

NOTRE PREMIÈRE INFRASTRUCTURE AWS DEPUIS TERRAFORM

Introduction

Dans notre chapitre précédent nous avons installé et configuré notre environnement Terraform. Dans cet article, je vais vous **présenter les bases de l'utilisation de Terraform** pour **définir et gérer votre infrastructure**. Pour démarrer, nous allons **construire une mini-infrastructure sur AWS**, ne vous inquiétez pas si vous n'avez jamais mis le pied sur le cloud AWS auparavant. Je vous guiderai tout au long du processus, étape par étape .

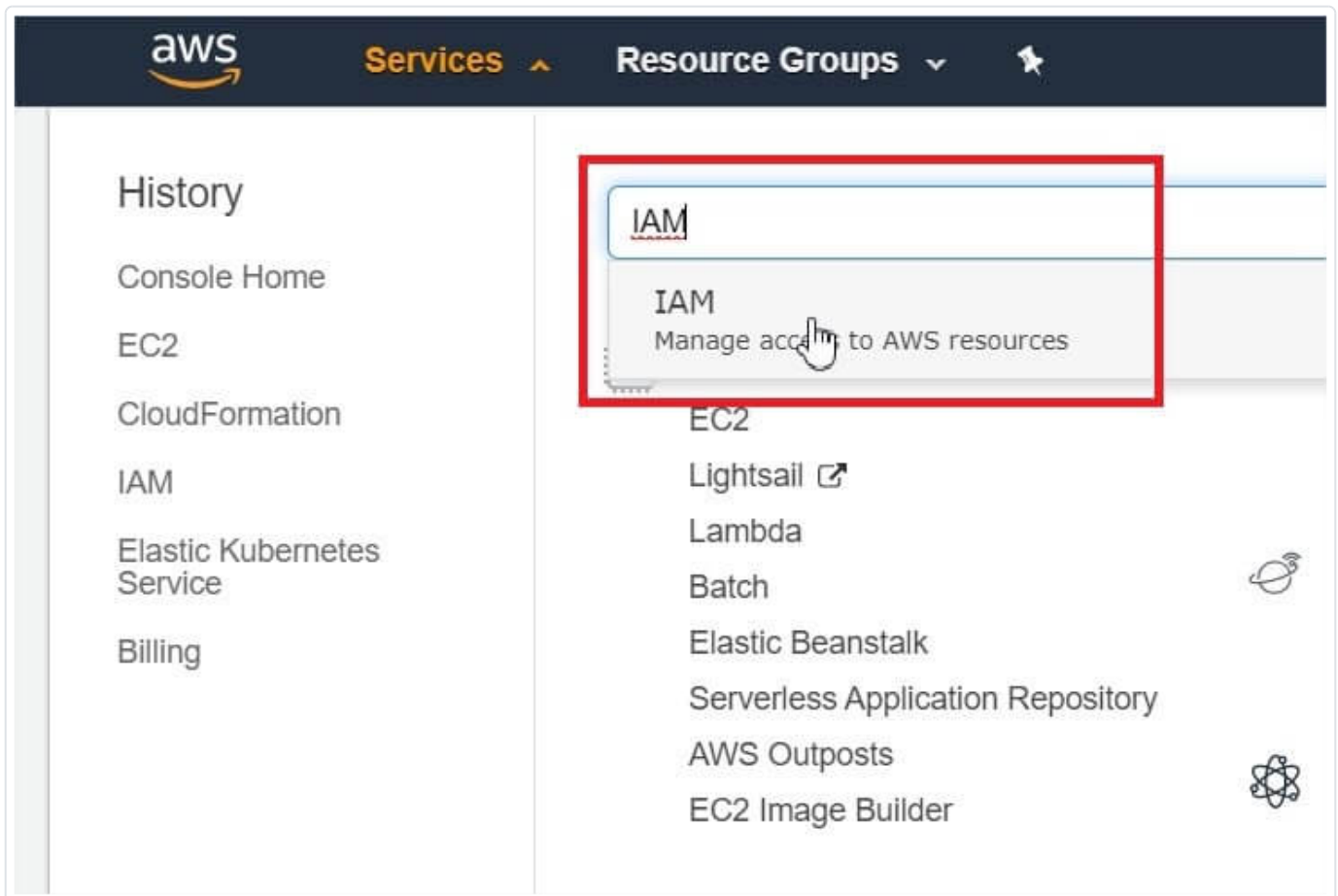
Comme vu précédemment, Terraform peut fournir une infrastructure à travers de nombreux types de fournisseurs de solutions Cloud (Azure, Google Cloud, DigitalOcean et bien d'autres ...). Mais pour démarrer, j'ai choisi de construire notre infrastructure sur le cloud Amazon Web Services (AWS) car déjà au jour d'aujourd'hui, c'est l'un des cloud providers que je maîtrise le mieux et qui est de loin le plus populaire des autres et fournit une vaste gamme de services d'hébergement cloud que nous aurons l'occasion d'utiliser dans ce cours. Enfin, il propose une **offre gratuite AWS** de 1 année qui devrait vous permettre d'exécuter tous ces exemples gratuitement ! (Plus d'informations sur les services [ici](#)).

Création de notre infrastructure AWS

Configurer votre compte AWS

Lorsque vous vous inscrivez pour la première fois à AWS, vous vous connectez initialement en tant qu'utilisateur root. Ce compte d'utilisateur a des autorisations d'accès total, donc du point de vue de la sécurité, je vous recommande de ne l'utiliser que pour créer d'autres comptes d'utilisateurs avec des autorisations plus limitées.

Pour **créer un compte d'utilisateur AWS plus limité**, rendez-vous sur la console du service "[Identity Access Management](#)" (IAM) , cliquez sur "Users", puis sur le bouton bleu "Add User". Saisissez un nom pour l'utilisateur et assurez-vous de cocher l'option que "Programmatic access" afin de générer des clés d'accès que nous utiliserons tout au long de ce cours. Cliquez ensuite sur le bouton "Next:Permissions" :





Services ▾

Resource Groups

Identity and Access Management (IAM)

Dashboard

▾ Access management

Groups

Users

Roles

Policies

Identity providers

Account settings

▾ Access reports

Access analyzer

Add user

Find users

User name

Add user

1

2

3

4

5

Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

terraform-user

+

Add another user

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Access type*

☒

Programmatic access
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

☐

AWS Management Console access
Enables a **password** that allows users to sign-in to the AWS Management Console.

* Required

Cancel

Next: Permissions

Vous arrivez ensuite sur la page de configuration des autorisations de votre nouvel utilisateur. Sélectionnez la section "Attach existing policies directly", et choisissez la policy déjà pré-configurée d'AWS nommée "AmazonEC2FullAccess", donnant ainsi les autorisations totales au service EC2 (Elastic Compute Cloud) qui est le service calcul évolutive d'Amazon Web Services (AWS) :

Add user

1 2 3 4 5

Set permissions

Add user to group

Copy permissions from existing user

Attach existing policies directly

Create policy

Filter policies

EC2Full

Showing 1 result

Policy name	Type	Used as
<input checked="" type="checkbox"/> AmazonEC2FullAccess	AWS managed	None

Set permissions boundary

Cancel Previous Next: Tags

Enfin, n'oubliez pas d'afficher et d'**enregistrer votre clé d'accès AWS** dans un endroit sécurisé :

Attention

Vous n'avez aucun moyen plus tard de réafficher la clé d'accès de votre compte, pensez donc à bien l'enregistrer !

Add user

1 2 3 4 5

✓ **Success**

You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: [https://\[redacted\].signin.aws.amazon.com/console](https://[redacted].signin.aws.amazon.com/console)

Download .csv

	User	Access key ID	Secret access key
▶	✓ terraform-user	A[redacted]7K[redacted]	***** Show

Notre utilisateur est dorénavant prêt à être utilisé, dans d'autres chapitres nous aurons l'occasion de réévaluer la policy de notre utilisateur afin de lui assigner davantage de droits, pour l'instant nous lui offrant le strict minimum pour les besoins de ce chapitre.

Création de notre code Terraform

Le code Terraform est écrit dans un langage appelé HCL dans des fichiers avec l'extension **.tf**. Il s'agit d'un langage déclaratif, qui est également utilisé par les langages de configuration dans d'autres applications, et en particulier par d'autres produits HashiCorp (créateur de Terraform). Notre objectif est donc de **décrire l'infrastructure qu'on souhaite dans le langage HCL**, et ensuite Terraform s'occupe de comment la créer.

Information

Si jamais comme moi, vous écrivez votre code Terraform l'éditeur de texte Visual Studio Code, je vous conseille alors de télécharger Terraform vous offrant ainsi la

prise en charge de la syntaxe Terraform, support pour le refactoring, etc ..

Création du Provider (fournisseur)

La première étape de l'utilisation de Terraform consiste généralement à **configurer le ou les providers** (fournisseurs) que vous souhaitez utiliser. Pour ce faire, créez un fichier avec l'extension **.tf**, dans mon cas je vais créer un fichier nommé **main.tf**. Ensuite, mettez-y le code suivant:

```
provider "aws" {  
  region = "us-east-2"  
  access_key = "votre-clé-dacces"  
  secret_key = "votre-clé-secrète"  
}
```

Cela indique à Terraform que vous allez utiliser le [fournisseur AWS](#) et que vous souhaitez déployer votre infrastructure dans la région "us-east-2". Vous devez également spécifier la paire de clés de votre utilisateur générée auparavant, pour le moment ce n'est pas sécurisé de partager une telle information sur votre code, c'est pour cela que nous aurons l'occasion de personnaliser et sécuriser ces informations dans le chapitre dédié aux variables.

Information

AWS possède des datacenters dans le monde entier, regroupés en région et Availability Zones (Zones de disponibilité) , et us-east-2 est le nom pour les datacenters situés en Ohio aux États-Unis).

Création de la Resource (ressource)

Pour chaque fournisseur, il existe de nombreux types de ressources que vous pouvez créer, tels que des serveurs, des bases de données, des équilibreurs de charge, etc. Avant de déployer des ressources complexes, voyons d'abord **comment déployer une instance** de calcul (machine virtuelle) qui s'exécutera en répondant "Hello devopssec" à nos requêtes HTTP. Dans le jargon AWS, un serveur/machine virtuelle est appelé une instance EC2 . Ajoutez le code suivant à votre fichier `main.tf` :

```
resource "aws_instance" "my_ec2_instance" {  
    ami = "ami-07c1207a9d40bc3bd"  
    instance_type = "t2.micro"  
}
```

La syntaxe générale d'une ressource Terraform est la suivante :

```
resource "<FOURNISSEUR>_<TYPE>" "<NOM>" {  
    [CONFIG ...]  
}
```

- **FOURNISSEUR** : c'est le nom d'un fournisseur (ici le provider "aws").
- **TYPE** : c'est le type de ressources à créer dans ce fournisseur (ici c'est une instance ec2)
- **NOM** : c'est un identifiant que vous pouvez utiliser dans le code Terraform pour faire référence à cette ressource (ici "my_ec2_instance")
- **CONFIG** : se compose de un ou plusieurs arguments spécifiques à cette ressource, dans notre cas :
 - **ami** : c'est l'acronyme d'"Amazon Machine Image" (AMI) , c'est donc l'image qui sera exécutée sur notre instance EC2. Vous pouvez trouver des AMI gratuites et payantes sur [AWS Marketplace](#) ou créer les vôtres

directement depuis la console AWS ou à l'aide d'outils tels que [Packer](#) que nous aurons sûrement l'occasion d'utiliser dans de futurs chapitres. Dans notre cas, nous utilisons l'identifiant "ami-07c1207a9d40bc3bd" qui est une AMI Ubuntu 18.04 (**Attention l'identifiant peut être modifié avec le temps !**). Cette AMI est gratuite et éligible à l'offre gratuite d'AWS.

- **instance_type** : Type d'instance EC2 à exécuter. Chaque type d'instance EC2 fournit une quantité différente de CPU, de mémoire, d'espace disque et de capacité réseau (plus d'informations ces différences sur cette [page](#)). Dans notre cas, nous utilisons le type **t2.micro**, qui fait partie du niveau gratuit AWS, et qui a comme caractéristiques 1 vCPU, ainsi que 1 Go de mémoire.

Création de notre instance ec2

En combinant les exemples précédents, nous nous retrouvons avec le code suivant :

```
provider "aws" {  
  region = "us-east-2"  
  access_key = "votre-clé-dacces"  
  secret_key = "votre-clé-secrète"  
}  
  
resource "aws_instance" "my_ec2_instance" {  
  ami = "ami-07c1207a9d40bc3bd"  
  instance_type = "t2.micro"  
}
```

Depuis un terminal, accédez au dossier dans lequel vous avez créé votre fichier

main.tf et exécutez la commande suivante :

```
terraform init
```

Résultat :

```
Initializing the backend...

Initializing provider plugins...
- Checking for available provider plugins...
- Downloading plugin for provider "aws" (hashicorp/aws) 2.57.0...
...
...
Terraform has been successfully initialized!
```

Pour info le binaire terraform téléchargé dans le chapitre précédent, ne contient que les fonctionnalités de base nécessaires au fonctionnement de Terraform, ce qui fait qu'il n'est pas fourni avec le code d'aucun des fournisseurs. Donc lorsque vous commencez à utiliser Terraform, vous devez exécuter cette fameuse commande `terraform init` pour **demander à Terraform de d'abord scanner votre code**, qui déterminera quel fournisseur vous utilisez et téléchargera le code pour vous.

Si vous listez les fichiers/dossiers de votre projet, vous retrouverez le dossier `.terraform` :

```
ls -a
```

Résultat :

```
.  .. main.tf .terraform
```

En effet, par défaut, le code du fournisseur sera téléchargé dans ce dossier `.terraform` qui est le répertoire de travail de Terraform (vous pouvez l'ajouter dans un fichier `.gitignore`). Pour l'instant, sachez que vous devez exécuter cette commande `init` chaque fois que vous démarrez avec du nouveau code Terraform.

Maintenant que vous avez téléchargé le code du fournisseur, exécutez la commande suivante :

```
terraform plan
```

Résultat :

```
Refreshing Terraform state in-memory prior to plan...
...
Resource actions are indicated with the following symbols:
+ create

Terraform will perform the following actions:

# aws_instance.my_ec2_instance will be created
+ resource "aws_instance" "my_ec2_instance" {
+   ami           = "ami-07c1207a9d40bc3bd"
+   ...
+   ...
}

Plan: 1 to add, 0 to change, 0 to destroy.
"terraform apply" is subsequently run
```

La commande `plan` est utilisée pour **créer un plan d'exécution**. Elle détermine les actions nécessaires pour atteindre l'état souhaité, spécifié dans les fichiers de configuration sans les exécuter. Elle n'est pas obligatoire mais ça reste un excellent moyen de vérifier vos modifications avant de les diffuser. La sortie de la commande plan ressemble un peu à la sortie de la commande Linux `diff`, avec le signe `+` qui indique les ressources qui vont être créées, le signe `-` pour les ressources qui vont être supprimées et enfin le signe `~` pour les ressources qui vont être modifiées.

Maintenant, pour véritablement **créer notre ressource Terraform**, exécutez la commande suivante :

```
terraform apply
```

Résultat :

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

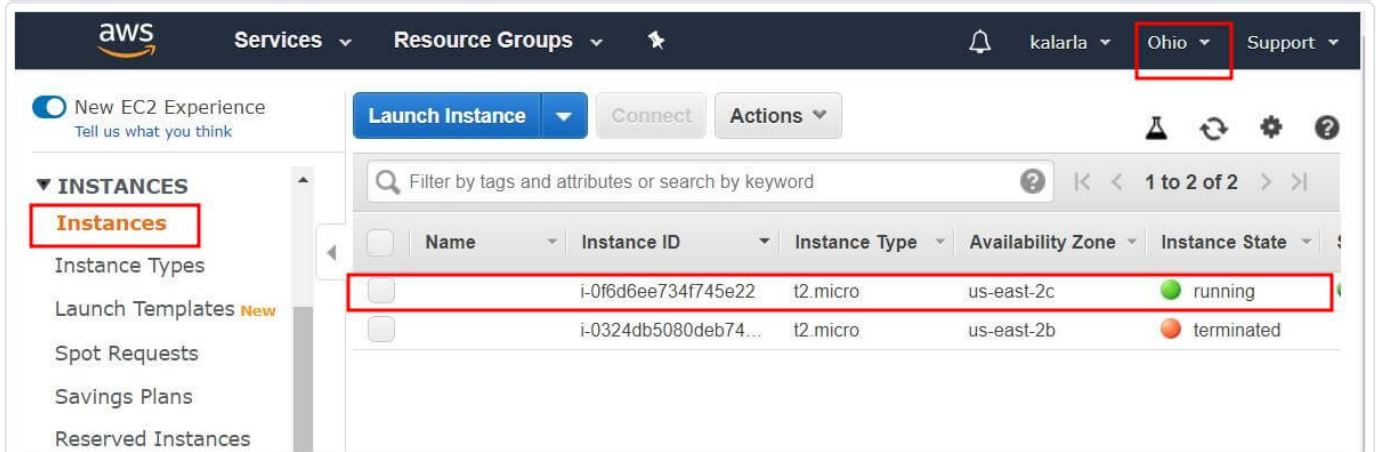
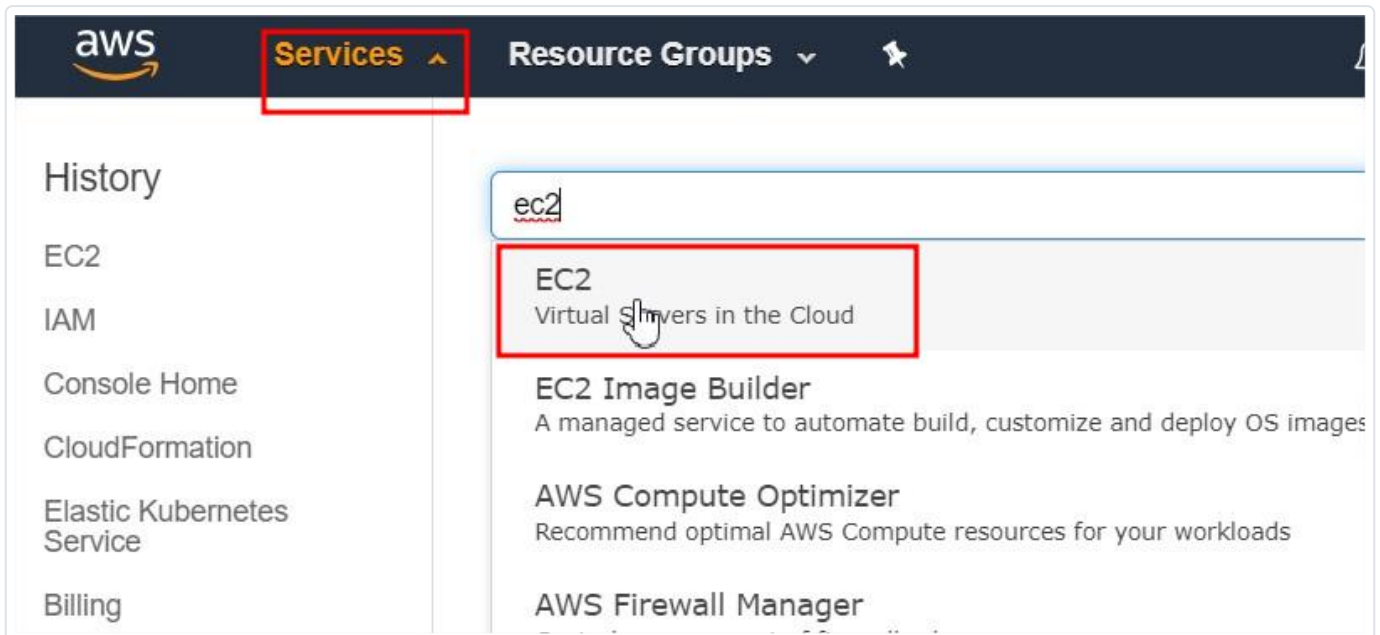
Enter a value: yes

aws_instance.my_ec2_instance: Creating...
```

```
aws_instance.my_ec2_instance: Still creating... [10s elapsed]
aws_instance.my_ec2_instance: Still creating... [20s elapsed]
aws_instance.my_ec2_instance: Still creating... [30s elapsed]
aws_instance.my_ec2_instance: Creation complete after 39s [id=i-0f6d6ee734f745e22]
```

Vous remarquerez que la commande `apply` vous affiche la même sortie que la commande `plan` et vous demande de confirmer, si vous souhaitez réellement poursuivre avec ce plan, alors tapez "yes" et appuyez sur Entrée pour déployer votre instance EC2.

Félicitations, vous venez de **déployer votre premier serveur avec Terraform** ! Pour vérifier cela, vous pouvez vous connecter à la [console EC2](#) et assurez-vous de bien sélectionner la région "us-east-2", vous verrez alors quelque chose comme ceci :



Les modifications de ressources

Terraform garde une trace de toutes les ressources qu'il a déjà créées. Si on rajoute par exemple une information Terraform saura détecter que votre instance EC2 existe déjà, et vous montrera la **différence entre ce qui est actuellement déployé** et ce qu'il y a actuellement dans votre code Terraform. Pour vous prouver que c'est bien le cas, nommons notre instance en créant une balise avec comme clé `Name` et comme valeur `terraform-test`, ce qui nous donne le code suivant :

```
provider "aws" {  
    region = "us-east-2"  
    access_key = "votre-clé-d'accès"  
    secret_key = "votre-clé-secrète"  
}  
  
resource "aws_instance" "my_ec2_instance" {  
    ami = "ami-07c1207a9d40bc3bd"  
    instance_type = "t2.micro"  
    tags = {  
        Name = "terraform test"  
    }  
}
```

Exécutons ensuite notre code :

```
terraform init && terraform apply
```

Résultat :

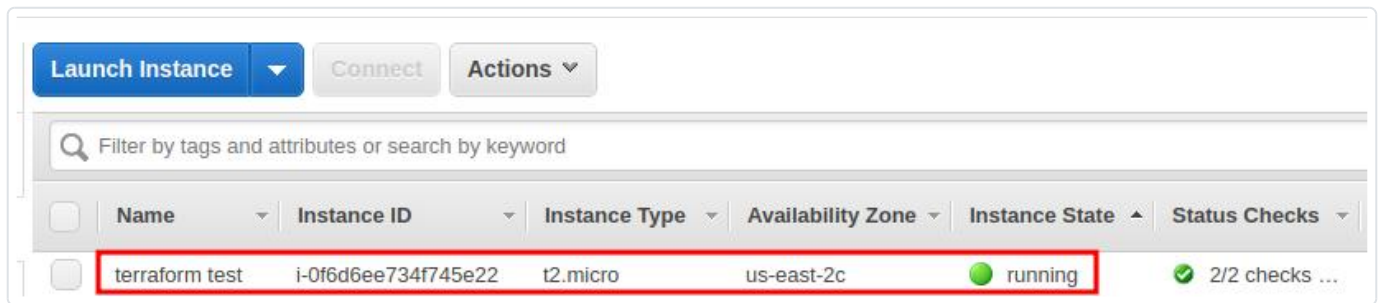
```
Resource actions are indicated with the following symbols:  
~ update in-place  
  
~ tags = {  
    + "Name" = "terraform test"  
}  
  
Plan: 0 to add, 1 to change, 0 to destroy.  
  
Do you want to perform these actions?  
Terraform will perform the actions described above.  
Only 'yes' will be accepted to approve.
```

```
Enter a value: yes
```

```
aws_instance.my_ec2_instance: Modifying... [id=i-0f6d6ee734f745e22]
```

```
aws_instance.my_ec2_instance: Modifications complete after 6s [id=i-0f6d6ee734f745e22]
```

Le résultat nous affiche clairement que Terraform souhaite créer seulement une Balise **Name** ce qui est exactement ce dont nous avons besoin. Lorsque vous actualisez votre console EC2, vous verrez ce changement :



The screenshot shows the AWS Management Console interface for EC2 instances. At the top, there are buttons for 'Launch Instance', 'Connect', and 'Actions'. Below these is a search bar labeled 'Filter by tags and attributes or search by keyword'. A table lists the instances with columns: Name, Instance ID, Instance Type, Availability Zone, Instance State, and Status Checks. One instance is highlighted with a red box: Name 'terraform test', Instance ID 'i-0f6d6ee734f745e22', Instance Type 't2.micro', Availability Zone 'us-east-2c', Instance State 'running' (indicated by a green dot), and Status Checks '2/2 checks ...' (indicated by a green checkmark).

	Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks
<input type="checkbox"/>	terraform test	i-0f6d6ee734f745e22	t2.micro	us-east-2c	running	2/2 checks ...

Intégration de notre service web

Il existe **différentes façons pour déployer un service web dans instance EC2**. On peut par exemple créer une AMI personnalisée sur laquelle le serveur Web déjà installé, on peut également utiliser certaines instructions terraform dans notre code (nous verrons comment faire cela dans le chapitre dédié aux provisionneurs dans un futur chapitre). Mais encore une fois, dans l'intérêt de garder cet exemple simple, nous allons exécuter un script dans le [user-data](#) de l'instance EC2, qu'AWS exécutera automatiquement au démarrage de l'instance en question. Soit l'exécution du script suivant :

```
#!/bin/bash
sudo apt-get update
sudo apt-get install -y apache2
sudo systemctl start apache2
sudo systemctl enable apache2
sudo echo "<h1>Hello devopssec</h1>" > /var/www/html/index.html
```

Dans ce script, les tâches suivantes seront exécutées dans le user-data :

1. Les packages logiciels de la distribution seront mis à jour.
2. Le service web apache sera installé.
3. Le service apache est lancé et activé via la commande `systemctl`.
4. Une page Web simple est créée pour tester le serveur Web qui renvoie toujours le texte "Hello devopssec".

Rajoutons donc notre script dans notre code Terraform :

```
provider "aws" {
  region = "us-east-2"
  access_key = "votre-clé-d'accès"
  secret_key = "votre-clé-secrète"
}

resource "aws_instance" "my_ec2_instance" {
  ami = "ami-07c1207a9d40bc3bd"
  instance_type = "t2.micro"

  user_data = Hello devopssec</h1>" > /var/www/html/index.html
  EOF

  tags = {
    Name = "terraform test"
  }
}
```

Si vous lancez votre code comme tel, vous ne pourrez pas accéder à votre serveur web depuis l'extérieur car vous devez faire encore une chose avant que votre serveur Web fonctionne. En effet, par défaut AWS n'autorise aucun trafic entrant ou sortant provenant d'une instance EC2. Pour permettre à l'instance EC2 de recevoir du trafic sur le port HTTP c'est-à-dire sur le port 80, vous devez **créer un [Security Group](#)** qui est le firewall interne de votre serveur :

```
resource "aws_security_group" "instance_sg" {
  name = "terraform-test-sg"

  egress {
```

```

    from_port      = 0
    to_port        = 0
    protocol       = "-1"
    cidr_blocks    = ["0.0.0.0/0"]
  }

  ingress {
    from_port      = 80
    to_port        = 80
    protocol       = "tcp"
    cidr_blocks    = ["0.0.0.0/0"]
  }
}

```

Ce code va donc créer une nouvelle ressource appelée `aws_security_group` et spécifie que ce Security Group autorise les requêtes TCP entrantes sur le port 80 à partir du bloc CIDR 0.0.0.0/0. Les blocs CIDR sont un moyen concis de spécifier des plages d'adresses IP. Par exemple, un bloc CIDR de 192.168.0.0/24 représente toutes les adresses IP entre 192.168.0.0 et 192.168.0.255. Dans notre cas , le bloc CIDR 0.0.0.0/0 est une plage d'adresses IP qui inclut toutes les adresses IP possibles (soit ouvert au réseau internet), donc ce Security Group autorisera les demandes entrantes sur le port 80 à partir de n'importe quelle IP.

La simple création d'un Security Group ne suffit pas, vous devez également indiquer à votre instance EC2 de l'utiliser en passant l'ID du Security Group dans l'argument `vpc_security_group_id`.

Sur la plus part des ressources, Terraform renvoie des valeurs. Voici les [attributs qui sont renvoyés par une ressource de type Security Group](#). Nous aurons d'ailleurs l'occasion de découvrir d'autres types d'attributs dans cette série d'articles. Pour **récupérer la valeur d'un attribut Terraform** nous utiliserons la syntaxe suivante :

```
<FOURNISSEUR>_<TYPE>.<NOM>.<ATTRIBUT>
```

- **FOURNISSEUR** : le nom du fournisseur (ici "aws").

- **TYPE** : le type de la ressource (ici "security_group").
- **NOM** : le nom de cette ressource (ici "instance_sg").
- **ATTRIBUT** : l'un des arguments de cette ressource (par exemple "name") ou l'un des attributs exportés (ici c'est l'attribut "id" qui est concerné).

Le code final, ressemblera donc à ceci :

```
provider "aws" {
  region = "us-east-2"
  access_key = "votre-clé-d'accès"
  secret_key = "votre-clé-secrète"
}

resource "aws_security_group" "instance_sg" {
  name = "terraform-test-sg"

  egress {
    from_port      = 0
    to_port        = 0
    protocol        = "-1"
    cidr_blocks     = ["0.0.0.0/0"]
  }

  ingress {
    from_port      = 80
    to_port        = 80
    protocol        = "tcp"
    cidr_blocks     = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "my_ec2_instance" {
  ami = "ami-07c1207a9d40bc3bd"
  instance_type = "t2.micro"
  vpc_security_group_ids = [aws_security_group.instance_sg.id]

  user_data = Hello devopssec</h1>" > /var/www/html/index.html
  EOF

  tags = {
    Name = "terraform test"
  }
}
```

Avant d'exécuter notre code, j'aimerais vous expliquer au préalable ce qui se passera en arrière plan. Lorsque vous exportez un attribut d'une ressource à une autre, vous créez une **dépendance implicite**. Terraform analyse ces dépendances, en construit un graphique de dépendances et l'utilise pour déterminer automatiquement dans quel ordre il doit créer les ressources. Par exemple, si vous déployez ce code à partir de zéro, Terraform sait qu'il doit créer le Security Group avant votre instance EC2, car l'instance EC2 fait référence à l'ID du Security Group. Il peut donc appliquer vos modifications assez efficacement.

Si vous exécutez la commande `apply`, vous verrez que Terraform ajoute un Security Group et remplace l'instance EC2 par une nouvelle instance contenant votre service apache :

```
terraform init && terraform apply
```

Résultat :

```
Terraform will perform the following actions:
# aws_security_group.instance_sg will be created
+ resource "aws_security_group" "instance_sg" {
  ...
  + ingress          = [
    + {
      + cidr_blocks      = [
        + "0.0.0.0/0",
      ]
      + from_port        = 80
      + protocol         = "tcp"
      ...
      + to_port          = 80
    },
  ]
}
...
aws_instance.my_ec2_instance: Modifications complete after 9s [id=i-0b25fb9f0b17211e7
```

Pour vérifier que votre Security Group a bien été créé, rendez-vous sur la [console EC2](#), et cliquez sur "Security Groups" sur le menu de droite :

The screenshot displays the AWS Management Console interface for the 'Security Groups' page. On the left-hand side, the navigation menu is visible, with 'Security Groups' highlighted under the 'NETWORK & SECURITY' category. The main content area shows the 'Security Groups (1/1)' overview. A search bar at the top allows filtering by 'Security group ID', with the value 'sg-07e6921e0c5dc4ca4' entered and highlighted by a red box. Below the search bar, a table lists the security groups. The first entry, 'sg-07e6921e0c5dc4ca4', is also highlighted by a red box. The table columns are 'Security group ID', 'Security group name', and 'VPC ID'. The security group name is 'terraform-test-sg' and the VPC ID is 'vpc-091ce362'. Below the table, the details for the selected security group are shown. The 'Inbound rules' tab is active, displaying a table of inbound rules. The first rule is highlighted by a red box, showing 'HTTP' as the type, 'TCP' as the protocol, and '80' as the port range. The table has columns for 'Type', 'Protocol', and 'Port range'. The 'Edit inbound rules' button is visible in the top right corner of the inbound rules section.

Security group ID	Security group name	VPC ID
sg-07e6921e0c5dc4ca4	terraform-test-sg	vpc-091ce362

Type	Protocol	Port range
HTTP	TCP	80

Copiez ensuite l'IP de votre instance ec2 en cochant votre instance, et vous verrez tout en bas de la page la description de votre instance avec son adresse IP, copiez la et rendez-vous sur votre navigateur afin de vérifier votre page web :

Name	Instance ID	Instance Type	Availability Zone	Instance State	Status Checks	Alarm
terraform test	i-0b25fb9f0b17211e7	t2.micro	us-east-2b	running	2/2 checks ...	Non
terraform test	i-0f6d6ee734f745e22	t2.micro	us-east-2c	terminated		Non

Hello devopssec

Nettoyer votre infrastructure

Lorsque vous avez terminé vos expériences avec Terraform, vous pouvez **supprimer toutes les ressources** que vous avez créées afin qu'AWS ne vous les facture pas (ne vous inquiétez les services utilisés dans cet article sont gratuits dans l'offre gratuite d'AWS). Puisque Terraform garde une trace des ressources que vous avez créées, le

nettoyage reste simple. Il vous suffit d'exécuter la commande suivante :

```
terraform destroy
```

Résultat :

```
Terraform will perform the following actions:
```

```
# aws_instance.my_ec2_instance will be destroyed
- resource "aws_instance" "my_ec2_instance" {
  ...
}
```

```
# aws_security_group.instance_sg will be destroyed
- resource "aws_security_group" "instance_sg" {
  ...
}
```

```
Enter a value: yes
```

```
aws_instance.my_ec2_instance: Destroying... [id=i-0b25fb9f0b17211e7]
aws_instance.my_ec2_instance: Still destroying... [id=i-0b25fb9f0b17211e7, 10s elapsed]
aws_instance.my_ec2_instance: Still destroying... [id=i-0b25fb9f0b17211e7, 20s elapsed]
aws_instance.my_ec2_instance: Destruction complete after 22s
aws_security_group.instance_sg: Destroying... [id=sg-07e6921e0c5dc4ca4]
aws_security_group.instance_sg: Destruction complete after 2s
```

```
Destroy complete! Resources: 2 destroyed.
```

Une fois que vous avez tapé "yes" et appuyé sur "Entrée", Terraform vous affichera les ressources à détruire et supprimera toutes les ressources dans le bon ordre. Votre compte AWS devrait à nouveau être neuf :).

Conclusion

Vous avez maintenant une compréhension de base de l'utilisation de Terraform. Vous remarquerez que le code Terraform reste facile à lire et simplifie la description exacte de l'infrastructure que vous souhaitez créer. Nous aurons l'occasion prochainement de créer une infrastructure plus complexe avec le fournisseur AWS, pour le moment j'ai voulu vous montrer comment il est simple de déployer des

ressources avec Terraform, dans le futur chapitre nous étudierons le système de variables dans Terraform afin de mieux sécuriser et personnaliser notre code actuel.