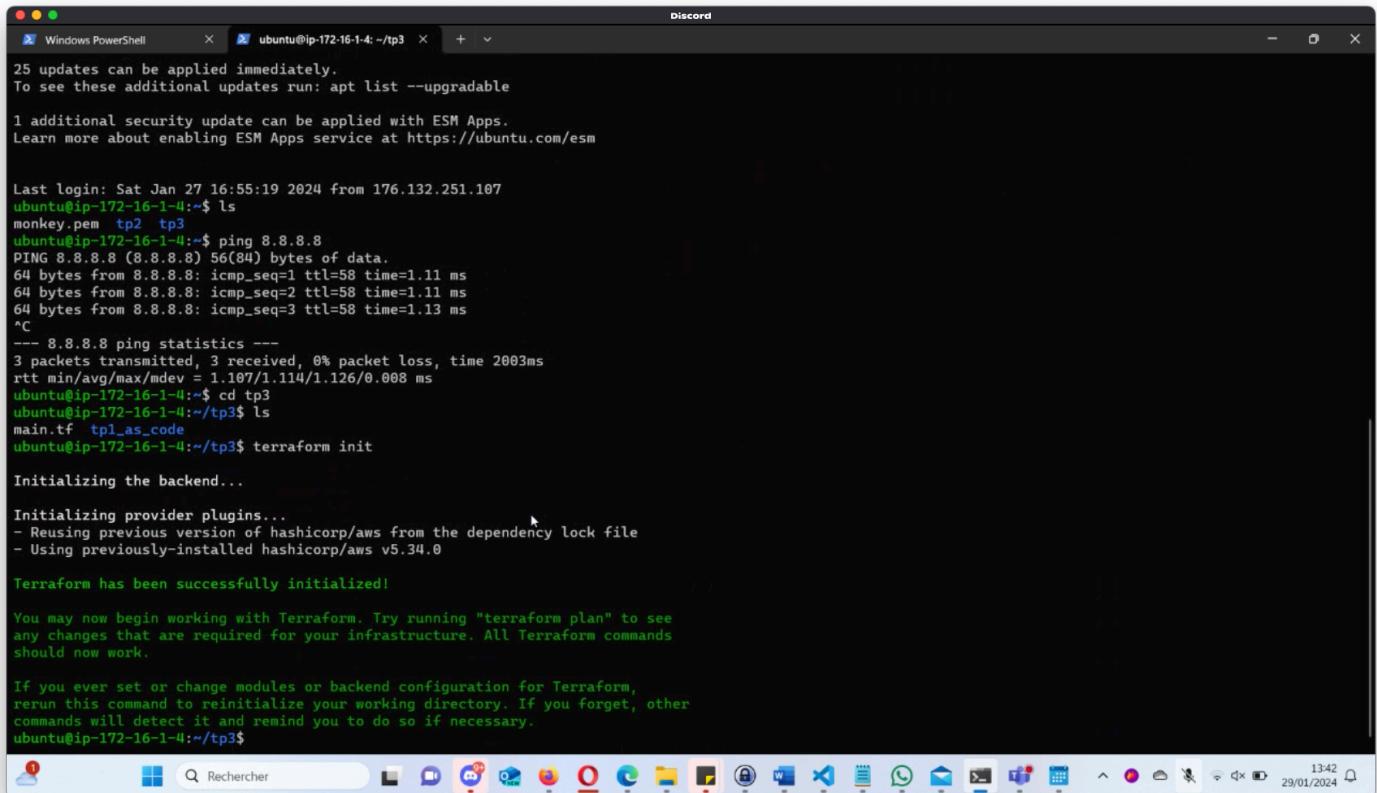


DevOps & Pipelines - Rendu TP03

Rendu effectué par Vincent LAGOGUÉ, Tom THIOULOUSE, Alexis PLEXIGLASSE, Thomas PEUGNET.

Initialisation

Initialisation du projet Terraform à l'aide des commandes `init` et `plan`.



The screenshot shows a Windows PowerShell window titled "Windows PowerShell" running on an Ubuntu system (ip-172-16-1-4). The terminal output shows the following steps:

- System updates: "25 updates can be applied immediately. To see these additional updates run: apt list --upgradable"
- Security update: "1 additional security update can be applied with ESM Apps. Learn more about enabling ESM Apps service at https://ubuntu.com/esm"
- Last login information: "Last login: Sat Jan 27 16:55:19 2024 from 176.132.251.107"
- File listing: "ls" command showing files like "monkey.pem", "tp2", and "tp3".
- Ping test: "ping 8.8.8.8" command showing results for three packets.
- Directory change: "cd tp3"
- File listing: "ls" command showing files "main.tf" and "tpl_as_code".
- Terraform initialization: "terraform init". The output indicates it's initializing the backend and provider plugins. It notes reusing a previous version of hashicorp/aws from the dependency lock file and using previously-installed v5.34.0.
- Success message: "Terraform has been successfully initialized!"
- Final note: "You may now begin working with Terraform. Try running "terraform plan" to see any changes that are required for your infrastructure. All Terraform commands should now work."
- Warning: "If you ever set or change modules or backend configuration for Terraform, rerun this command to reinitialize your working directory. If you forget, other commands will detect it and remind you to do so if necessary."

TP1 as Code

Initialisation du projet `tp_1_as_code` après avoir créé les alias.

```

+ description      = ""
+ from_port       = 22
+ ipv6_cidr_blocks = []
+ prefix_list_ids = []
+ protocol        = "tcp"
+ security_groups = []
+ self             = false
+ to_port          = 22
},
{
+ cidr_blocks     = []
+ from_port       = 0
+ ipv6_cidr_blocks = []
+ prefix_list_ids = []
+ protocol        = "-1"
+ security_groups = [
+   "sg-0c29eed3f3c0ac4bf",
]
+ self             = false
+ to_port          = 0
},
],
name           = "public-20240127171738982200000002"
tags           = {
  "Name" = "public"
}
# (8 unchanged attributes hidden)
}

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

Changes to Outputs:
~ nat_public_ip  = "52.55.87.96" -> ""

Note: You didn't use the -out option to save this plan, so Terraform can't
guarantee to take exactly these actions if you run "terraform apply" now.
ubuntu@ip-172-16-1-4:~/tp3/tpl1_as_code$ |

```

Déploiement du code Terraform à l'aide de la commande `terraform apply`.

```

+ security_groups = [
+   "sg-0c29eed3f3c0ac4bf",
]
+ self             = false
+ to_port          = 0
},
],
name           = "public-20240127171738982200000002"
tags           = {
  "Name" = "public"
}
# (8 unchanged attributes hidden)
}

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

Changes to Outputs:
~ nat_public_ip  = "52.55.87.96" -> ""

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_security_group.public: Modifying... [id=sg-0e4435756a3edad76]
aws_security_group.public: Modifications complete after 0s [id=sg-0e4435756a3edad76]

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

Outputs:
nat_private_ip = "172.16.1.48"
nat_public_ip  = ""
private_host_ip = "172.16.2.10"
ubuntu@ip-172-16-1-4:~/tp3/tpl1_as_code$ |

```

Si vous exécutez `terraform apply`, terraform vous redemandera votre IP et va vous demander validation (tapez yes) puis provisioningera les ressources.

⚠ Notez que les ressources vont avoir les mêmes noms "logiques" que certaines déjà existantes du TP1 (VPC, Routes, ...). Attention à ne pas les confondre dans la console.

Vous pouvez alors reproduire les tests du TP1 :

- se connecter en SSH sur l'ip publique de l'instance `NAT_JumpHost`
- sur `NAT_JumpHost`, copier la clé privée (format PEM, droits 600) et se connecter en SSH sur l'ip privée de l'instance `PrivateHost`
- depuis `PrivateHost`, effectuez les tests `ping 8.8.8.8` et `curl https://www.google.fr` pour vérifier les accès vers Internet à travers l'instance NAT

A présent exécutez `terraform destroy` ou `tfd` dans le même dossier où vous avez lancé `terraform apply`, terraform vous redemandera votre IP et validation (tapez yes) puis il supprimera les ressources.

Ask ChatGPT

Please give me the terraform code for an AWS VPC named "TP_DevOps" on CIDR "172.16.0.0/16" with a public subnet named "TP_DevOps_Public" on CIDR "172.16.1.0/24" and a private subnet named "TP_DevOps_Private" on CIDR "172.16.2.0/24", a NAT Instance named "Nat_JumpHost" using Linux Ubuntu Server and the already existing IAM Profile named LabInstanceProfile, a private instance named

Nous pouvons constater une nouvelle instance en attente de création `NAT_JumpHost`.

The screenshot shows the AWS CloudWatch Metrics Insights interface. A search bar at the top contains the query: `CloudWatch Metrics Insights`. Below the search bar, a table displays metrics from the CloudWatch Metrics Insights service. The table has columns for Metric Name, Unit, Value, and Last Update. One row is highlighted in yellow, showing the metric `CloudWatch Metrics Insights:CloudWatch Metrics Insights` with a value of `1` and last updated at `2024-01-29T10:45:00Z`.

Metric Name	Unit	Value	Last Update
CloudWatch Metrics Insights:CloudWatch Metrics Insights	1	1	2024-01-29T10:45:00Z

Nous avons rencontré un problème concernant la paire de clés, `vockey.pem`. Nous avons donc créé une nouvelle paire de clés nommée `monkey.pem` depuis AWS, donc nous connaissons les crédentails.

Nous avons modifié `main.tf` pour pouvoir utiliser notre paire de clés en lieu et place de `vockey`.

Nous avons ensuite résilié notre instance, puis avons de nouveau exécuté un `terraform apply`.

Dès lors, nous avons réussi à nous connecter sur `NAT_JumpHost` en SSH.

The screenshot shows a dual-monitor setup. The left monitor displays a terminal window with the following content:

```
Windows PowerS x ubuntu@ip-172-1 x ubuntu@ip-172-1 x + - x

Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '54.234.128.39' (ED25519) to the list of known hosts.
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-1052-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/pro

System information disabled due to load higher than 1.0

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

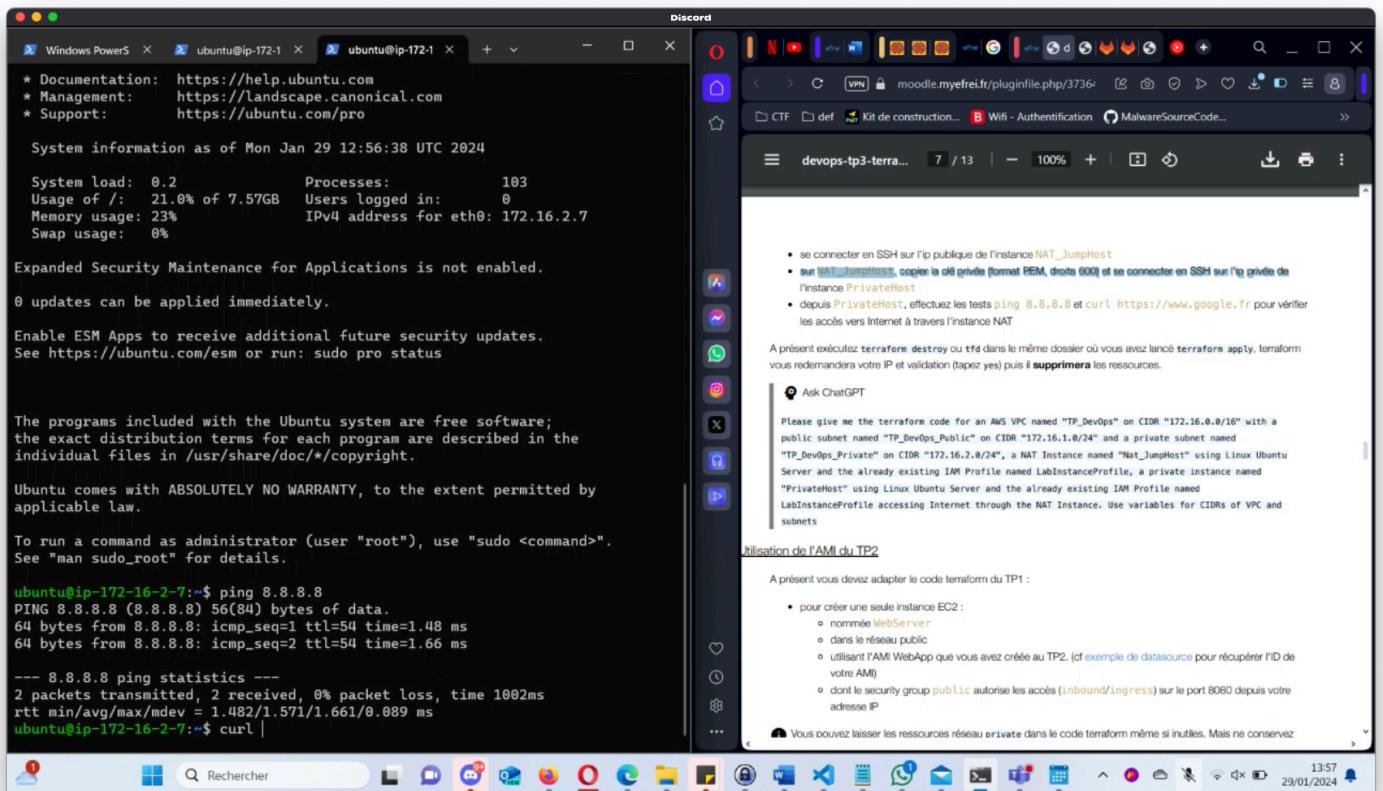
ubuntu@ip-172-16-1-48:~$
```

The right monitor displays a browser window with the URL moodle.myfreel.fr/pluginfile.php/37364. The page content includes:

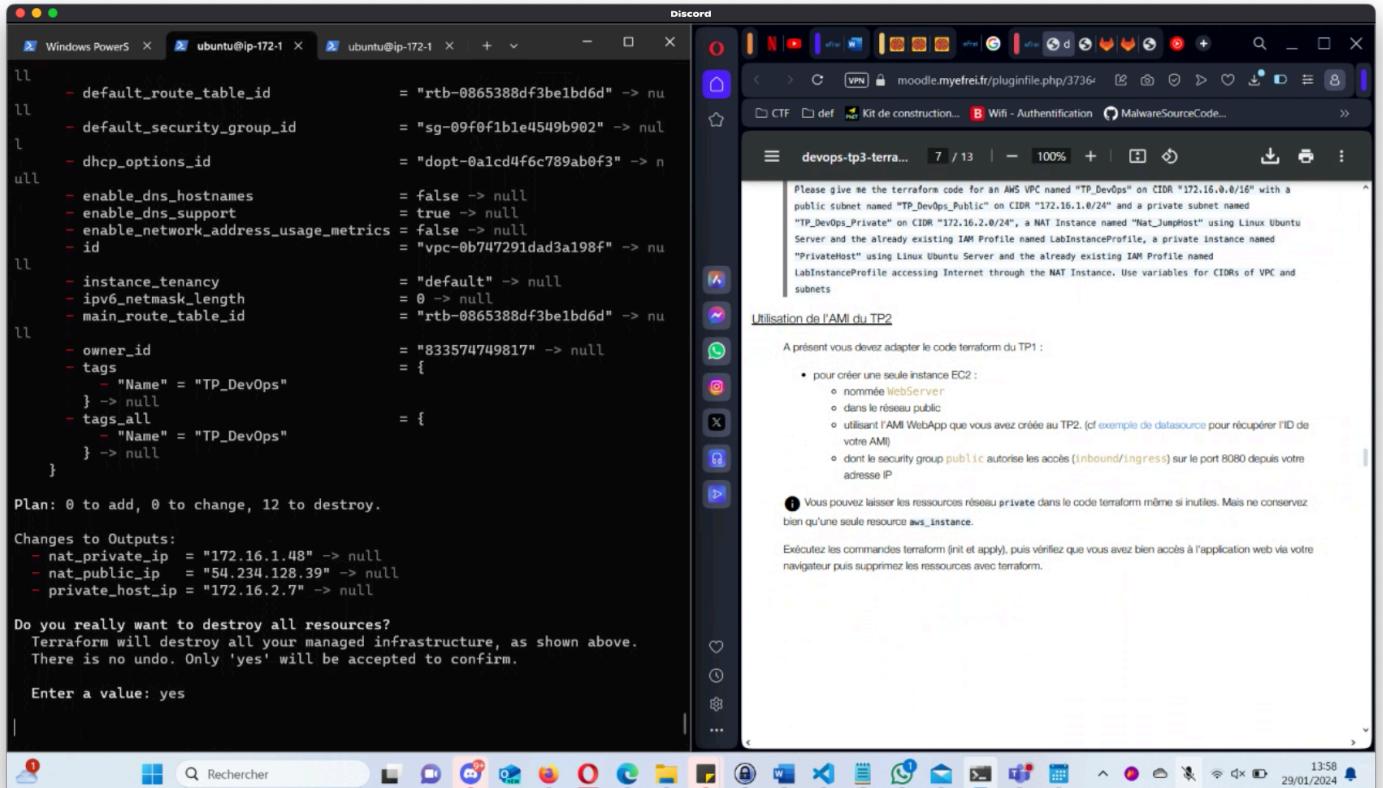
- A note: "Notez que les ressources vont avoir les mêmes noms "logiques" que certaines déjà existantes du TP1 (VPC, Routes, ...). Attention à ne pas les confondre dans la console."
- A message: "Vous pouvez alors reproduire les tests du TP1 :"
- A list of steps:
 - se connecter en SSH sur l'ip publique de l'instance NAT_JumpHost
 - sur NAT_JumpHost, copier la clé privée (format PEM, droits 600) et se connecter en SSH sur l'ip privée de l'instance PrivateHost
 - depuis PrivateHost, effectuez les tests ping 8.8.8.8 et curl <https://www.google.fr> pour vérifier les accès vers Internet à travers l'instance NAT
- A message: "A présent exécutez terraform destroy ou tfd dans le même dossier où vous avez lancé terraform apply, terraform vous redemandera votre IP et validation (tapez yes) puis il supprimera les ressources."
- A ChatGPT interface asking: "Please give me the terraform code for an AWS VPC named "TP_DevOps" on CIDR "172.16.0.0/16" with a public subnet named "TP_DevOps_Public" on CIDR "172.16.1.0/24" and a private subnet named "TP_DevOps_Private" on CIDR "172.16.2.0/24", a NAT Instance named "Nat_JumpHost" using Linux Ubuntu Server and the already existing IAM Profile named LabInstanceProfile, a private instance named "PrivateHost" using Linux Ubuntu Server and the already existing IAM Profile named LabInstanceProfile accessing Internet through the NAT Instance. Use variables for CIDRs of VPC and subnets"
- A message: "Utilisation de l'AMI du TP2"
- A message: "A présent vous devez adapter le code terraform du TP1 :"

Nous nous connectons à présent sur l'instance `PrivateHost`, en ayant modifié les permissions de notre clé privée nouvellement copiée.

En effectuant la commande `ping 8.8.8.8`, nous pouvons constater que `PrivateHost` a bien accès à Internet.



Nous pouvons maintenant faire un `terraform destroy`.



Utilisation de l'AMI du TP2

Nous allons modifier le code de `main.tf` pour pouvoir répondre aux exigences de cette partie.

Premièrement, nous supprimons l'instance `NAT`, ainsi que tous ses cas d'usages dans le code.

Puis, on adapte l'instance `PrivateHost` pour être dans le `public_subnet`, et récupérer l'AMI correspondant à celle que nous avions créée, tout en modifiant les `owners`.

```
#####
# Instances
#####
data "aws_ami" "webapp" {
  most_recent = true
  owners      = ["833574749817"]
  filter {
    name   = "image-id"
    values = ["ami-009bc99353224f1ec"]
  }
}

# Create a WebApp instance
resource "aws_instance" "tp_devops_webapp_instance" {
  ami           = data.aws_ami.webapp.id
  instance_type = var.instance_type
  key_name       = "monkey"
  iam_instance_profile = "LabInstanceProfile"
  subnet_id      = aws_subnet.tp_devops_public_subnet.id

  tags = {
    Name = "WebServer"
  }

  vpc_security_group_ids = [aws_security_group.public.id]
}
```

Enfin, nous avons modifié le `security group` pour permettre au port 80 d'être un port d'écoute.

```
resource "aws_security_group" "public" {
# [...]

  # allow HTTP access from 8080
  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = [for ip in local.your_ip_addresses : "${ip}/32"]
  }

# [...]
}
```

Nous exécutons les commandes `terraform init` et `terraform apply`.

```
+ nat_public_ip = (known after apply)

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_vpc.tp_devops_vpc: Creating...
aws_vpc.tp_devops_vpc: Creation complete after 1s [id=vpc-05db7e337f8200013]
aws_subnet.tp_devops_public_subnet: Creating...
aws_default_route_table.private_route_table: Creating...
aws_internet_gateway.tp_devops_igw: Creating...
aws_security_group.private: Creating...
aws_subnet.tp_devops_private_subnet: Creating...
aws_default_route_table.private_route_table: Creation complete after 1s [id=rtb-001ef92ac9bbe75de]
aws_subnet.tp_devops_private_subnet: Creation complete after 1s [id=subnet-05d319ac7f475c79b]
aws_internet_gateway.tp_devops_igw: Creation complete after 1s [id=igw-00fdca5957ec82b94]
aws_route_table.public_route_table: Creating...
aws_route_table.public_route_table: Creation complete after 1s [id=rtb-0426a61c2b4f88561]
aws_security_group.private: Creation complete after 2s [id=sg-088b47e34dc43696d]
aws_security_group.public: Creating...
aws_security_group.public: Creation complete after 2s [id=sg-00ddcbc0e85b280b8]
aws_subnet.tp_devops_public_subnet: Still creating... [10s elapsed]
aws_subnet.tp_devops_public_subnet: Creation complete after 11s [id=subnet-0864330494e36db2d]
aws_instance.tp_devops_webapp_instance: Creating...
aws_route_table_association.public: Creating...
aws_route_table_association.public: Creation complete after 0s [id=rbassoc-01eb71a96a885fc41]
aws_instance.tp_devops_webapp_instance: Still creating... [10s elapsed]
aws_instance.tp_devops_webapp_instance: Still creating... [20s elapsed]
aws_instance.tp_devops_webapp_instance: Still creating... [30s elapsed]
aws_instance.tp_devops_webapp_instance: Creation complete after 32s [id=i-03651f654fc330030]

Apply complete! Resources: 10 added, 0 changed, 0 destroyed.

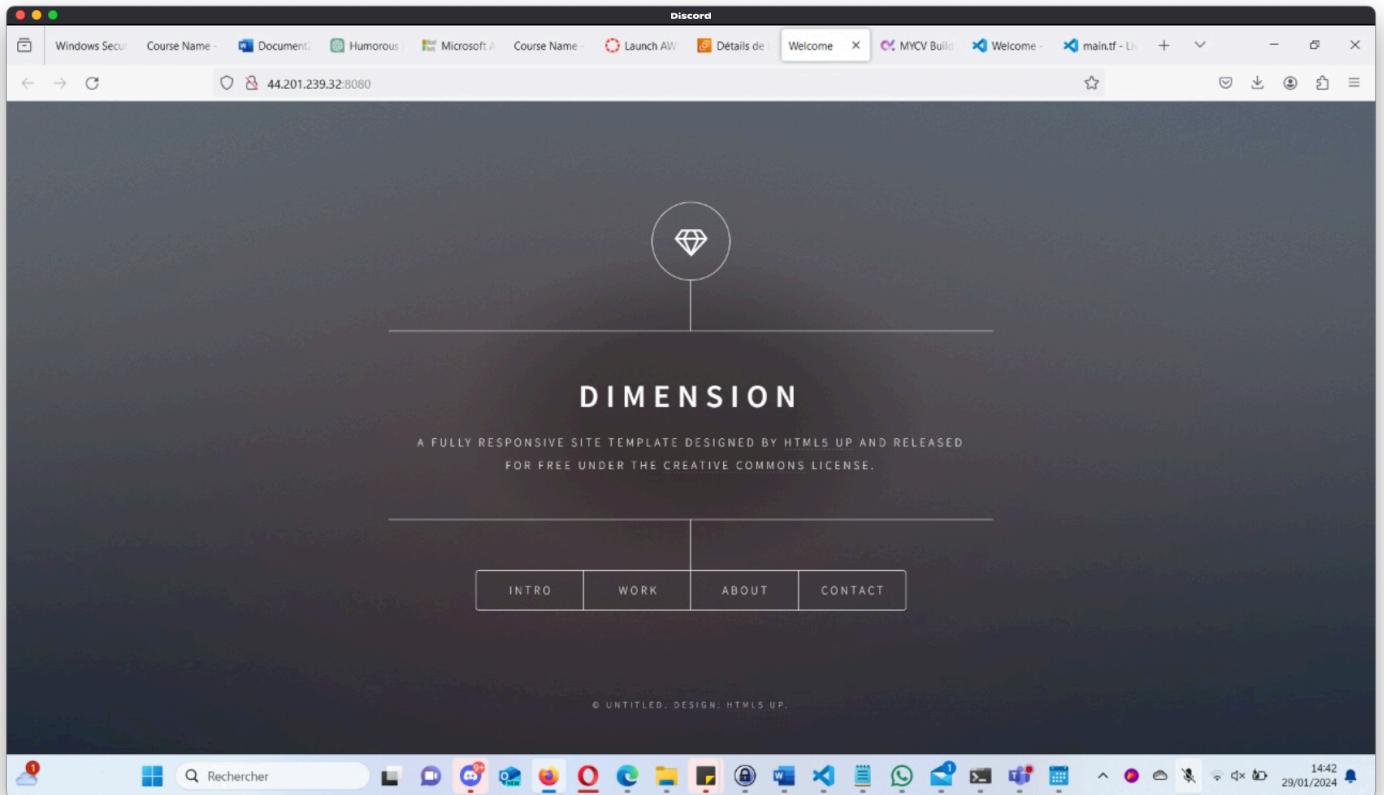
Outputs:

nat_private_ip = "172.16.1.170"
nat_public_ip = "44.201.239.32"
ubuntu@ip-172-16-1-4:~/tp$ |
```

Après avoir corrigé quelques erreurs, nous finissons par observer que le `apply` s'est effectué avec succès. Nous pouvons donc constater la création d'une instance `WebServer` sur l'interface AWS.

The screenshot shows the AWS CloudWatch Instances page. On the left, there's a sidebar with navigation links for EC2, Services, Images, and more. The main area displays a table titled "Instances (11) Informations". The table lists 11 instances, each with a checkbox, name, ID, state, type, monitoring status, alarm status, availability zone, public IP, and address. The "WebServer" instance is highlighted with a green circle and the text "En cours d'initialisation". The table has columns for Name, ID d'instance, État de l'instance, Type d'instance, Contrôle des stat., Statut d'alarme, Zone de disponibilité, DNS IPv4 public, and Adresse. The "Instances" section in the sidebar is expanded, showing sub-options like Types d'instances, Modèles de lancement, Demandes Spot, Savings Plans, Instances réservées, Hôtes dédiés, Réservations de capacité, and Nouveau. The bottom of the screen shows the Windows taskbar with various icons and the system tray.

Enfin, nous testons de nous connecter sur le port 8080 en HTTP sur l'IP publique de notre instance, puis constatons que nous voyons le site web s'afficher.



Questions

Terraform

Conceptuelle : Quels sont les avantages de l'utilisation de Terraform pour le déploiement d'infrastructures par rapport à des méthodes manuelles ou à d'autres outils d'automatisation ?

Terraform permet de définir les infrastructures de manière itérative, via l'utilisation de codes HashiCorp Configuration Language (HCL) ou JSON. Cet outil repose donc sur le principe de l'Infrastructure as Code ce qui améliore la reproductibilité, la transparence et la traçabilité des environnements. Avec Terraform, par rapport aux méthodes manuelles, les erreurs humaines sont plus facilement évitables et le déploiement est plus rapide. Cet outil supporte la majorité des fournisseurs de services cloud, facilitant la gestion des configurations multi-cloud et enfin, il propose des fonctionnalités de Monitoring as Code, facilitant le monitoring à grande échelle.

Application : Après avoir déployé une application web avec Terraform, comment vérifieriez-vous son bon fonctionnement ?

Pour s'assurer du bon fonctionnement d'une application web déployée avec Terraform, la première étape serait de vérifier la mise en place correcte des ressources à l'aide de la commande `terraform plan`. Ensuite, nous pourrions tester la connectivité et l'accessibilité du serveur avec des outils comme curl. Puis, nous pourrions réaliser des tests fonctionnels pour valider les principales fonctionnalités de l'application et analyser les logs pour détecter d'éventuelles erreurs. Enfin, exécuter des tests de charge permettrait de valider la capacité de l'application à supporter un trafic important.

Analyse : Si Terraform échoue lors de l'application d'un plan, quelles étapes suivriez-vous pour diagnostiquer et résoudre le problème ?

Lorsque Terraform échoue durant l'application d'un plan, plusieurs étapes peuvent être suivies pour diagnostiquer puis résoudre ce problème. La première étape est d'analyser les messages d'erreur affichés par Terraform, car ils fournissent souvent des indications sur la nature du problème. La deuxième étape est la vérification du fichier de configuration Terraform.

Ensuite, il est pertinent de consulter les logs. Finalement, il faudrait vérifier que toutes les dépendances et les versions des providers sont correctement configurées et à jour.

Si après toutes ces étapes, le problème est toujours présent, la réduction de la portée du plan Terraform peut aider.

La documentation est évidemment une source intéressante aussi. Étant un outil open source, Terraform dispose d'une grande communauté qui échange sur des forums où les mêmes problématiques ont potentiellement été résolues.

Critique : *Discutez des limites de Terraform en matière de gestion de l'infrastructure. Y a-t-il des scénarios où il ne serait pas l'outil idéal ?*

Nous pouvons citer plusieurs inconvénients de Terraform :

- Présence de bugs dans les nouvelles versions.
- Pas de fonction de retour en arrière automatique en cas de modification incorrecte des ressources.
- Fonctions de collaboration et de sécurité disponibles uniquement dans des solutions entreprise onéreuses.

Terraform n'est pas l'outil idéal si l'on souhaite réaliser de l'orchestration de conteneurs. Dans ce cas, il est préférable d'utiliser Kubernetes.

CloudFormation, la solution d'AWS pourrait aussi être une bonne alternative dans la situation où l'unique cloud provider utilisé est AWS.

Extension : *Comment intégreriez-vous Terraform avec d'autres outils de DevOps pour un pipeline de déploiement continu ?*

On pourrait par exemple imaginer de mettre en place un système de monitoring de l'interface avec des technologies comme Prometheus, Grafana et InfluxDB.

Sécurité : *Quelles sont les meilleures pratiques pour assurer la sécurité lors du déploiement d'infrastructures avec Terraform sur AWS ?*

Plusieurs bonnes pratiques sont importantes lors du déploiement avec Terraform :

- Utiliser des modules reconnus et maintenus pour éviter les vulnérabilités dans le code.
- Gérer et stocker les états de Terraform de manière sécurisée, avec l'utilisation d'un chiffrement.
- Appliquer le principe de moindre privilège pour les accès IAM.

Optimisation : *Comment optimiserez-vous les coûts lors du déploiement d'une infrastructure avec Terraform sur AWS ?*

Plusieurs actions sont possibles pour réduire les coûts liés au déploiement d'infrastructures via Terraform sur AWS.

- Analyser les processus qui consomment le plus de ressources, par exemple, les instances EC2, les

- stockages S3 et les bases de données RDS.
- Utiliser des solutions serverless comme Amazon Lambda.
- Utiliser les tags pour suivre et gérer avec plus de précisions les coûts. Utiliser les modules mis à disposition par la communauté Terraform.

Évolution : Comment Terraform peut-il être utilisé pour gérer les changements dans une infrastructure AWS existante ?

Plusieurs outils sont disponibles pour gérer les changements dans une infrastructure AWS existante avec Terraform (liste non exhaustive) : Terraform lui-même, AWS CLI et l'outil Terraforming. Il est possible d'importer intégralement ou partiellement l'infrastructure actuelle dans Terraform, si l'on souhaite la gérer partiellement ou intégralement avec Terraform.

Si l'infrastructure est complexe, utiliser l'automatisation facilite grandement l'import. A l'inverse, si l'infrastructure est "simple", il peut être pertinent de la récréer manuellement dans Terraform.

Cas d'usage : Proposez un scénario où l'utilisation de Terraform serait particulièrement bénéfique dans un contexte de production.

Dans un contexte de production Terraform est particulièrement utile pour déployer des applications sur plusieurs fournisseurs de cloud comme AWS, Azure, et autres, en utilisant un workflow unique. Cela aide à prévenir les interruptions de service dues à des pannes d'un fournisseur spécifique. En configurant l'infrastructure sur plusieurs clouds avec Terraform, les organisations simplifient la gestion et l'orchestration de grandes infrastructures multi-cloud.

L'utilisation d'un seul workflow réduit également les coûts et le temps, évitant le besoin de multiples outils de déploiement cloud.

Réflexion : Quelles sont les conséquences potentielles de la suppression manuelle de ressources AWS qui ont été déployées via Terraform ?

Supprimer manuellement des ressources AWS qui ont été créées via Terraform peut empêcher l'application, la planification ou la destruction d'autres ressources qui sont dépendantes de celles qui ont été supprimées.

Il est possible de réparer cette erreur en utilisant la commande `terraform state rm`. Ainsi, à la prochaine exécution de `terraform apply`, la ressource sera considérée comme non existante et donc sera de nouveau créée.

Autoscaling

Compréhension : Comment l'autoscaling fonctionne-t-il avec AWS et quel est son rôle dans la gestion des ressources ?

Sur AWS l'autoscalling fonctionne via les EC2 et le service EC2 Auto Scaling, une fois les machines EC2 configurés on va créer un groupe d'Auto Scaling et le paramétrier avec les options de notre choix pour le lancement de l'auto scaling et définir les zones de disponibilités qui seront affectés.

Configuration : Quelles sont les étapes pour configurer une politique d'autoscaling avec Terraform sur AWS ?

La première étape est de créer un environnement Terraform, une fois celui-ci créé, on va ensuite définir un groupe de dimensionnement qui va nous permettre de définir le nombre d'instances que l'on va vouloir créer, l'interface réseau.

On va ensuite mettre en place les règles de mise à l'échelle quand on va avoir besoin d'allouer plus de ressources et selon quel paramètre (RAM, CPU, etc.). Il faudra ensuite appliquer la configuration Terraform et voir le résultat.

Scénarios : *Dans quels scénarios l'utilisation de l'autoscaling serait-elle particulièrement utile ? Donnez des exemples concrets.*

L'auto Scaling peut être pratique sur un site web avec un trafic variable, si une pizzeria met en place un site pour commander des pizzas, il y aura beaucoup plus de trafic le midi et le soir que le reste de la journée. Il peut être pertinent d'avoir une infrastructure variable pour toujours répondre à la demande tout en limitant les coûts.

Dépannage : *Si une instance ne se lance pas comme prévu dans un groupe d'autoscaling configuré via Terraform, quelles seraient vos premières étapes de dépannage ?*

La première étape est d'observer les logs Terraform pour voir s'il y a des erreurs détectées par celui-ci. S'il n'y a rien du côté de Terraform, on peut, via CloudWatch, regarder si une instance a eu des soucis. On peut ensuite faire de multiples choses, par exemple vérifier si les règles de mise à l'échelle sont correctes. Regarder la santé des instances ou bien encore que l'ont atteint des limites au niveau des instances créées dans AWS.

Optimisation : *Comment optimiserez-vous les coûts tout en garantissant la performance avec l'autoscaling sur AWS ?*

Pour optimiser les coûts sans renier sur le service, on peut par exemple définir des groupes de dimensionnement cible. Ces groupes vont définir une cible à atteindre au niveau de certaines métriques et faire varier l'infrastructure pour atteindre cette cible.

Par exemple, si on veut que la charge CPU reste à 80% et bien, on rajoutera autant d'EC2 requis pour rester à cette charge. Et si la charge baisse, on va supprimer des EC2. Ainsi, on maîtrise le coût.

Sécurité : *Quels sont les enjeux de sécurité à considérer lors de la mise en place de l'autoscaling avec Terraform sur AWS ?*

Les enjeux de sécurité sont dans un premier temps de vérifier que les ressources créées par Terraform soient conformes à la sécurité de l'environnement. On peut ensuite vérifier comment sont gérées les clés et si elles sont stockées de manière sûre.

Évaluation : *Comment évalueriez-vous l'efficacité d'une stratégie d'autoscaling mise en place via Terraform sur AWS ?*

Dans un premier temps, pour évaluer l'efficacité, on doit se fixer des objectifs de performance clairs pour savoir ce que l'on va mesurer. Ensuite, on doit mesurer la performance de base du système sans cette stratégie. Pour finir, on met en place notre stratégie et on observe la différence de performance.

Mise à jour : *Quelles sont les meilleures pratiques pour mettre à jour les AMI dans un groupe d'autoscaling sans interrompre le service ?*

L'utilisation de AWS Systems Manager Automation est recommandée. Ce service permet de mettre à jour les AMI dans un groupe d'autoscaling, via le *runbook PatchAMIAndUpdateASG*.

Ainsi, il est possible d'intégrer des actions d'automation, comme la mise à jour des AMI lors des maintenances et le déploiement progressif des mises à jour.

*AMI (Amazon Images) *

Optimisation : *Comment construire une AMI pour optimiser les performances et la réactivité dans un environnement d'autoscaling ?*

Pour créer une AMI améliorant performances et réactivité dans un contexte d'autoscaling, voici les points importants :

- Choisir judicieusement l'image de base, car le nombre de composants logiciels doit être minimisé, en sélectionnant ceux qui sont strictement nécessaires.
- Configurer la mise en cache et les services correctement.
- Vérifier que l'image est compatible avec divers types d'instances. Cela garantit une flexibilité maximale lors des changements d'échelle.

Sécurité : *Quelles mesures de sécurité spécifiques doivent être prises lors de l'utilisation d'AMI personnalisées dans un groupe d'autoscaling ?*

Lors de l'usage d'AMI personnalisées en autoscaling, diverses mesures de sécurité sont importantes.

- Chaque AMI doit être analysée pour détecter des malwares ou des vulnérabilités.
- Les patchs de sécurité doivent être appliqués régulièrement.
- Les accès inutiles ou donnant des priviléges inutiles doivent être désactivés.
- La transmission des données doit être sécurisée, notamment lors de l'initialisation des instances.
- Un processus de révision continue doit être mis en place.

Dépannage : *Comment identifier et résoudre les problèmes liés à l'utilisation d'une AMI spécifique dans un groupe d'autoscaling ?*

Voici l'approche que nous adopterions pour identifier et corriger les problèmes liés à une AMI spécifique en autoscaling :

- Commencer par examiner les journaux de démarrage des instances.
- Surveiller les métriques de performance.
- Utiliser des outils de diagnostic AWS pour déceler les anomalies.
- Assurer que les configurations de l'AMI sont bien adaptées au type d'instance.
- Parfois, il est nécessaire de reconstruire l'AMI avec des mises à jour ou des ajustements pour résoudre le problème.

Cloud Watch (Contexte Autoscaling)

Fonctionnement : *Comment les alarmes CloudWatch interagissent-elles avec les groupes d'autoscaling dans AWS ?*

Dans AWS, les alarmes CloudWatch surveillent des métriques spécifiques. Si une métrique franchit un seuil défini, l'alarme se déclenche. Cette action peut signaler à un groupe d'autoscaling d'ajuster sa capacité.

Par exemple, si la charge CPU dépasse 70%, l'alarme peut ordonner l'ajout d'instances. Inversement, une baisse significative déclenche la réduction du nombre d'instances.

Configuration : Quelles étapes suivez-vous pour configurer une alarme CloudWatch qui déclenche l'autoscaling sur AWS avec Terraform ?

Pour configurer une alarme CloudWatch déclenchant l'autoscaling avec Terraform, il faut:

- Définir un "aws_cloudwatch_metric_alarm" dans le code Terraform.
- Spécifier les métriques à surveiller, comme l'utilisation CPU.
- Définir des seuils pour l'activation de l'alarme.
- Associer cette alarme à un groupe d'autoscaling en utilisant l'argument "alarm_actions".
- Vérifier que Terraform inclut également le groupe d'autoscaling ciblé.

Sensibilité : Comment déterminer les seuils appropriés pour les alarmes CloudWatch dans un contexte d'autoscaling ?

On fixe des seuils pour les alarmes CloudWatch en analysant les tendances de l'application. Il faut consulter les pics d'utilisation et les minimums de performance passés. Il ajuste les seuils pour anticiper les besoins, non pour réagir après coup. Une bonne pratique consiste à tester différents seuils dans des environnements de staging avant de les appliquer en production.

Analyse : Quels types de métriques CloudWatch sont les plus utiles pour gérer efficacement l'autoscaling ?

Pour gérer l'autoscaling, les métriques utiles dans CloudWatch incluent :

- l'utilisation du CPU "CPUUtilization" ,
- le trafic entrant et sortant "NetworkIn/NetworkOut"
- "StatusCheckFailed" qui signale les problèmes au niveau de l'instance ou du système.
- "RequestCount" nombre de requêtes
- "latency" pour la latence

Il est nécessaire d'aller dans AWS et d'analyser ces métriques pour comprendre le comportement de l'application sous charge. Cela aide à configurer l'autoscaling de manière plus précise.

Optimisation : Comment l'utilisation des alarmes CloudWatch peut-elle améliorer l'efficacité de l'autoscaling en termes de coûts et de performance ?

Les alarmes CloudWatch, utilisées judicieusement, réduisent les coûts en évitant le surdimensionnement. Elles améliorent la performance en assurant une montée en charge rapide lors des pics de demande.

Le but étant de définir des seuils minimaux et maximaux afin d'encadrer une utilisation inutile ou trop importante de ressources. Cela maximise l'efficacité sans compromettre les performances.

*Load balancer (Autoscaling)**

Load balancer : Pouvez-vous expliquer les différences clés entre un Classic Load Balancer, un Network Load Balancer et un Application Load Balancer sur AWS ?

Load balancer : Un Classic Load Balancer, distribue le trafic de façon équilibrée entre les instances EC2. Conçu pour des applications simples. Il détecte les instances en mauvaise santé (performances) et route le trafic vers celles en bonne santé.

Un Network Load Balancer prend ses décisions au niveau de la couche transport, sa qualité première est sa rapidité et, il est capable de gérer des millions de requêtes par seconde.

L'Application Load Balancer, route le trafic au niveau de la couche applicative, des requêtes HTTP et HTTPS, offrant des fonctions avancées telles que le routage basé sur le contenu.

Rôle fondamental : *Quel est le rôle principal d'un load balancer dans un environnement avec autoscaling sur AWS ?*

Dans un milieu d'autoscaling AWS, un load balancer joue un rôle de chef d'orchestre. Il répartit le trafic entrant entre diverses instances EC2, assurant ainsi une utilisation efficace des ressources et une haute disponibilité. Ce processus favorise une expérience utilisateur fluide, même en cas de variations soudaines du trafic ou de pannes d'instance.

Distribution de trafic : *Comment un load balancer gère-t-il la distribution du trafic entrant dans un groupe d'autoscaling ?*

Un Load Balancer gère la distribution du trafic entrant dans un groupe d'autoscaling en répartissant équitablement la charge entre les instances.

Pour ce faire, il vérifie en permanence les instances qui peuvent recevoir du trafic en fonction de leurs états. Ensuite, lorsqu'une requête arrive, un algorithme de répartition va distribuer équitablement le trafic entre les instances disponibles.

Scalabilité : *Comment le load balancing contribue-t-il à la scalabilité d'une application déployée sur AWS avec l'autoscaling ?*

Le load balancing permet une scalabilité horizontale et verticale via une répartition équilibrée des charges, comme nous l'avons vu dans les questions précédentes, il y a différents types de LB (Classic, Network, Application) qui vont répartir la charge selon des critères différents.

Le load balancer joue aussi un rôle dans la résilience d'une application. La résilience rejoint le concept de HA (question suivante).

La répartition de charges et la résilience sont deux caractéristiques clés qui vont prendre en importance lorsque la demande d'une application augmente. Ainsi le LB, via l'autoscaling joue un rôle crucial dans la scalabilité d'une application.

Haute disponibilité : *En quoi un load balancer est-il essentiel pour assurer la haute disponibilité dans un système avec l'autoscaling ?*

Le load balancing va permettre de répartir la charge sur les différentes machines qui vont supporter la charge. Si jamais nous avons un autoscalling efficace, mais que la charge de donnée n'est pas répartie correctement par le load balancer, alors l'autoscalling sera inutile.