ...
import {
  validateProduct,
  validateProductUpdate,
  validateId
} from '../middleware/validation.js';
import {
  uploadSingle,
  processImage,
  deleteOldImage,
  handleUploadError
} from '../middleware/uploadMiddleware.js';

const router = express.Router();

// Routes
router.route('/')
  .get(getProducts) // GET /api/products
  .post(uploadSingle, handleUploadError, processImage, validateProduct, createProduct); // POST /api/products

router.route('/stats')
  .get(getProductStats); // GET /api/products/stats

router.route('/bulk-delete')
  .delete(deleteMultipleProducts); // DELETE /api/products/bulk-delete

router.route('/:id')
  .get(validateId, getProduct) // GET /api/products/:id
  .put(validateId, uploadSingle, handleUploadError, deleteOldImage, processImage, validateProductUpdate, updateProduct) // PUT /api/products/:id
  .delete(validateId, deleteProduct); // DELETE /api/products/:id

export default router;

```

Maintenant on va vérifier le bon fonctionnement des API à travers un frontend développer avec React :

http://localhost:5193/create

### Create a Product

Name \*

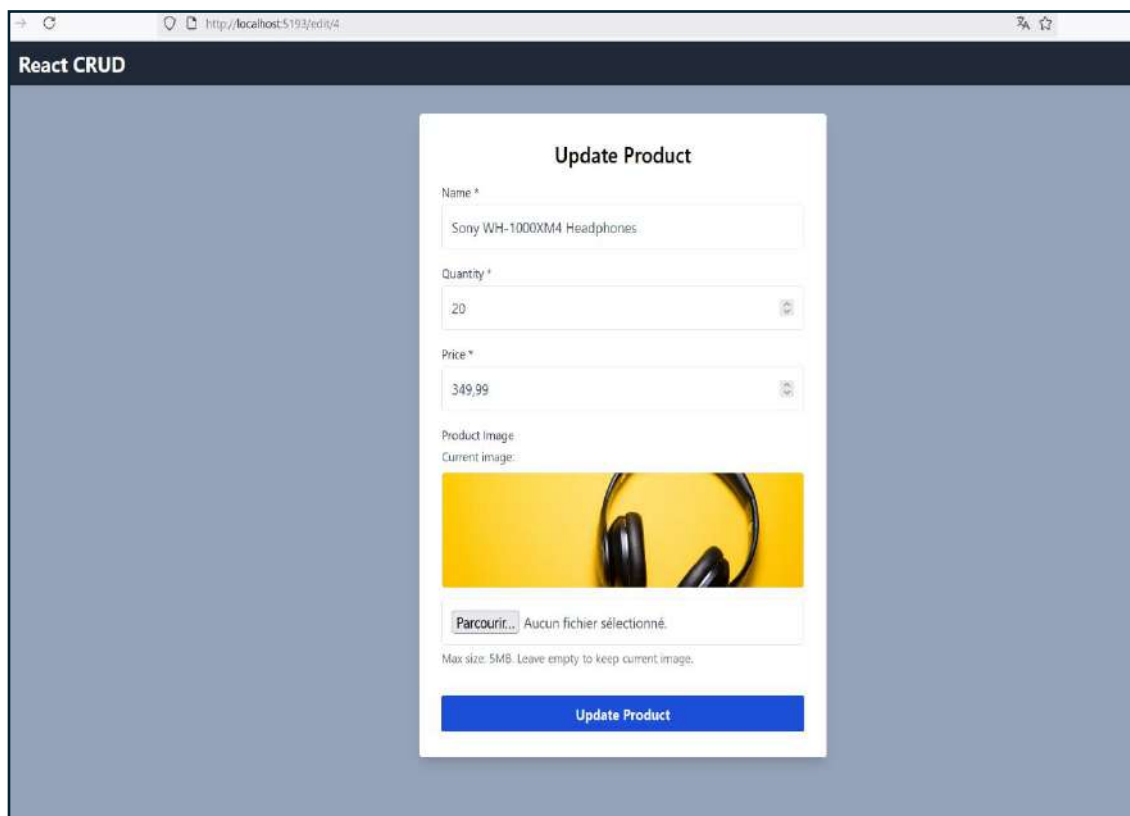
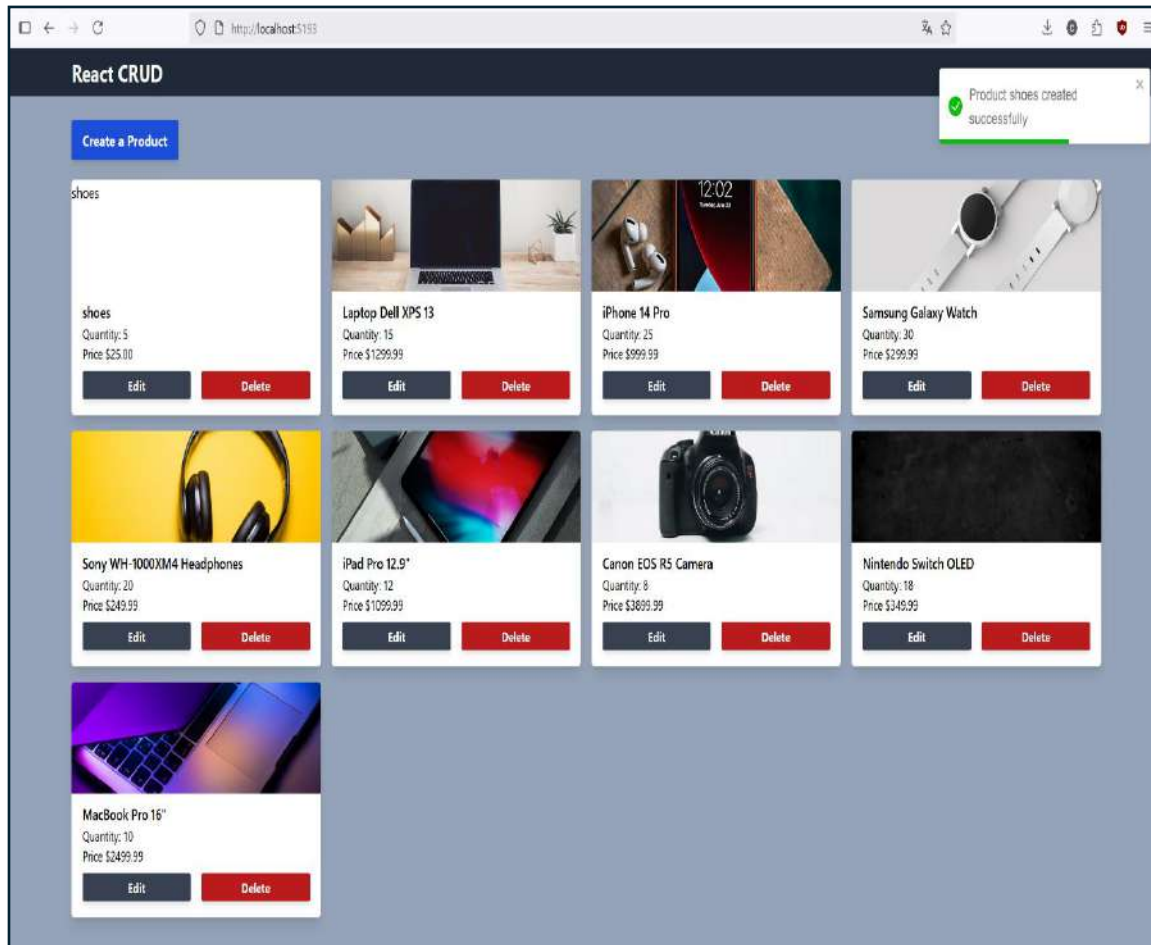
Quantity \*

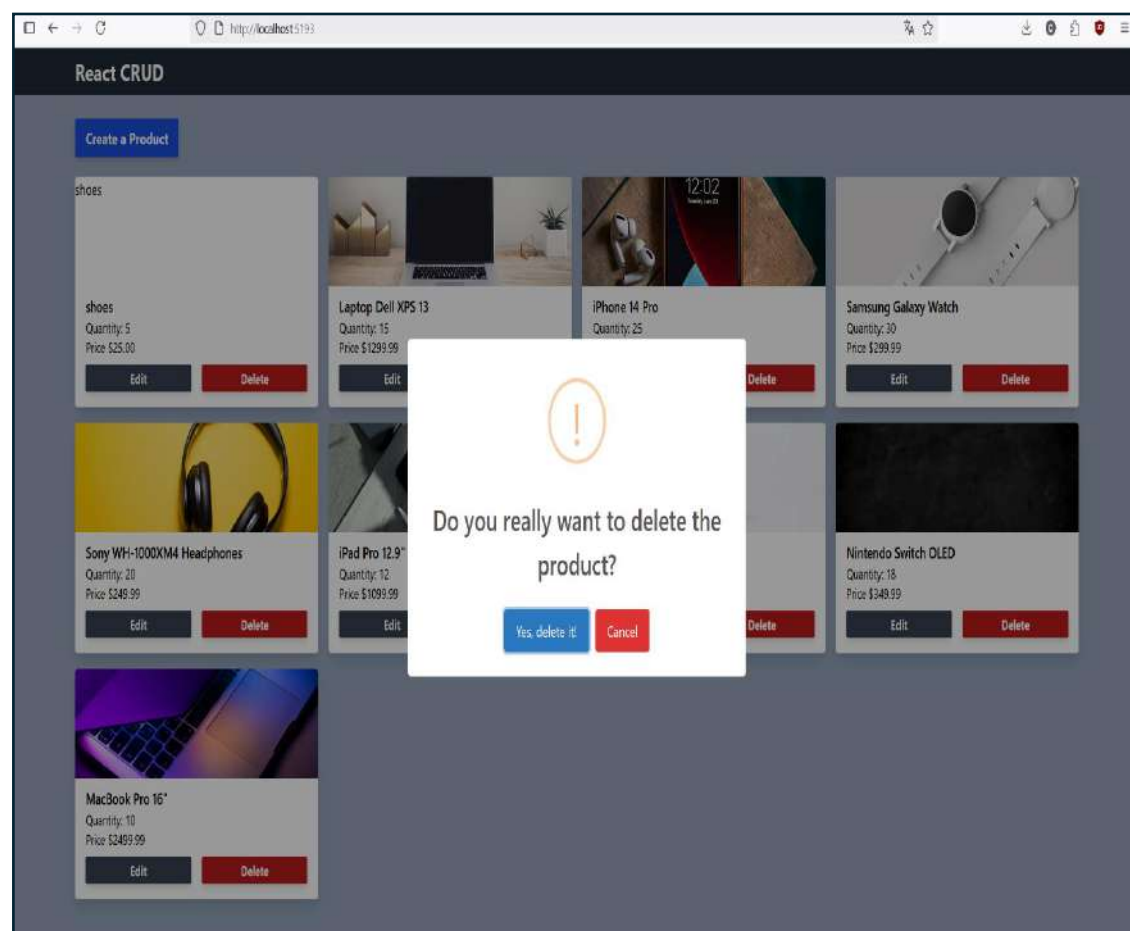
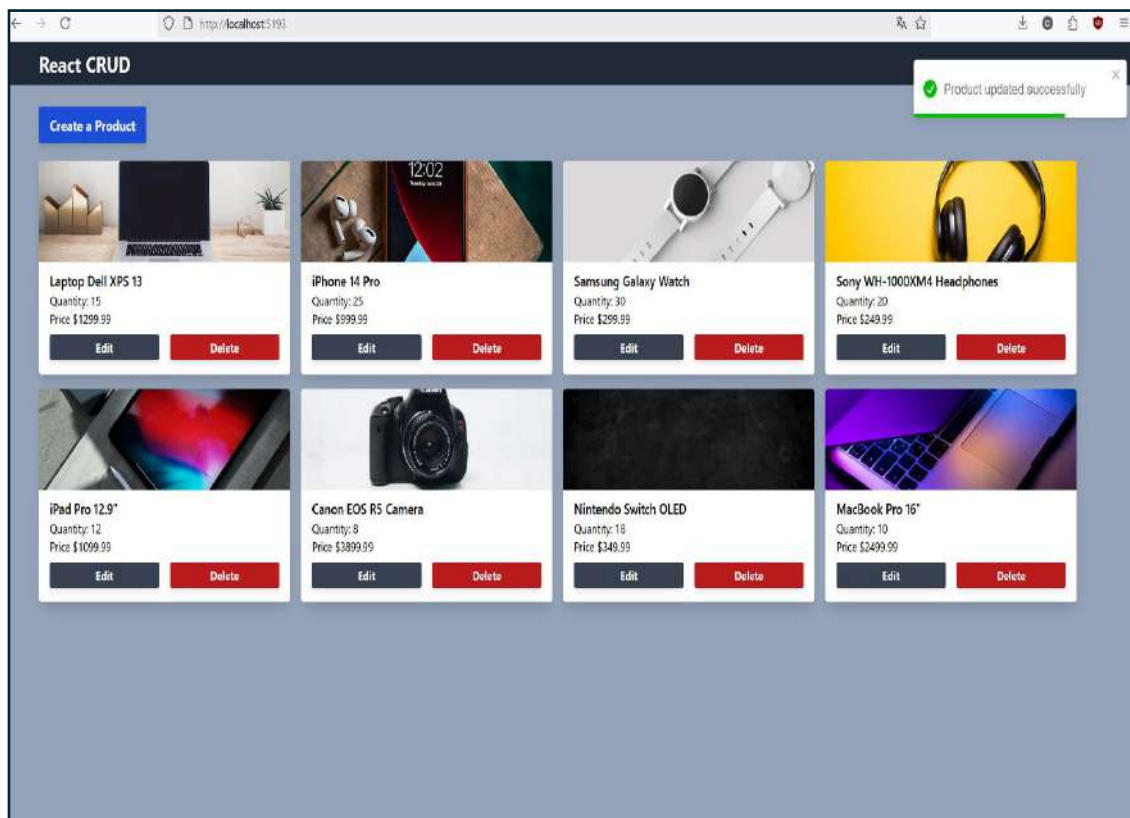
Price \*

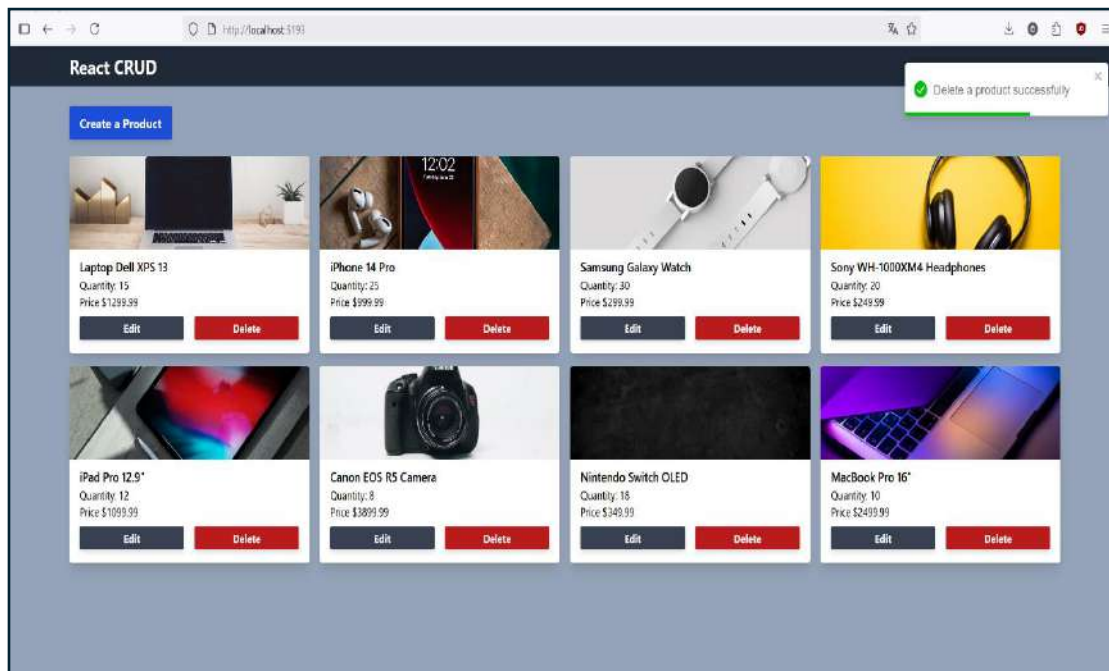
Product Image

Aucun fichier sélectionné.

Max size: 5MB. Supported formats: JPG, PNG, GIF, WebP







## Etape4 :

### A – Gestion des Variables et Outputs

Le fichier **variables.tf** est un composant essentiel de Terraform qui permet de définir les variables d'entrée pour notre infrastructure.

```

103 }
104
105 variable "mysql_sku_name" {
106   description = "MySQL SKU name (B_Standard_B1ms, GP_Standard_D2s_v3, etc.)"
107   type        = string
108   default     = "B_Standard_B1ms"
109 }
110
111 variable "mysql_version" {
112   description = "MySQL version"
113   type        = string
114   default     = "8.0.21"
115 }
116
117 variable "mysql_storage_size_gb" {
118   description = "MySQL storage size in GB"
119   type        = number
120   default     = 20
121 }
122
123 variable "mysql_backup_retention_days" {
124   description = "MySQL backup retention days"
125   type        = number
126   default     = 7
127 }
128
129 variable "mysql_database_name" {
130   description = "Name of the MySQL database"
131   type        = string
132   default     = "appdb"
133 }
134
135 storage_account_primary_access_key = <sensitive>
136 storage_blob_endpoints = {
137   "primary_blob_endpoint" = "https://stappsc.blob.core.windows.net/"
138   "primary_blob_host"     = "stappsc.blob.core.windows.net"
139 }
140
141 storage_containers = {
142   "app_logs" = "app-logs"
143   "backup_files" = "backup-files"
144   "private_files" = "private-files"
145   "public_files" = "public-files"
146 }
147
148 vm_name = "vm-app-sc"
149 vm_public_ip = "52.169.106.107"
150 vm_ssh_connection = "ssh azureuser@52.169.106.107"
151
152 PS C:\Users\Stalenes\Desktop\H9_L3_DevMob\CloudComputing\MiniProjetCloud\TerraformUse>
  
```

Il sert plusieurs objectifs importants :

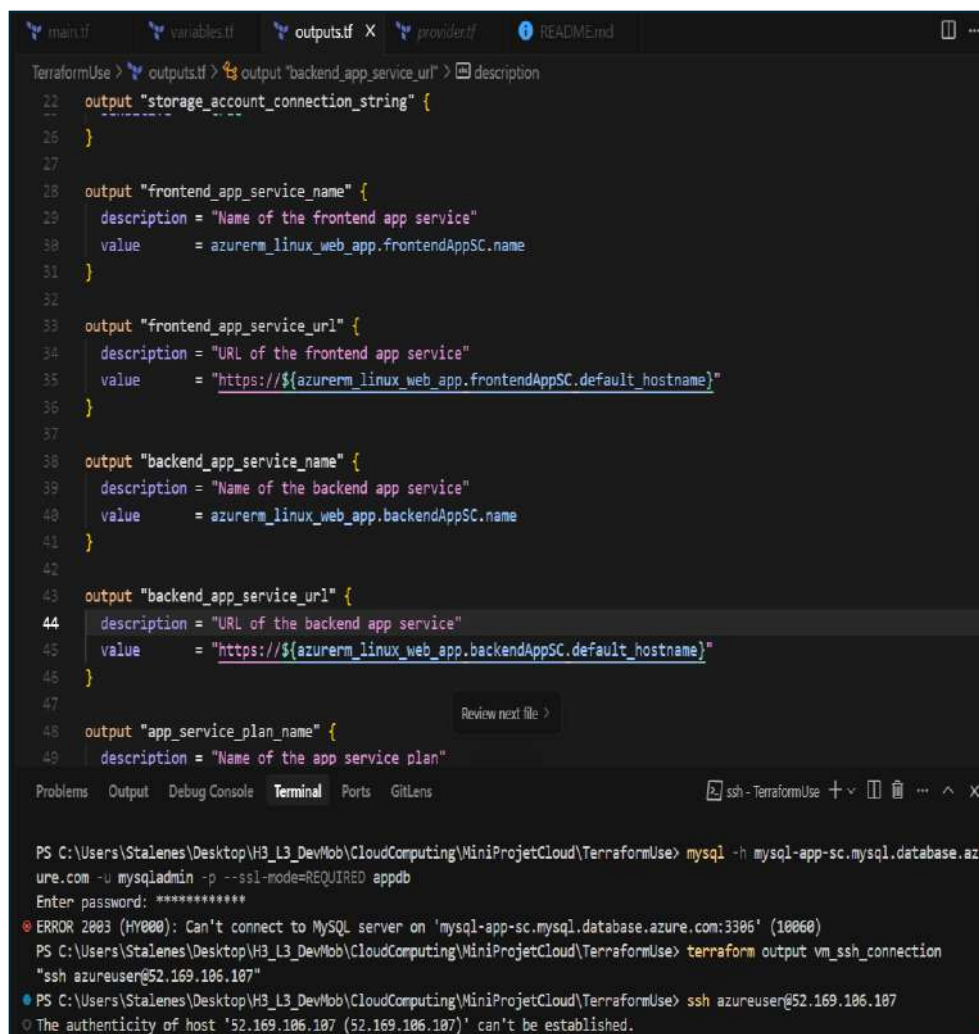
- **Définition des variables d'entrée :**

Le fichier `variables.tf` contient les déclarations de toutes les variables que votre configuration Terraform peut accepter. Chaque variable est définie avec sa structure, son type, et ses contraintes.

- **Paramétrage flexible :**

Il permet de rendre votre code Terraform réutilisable en évitant les valeurs codées en dur. Les valeurs de ces variables peuvent ensuite être fournies via des fichiers `tfvars`, des variables d'environnement. Notre sécurité est donc accrue car on évite d'exposer les valeurs sensibles.

Le fichier **outputs.tf** dans Terraform sert à définir les valeurs de sortie de votre infrastructure.



```
main.tf  variables.tf  outputs.tf X  provider.tf  README.md  ...
TerraformUse > outputs.tf > output "backend_app_service_url" > description
22  output "storage_account_connection_string" {
26  }
27
28  output "frontend_app_service_name" {
29    description = "Name of the frontend app service"
30    value      = azurelinux_web_app.frontendAppSC.name
31  }
32
33  output "frontend_app_service_url" {
34    description = "URL of the frontend app service"
35    value      = "https://${azurelinux_web_app.frontendAppSC.default_hostname}"
36  }
37
38  output "backend_app_service_name" {
39    description = "Name of the backend app service"
40    value      = azurelinux_web_app.backendAppSC.name
41  }
42
43  output "backend_app_service_url" {
44    description = "URL of the backend app service"
45    value      = "https://${azurelinux_web_app.backendAppSC.default_hostname}"
46  }
47
48  output "app_service_plan_name" {
49    description = "Name of the app service plan"
50  }
Review next file >
Problems  Output  Debug Console  Terminal  Ports  GitLens
ssh - TerraformUse
PS C:\Users\Stalenes\Desktop\H3_L3_DevMob\CloudComputing\MiniProjetCloud\TerraformUse> mysql -h mysql-app-sc.mysql.database.azure.com -u mysqladmin -p --ssl-mode=REQUIRED appdb
Enter password: *****
ERROR 2003 (HY000): Can't connect to MySQL server on 'mysql-app-sc.mysql.database.azure.com:3306' (10060)
PS C:\Users\Stalenes\Desktop\H3_L3_DevMob\CloudComputing\MiniProjetCloud\TerraformUse> terraform output vm_ssh_connection
"ssh azureuser@52.169.106.107"
PS C:\Users\Stalenes\Desktop\H3_L3_DevMob\CloudComputing\MiniProjetCloud\TerraformUse> ssh azureuser@52.169.106.107
The authenticity of host '52.169.106.107 (52.169.106.107)' can't be established.
```



Il permet d'extraire et d'afficher des informations importantes sur les ressources créées, comme des adresses IP ou des URL, etc....

```
TerraformUse > outputs.tf > output "resource_group_name" > value
10   output "resource_group_location" {
11     value = azurerm_resource_group.rgAppSC.location
12   }
13
14
15   # =====
16   # VM AZURE - IPs et Accès
17   # =====
18
19   output "vm_name" {
20     description = "Name of the virtual machine"
21     value       = azurerm_linux_virtual_machine.vmAppSC.name
22   }
23
24   output "vm_public_ip" {
25     description = "Public IP address of the virtual machine"
26     value       = azurerm_public_ip.publicIPAppSC.ip_address
27   }
28
29   output "vm_private_ip" {
30     description = "Private IP address of the virtual machine"
31     value       = azurerm_network_interface.nicAppSC.private_ip_address
32   }
33
34   output "vm_ssh_connection" {
35     description = "SSH connection string for the virtual machine"
36     value       = "ssh ${var.vm_admin_username}@${azurerm_public_ip.publicIPAppSC.ip_address}"
37   }
38
39   output "vm_applications_urls" {
40     description = "URLs des applications sur la VM"
41     value = {
42       backend_health = "http://${azurerm_public_ip.publicIPAppSC.ip_address}:5170/health"
43       backend_api     = "http://${azurerm_public_ip.publicIPAppSC.ip_address}:5170/api/products"
44       frontend        = "http://${azurerm_public_ip.publicIPAppSC.ip_address}:5193"
45       ssh_access      = "ssh ${var.vm_admin_username}@${azurerm_public_ip.publicIPAppSC.ip_address}"
46     }
47   }
48
```

```
TerraformUse > outputs.tf > output "mysql_server_name"
88   # =====
89   # MYSQL DATABASE - Connexion
90   # =====
91
92   output "mysql_server_name" {
93     description = "Name of the MySQL server"
94     value       = azurerm_mysql_flexible_server.mysqlAppSC.name
95   }
96
97   output "mysql_server_fqdn" {
98     description = "FQDN of the MySQL server"
99     value       = azurerm_mysql_flexible_server.mysqlAppSC.fqdn
100  }
101
102   output "mysql_database_name" {
103     description = "Name of the MySQL database"
104     value       = azurerm_mysql_flexible_database.appDatabase.name
105   }
106
107   output "mysql_admin_username" {
108     description = "MySQL administrator username"
109     value       = var.mysql_admin_username
110   }
111
112   output "mysql_connection_string" {
113     description = "MySQL connection string"
114     value       = "mysql://${var.mysql_admin_username}:${var.mysql_admin_password}@${azurerm_mysql_flexible_server.mysqlAppSC.fqdn}"
115     sensitive   = true
116   }
117
118   output "mysql_connection_info" {
119     description = "Informations de connexion MySQL"
120     value = {
121       host     = azurerm_mysql_flexible_server.mysqlAppSC.fqdn
122       port     = 3306
123       database = azurerm_mysql_flexible_database.appDatabase.name
124       username = var.mysql_admin_username
125       ssl_mode = "require"
126     }
127   }
128
```



```

# STORAGE ACCOUNT - URLs et Accès
# =====

output "storage_account_name" {
  description = "Name of the created storage account"
  value       = azurerm_storage_account.storageAppSC.name
}

output "storage_account_primary_access_key" {
  description = "Primary access key for the storage account"
  value       = azurerm_storage_account.storageAppSC.primary_access_key
  sensitive   = true
}

output "storage_account_connection_string" {
  description = "Connection string for the storage account"
  value       = azurerm_storage_account.storageAppSC.primary_connection_string
  sensitive   = true
}

output "storage_containers" {
  description = "Storage containers created"
  value = {
    public_files = azurerm_storage_container.publicFiles.name
    private_files = azurerm_storage_container.privateFiles.name
    app_logs     = azurerm_storage_container.appLogs.name
    backup_files = azurerm_storage_container.backupFiles.name
  }
}

output "storage_blob_endpoints" {
  description = "Storage blob endpoints"
  value = {
    primary_blob_endpoint = azurerm_storage_account.storageAppSC.primary_blob_endpoint
    primary_blob_host     = azurerm_storage_account.storageAppSC.primary_blob_host
  }
}

output "storage_urls" {
  description = "URLs des services de stockage"
  value = {
    public_files_direct = "${azurerm_storage_account.storageAppSC.primary_blob_endpoint}public-files/"
    private_files       = "${azurerm_storage_account.storageAppSC.primary_blob_endpoint}private-files/"
    app_logs            = "${azurerm_storage_account.storageAppSC.primary_blob_endpoint}app-logs/"
    backup_files        = "${azurerm_storage_account.storageAppSC.primary_blob_endpoint}backup-files/"
  }
}

```

```

TerraformUse > outputs.tf > output "mysql_server_name"
280
281 # =====
282 # RÉSUMÉ INFRASTRUCTURE
283 # =====
284
285 output "infrastructure_summary" {
286   description = "Résumé complet de l'infrastructure"
287   value = {
288     # Informations générales
289     resource_group = azurerm_resource_group.rgAppSC.name
290     location       = azurerm_resource_group.rgAppSC.location
291
292     # VM Azure
293     vm_public_ip    = azurerm_public_ip.publicIPAppSC.ip_address
294     vm_private_ip   = azurerm_network_interface.nicAppSC.private_ip_address
295
296     # App Services
297     frontend_url    = "https://${azurerm_linux_web_app.frontendAppSC.default_hostname}"
298     backend_url     = "https://${azurerm_linux_web_app.backendAppSC.default_hostname}"
299
300     # Base de données
301     mysql_host      = azurerm_mysql_flexible_server.mysqlAppSC.fqdn
302     mysql_database  = azurerm_mysql_flexible_database.appDatabase.name
303
304     # Stockage
305     storage_account = azurerm_storage_account.storageAppSC.name
306     cdn_endpoint    = "https://${azurerm_cdn_endpoint.cdnEndpoint.fqdn}"
307
308     # Ports configurés
309     backend_port    = "5170"
310     frontend_port   = "5193"
311     mysql_port      = "3306"
312   }
313 }

```

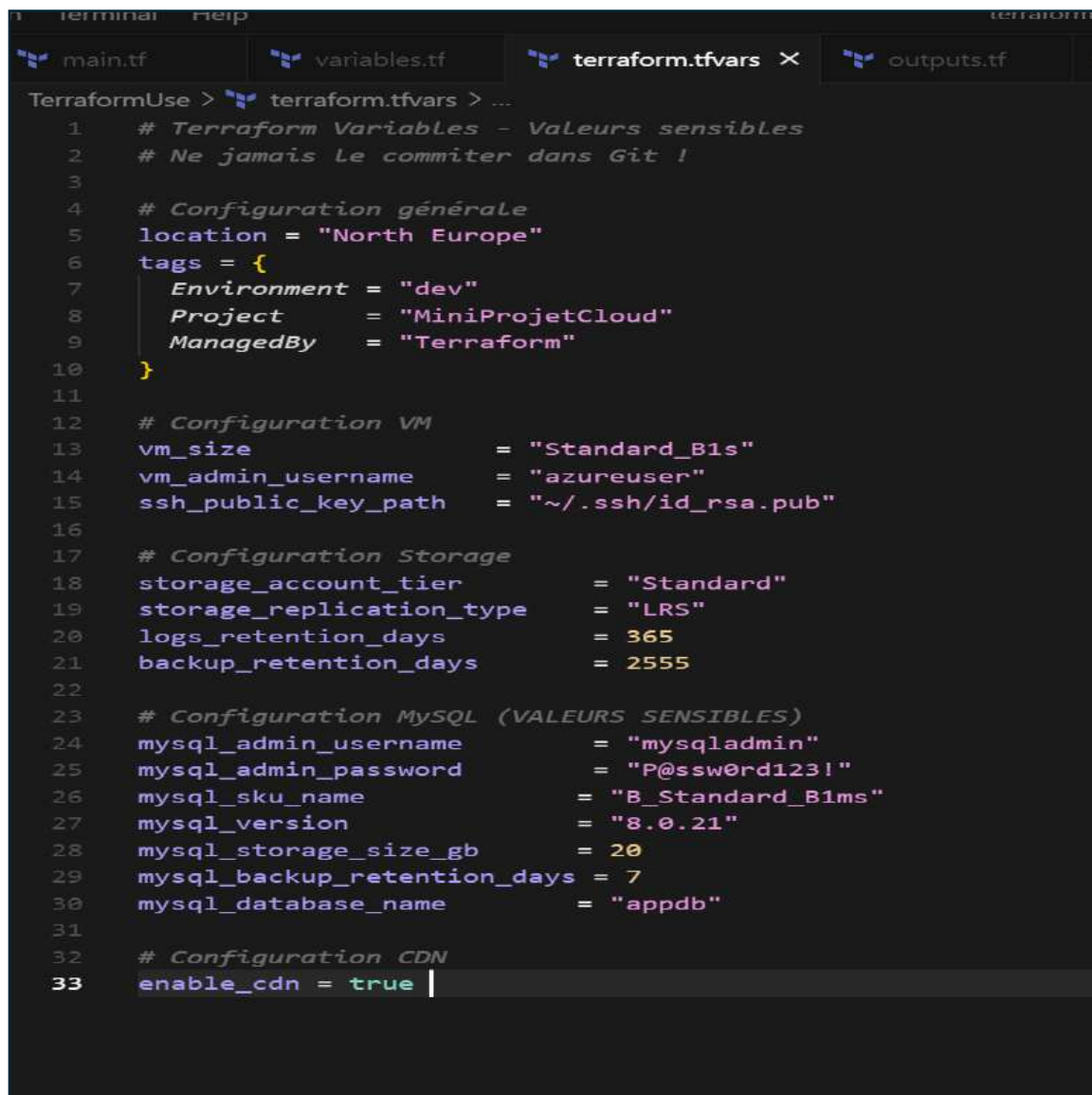
Utilisations courantes :

- **Affichage d'informations** après déploiement (adresses IP, DNS, etc.)
- **Partage de données** entre différents modules Terraform
- **Intégration** avec des scripts externes ou des pipelines CI/CD

Les outputs sont affichés à la fin de **terraform apply** et peuvent être consultés avec **terraform output**.

## **B – Terraforms.tfvars :**

Le fichier terraforms.tfvars permet de séparer la **définition** des variables (dans variables.tf) de leurs **valeurs** (dans .tfvars), offrant ainsi flexibilité et sécurité.



```
1  # Terraform Variables - Valeurs sensibles
2  # Ne jamais Le commiter dans Git !
3
4  # Configuration générale
5  location = "North Europe"
6  tags = {
7    Environment = "dev"
8    Project      = "MiniProjetCloud"
9    ManagedBy    = "Terraform"
10 }
11
12 # Configuration VM
13 vm_size              = "Standard_B1s"
14 vm_admin_username    = "azureuser"
15 ssh_public_key_path  = "~/ssh/id_rsa.pub"
16
17 # Configuration Storage
18 storage_account_tier = "Standard"
19 storage_replication_type = "LRS"
20 logs_retention_days  = 365
21 backup_retention_days = 2555
22
23 # Configuration MySQL (VALEURS SENSIBLES)
24 mysql_admin_username = "mysqladmin"
25 mysql_admin_password = "P@ssw0rd123!"
26 mysql_sku_name       = "B_Standard_B1ms"
27 mysql_version        = "8.0.21"
28 mysql_storage_size_gb = 20
29 mysql_backup_retention_days = 7
30 mysql_database_name   = "appdb"
31
32 # Configuration CDN
33 enable_cdn = true
```

## Etape5 :

Donc on va procéder ainsi pour le déploiement final avec Azure

```
See "man sudo_root" for details.

azureuser@vm-app-sc:~$ mysql -h mysql-app-sc.mysql.database.azure.com -u mysqladmin -p --ssl-mode=REQUIRED appdb

Command 'mysql' not found, but can be installed with:

sudo apt install mysql-client-core-8.0 # version 8.0.41-0ubuntu0.20.04.1, or
mysql-client-core-8.0
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 5091 kB of archives.
After this operation, 74.6 MB of additional disk space will be used.
Get:1 http://azure.archive.ubuntu.com/ubuntu focal-updates/main amd64 mysql-client-core-8.0 amd64 8.0.42-0ubuntu0.20.04.1 [5091 kB]
Fetched 5091 kB in 0s (22.1 MB/s)
Selecting previously unselected package mysql-client-core-8.0.
(Reading database ... 59172 files and directories currently installed.)
Preparing to unpack .../mysql-client-core-8.0_8.0.42-0ubuntu0.20.04.1_amd64.deb ...
Unpacking mysql-client-core-8.0 (8.0.42-0ubuntu0.20.04.1) ...
Setting up mysql-client-core-8.0 (8.0.42-0ubuntu0.20.04.1) ...
Processing triggers for man-db (2.9.1-1) ...
azureuser@vm-app-sc:~$ mysql -h mysql-app-sc.mysql.database.azure.com -u mysqladmin -p --ssl-mode=REQUIRED appdb
Enter password:

Ctrl+K to generate a command
```

Pour procéder à la suppression de notre VM et de toute les ressources en rapport avec notre projet cloud on utilisera la ligne de commande « terraform destroy ».

## Conclusion :

Ce projet a permis la mise en place de pratiques recommandées pour créer une architecture cloud et déployer une application web à l'aide du cloud. Les compétences acquises incluent l'Infrastructure as Code, la gestion d'Azure Cloud, l'automatisation DevOps et le développement full-stack. L'utilisation d'outils comme Terraform, Azure et des scripts d'automatisation, associée à une approche sécurisée et documentée, permet de concevoir, déployer et maintenir des environnements adaptés aux besoins actuels des projets cloud-natifs.