LES BACKENDS ET LES WORKSPACES

Introduction

Comme je vous l'avais promis, aujourd'hui nous allons **expliquer ce qu'est le state dans Terraform**, et comment vous devez le gérer. nous aurons également l'occasion de voir **comment gérer plusieurs environnements avec les workspaces**. Sans plus tarder, commençons !

Les states (états)

Pourquoi les états ?

Terraform nécessite une sorte de base de données pour pouvoir **cartographier votre infrastructure**. Cette base de données est ce qu'on appelle communément "State" ou "état" traduit en français. Cet état est stocké par défaut au format json dans un fichier local nommé **terraform.tfstate**, mais il peut également être stocké à distance dans le cloud.

Ce fichier d'état est donc indispensable pour créer des plans et apporter ainsi des modifications à votre infrastructure. En effet, lorsque vous exécutez la commande `terraform apply` pour la première fois, Terraform créera le nouveau fichier **terraform.tfstate** en local. Et quand vous souhaitez effectuer des modifications dans votre code Terraform, et que vous appuyez à nouveau sur la commande `terraform apply`, alors Terraform apportera des modifications sur ce même fichier pour ainsi tracer les changements à apporter sur votre infrastructure. Et c'est ainsi que Terraform assure le suivi de votre infrastructure réelle.

> **Information**
>
> Il existe également une sauvegarde de l'état précédent dans un fichier appelé `terraform.tfstate.backup`.

Vous l'aurez compris, ce fichier reste très important pour **suivre les changements effectués sur votre infrastructure**, vous aurez donc tendance à le stocker séparément des autres fichiers de votre projet Terraform.

Vous pouvez par exemple conserver ce fichier `terraform.tfstate` dans votre contrôleur de version git, car au final ça ne reste qu'un gros fichier Json. Cependant, gardez à l'esprit que lorsque vous travaillez avec Terraform en équipe, l'utilisation d'un fichier d'état local ou sur git complique l'utilisation de Terraform. Pourquoi ? Car chaque utilisateur doit s'assurer qu'il dispose toujours des dernières données d'état avant d'exécuter Terraform et doit s'assurer que personne d'autre n'exécute Terraform en même temps.

L'état en local fonctionne très bien au début, mais lorsque votre produit devient plus gros, vous voudrez peut-être **stocker votre state à distance** et c'est ce dont je vais parler maintenant.

Les backends

Avant de vous parler de stockage distant, laissez-moi d'abord vous **définir ce que c'est les** `backends`. Un "backend" dans Terraform détermine comment l'état est chargé et comment une telle opération doit être exécutée avec la commande `terraform apply`. C'est grâce à cette instruction que nous pouvons par exemple définir le type de stockage distant.

Par défaut, Terraform utilise le backend `local`, qui est le comportement normal de Terraform auquel vous êtes habitué. C'est le backend qui était utilisé tout au long de ce cours.

Les backends sont complètement facultatifs. Vous pouvez tout à fait utiliser Terraform avec succès sans jamais avoir à apprendre ou à utiliser des backends. Cependant, ils résolvent les problèmes de synchronisation présentés précédemment. Donc même si vous avez uniquement l'intention d'utiliser le backend "local", il peut être utile de vous renseigner sur les backends distants afin de modifier le comportement du backend par défaut.

Définition du Remote state (État distant)

Avec le remote state, Terraform écrit les données d'état dans un stockage distant, qui peut ensuite être partagé entre tous les membres d'une équipe en lecture seule. Si j'ai besoin par exemple de récupérer l'IP publique, je peux facilement retrouver cette information dans notre fichier **terraform.tfstate** (rappel : c'est ce fichier qui sera stocké à distance) :

```
grep public_ip terraform.tfstate
```

Résultat :

```
"public_ip": "54.81.193.196"
```

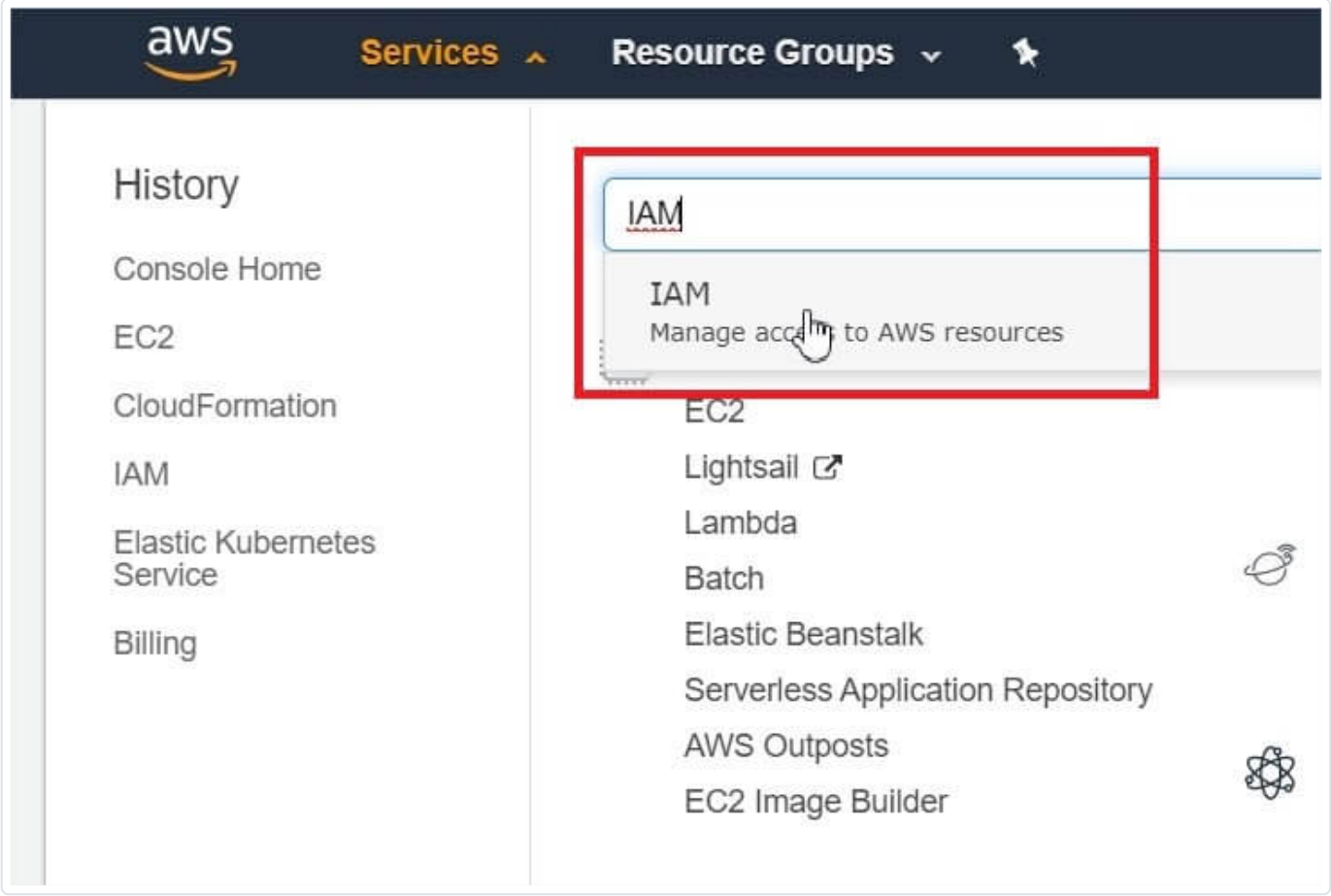
Terraform prend en charge le stockage de l'état dans plusieurs providers dont le service S3 (Simple Storage Service) d'AWS, qui est le service de stockage de données en ligne dans le cloud AWS, et **nous utiliserons le service S3 dans notre remote backend** en tant qu'exemple pour cet article.

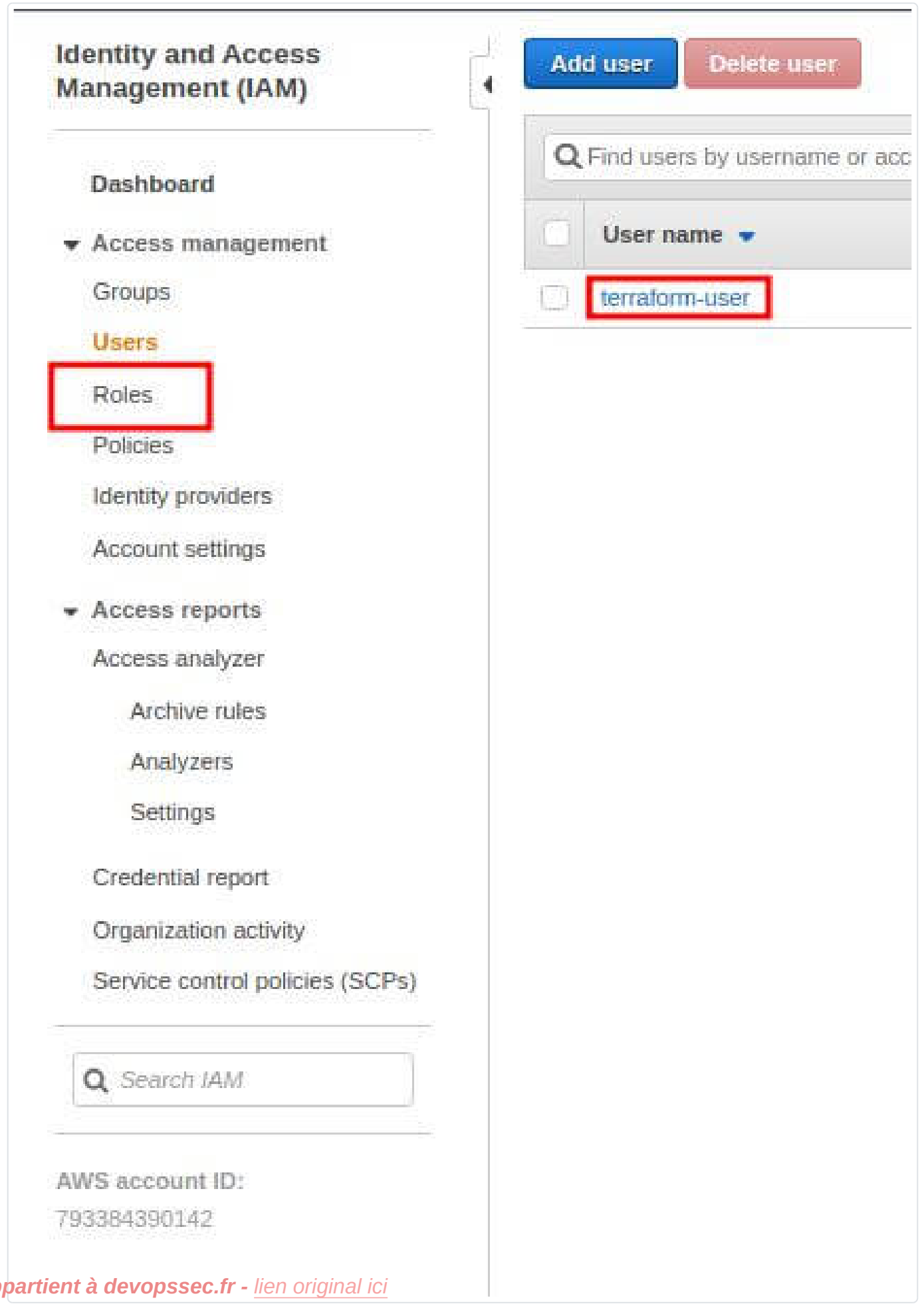
Préparation des prérequis

Configuration de notre utilisateur IAM

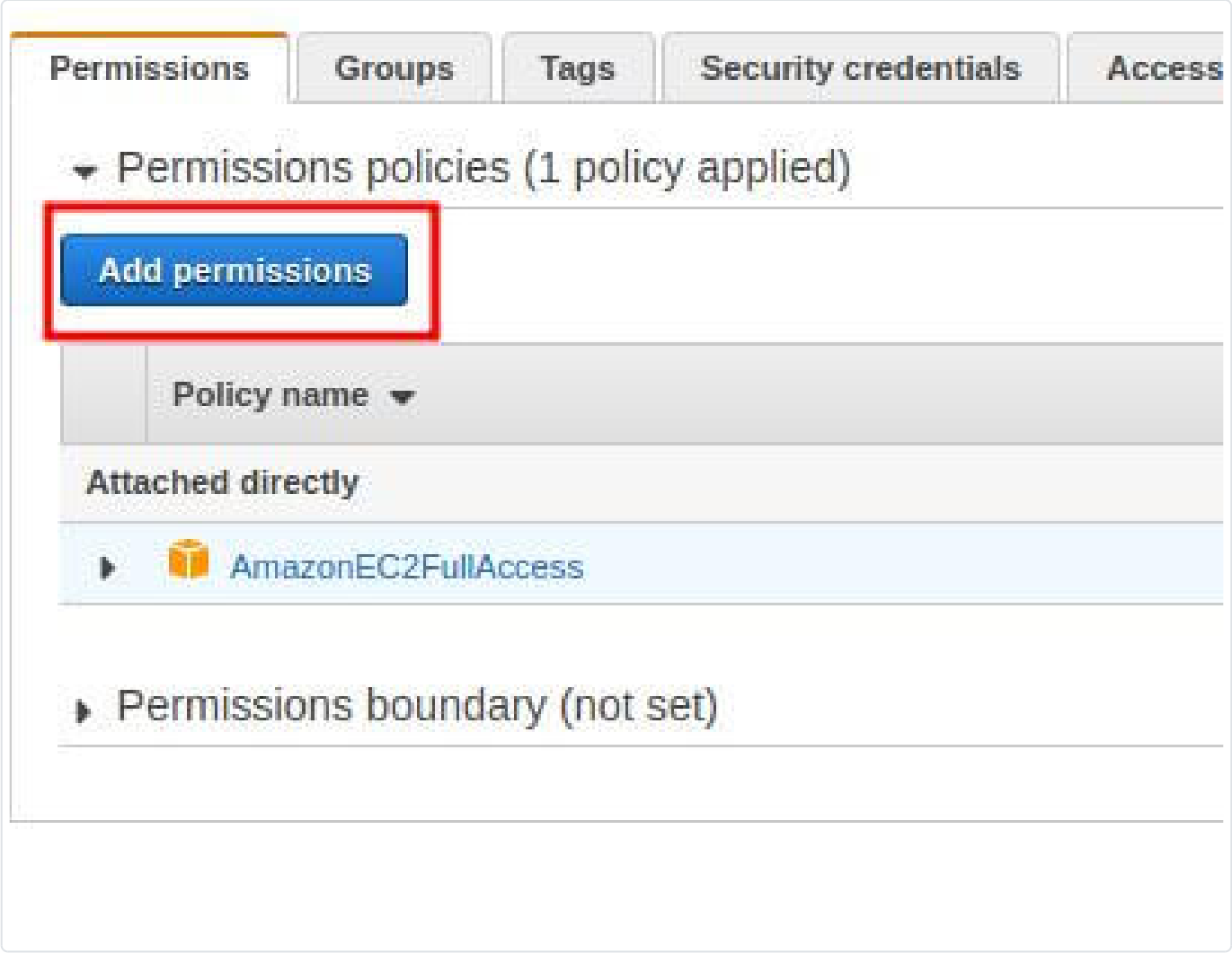
Rappelez-vous, les stratégies IAM sont un mécanisme permettant d'affiner les privilèges d'accès des utilisateurs IAM. Nous avions déjà eu l'occasion de créer notre utilisateur IAM dans cet article avec la policy "AmazonEC2FullAccess", maintenant pour pouvoir utiliser le service S3, nous devons rajouter notre nouvelle stratégie préconfigurée par AWS nommée "AmazonS3FullAccess", qui comme son nom l'indique fournira les autorisations totales au service S3 pour notre utilisateur IAM anciennement créé. Je vais donc vous présenter ci-dessous les **étapes à suivre pour créer la stratégie "AmazonS3FullAccess"** :

Après vous être connecté à votre console AWS, passez à la section IAM, et cliquez ensuite sur "Users" et assurez-vous de sélectionner votre utilisateur IAM.

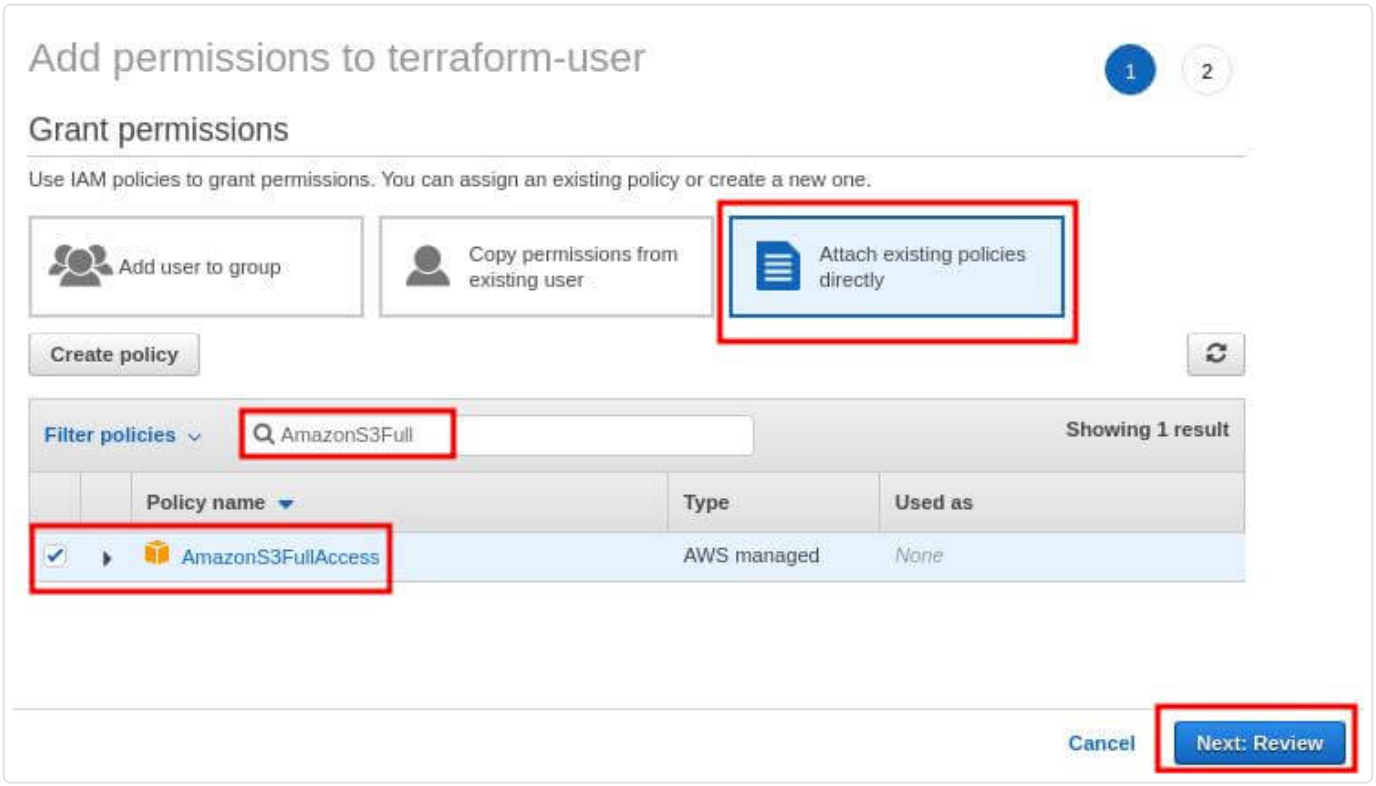




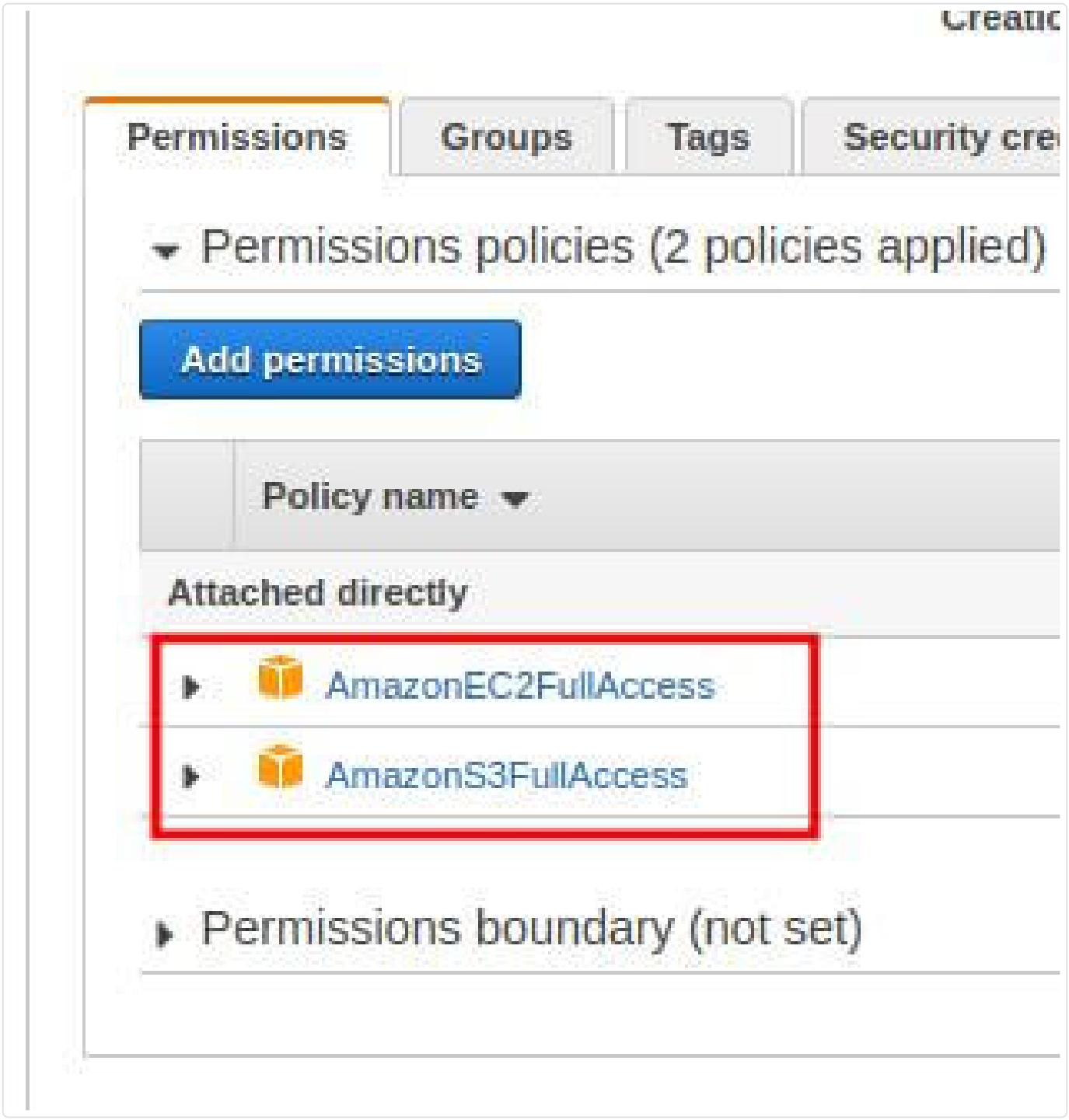
Une fois votre utilisateur sélectionné, une nouvelle page apparaît listant les permissions actuelles de votre utilisateur IAM, cliquez ensuite sur le bouton "Add permissions".



Sur l'écran suivant, vous devez choisir "Attach existing policies directly" et sélectionnez la policy "AmazonS3FullAccess" et ensuite cliquez sur le bouton "Next Review" et enfin "Add permissions" :



Voici à quoi ressemblent maintenant nos nouvelles policies :



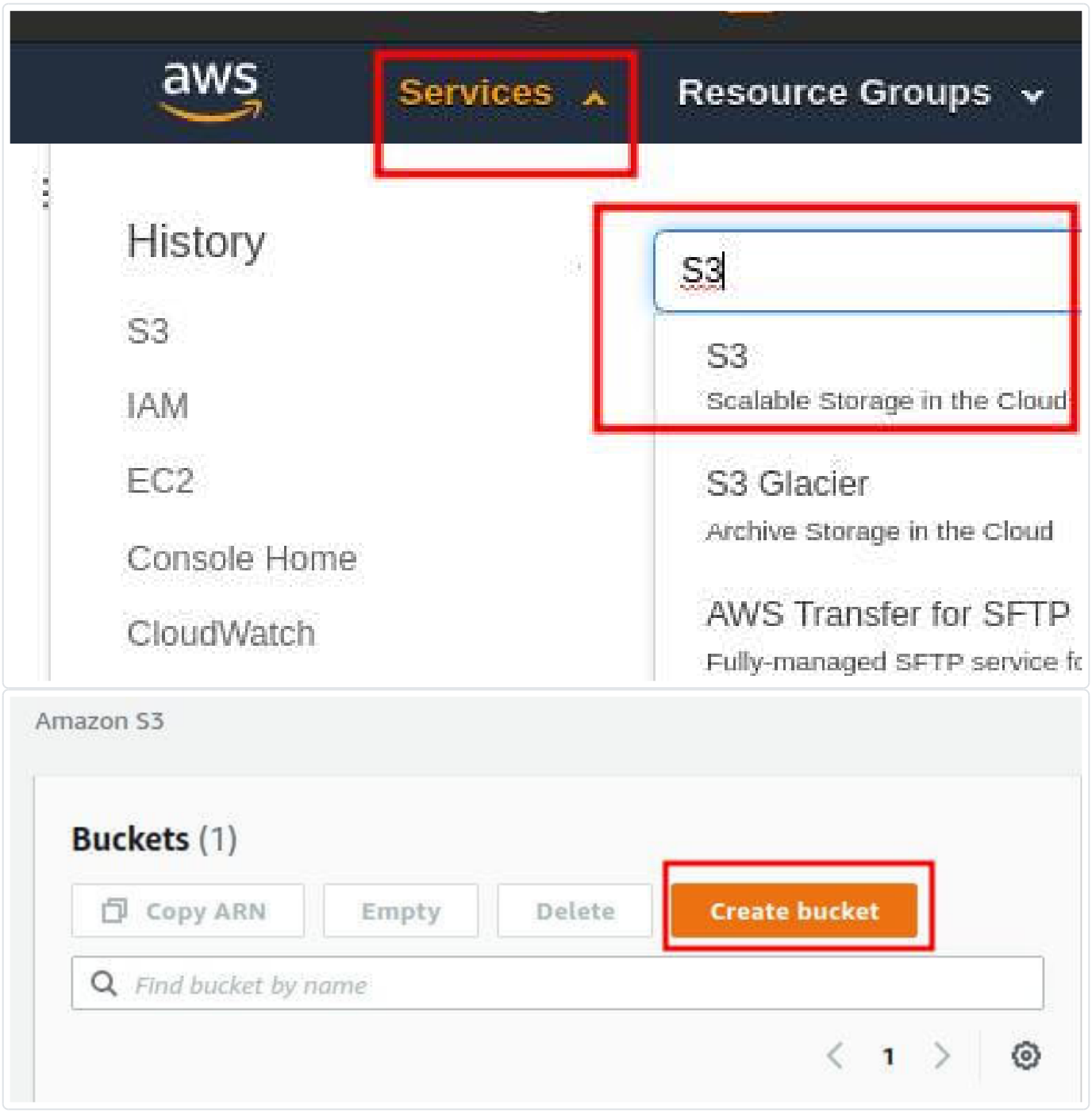
Création de notre bucket S3

Maintenant que vous avez configuré les autorisations de votre utilisateur IAM, vous êtes prêt à **créer un bucket S3** ("compartiment" en français) à l'aide de la console AWS où y sera stocké votre fichier d'état.

Information

Un objet est le nom donné aux fichiers stockés dans un bucket S3.

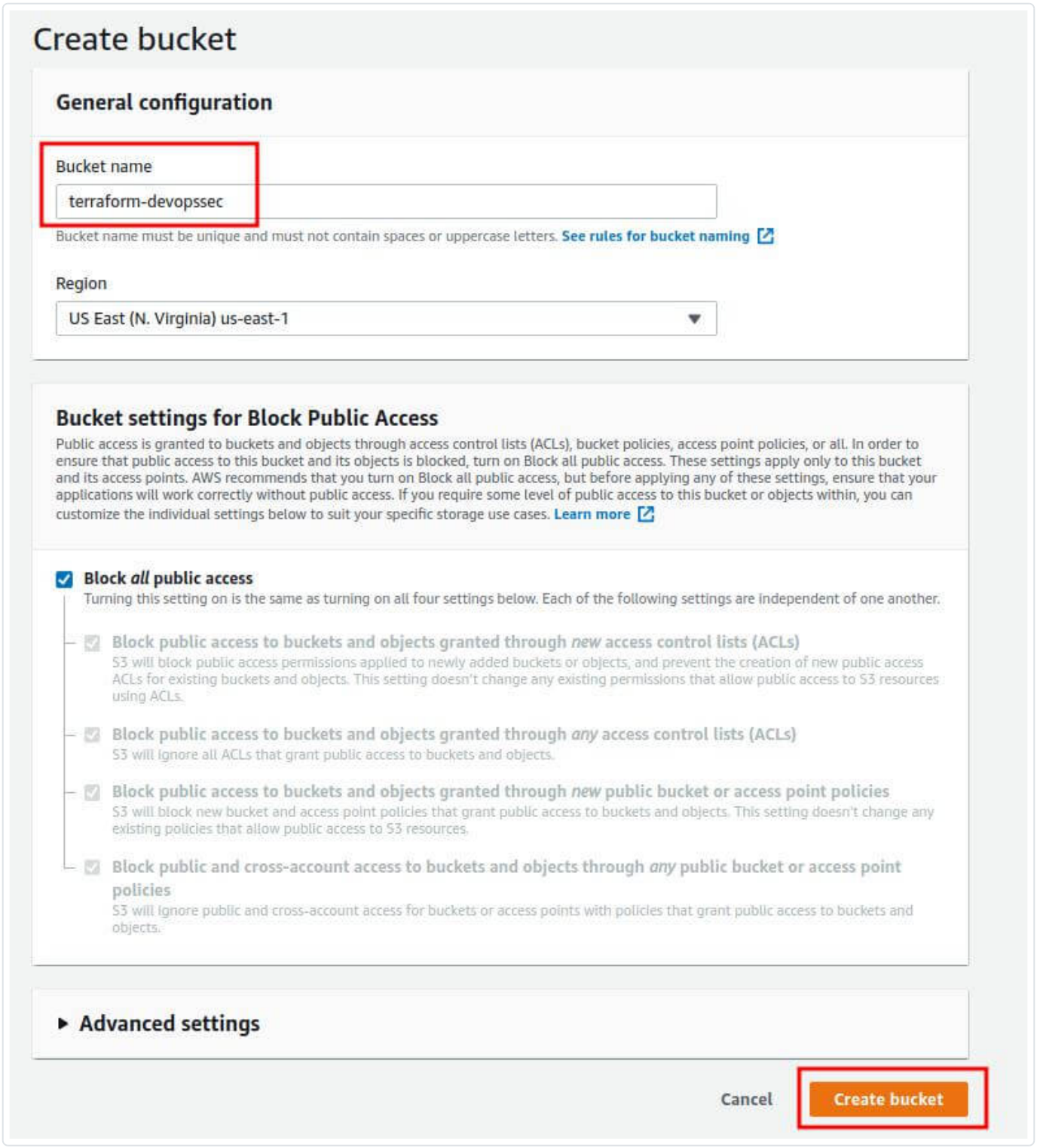
Commencez déjà par ouvrir la console S3 et cliquez sur le bouton "Create bucket" :



Ensuite assurez-vous que le nom de votre bucket respecte les règles suivantes :

- Il doit être unique sur l'ensemble des noms de bucket mondiaux d'Amazon S3.
- Compter entre 3 et 63 caractères.
- Ne contient pas de caractères majuscules.
- Commencez par une lettre minuscule ou un chiffre.

Une fois ces règles respectées, créez votre bucket en cliquant sur le bouton "Create bucket" :



Installation du CLI AWS

Sauvegardez dans un coin la région et le nom de votre bucket car nous l'utiliserons plus tard dans notre code Terraform. Car pour le moment nous **installerons et**

configurons l'AWS CLI qui permet de contrôler par ligne de commande, divers services AWS. Pour ce faire suivez les étapes suivantes :

Installation du CLI AWS sur Linux

Si ce n'est pas déjà fait, commencez par télécharger les packages suivants nécessaires à l'installation de la cli :

```
sudo apt update -y
sudo apt install -y curl unzip
```

Téléchargez ensuite le programme d'installation fourni avec AWS CLI à l'aide de la commande suivante :

```
curl "https://s3.amazonaws.com/aws-cli/awscli-bundle.zip" -o "awscli-bundle.zip"
```

Ensuite, extrayez les fichiers du package :

```
unzip awscli-bundle.zip
```

Enfin, exécutez le programme d'installation :

```
sudo ./awscli-bundle/install -i /usr/local/aws -b /usr/local/bin/aws
```

Vérifiez ensuite votre installation avec la commande suivante :

```
aws --version
```

Résultat :

```
aws-cli/1.18.42 Python/2.7.17 Linux/5.3.0-46-generic botocore/1.15.42
```

Installation du CLI AWS sur Windows

Pour la partie windows, il suffit de choisir le msi correspondant à votre architecture (dans mon cas c'est le 64 bits), retrouvable sur la page d'installation du cli aws pour Windows.

Configuration de l'AWS CLI

Nous allons maintenant configurer notre AWS CLI afin d'utiliser par défaut notre compte IAM modifié précédemment, pour cela il suffit de lancer la commande suivante :

```
aws configure
```

Résultat :

```
AWS Access Key ID [None]: VOTRE_CLE_D_ACCES
AWS Secret Access Key [None]: VOTRE_CLE_SECRETE
Default region name [None]: us-west-1 (REMPLACEZ PAR VOTRE REGION)
Default output format [None]: json
```

Sur linux, vous retrouverez ces informations saisies directement dans le chemin suivant sur Linux:

```
ls -l   ~/.aws/
```

Résultat :

```
-rw------- 1 hatim hatim  43 avril 21 10:40 config
-rw------- 1 hatim hatim 116 avril 21 10:40 credentials
```

Sinon sur Windows, ces informations sont stockées ici : **C:\Users\%username%\.aws**

Le chemin du fichier **credentials** est le chemin d'accès au fichier d'informations d'identification qui sera utilisé plus tard dans notre backend s3.

Déclenchement de notre remote state

Voilà nous avons configuré les prérequis nécessaires pour lancer notre backend s3. Il ne reste plus qu'à l'utiliser dans notre code terraform, voici un exemple d'intégration :

```
provider "aws" {
    region = "us-east-1"
}

resource "aws_instance" "my_ec2_instance" {
    ami = "ami-085925f297f89fce1"
    instance_type = "t2.micro"
}

terraform {
  backend "s3" {
    bucket = "terraform-devopssec"
    key    = "states/terraform.state"
    region = "us-east-1"
  }
}
```

Voici une petite explication des arguments de notre bloc de code `backend` :

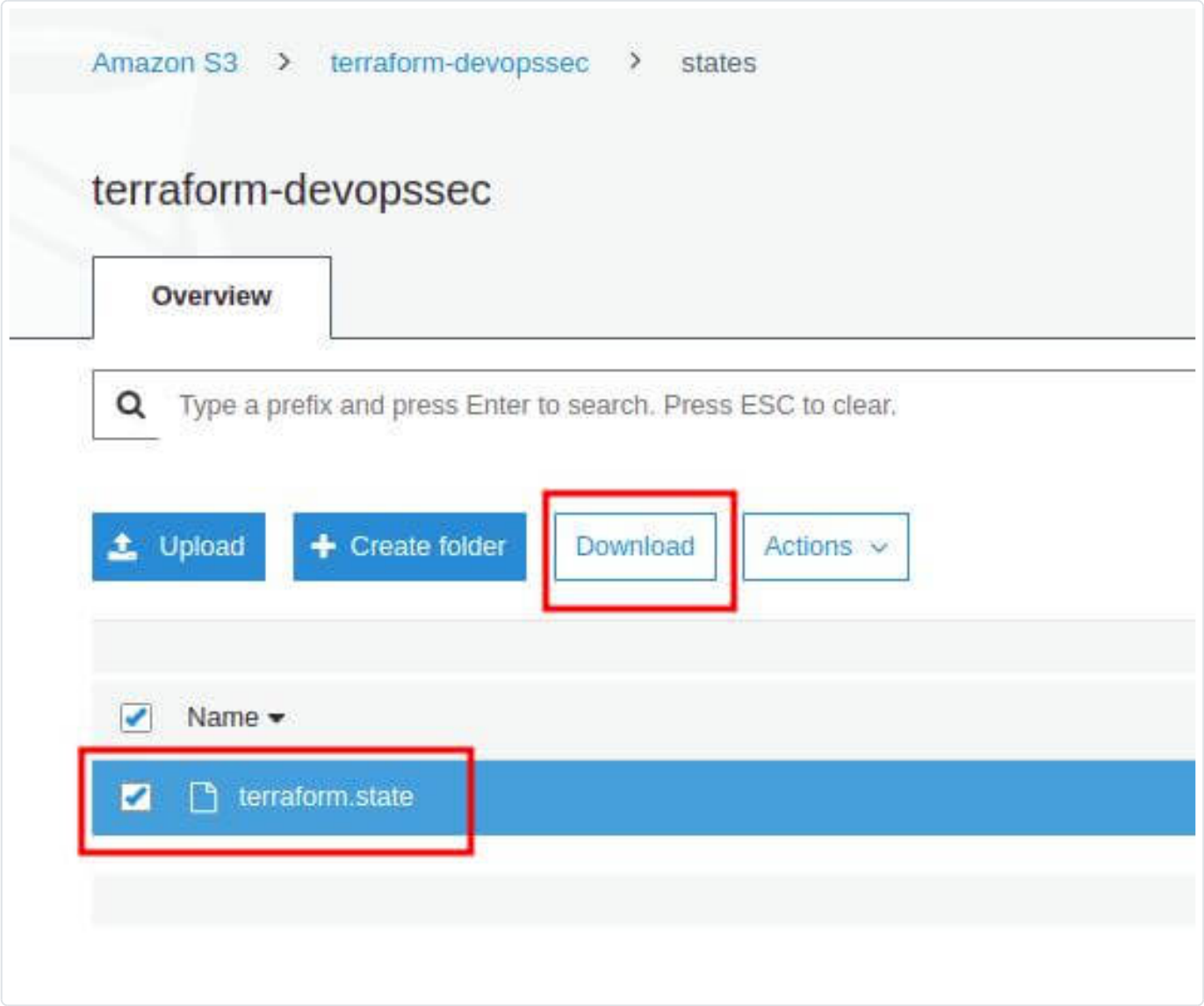
- `bucket` : le nom du bucket S3 dans votre compte.
- `key` : chemin dans votre bucket et le nom de votre objet S3.
- `region` : région de votre bucket s3.

Vous remarquerez que je n'ai pas spécifié cette fois-ci dans le bloc `provider` notre `access_key` `secret_key` car par défaut Terraform vérifie directement le fichier **~/.aws/credentials** quand ces informations ne sont pas spécifiées.

Vous pouvez maintenant lancer votre commande d'exécution Terraform :

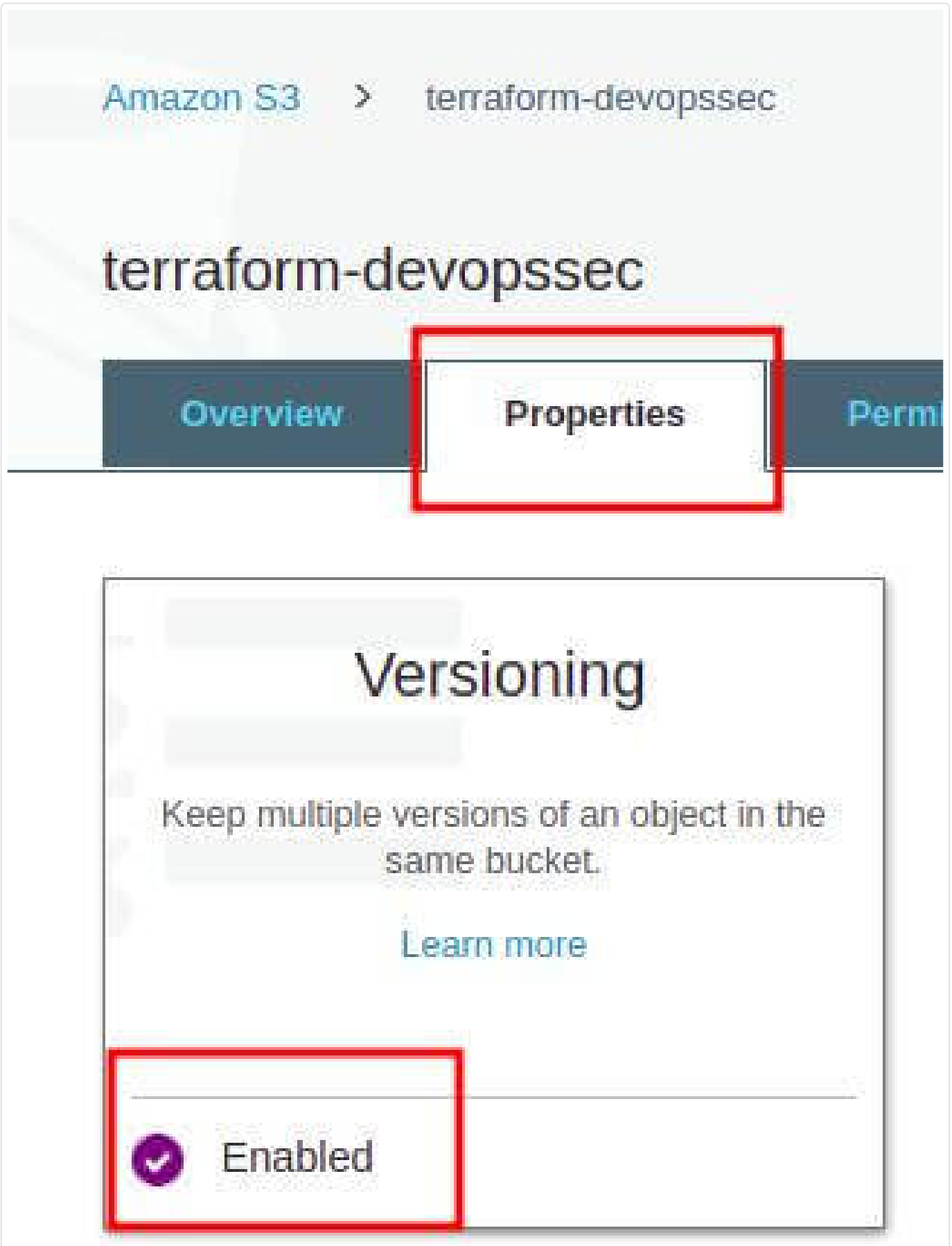
```
terraform init && terraform apply
```

Pour finir, rendez-vous dans votre bucket S3 et vérifiez si le fichier est bien existant. Vous pouvez même le télécharger pour vérifier son contenu :

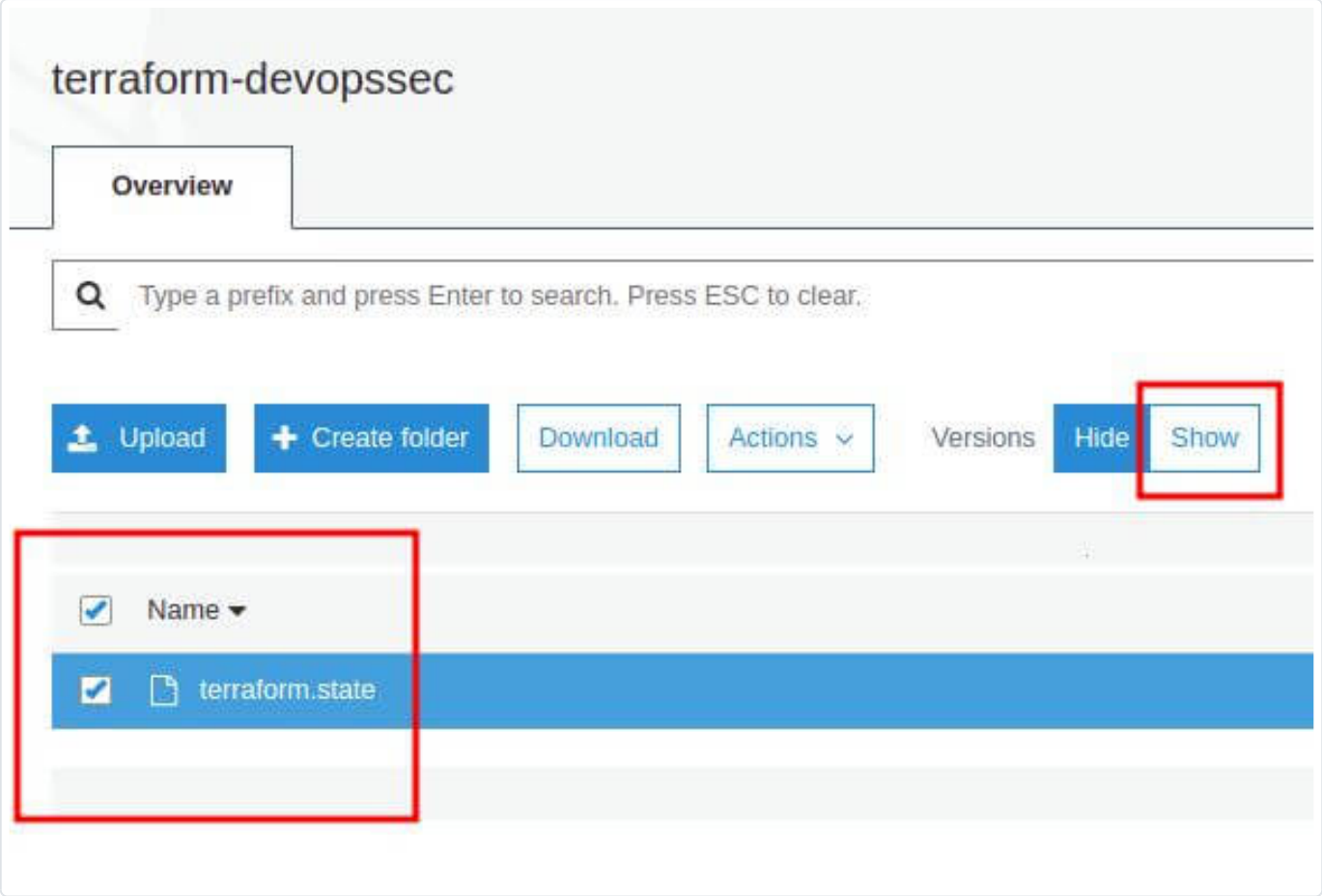


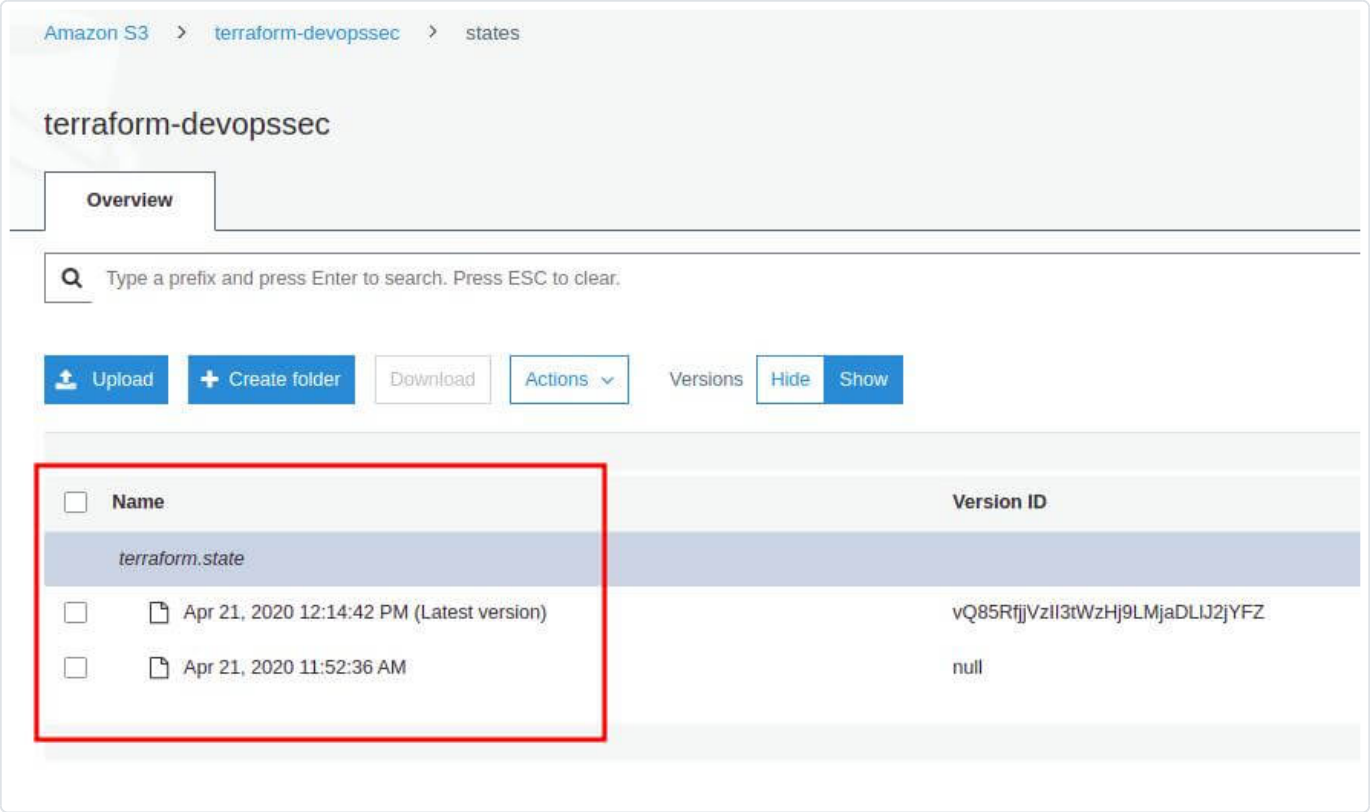
Activer le versioning dans S3

Il est fortement recommandé d'activer le "Versioning" sur notre bucket S3 pour permettre la récupération de l'état en cas de suppressions accidentelles, d'erreurs humaines et de pouvoir tracer tous les changements de votre infrastructure. Pour ce faire, rendez-vous dans votre console S3 et sélectionnez votre bucket, ensuite allez sur "Properties" et activez le "Versioning" :



Dorénavant, si vous modifiez votre infrastructure, alors votre état précédent sera également enregistré. Pour visualiser vos anciennes versions du fichier d'état, sélectionnez votre objet S3 et clique sur le bouton "Show" prêt du champ "Versions" :





Sécurité

J'aimerais ajouter d'autres avantages de l'état distant. Les backends exposent généralement des moyens de **configurer les autorisations d'accès**.

Par exemple, pour AWS, en utilisant des stratégies IAM dans votre bucket S3, vous pouvez contrôler qui a accès à vos fichiers d'état, ce qu'on a eu tendance à faire avant pour donner un accès total à notre utilisateur IAM terraform. Vous pouvez reproduire cette action pour vos autres utilisateurs afin de limiter leurs droits. Dans ce cas, vous pouvez soit personnaliser vous-même votre stratégie en utilisant la syntaxe json, ou prendre une stratégie déjà managée, comme celle de ci-dessous qui donne un accès en lecture seulement à tous vos buckets S3 :

Vous pouvez également **créer des Bucket Policies**, qui sont attachées uniquement à votre bucket et non à un utilisateur IAM, comme peuvent l'être les stratégies IAM. Ces stratégies sont plus granulaires et spécifient les conditions d'actions autorisées ou refusées de votre bucket S3. Dans notre exemple nous allons refuser l'accès à la plage d'IP 192.168.1.0/24 :

Pour cela, sélectionnez votre bucket S3, et rendez-vous sur l'onglet "Permissions" et cochez le bouton "Bucket Policy", enfin rajoutez-y le code suivant :

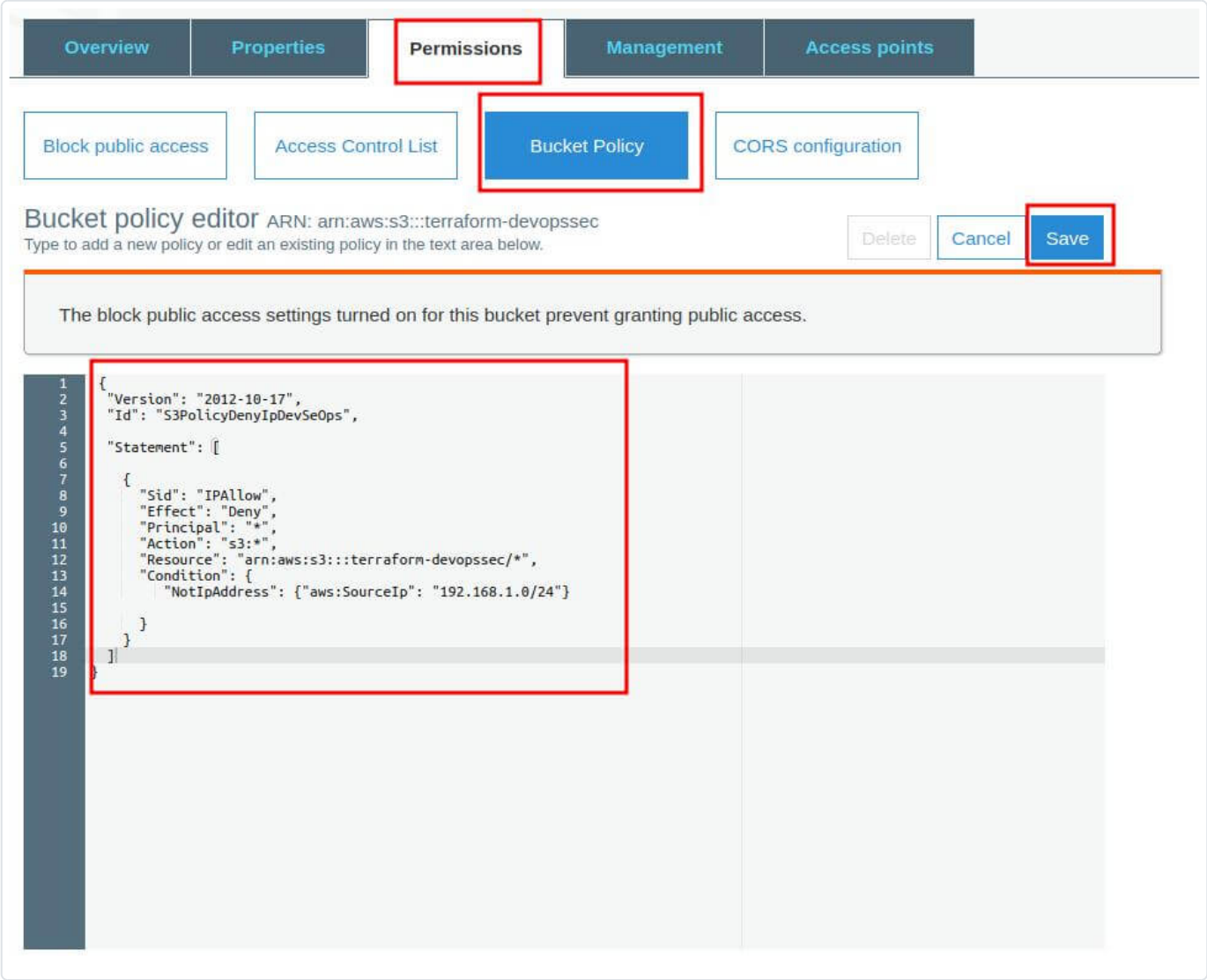
```
{
 "Version": "2012-10-17",
 "Id": "S3PolicyDenyIpDevSeOps",

 "Statement": [

   {
     "Sid": "IPAllow",
     "Effect": "Deny",
     "Principal": "*",
     "Action": "s3:*",
     "Resource": "arn:aws:s3:::terraform-devopssec/*",
     "Condition": {
        "NotIpAddress": {"aws:SourceIp": "192.168.1.0/24"}

     }
   }
 ]
}
```

Cliquez ensuite sur le bouton "Save".



Information

Si vous souhaitez créer des politiques personnalisées qui contrôlent l'accès à vos ressources AWS, alors vous pouvez utiliser l'outil de génération de stratégies AWS pour vous faciliter la vie.

Toujours en parlant de sécurité, il faut toujours garder à l'esprit que le service S3 reste un service géré par AWS, vous n'avez donc pas à déployer et à gérer une infrastructure supplémentaire pour l'utiliser, d'ailleurs il est conçu pour une

durabilité de 99,99999999999% et une disponibilité de 99,99% , ce qui signifie que vous n'avez pas à vous soucier trop de la perte de données ou des pannes.

Enfin pour finir, la plupart des backends distants prennent en charge nativement le **chiffrement en transit** (TLS/SSL) et le **chiffrement sur disque** de votre fichier d'état. Sur AWS, je vous conseille d'**activer le chiffrement au repos**, c'est-à-dire sur disque dûr. Plus d'informations sur l'article sur le chiffrement de votre bucket S3.

Workspace (Espaces de travail)

Vous le savez maintenant, chaque configuration Terraform a un `backend` associé qui définit comment les opérations sont exécutées et où les données persistantes telles que l'état Terraform ou les variables Outputs sont stockées.

Ces données persistantes stockées dans le backend, appartiennent à ce qu'on appelle un workspace ou "espace de travail" en français. Initialement, le backend n'a qu'un seul espace de travail, nommé `default` et donc il n'y a qu'un seul état Terraform associé à cette configuration.

Cependant certains backends prennent en charge plusieurs espaces de travail nommés, **permettant à plusieurs états d'être associés à une même configuration Terraform**. Cette fonctionnalité nous permet donc en matière de gestion d'infrastructure de **créer plusieurs environnements**, tels que des environnements de tests et de production, avec principalement la même configuration mais en gardant différents états et variables. Le but est de **créer des états différents et indépendants sur la même configuration**. Et comme il est compatible avec le backend distant, ces espaces de travail sont partagés avec votre équipe.

Pour **créer un nouveau workspace**, placez vu sur votre projet Terraform et utilisez la commande suivante :

```
terraform workspace new prod
```

Dans cet exemple nous créons un espace de travail nommé `prod`, maintenant nous devons **sélectionner notre workspace** avec la commande suivante :

```
terraform workspace select prod
```

Résultat :

```
You're now on a new, empty workspace. Workspaces isolate their state,
so if you run "terraform plan" Terraform will not see any existing state
for this configuration.
```

Comme c'est juste un nouveau espace de travail, il n'a aucun autre état précédent sur le quel il doit se baser.

Pour revenir à l'état par défaut , il suffit de rentrer la commande suivante :

```
terraform workspace select default
```

D'ailleurs, dans votre configuration Terraform, vous pouvez inclure le nom de l'espace de travail actuel à l'aide de la variable `${terraform.workspace}`. Dans notre example je vais nommer mon instance EC2 différemment selon le workspace où je suis positionnés (`default` ou `prod`)

```
provider "aws" {
    region = "us-east-1"
}

resource "aws_instance" "my_ec2_instance" {
    ami = "ami-085925f297f89fce1"
    instance_type = "t2.micro"

    tags = {
        Name = "${terraform.workspace == "prod" ? "prod-ec2" : "default-ec2"}"
```

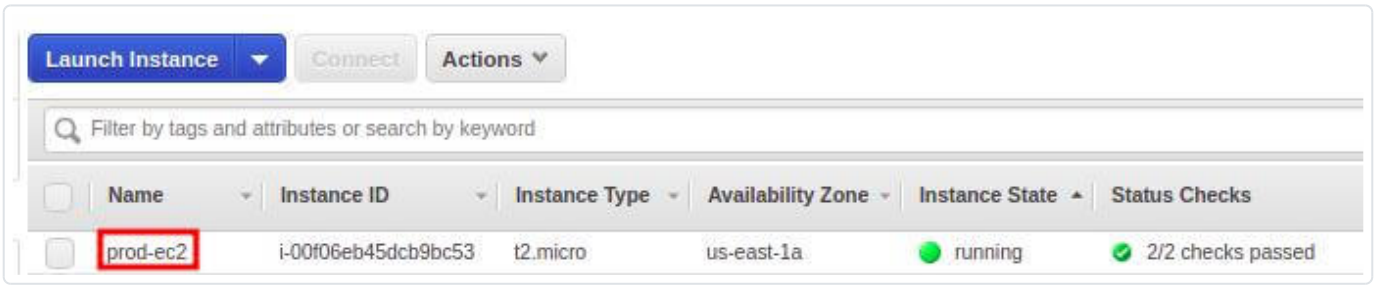
```
    }
}

terraform {
  backend "s3" {
    bucket = "terraform-devopssec"
    key    = "states/terraform.state"
    region = "us-east-1"
  }
}
```

Dans le cas où je suis dans un workspace de prod mon instance EC2 sera nommée `prod-ec2` sinon elle sera nommée `default-ec2`. Exécutons maintenant notre code et vérifions cela :

```
terraform workspace select prod
terraform init && terraform apply
```

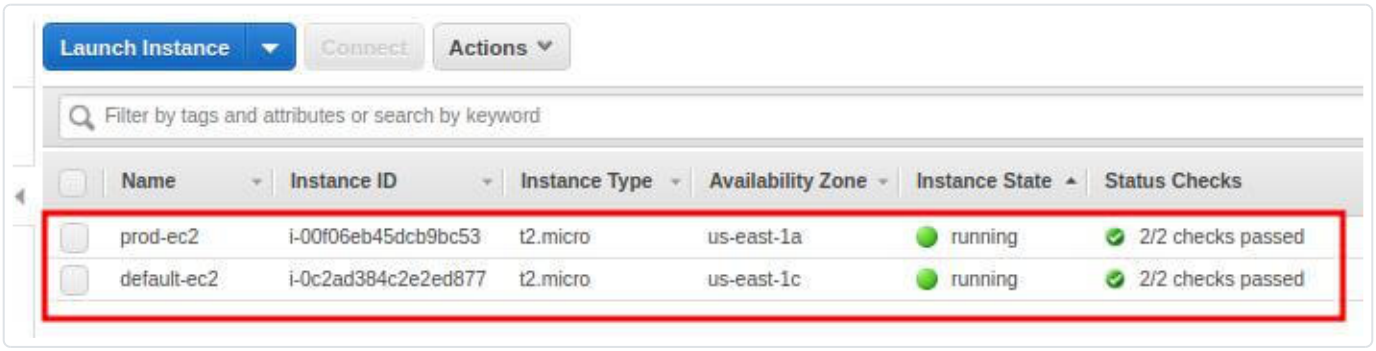
Rendez-vous sur votre console EC2, et regardez vos instances ec2 :



Si vous revenez sur votre workspace par défaut, votre fichier state ne saura pas que vous aviez créé votre instance ec2 avec comme nom `prod-ec2`. Si vous relancez votre code Terraform sur ce workspace, alors il créera une nouvelle instance ec2 au lieu de juste modifier son nom :

```
terraform workspace select default
terraform init && terraform apply
```

Si on revérifie nos instances ec2, nous verrons une nouvelle instance qui apparaît :



Conclusion

Dans ce chapitre, nous avons vu l'intérêt des backends dans Terraform. Ils restent un bon moyen de suivre l'évolution des ressources de votre infrastructure. Nous avons également appris à utiliser les remotes states afin de faciliter et sécuriser le partage du fichier d'état à vos différentes équipes, en gardant ainsi une meilleure synchronisation de ce fichier. Enfin, nous avons finit par étudier l'objectif et l'utilisation des workspaces dans Terraform, afin de vous permettre de travailler sur différents environnements depuis un seul même projet Terraform.