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 le 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 azurerm avec les fonctionnalités de base d'Azure. Puis l'on va créer notre fichier main.tf ou l'on mettra nos instructions pour la création de l'infrastructure. Nos différents fichiers contiendront :

- main.tf - Ressources principales

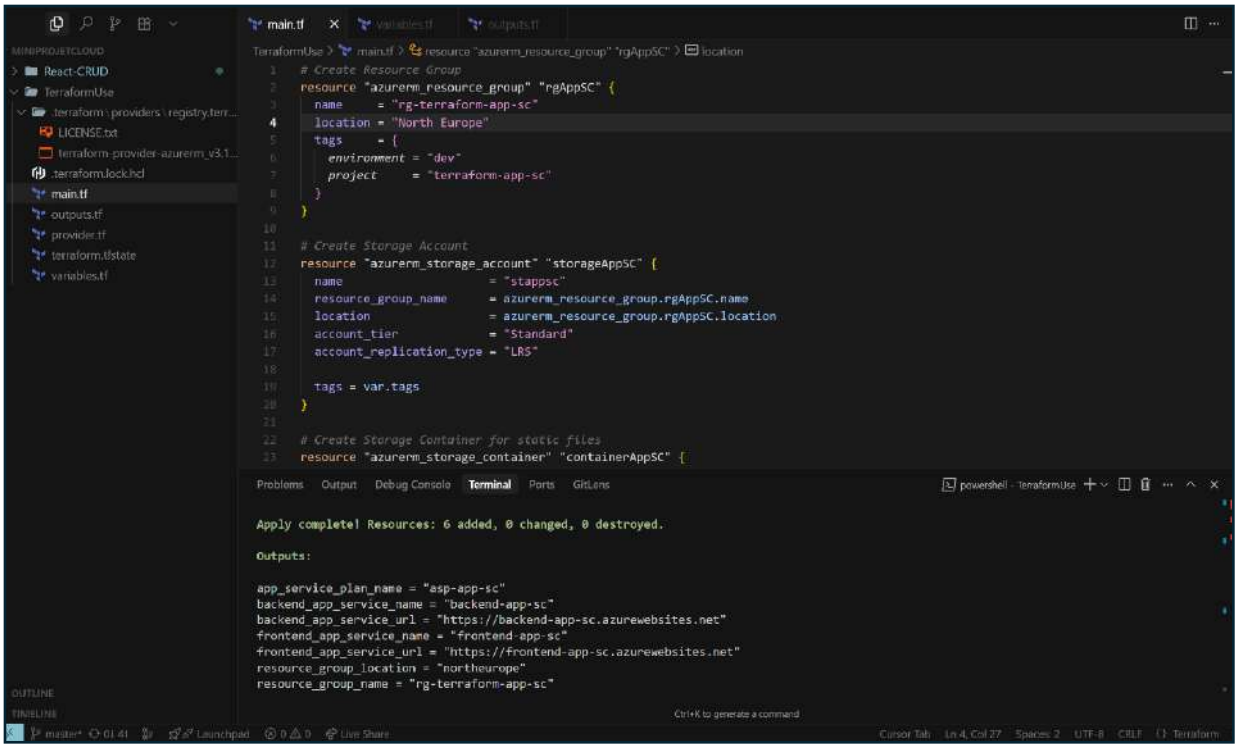
- **Resource Group** : Groupe de ressources principal
- **Storage Account** : Compte de stockage avec container pour fichiers statiques
- **App Service Plan** : Plan d'hébergement Linux (B1)
- **Frontend App Service** : Service web pour le frontend React
- **Backend App Service** : Service web pour le 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 deploiement)
- **terraform apply** (pour déployer nos ressources)

La configuration est prête et on deploiera 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**.

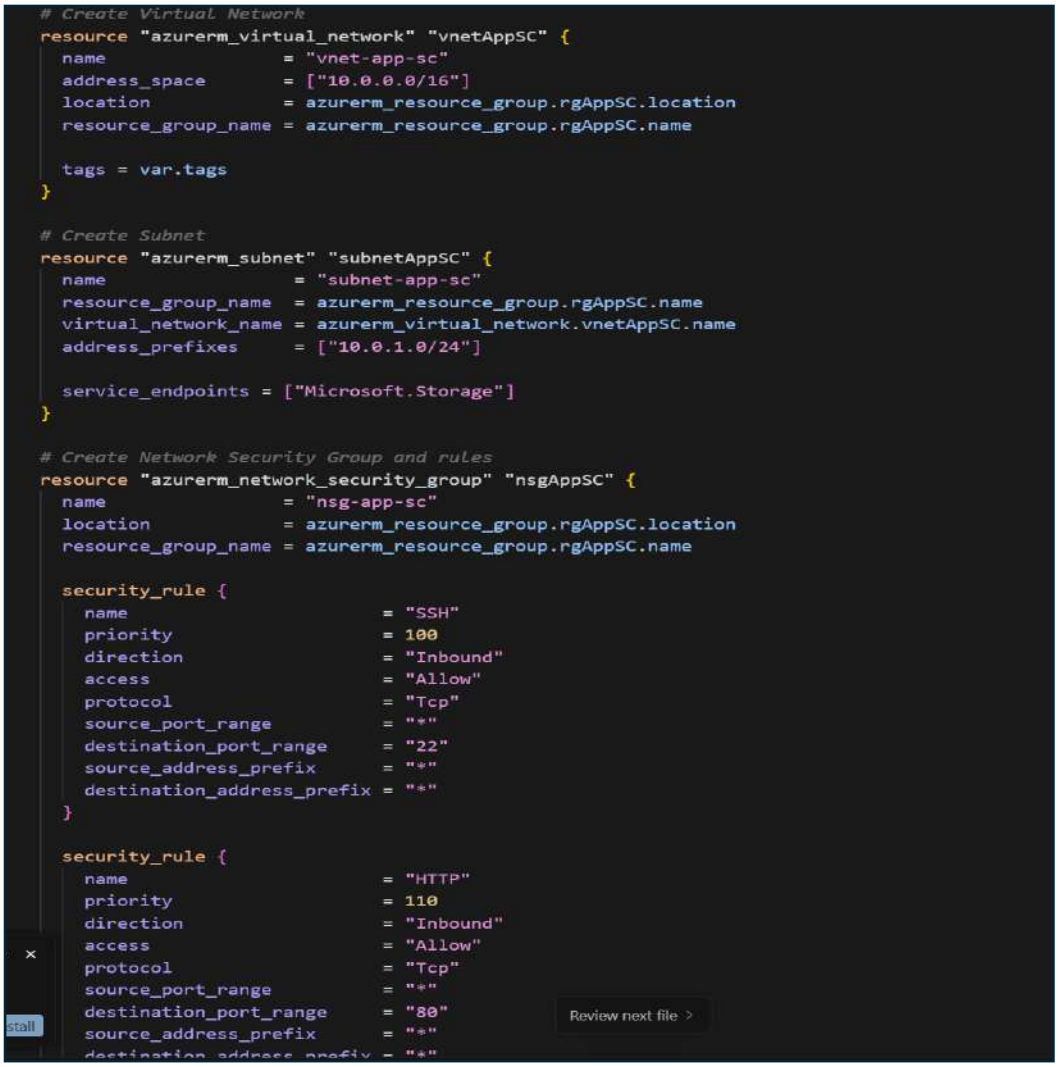
On commence donc par générer et verifier une clef ssh avec ces lignes de commandes dans un terminal :

- 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      = "*"
```

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.

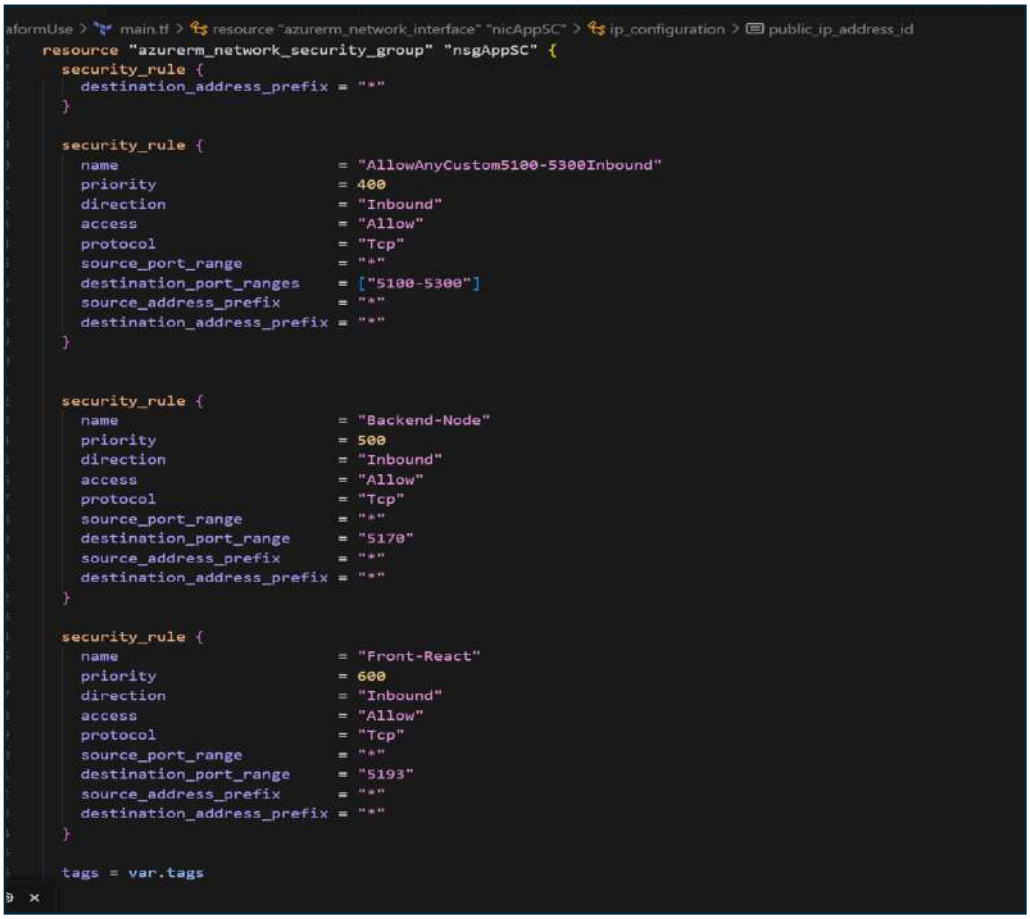
3.Création du NGS

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 crée 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 toute 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 :



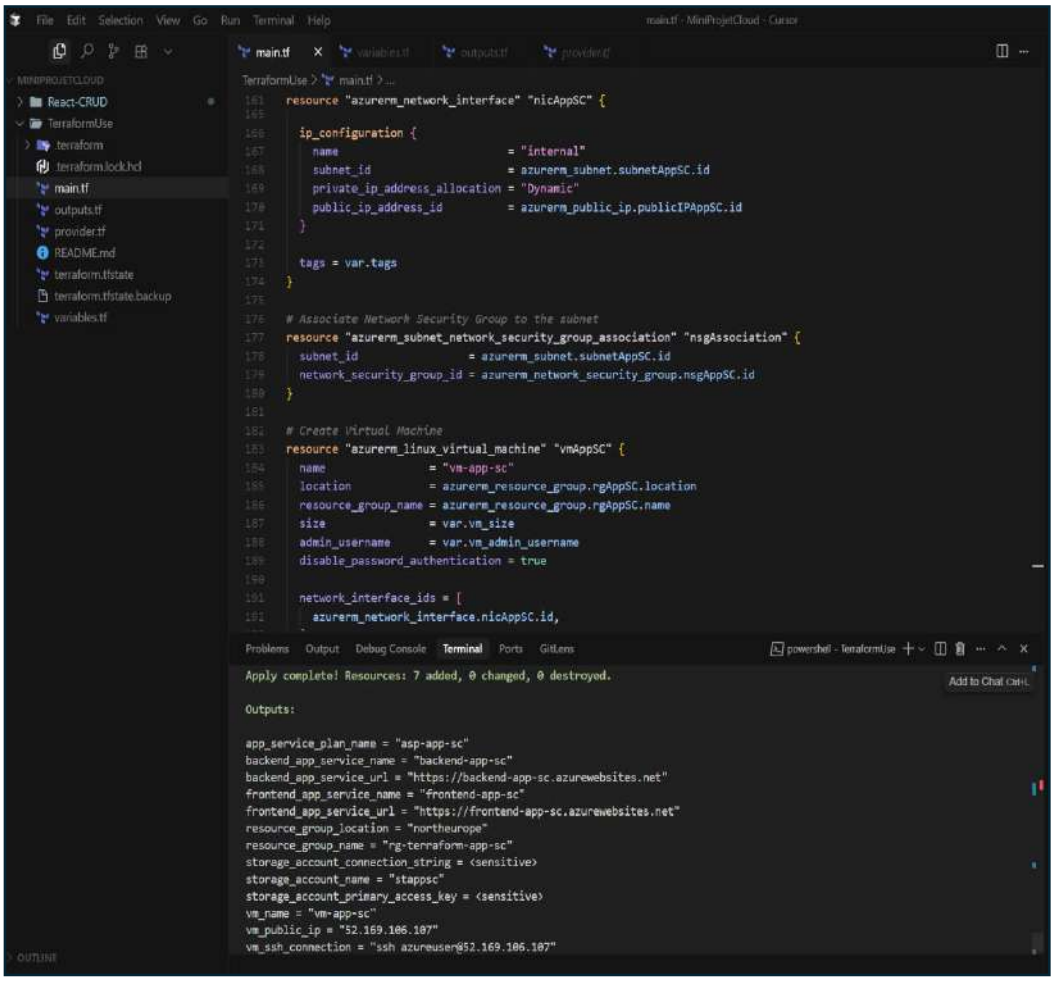
4.Création de l'Interface Réseau(NIC-Network Interface Card)

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 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



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 entant 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
}
```

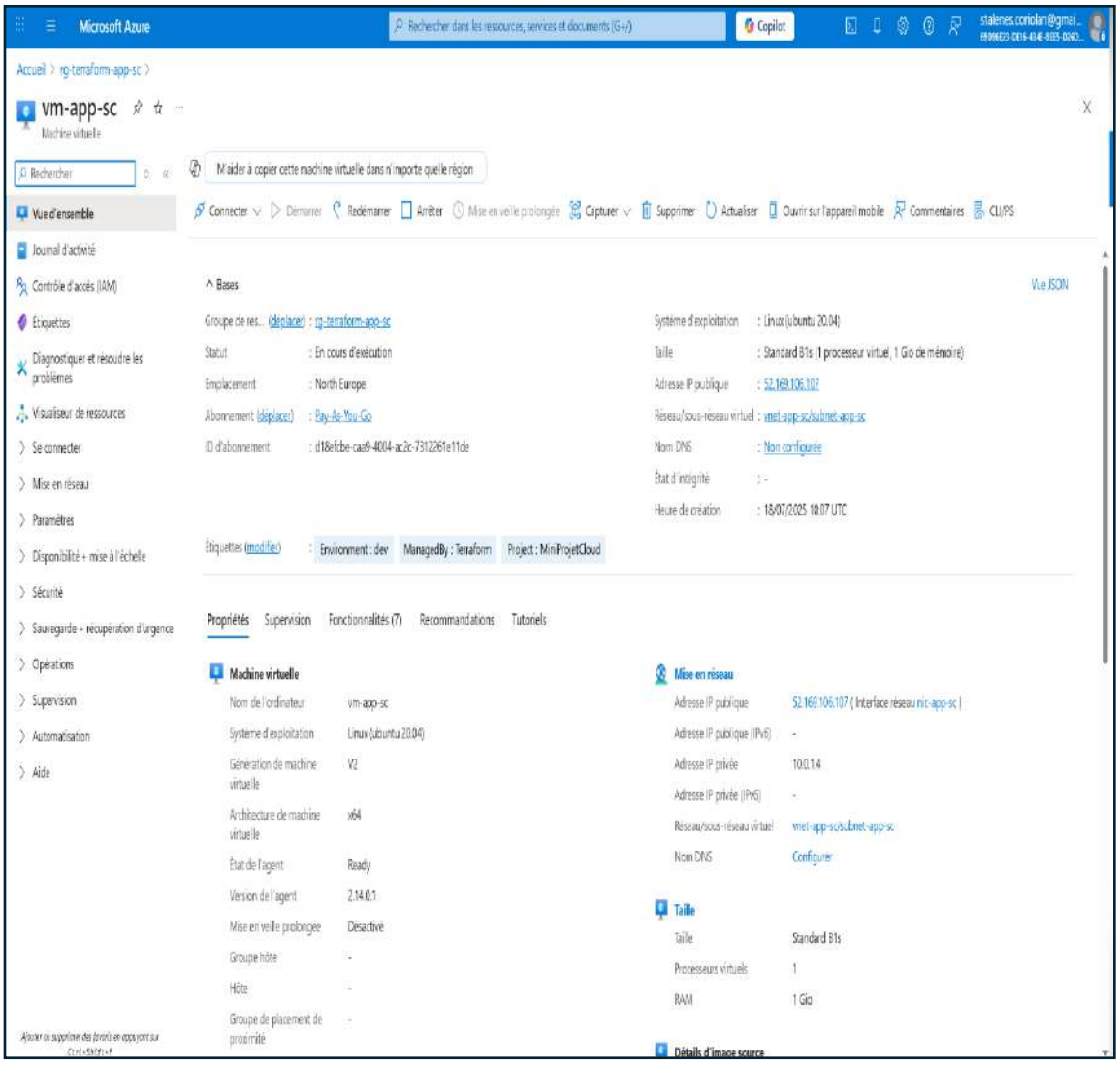
On déploie avec les commande terraform init -> terraform plan -> terraform apply.
Ensuite dans notre fichier outputs.tf on ajoute :

- 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

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

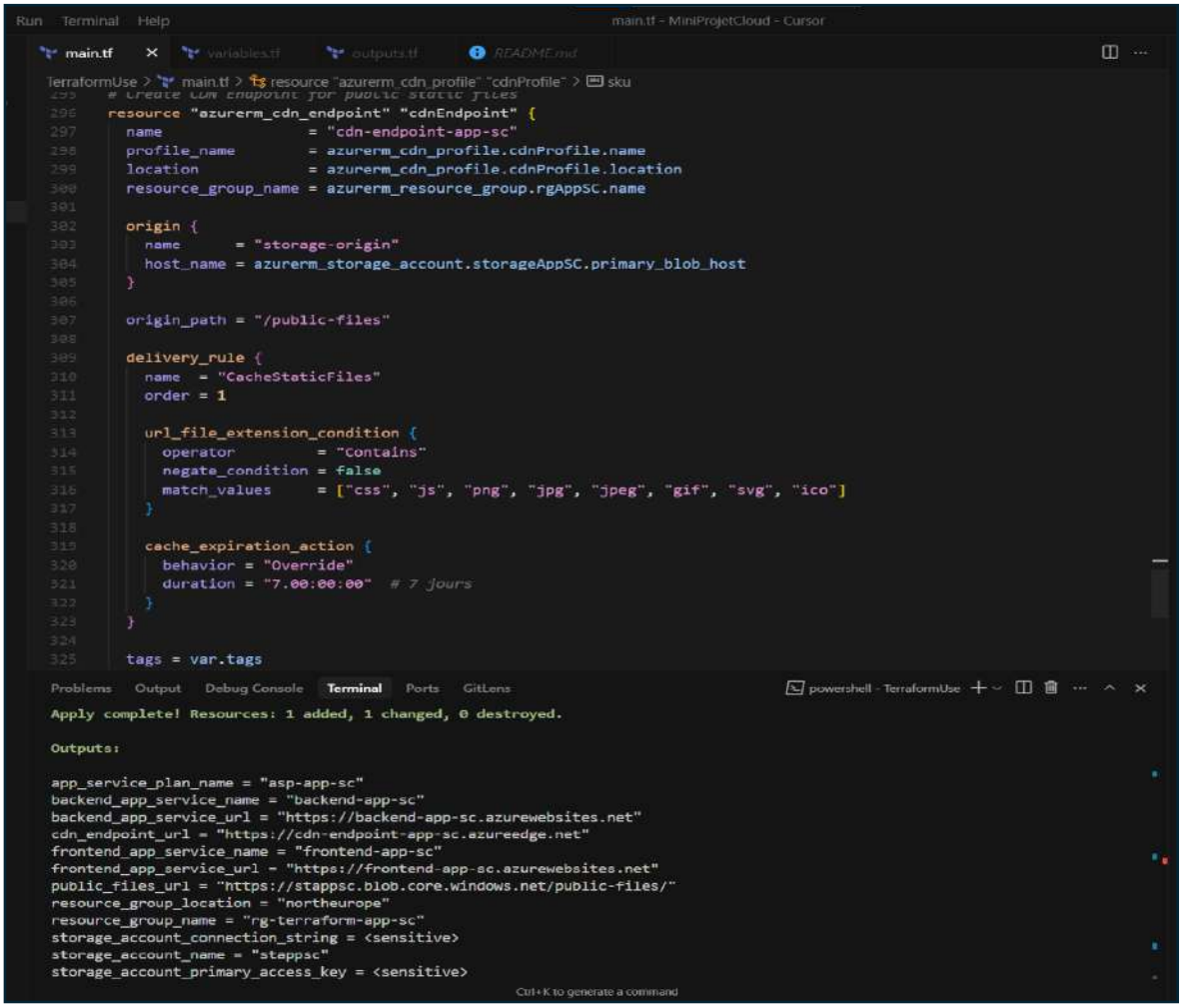


B - Création du stockage cloud avec Blob :

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 fichier statique publics(image, vidéo, JS...) stockés dans un Azure Storage Account(blob container).



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

2. Ressource Storage Container

Maintenant pour **héberge**r des **fichiers publics**(image, pdf, js, videos...) 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 :

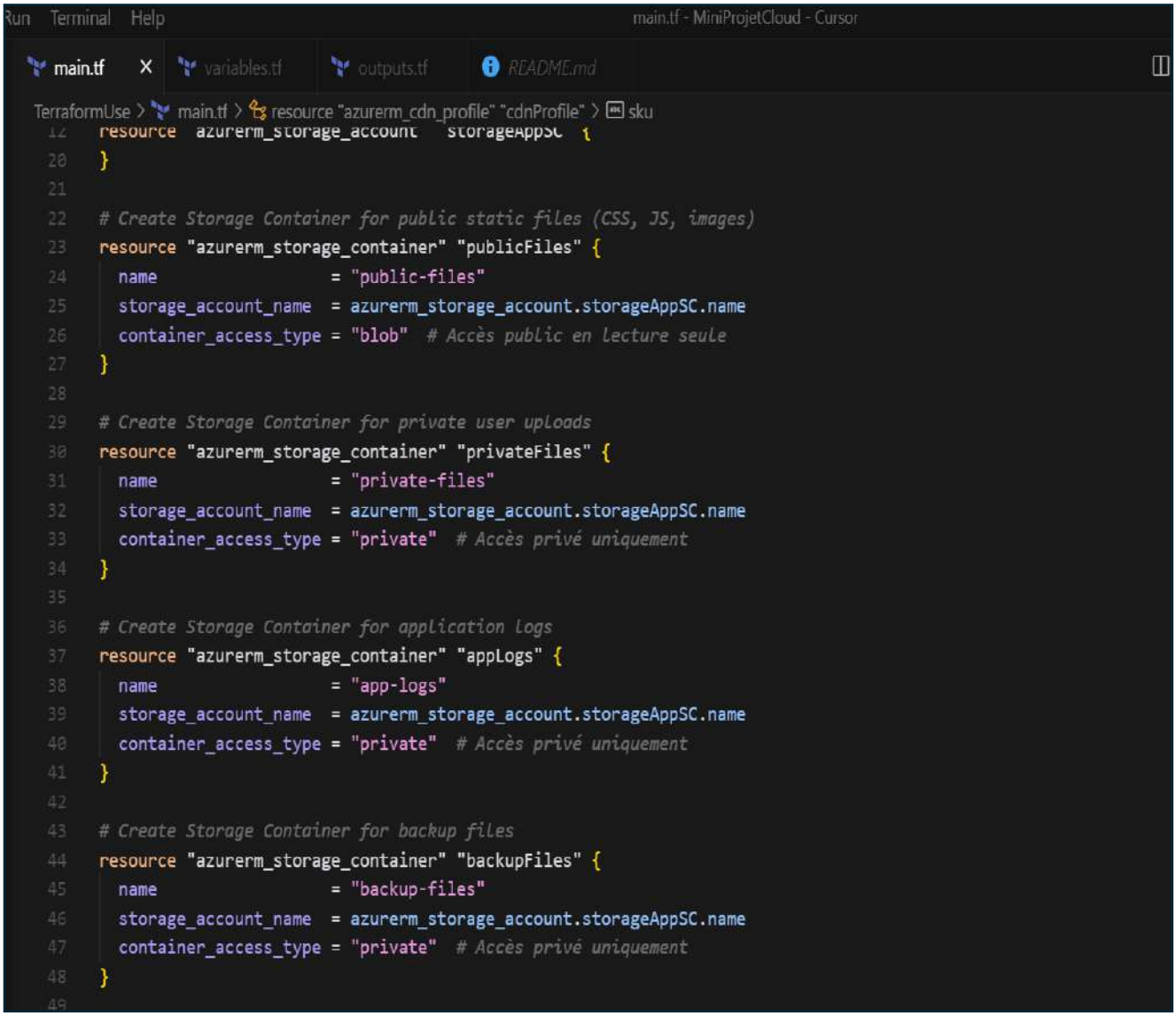
« ***https://stappsc.blobcore.windows.net/public-files/*** »

Ensuite pour **uploader des fichiers privés** par les utilisateurs on a créé une ressource pour les fichiers privée nommée « **PrivateFiles** ». Ainsi on pourra uploade 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 **rétention 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



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
}
```

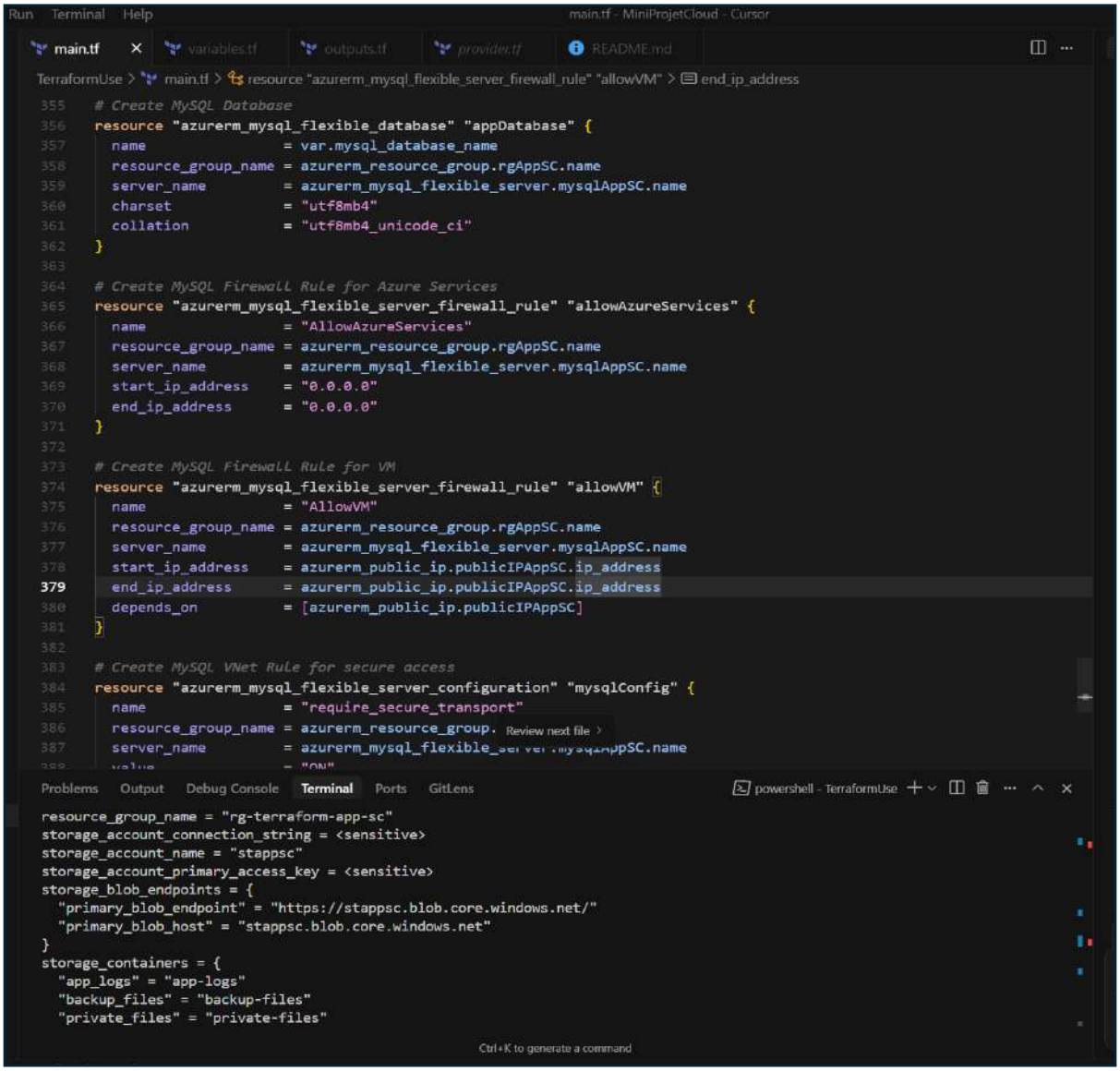
2.Creation de la BDD Mysql :

Pour crée 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 crée notre base de donnée **var.mysql_database_name** sur notre serveur Azure **Database for MySQL Flixible Server** existant.

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

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



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 :

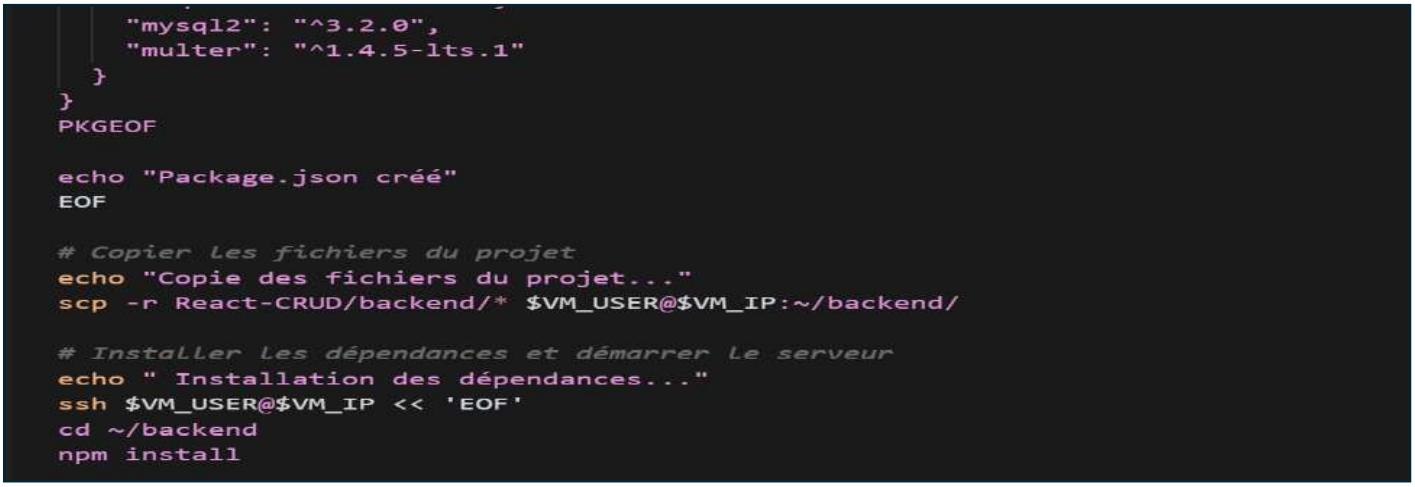
1)

```
#!/bin/bash

# Script de déploiement sur VM Azure
# Utilisation: ./deploy-to-vm.sh

echo "Déploiement sur VM Azure (52.169.106.107)"

# Variables
VM_IP="52.169.106.107"
VM_USER="azureuser"

# Vérifier la connectivité SSH
echo "Vérification de la connectivité SSH..."
if ! ssh -o BatchMode=yes -o ConnectTimeout=5 $VM_USER@$VM_IP exit 2>/dev/null; then
    echo "Impossible de se connecter à la VM Azure"
    echo "Vérifiez que vous avez accès SSH à $VM_USER@$VM_IP"
    exit 1
fi

echo "Connexion SSH établie"

# Installer Node.js sur la VM
echo "Installation de Node.js sur la VM..."
ssh $VM_USER@$VM_IP << 'EOF'
# Vérifier si Node.js est déjà installé
if command -v node &> /dev/null; then
    echo "Node.js déjà installé: $(node --version)"
else
    echo "Installation de Node.js..."
    curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
    sudo apt-get install -y nodejs
    echo "Node.js installé: $(node --version)"
fi

# Installer MySQL client
if ! command -v mysql &> /dev/null; then
    echo "Installation du client MySQL..."
    sudo apt-get install -y mysql-client-core-8.0
fi
```

2)

```
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=5170
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}"
  }
}
```

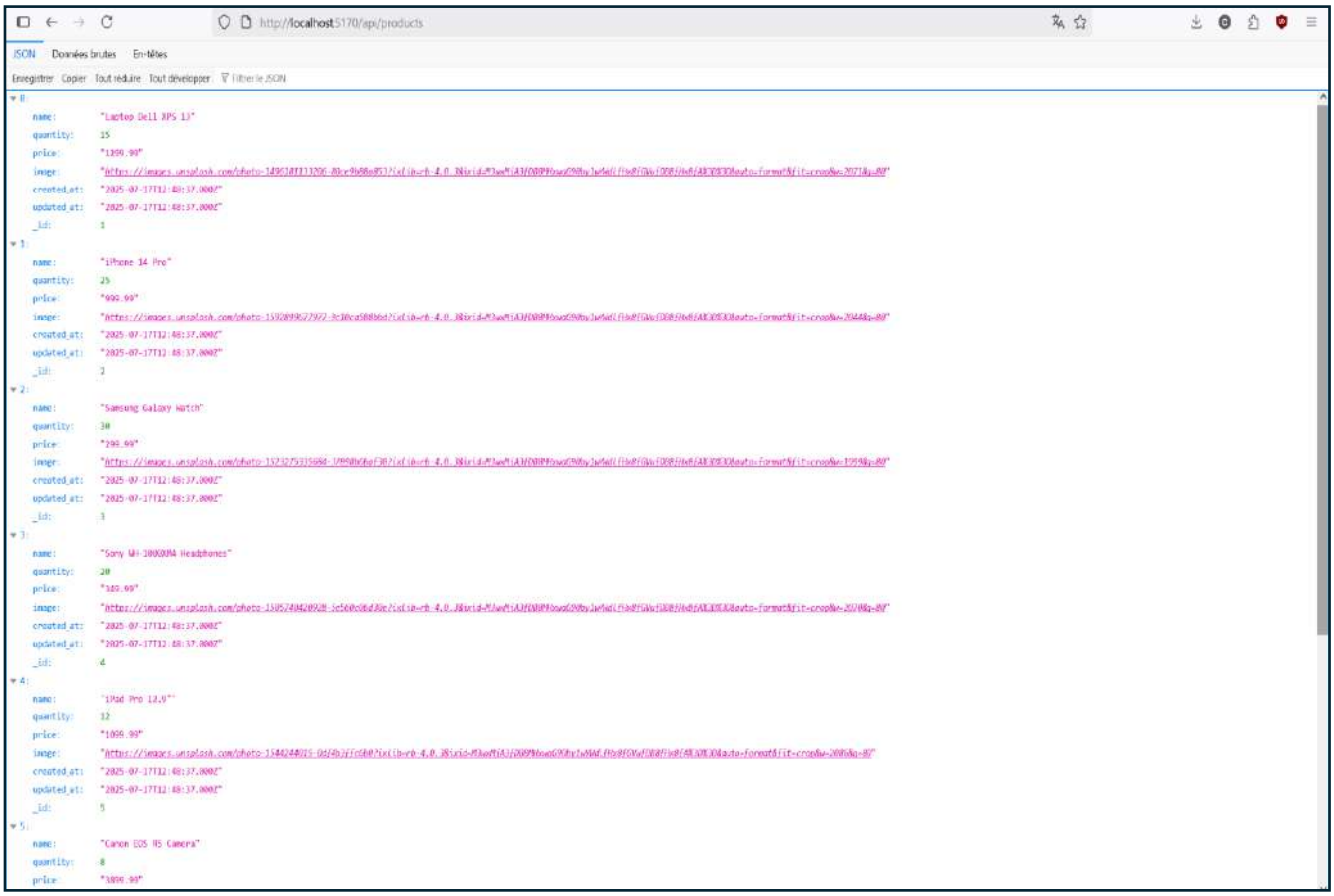
Connection via la VM avec : ssh azureuser@52.169.106.107

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 :

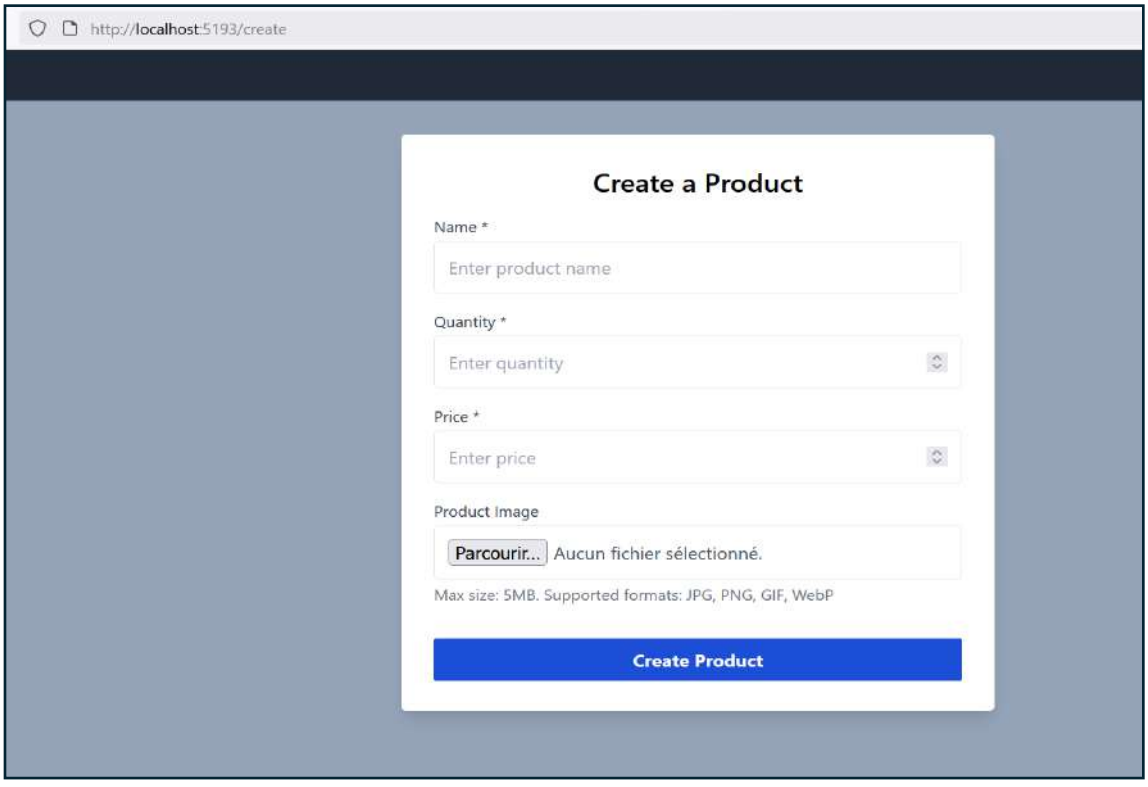


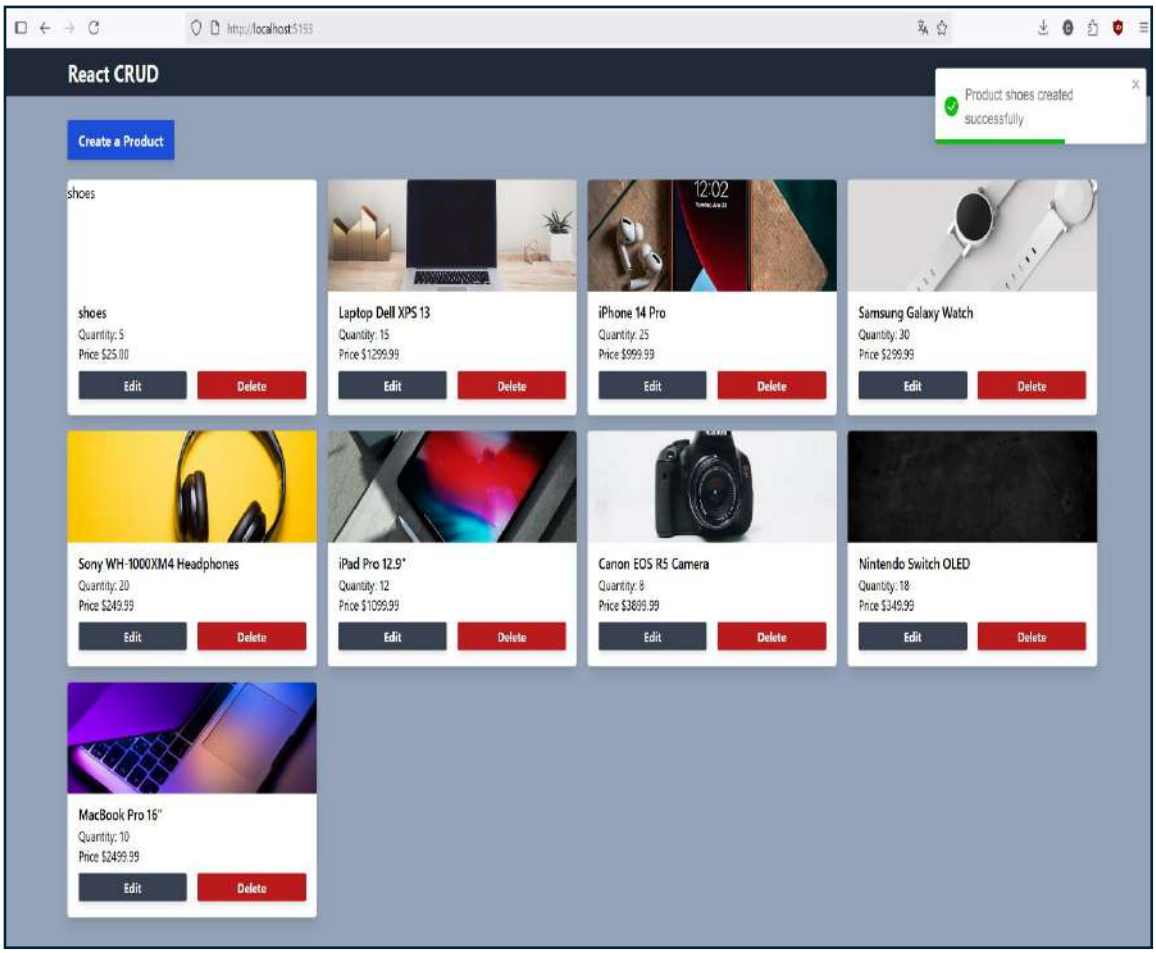
Dans le code toute nos routes sont aussi bien définies, on le voit avec les méthodes :

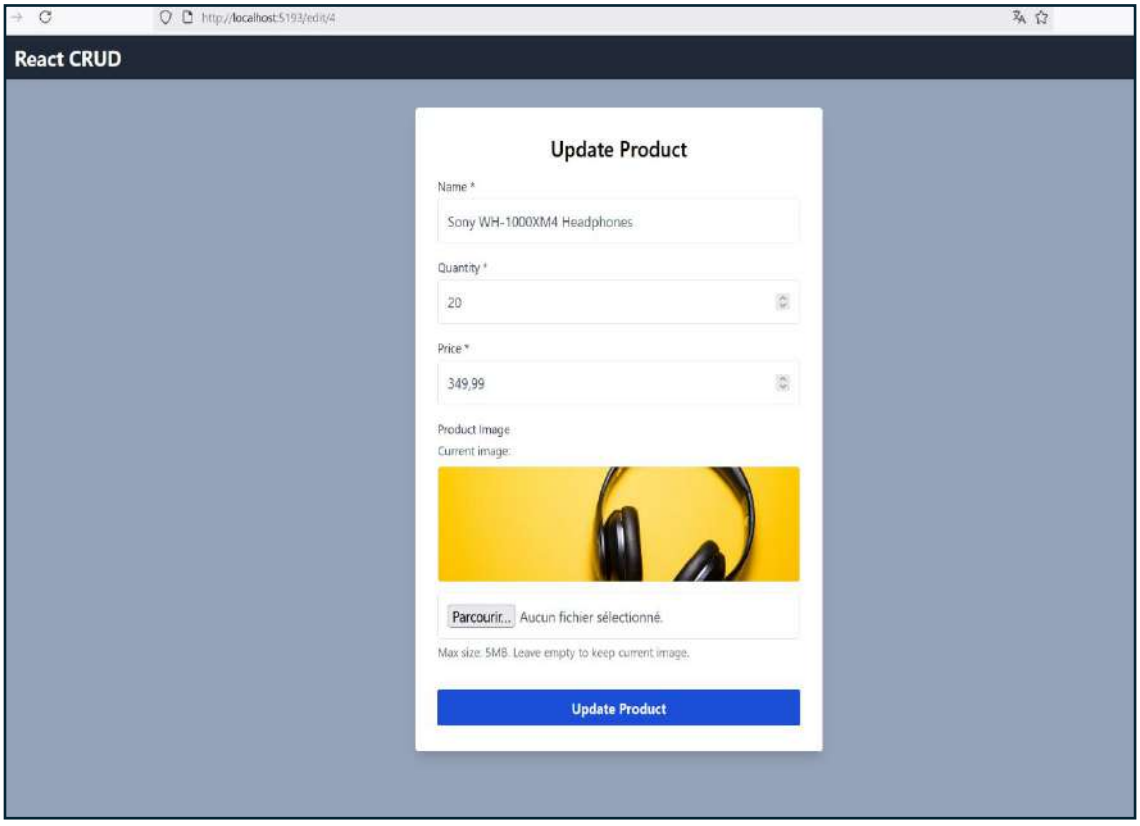
Ensuite on va voir les routes disponibles sur le port 5170 afin qu'on puisse exécuter nos requête api.

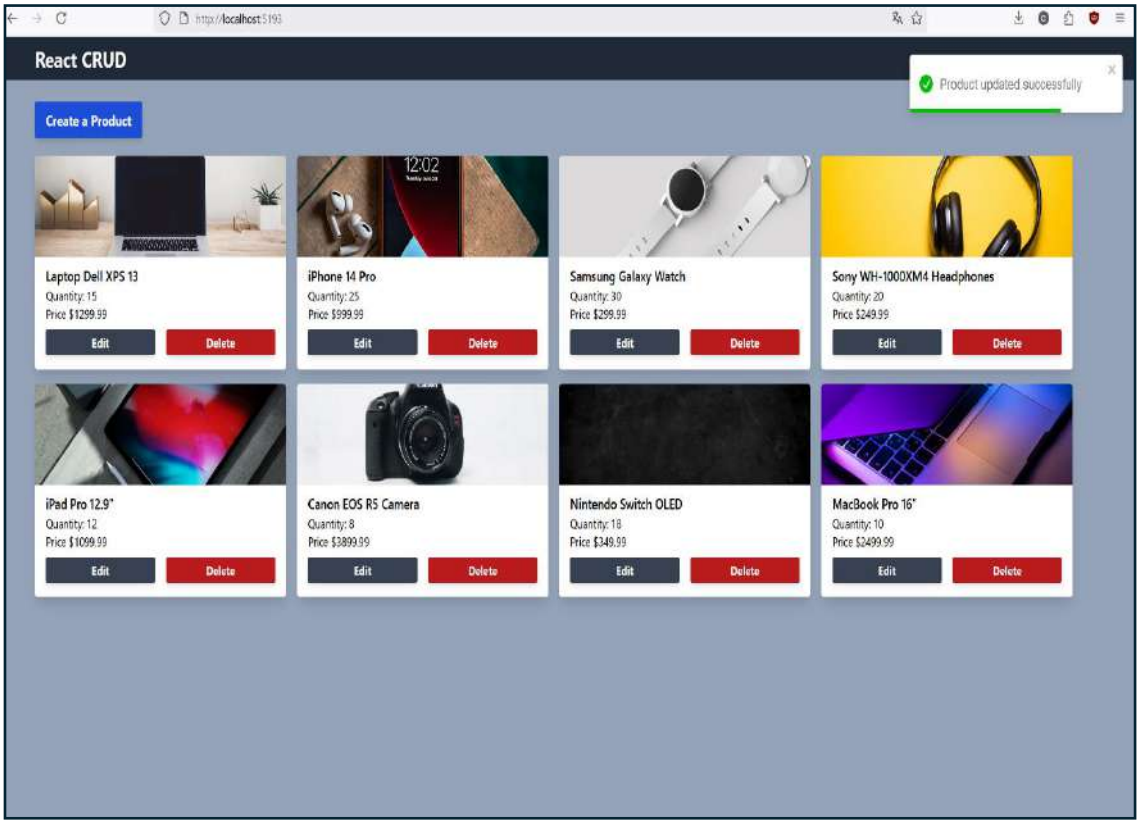
```
t-CRUD > backend > routes > productRoutes.js > ...
import {
  validateProduct,
  validateProductUpdate,
  validateId
} from '../middleware/validation.js';
import {
  uploadSingle,
  processImage,
  deleteOldImage,
  handleUploadError
} from '../middleware/uploadMiddleware.js';      You, 5 days ago • first commit …

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)  /
  .delete(validateId, deleteProduct);  // DELETE /api/products/:id

export default router;
```

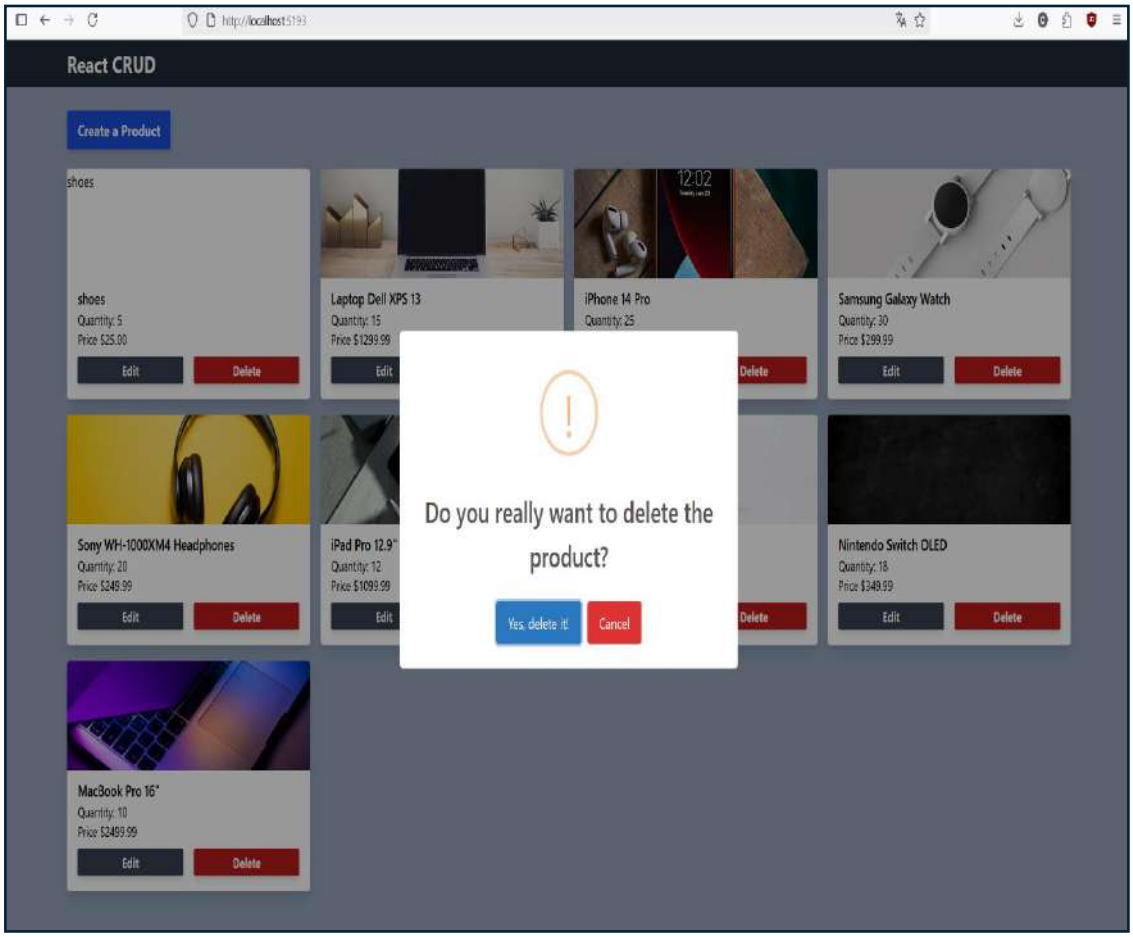
Maintenant on va vérifier le bon fonctionnement des API à travers un frontend développer avec React :

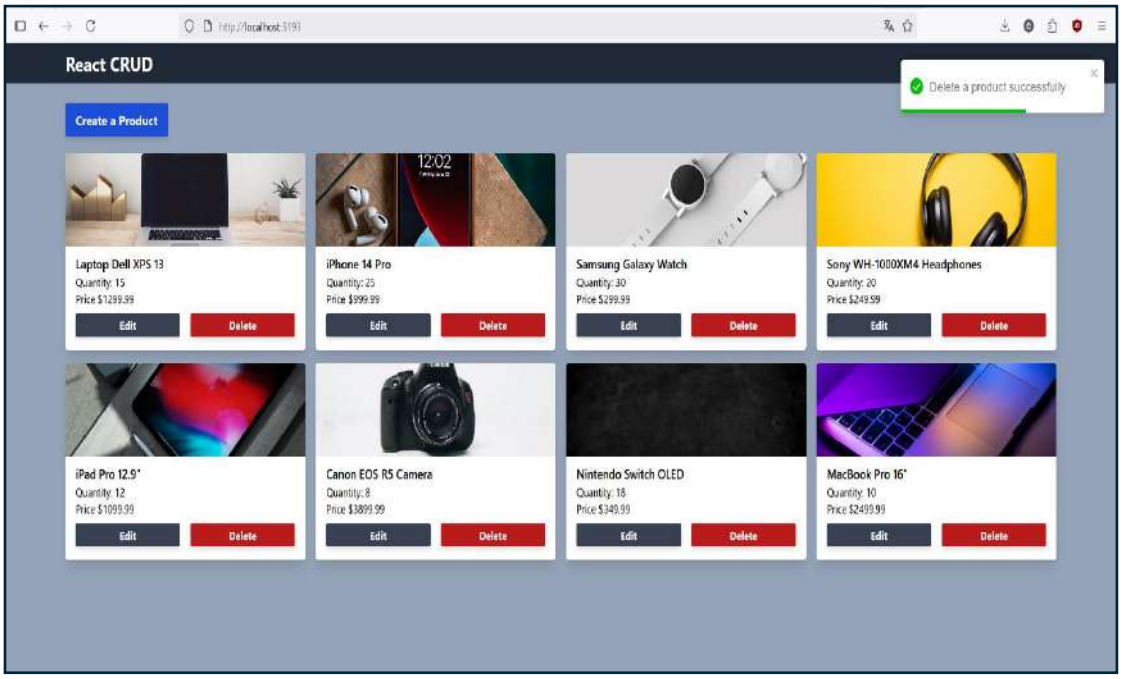



http://localhost:5193
React CRUD
Product shoes created successfully
Create a Product
shoes
shoes
Quantity: 5
Price $25.00
Edit
Delete
Laptop Dell XPS 13
Quantity: 15
Price $1299.99
Edit
Delete
iPhone 14 Pro
Quantity: 25
Price $999.99
Edit
Delete
Samsung Galaxy Watch
Quantity: 30
Price $299.99
Edit
Delete
Sony WH-1000XM4 Headphones
Quantity: 20
Price $249.99
Edit
Delete
iPad Pro 12.9"
Quantity: 12
Price $1099.99
Edit
Delete
Canon EOS R5 Camera
Quantity: 8
Price $3899.99
Edit
Delete
Nintendo Switch OLED
Quantity: 18
Price $349.99
Edit
Delete
MacBook Pro 16"
Quantity: 10
Price $2499.99
Edit
Delete


http://localhost:5193/edit/4
React CRUD
Update Product
Name *
Sony WH-1000XM4 Headphones
Quantity *
20
Price *
349,99
Product Image
Current image:
Parcourir...
Aucun fichier sélectionné.
Max size: 5MB. Leave empty to keep current image.
Update Product


http://localhost:5193
React CRUD
Product updated successfully
Create a Product
Laptop Dell XPS 13
Quantity: 15
Price $1299.99
Edit
Delete
iPhone 14 Pro
Quantity: 25
Price $999.99
Edit
Delete
Samsung Galaxy Watch
Quantity: 30
Price $299.99
Edit
Delete
Sony WH-1000XM4 Headphones
Quantity: 20
Price $249.99
Edit
Delete
iPad Pro 12.9"
Quantity: 12
Price $1099.99
Edit
Delete
Canon EOS R5 Camera
Quantity: 8
Price $3899.99
Edit
Delete
Nintendo Switch OLED
Quantity: 18
Price $349.99
Edit
Delete
MacBook Pro 16"
Quantity: 10
Price $2499.99
Edit
Delete

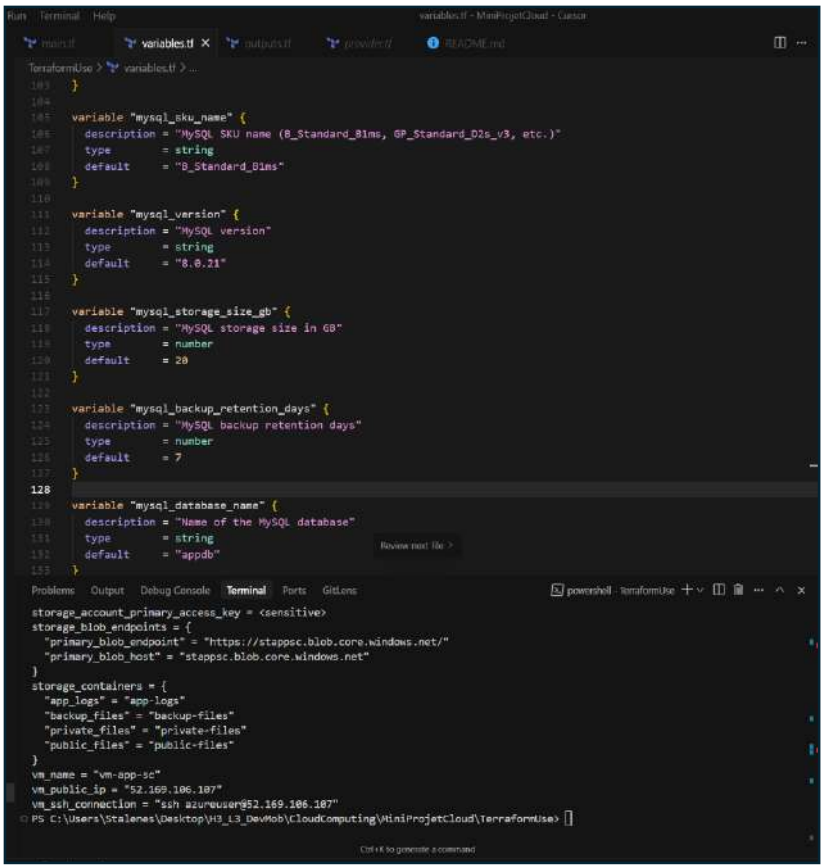

http://localhost:5193
React CRUD
Create a Product
shoes
shoes
Quantity: 5
Price $25.00
Edit
Delete
Laptop Dell XPS 13
Quantity: 15
Price $1299.99
iPhone 14 Pro
Quantity: 25
Samsung Galaxy Watch
Quantity: 30
Price $299.99
Edit
Delete
Sony WH-1000XM4 Headphones
Quantity: 20
Price $249.99
Edit
Delete
iPad Pro 12.9"
Quantity: 12
Price $1099.99
Nintendo Switch OLED
Quantity: 18
Price $349.99
Edit
Delete
MacBook Pro 16"
Quantity: 10
Price $2499.99
Edit
Delete
Do you really want to delete the product?
Yes, delete it!
Cancel



Etape4 :

A – Gestion des Variables et Outputs

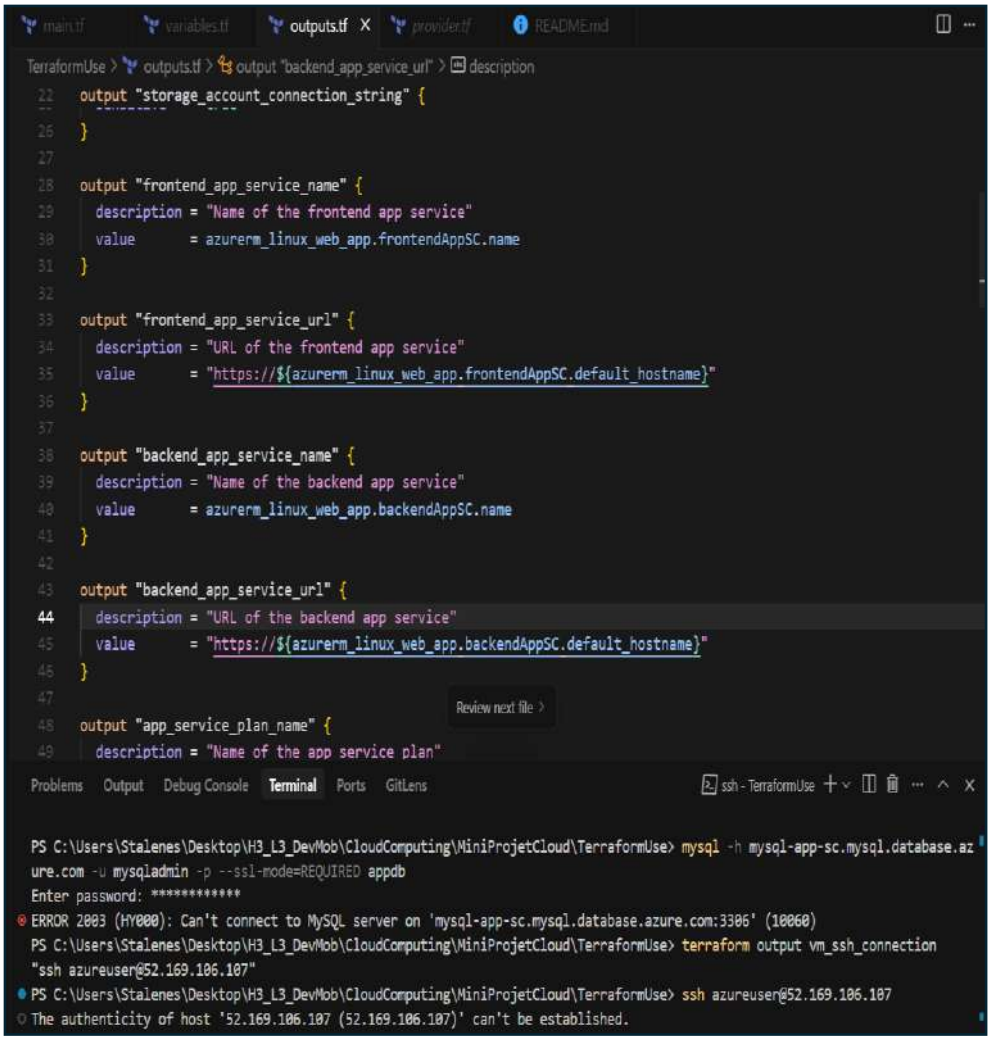
Le fichier **variables.tf** est un composant essentiel de Terraform qui permet de définir les variables d'entrée pour notre infrastructure.



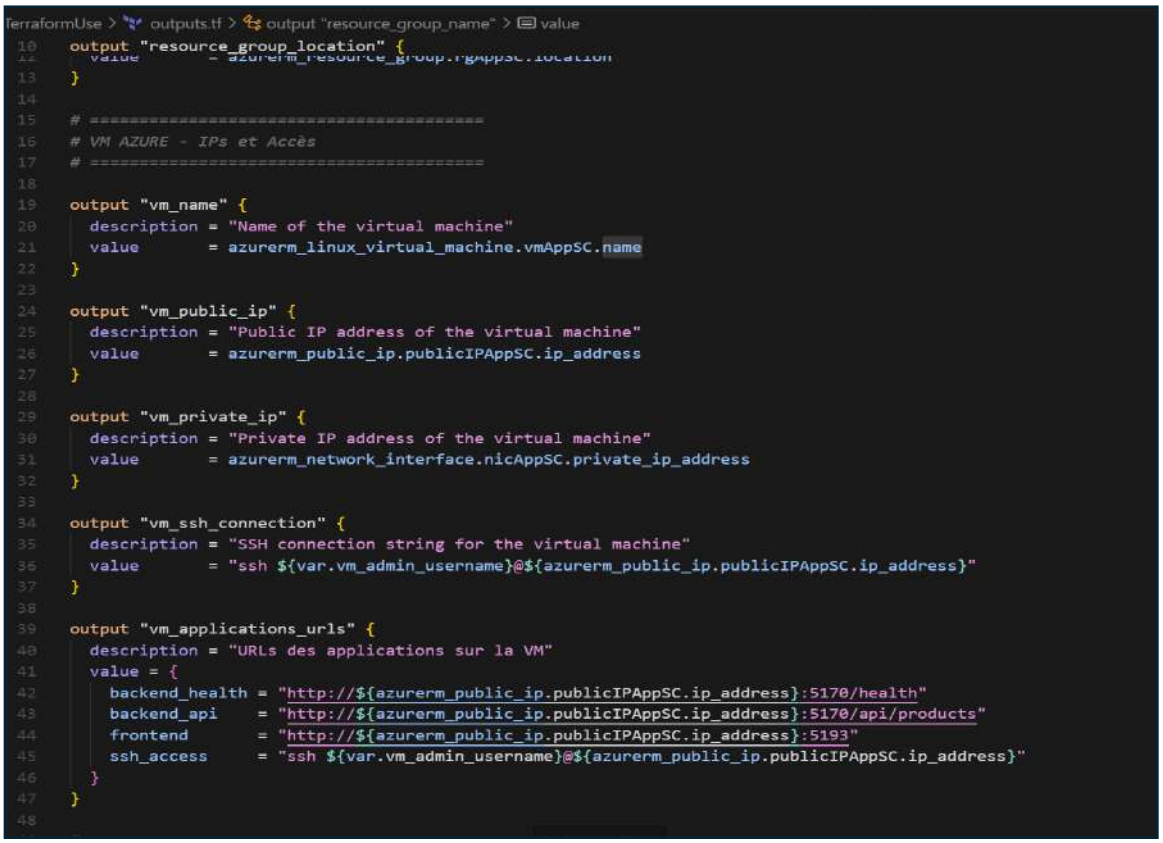
Il sert plusieurs objectifs importants :

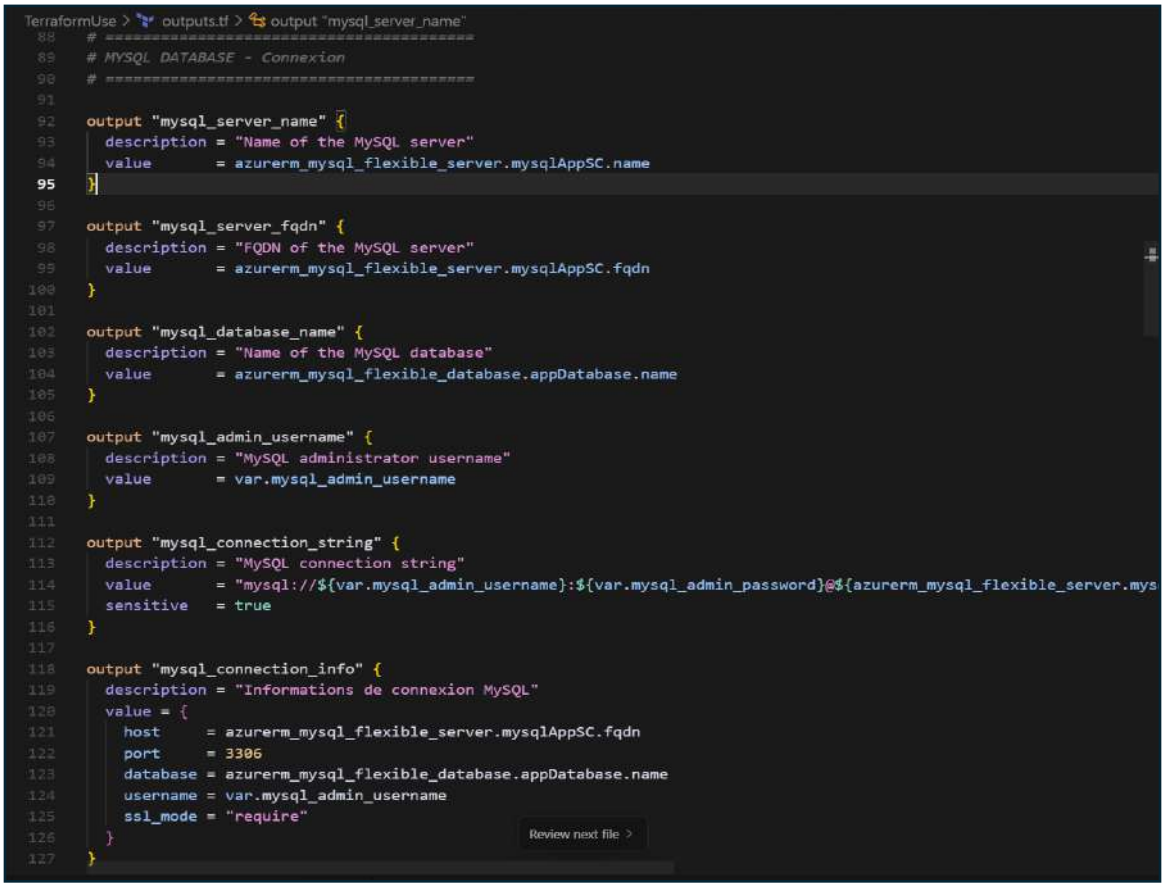
- **Définition des variables d'entrée :**
  Le fichier variables.tf contient les déclarations de toutes les variables que votre configuration Terraform peut accepter. Chaque variable est définie avec sa structure, son type, et ses contraintes.
- **Paramétrage flexible :**
  Il permet de rendre votre 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.



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





```hcl
# 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"

```hcl
# ========================================
# RÉSUMÉ INFRASTRUCTURE
# ========================================

output "infrastructure_summary" {
  description = "Résumé complet de l'infrastructure"
  value = {
    # Informations générales
    resource_group        = azurerm_resource_group.rgAppSC.name
    location              = azurerm_resource_group.rgAppSC.location

    # VM Azure
    vm_public_ip          = azurerm_public_ip.publicIPAppSC.ip_address
    vm_private_ip         = azurerm_network_interface.nicAppSC.private_ip_address

    # App Services
    frontend_url          = "https://${azurerm_linux_web_app.frontendAppSC.default_hostname}"
    backend_url           = "https://${azurerm_linux_web_app.backendAppSC.default_hostname}"

    # Base de données
    mysql_host            = azurerm_mysql_flexible_server.mysqlAppSC.fqdn
    mysql_database        = azurerm_mysql_flexible_database.appDatabase.name

    # Stockage
    storage_account       = azurerm_storage_account.storageAppSC.name
    cdn_endpoint          = "https://${azurerm_cdn_endpoint.cdnEndpoint.fqdn}"

    # Ports configurés
    backend_port          = "5170"
    frontend_port         = "5193"
    mysql_port            = "3306"
  }
}
```

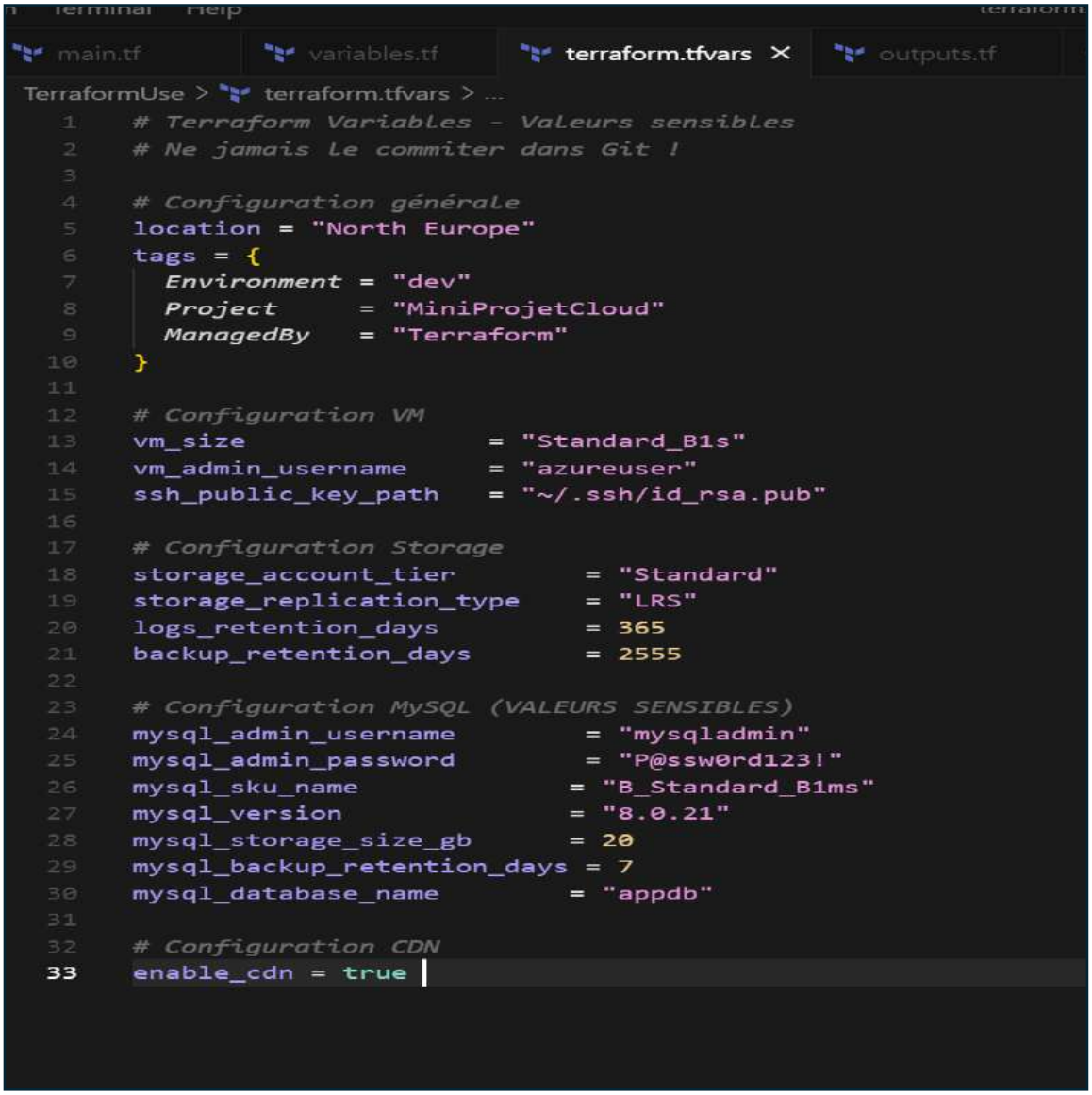
Utilisations courantes :

- **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**.

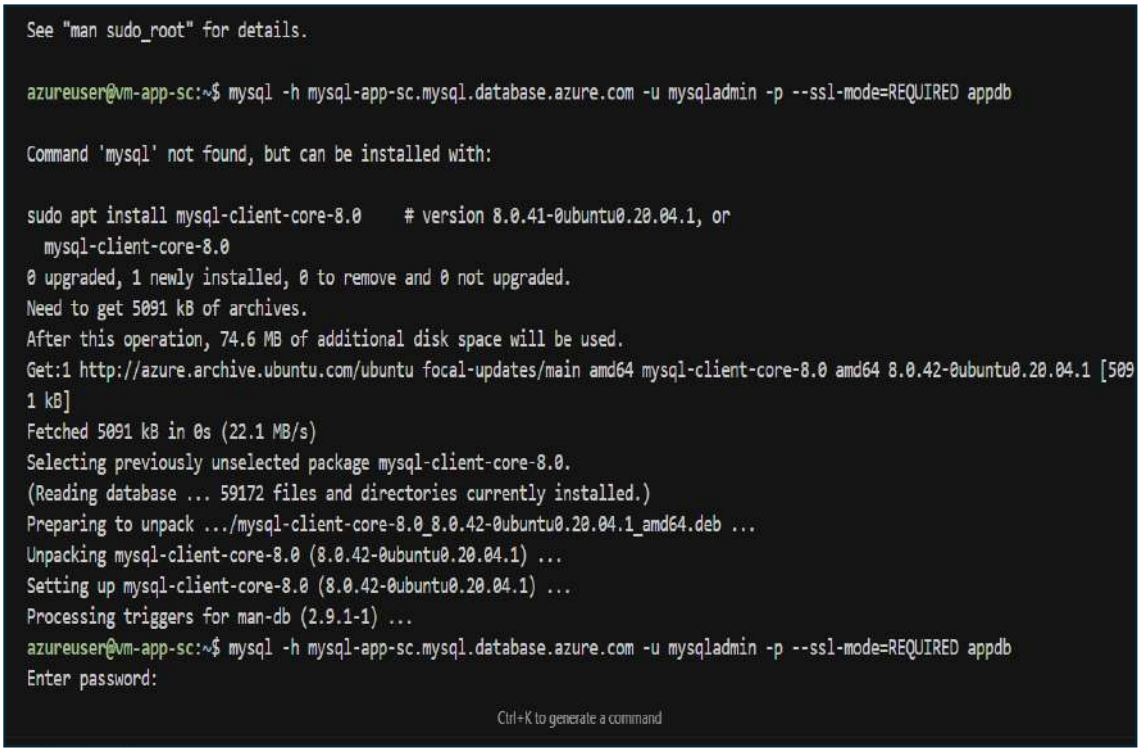
B – Terraforms.tfvars :

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



Etape5 :

Donc on va procéder ainsi pour le déploiement final avec Azure



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.