praxis.md

Praxis: Terraform VPC

0. Überprüfung der Dependencies

1. AWS CLI mit konfiguriertem Profil für Techstarter Sandbox

Überprüfe ob die `aws-cli` installiert ist

```
aws --verison
```

Stelle sicher, dass ein Profil für die Techstarter Sandbox eingerichtet ist:

```
aws configure list-profiles
# techstarter
```

Mein Profil wurde zum Beispiel als `techstarter` konfiguriert, das heißt: **bei allen weiteren `aws` Befehlen muss am Ende `--profile techstarter` eingegeben werden!**

Wenn bei dir das Profil anders benannt war (output des letzten Commands) nutze dieses Profil

Starte die SSO Session:

```
aws sso login --profile techstarter
```

2. Terraform installation überprüfen

```
terraform --version
```

Falls die Versionsnummer nicht angezeigt wird, muss Terraform noch installiert werden.

1. Erstellung der Projektumgebung

1. Erstelle einen neuen Projekt Ordner

```
mkdir terraform-vpc && cd terraform-vpc
```

2. Erstelle die wichtigen HCL Dateien

```
touch main.tf versions.tf variables.tf outputs.tf .gitignore
```

3. Öffne das Projekt in dem Texteditor deiner Wahl

```
code .
```

4. Editiere die `.gitignore` Datei und füge folgenden Inhalt hinzu

```
# Local .terraform directories
**/.terraform/*

# .tfstate files
```

```
*.tfstate
*.tfstate.*
```

Terraform lädt Provider herunter und speichert den State lokal ab. Das wollen wir aber nicht in unserem Git repository speichern (ähnlich wie `node_modules`)

2. Terraform Versions, Provider und Variables

1. Editiere die `versions.tf` Datei:

```
terraform {
  required_version = ">= 1.0" # Wir wollen mindestens terraform version 1.0 verwenden

  required_providers {
    aws = {  # Der AWS Provider ermöglicht es AWS Resourcen zu erstellen
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
}

provider "aws" {
  region = var.region # Diese Terraform variable definieren wir im nächsten Schritt
  profile = "techstarter" # BITTE DEIN AWS PROFILE EINTRAGEN
}
```

Stelle sicher, dass du bei dem Punkt `profile` das von dir konfigurierte Aws Profil verwendest.

Die Region geben wir hier als verweis auf eine Terraform Variable an, sodass wir das gesamte Setup auch schnell in einer anderen Region deployen könnten.

2. Editiere die `variables.tf` Datei:

```
variable "region" {
  type = string # Welcher Datentyp ist die Variable?
  default = "eu-central-1" # Welchen Wert hat die Variable, wenn nichts angegeben wird?
}
```

4. Erster Test

1. Um die verwendeten Provider herunterzuladen und den Terraform State zu initialisieren, müssen wir folgenden Befehl ausführen:

```
terraform init
```

WICHTIG: Stelle sicher, dass du dich im Projektordner befindest

Wenn alles richti ist, solltest du folgenden Text als output erhalten:

```
Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 5.0"...
- Installing hashicorp/aws v5.21.0...
- Installed hashicorp/aws v5.21.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!
...
```

2. Ab jetzt können wir immer testen, welche Resourcen Terraform erstellen würde. Hierfür nutzen wir `terraform plan`

```
terraform plan
```

```
No changes. Your infrastructure matches the configuration.

Terraform has compared your real infrastructure against your configuration and found no differences, so no ch
```

5. Erstellung der Resourcen

1. Im ersten Schritt erstellen wir die VPC Resource:

```
# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/vpc
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"

  tags = {
    Name = "TF VPC"
  }
}
```

2. Jetzt können wir das leere VPC erstellen

```
terraform apply
```

Nach dem Ausführen wird uns eine Übersicht aller Schritte angezeigt, die Terraform ausführen wird.

```
Terraform will perform the following actions:

  # aws_vpc.main will be created
  + resource "aws_vpc" "main" {
      + arn                                  = (known after apply)
      + cidr_block                           = "10.0.0.0/16"
      + default_network_acl_id               = (known after apply)
      + default_route_table_id               = (known after apply)
      + default_security_group_id            = (known after apply)
      + dhcp_options_id                      = (known after apply)
      + enable_dns_hostnames                 = (known after apply)
      + enable_dns_support                   = true
      + enable_network_address_usage_metrics = (known after apply)
      + id                                   = (known after apply)
      + instance_tenancy                     = "default"
      + ipv6_association_id                  = (known after apply)
      + ipv6_cidr_block                      = (known after apply)
      + ipv6_cidr_block_network_border_group = (known after apply)
      + main_route_table_id                  = (known after apply)
      + owner_id                             = (known after apply)
      + tags                                 = {
          + "Name" = "TF VPC"
        }
      + tags_all                             = {
          + "Name" = "TF VPC"
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.
```

```
  Enter a value:
```

Um die Erstellung zu starten müssen wir `yes` eingegeben.

```
aws_vpc.main: Creating...
aws_vpc.main: Creation complete after 1s [id=vpc-0d0552b7f531601ec]

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

Jetzt wurde das VPC auch in der AWS Console erstellt.

3. Locals für Subnetze definieren

Um die Subnetzte zu provisionieren benötigen wir die CIDR Blöcke und Availability Zones. Hier macht es also Sinn auf Locals zurück zu greifen.

Füge folgenden Inhalt an den Anfang der `main.tf` hinzu:

```
# Alle Locals werden innerhalb des locals Block definiert
locals {
  az_a = "${var.region}a" # eu-central-1a
  az_b = "${var.region}b"

  cidr_a = "10.0.1.0/24"
  cidr_b = "10.0.2.0/24"
}
```

4. Jetzt können die Subnetze mit den definierten Lokals erstellt werden

Die `main.tf` sollte nun so aussehen:

```
# Alle Locals werden innerhalb des locals Block definiert
locals {
  az_a = "${var.region}a" # eu-central-1a
  az_b = "${var.region}b"

  cidr_a = "10.0.1.0/24"
  cidr_b = "10.0.2.0/24"
}

# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/vpc
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"

  tags = {
    Name = "TF VPC"
  }
}

# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/subnet
resource "aws_subnet" "subnet_a" {
  vpc_id     = aws_vpc.main.id
  cidr_block = local.cidr_a
  availability_zone = local.az_a
  map_public_ip_on_launch = true # Wir wollen, dass die Instanzen eine öffentliche IP bekommen

  tags = {
    Name = "TF Subnet A"
  }
}

resource "aws_subnet" "subnet_b" {
  vpc_id     = aws_vpc.main.id
  cidr_block = local.cidr_b
  availability_zone = local.az_b
```

```
  map_public_ip_on_launch = true # Wir wollen, dass die Instanzen eine öffentliche IP bekommen

  tags = {
    Name = "TF Subnet B"
  }
}
```

5. Wende die Änderungen an und stelle in der AWS Console sicher, dass alles funktioniert hat:

```
terraform apply
```

6. Nach dem gleichen Prinzip erstellen wir den Internet Gateway

Füge folgendes zum Ende der `main.tf` hinzu:

```
# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/internet_gateway
resource "aws_internet_gateway" "gw" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "TF Internet Gateway"
  }
}
```

```
terraform apply
```

7. Um das Internet Gateway auch zu benutzen, fehlt noch der Routing Table.

Bei jedem Erzeugen eines VPCs wird gleichzeitig ein (default) Routing Table erzeugt. Es empfiehlt sich, diesen nicht zu verändern und einfach einen neuen anzulegen.

```
# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/route_table
resource "aws_route_table" "rt" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0" # Das gesamte Internet
    gateway_id = aws_internet_gateway.gw.id # Link zu unserem erstellten Internet Gateway
  }

  tags = {
    Name = "TF Route Table"
  }
}
```

```
terraform apply
```

8. Dieser Route Table wird aktuell noch nicht verwendet. Die Subnetze müssen dem Route Table zugewiesen werden.

Füge auch diese Zeilen am Ende der `main.tf` Datei hinzu:

```
# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/route_table_association
resource "aws_route_table_association" "a" {
  subnet_id      = aws_subnet.subnet_a.id
  route_table_id = aws_route_table.rt.id
}

resource "aws_route_table_association" "b" {
  subnet_id      = aws_subnet.subnet_b.id
  route_table_id = aws_route_table.rt.id
}
```

9. Um das Setup zu testen, definieren wir noch eine EC2 Instanz + Security Group im neuen VPC:

```
# ------------------------------------------------------------------------------
# EC2 Instanz zum Testen des Setupt

# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/security_group
resource "aws_security_group" "sg" {
  name = "tf_sg"
  description = "Allow SSH inbound traffic"
  vpc_id = aws_vpc.main.id

  ingress {
    description = "SSH from VPC"
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    description = "Allow all outbound traffic"
    from_port = 0
    to_port = 0
    protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
resource "aws_instance" "test" {
  ami           = "ami-065ab11fbd3d0323d"
  instance_type = "t2.micro"
  subnet_id = aws_subnet.subnet_a.id
  vpc_security_group_ids = [aws_security_group.sg.id]
}
```

Finaler Code in `main.tf`

```
# Alle Locals werden innerhalb des locals Block definiert
locals {
  az_a = "${var.region}a" # eu-central-1a
  az_b = "${var.region}b"

  cidr_a = "10.0.1.0/24"
  cidr_b = "10.0.2.0/24"
}

# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/vpc
resource "aws_vpc" "main" {
  cidr_block = "10.0.0.0/16"

  tags = {
    Name = "TF VPC"
  }
}

# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/subnet
resource "aws_subnet" "subnet_a" {
  vpc_id     = aws_vpc.main.id
  cidr_block = local.cidr_a
  availability_zone = local.az_a
  map_public_ip_on_launch = true # Wir wollen, dass die Instanzen eine öffentliche IP bekommen

  tags = {
    Name = "TF Subnet A"
  }
}
```

```hcl
resource "aws_subnet" "subnet_b" {
  vpc_id     = aws_vpc.main.id
  cidr_block = local.cidr_b
  availability_zone = local.az_b
  map_public_ip_on_launch = true # Wir wollen, dass die Instanzen eine öffentliche IP bekommen

  tags = {
    Name = "TF Subnet B"
  }
}

# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/internet_gateway
resource "aws_internet_gateway" "gw" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "TF Internet Gateway"
  }
}

# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/route_table
resource "aws_route_table" "rt" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0" # Das gesamte Internet
    gateway_id = aws_internet_gateway.gw.id # Link zu unserem erstellten Internet Gateway
  }

  tags = {
    Name = "TF Route Table"
  }
}

# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/route_table_association
resource "aws_route_table_association" "a" {
  subnet_id      = aws_subnet.subnet_a.id
  route_table_id = aws_route_table.rt.id
}

resource "aws_route_table_association" "b" {
  subnet_id      = aws_subnet.subnet_b.id
  route_table_id = aws_route_table.rt.id
}

# ------------------------------------------------------------------------------
# EC2 Instanz zum Testen des Setupt

# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/security_group
resource "aws_security_group" "sg" {
  name = "tf_sg"
  description = "Allow SSH inbound traffic"
  vpc_id = aws_vpc.main.id

  ingress {
    description = "SSH from VPC"
    from_port = 22
    to_port = 22
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    description = "Allow all outbound traffic"
    from_port = 0
    to_port = 0
    protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

```
# https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/instance
resource "aws_instance" "test" {
  ami           = "ami-065ab11fbd3d0323d"
  instance_type = "t2.micro"
  subnet_id = aws_subnet.subnet_a.id
  vpc_security_group_ids = [aws_security_group.sg.id]
}
```

6. Terraform Output

1. Um am Ende die Id der erstellten EC2 Instanz anzuzeigen, erstellen wir noch einen Output:

Editiere die `outputs.tf` Datei:

```
output "instance_id" {
  value = aws_instance.test.id
}
```

7. Aufräumen

Um alle erstellten Resourcen wieder zu löschen, führe den folgenden Befehl aus und bestätige mit `yes`:

```
terraform destroy
```