

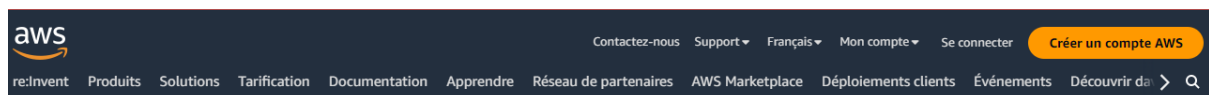
Élèves:

Akobé Kobon
Brou Justin
Ismael Sylla
Siaka Traoré

Dans le cadre de l'évaluation de réalisation technique, nous avons entrepris la mise en place d'un outil lié aux pratiques DevOps sur des projets sélectionnés. Notre objectif était d'évaluer la réalisation technique de cette mise en place en utilisant deux approches distinctes : l'utilisation de Terraform et d'Ansible. Ce rapport présente en détail nos démarches ainsi que les étapes suivies pour intégrer ces outils dans notre réalisation. Nous discuterons également des défis rencontrés et des solutions trouvées tout au long du processus.

Dans les deux projets, une première tâche commune a consisté à créer un compte sur AWS afin de disposer d'un environnement cloud pour nos déploiements. Cette étape préliminaire a été essentielle pour permettre l'utilisation de Terraform et d'Ansible dans nos réalisations.

AWS fournit une infrastructure cloud évolutive et rentable qui permet aux entreprises de déployer leurs applications et leurs services en ligne. Cela permet aux entreprises de se concentrer sur leur activité principale sans se soucier de la gestion de l'infrastructure physique sous-jacente.



Pour créer un compte sur AWS, il suffit tout simplement de se rendre sur le site: <https://aws.amazon.com/> puis cliquer sur le bouton “Créer un compte AWS” comme indiqué sur l'image ci-dessus et de suivre les instructions.

I) Projet 1: Utilisation de Terraform dans la construction de l'infrastructure et le déploiement d'un projet sur AWS

Terraform est un outil logiciel d'infrastructure en tant que code qui permet aux équipes DevOps d'automatiser le provisionnement de l'infrastructure à l'aide de fichiers de configuration déclaratifs, qui décrivent donc l'état final de l'infrastructure.

1. Installation de Terraform :

Nous avons choisi de travailler sur linux.

Pour installer TerraForm sur linux, il faut vous rendre sur le site officiel de HashiCorp:

<https://developer.hashicorp.com/terraform/install>

Puis choisir votre distribution linux (dans notre cas ubuntu) et saisir les 3 commandes de l'image ci-dessous:

The screenshot shows the 'Linux' section of the Terraform installation guide. It features a 'Directeur chargé d'emballage' (Package manager director) with buttons for 'Ubuntu/Debian', 'CentOS/RHEL', 'Feutre', 'Amazon Linux', and 'Homebrew'. The 'Ubuntu/Debian' button is selected. Below this, a terminal window displays the following commands:

```
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/hashicorp-archive-keyring.gpg
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.hashicorp.com/ ubuntu main" | sudo tee /etc/apt/sources.list.d/hashicorp.list
sudo apt update && sudo apt install terraform
```

Below the terminal window, the 'Téléchargement binaire' (Binary download) section is visible, showing four download options for version 1.8.2:

Architecture	Version	Action
386	Version : 1.8.2	Télécharger
AMD64	Version : 1.8.2	Télécharger
BRAS	Version : 1.8.2	Télécharger
ARM64	Version : 1.8.2	Télécharger

Pour vérifier que terraform est bien installé, vous pouvez saisir terraform et voir l'affichage suivant des différentes commandes disponibles sur terraform :

```
server@server:~$ terraform
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
  init      Prepare your working directory for other commands
  validate  Check whether the configuration is valid
  plan      Show changes required by the current configuration
  apply     Create or update infrastructure
  destroy   Destroy previously-created infrastructure

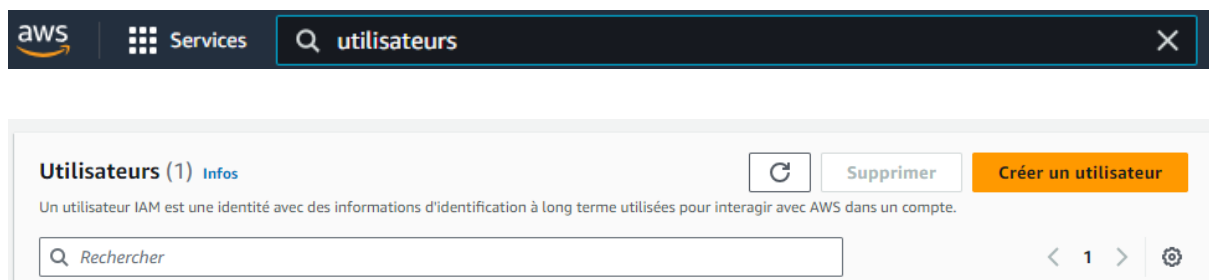
All other commands:
  console   Try Terraform expressions at an interactive command prompt
  fmt       Reformat your configuration in the standard style
```

2. Projet :

2.1) Création d'un IAM

Dans la réalisation technique avec Terraform, nous avons créé un IAM (Identity and Access Management) pour gérer les identités et les permissions au sein de notre environnement cloud sur AWS. En d'autres termes, cette IAM nous permet de contrôler qui est autorisé à accéder à nos instances EC2.

Pour créer un utilisateur sur AWS, il suffit de saisir "utilisateur" dans la barre de recherche et de cliquer sur le bouton "Créer un utilisateur"



2.2) Récupération des clés d'accès d'IAM

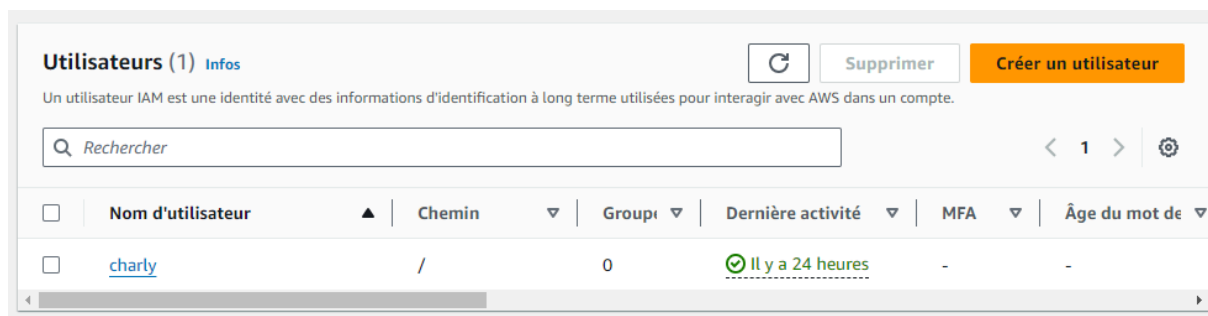
Après avoir créer l'utilisateur IAM, nous avons récupéré les clés d'accès de notre utilisateur IAM et nous les avons stocker dans un fichier ~/.aws/credentials comme suit

[default]

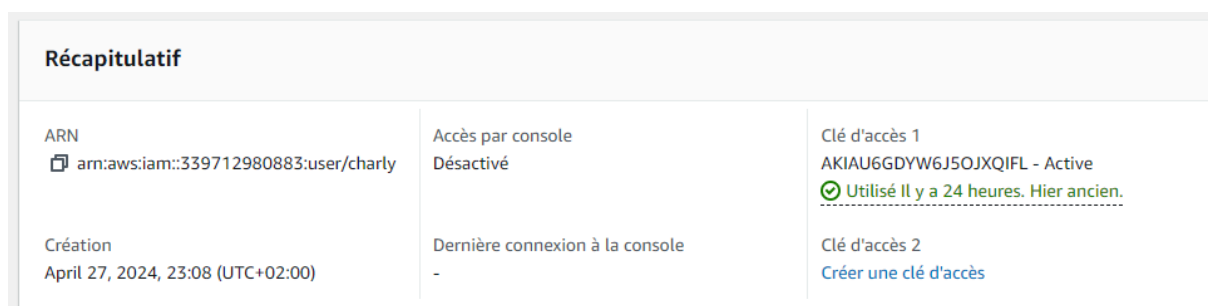
aws_access_key_id = VOTRE_ACCESS_KEY_ID

aws_secret_access_key = VOTRE_SECRET_ACCESS_KEY

Pour récupérer les accès d'un utilisateur IAM, il faut se rendre dans la liste des utilisateurs en saisissant "utilisateur" dans la barre de recherche puis de cliquer sur le nom de l'utilisateur (par exemple 'charly' dans cet exemple):



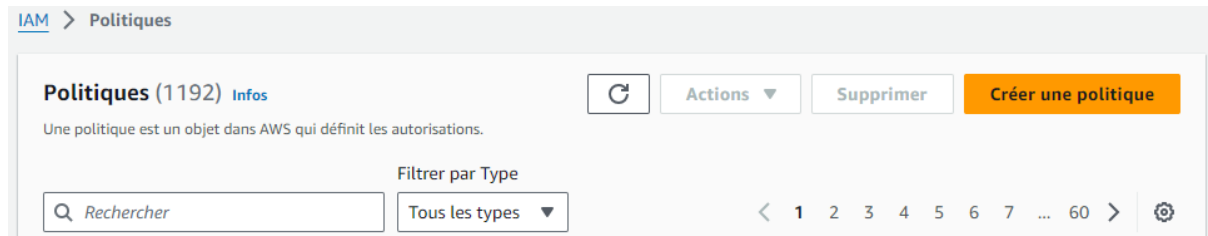
Ensuite, il suffit de copier la clé d'accès (Clé d'accès 1) et la clé d'accès secrète (Clé d'accès 2):



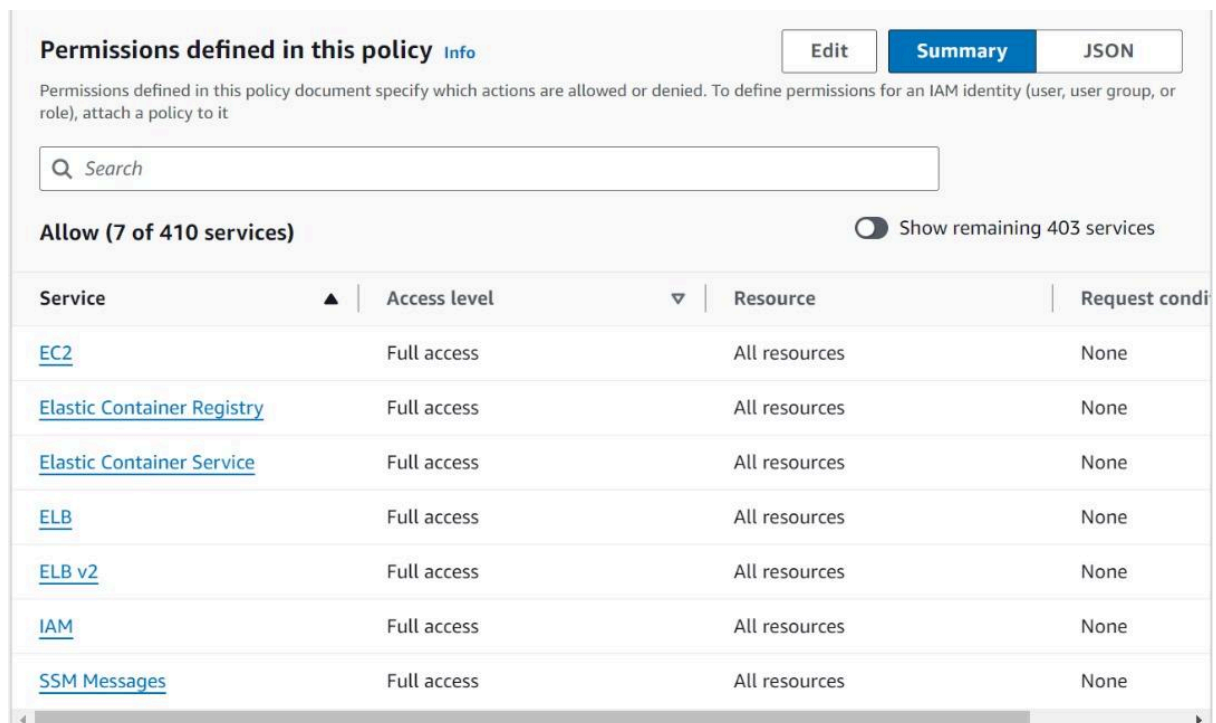
2.3) Politiques

Pour que notre utilisateur puisse créer une instance, il a fallu lui donner des autorisations appelées politiques sur AWS.

Pour accorder des autorisations à un utilisateur IAM, il suffit de saisir “Politique” dans la barre de recherche puis de sélectionner les politiques nécessaires en cliquant sur le bouton “Créer une politique”:



Les politiques que nous avons appliqué sur notre utilisateur:



Une fois que les permissions ont été accordées à notre utilisateur, nous pouvons commencer les configurations avec terraform sans risque lié aux autorisations.

NOTRE INFRASTRUCTURE

Nous avons décidé de mettre en place deux serveurs (deux ec2) sur AWS sur lesquels notre projet serait déployer, ainsi qu'un equilibreur de charge qui nous permet de rediriger le trafic vers nos service en fonction de l'affluence de

la demande a notre service afin de ne pas faire effondrer notre service (site web).

En plus de cela nous avons autorisé le trafic entrant sur le port 80 et la connexion SSH.

Pour le déploiement de notre architecture, notre outil terraform s'avère très efficace vu qu'il nous permet de décrire l'ensemble de notre architecture sur AWS.

Voici le contenu de notre fichier de configuration ainsi que la description.

```
provider "aws" {  
  region = "eu-north-1"  
  access_key = "AKIA5FTZCTGVQ76G7XPD"  
  secret_key = "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"  
}
```

Cette première partie nous permet d'accéder à AWS avec nos clés d'accès que nous avons récupérées sur AWS.

Pour récupérer nos clés, nous entrons dans la barre de recherche '**IAM**' ensuite on sélectionne notre utilisateur et nous cliquons sur '**Créer une clé**' une fois que c'est fait nous aurons accès à notre clé privée et notre clé publique et nous les remplaçons dans le champ provider AWS.

Ensuite, nous passons à la création de nos EC2

```
resource "aws_instance" "example" {  
  ami           = "ami-0705384c0b33c194c"  
  instance_type = "t3.micro"  
  key_name      = "my_key_pair"  
  
  tags = {  
    Name = "MonInstance"  
  }  
  
  connection {  
    type        = "ssh"  
    user        = "ubuntu"  
    private_key = file("/home/ismael/Téléchargements/my_key_pair.pem")  
    host        = self.public_ip  
  }  
}
```

Nous créons notre Ec2 en utilisant l'ami d'une machine AWS et en utilisant une instance de type t3.micro. La paire de clé est créée sur AWS et on

s'assure de donner un nom et ce dernier est celui utilisé pour créer notre machine, on lui donne le nom MonInstance. la balise qui suit est une configuration pour se connecter à la machine et effectuer de commandes au démarrage par exemple pour installer des packages comme docker afin de télécharger des containers de notre projet

```
provisioner "remote-exec" {
  inline = [
    "sudo apt-get update -y",
    "sudo apt-get install -y docker.io",
    "sudo systemctl start docker",
    "sudo usermod -a -G docker ubuntu" # Ajoutez l'utilisateur à docker group
    "sudo docker run -d --name=mysql_container -p 3306:3306 -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=tlc -e MYSQL_USER=tlc -e MYSQL_PASSWORD=tlc mysql",
    "sudo docker run -d --name=etherpad_container -p 9001:9001 -e etherpad/etherpad",
    "sudo docker run -d --name=smtp_container -p 2525:25 bytemark/smtp"
  ]
}
```

Dans cette partie, nous exécutons des commandes pour l'installation de certain packages et pour télécharger les images de containers dont nous avons besoin pour exécuter le projet comme l'installation de docker pour la conteneurisation de notre projet

Ensuite, nous lançons nos différents containers.

Ce processus est répété sur notre seconde machine. Pour ce qui est de la configuration réseau autorisant la trafic sur notre machine, nous l'avons défini comme suit:

```
resource "aws_security_group" "example" {
  name           = "example-sg"
  description    = "Allow HTTP and SSH inbound traffic"
  vpc_id        = "vpc-0e6f0d0f29ce293a9"

  ingress {
    from_port = 80
    to_port   = 80
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"] # Cette règle autorise tout le trafic entrant sur le port 80 (HTTP)
  }

  ingress {
    from_port = 22
    to_port   = 22
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"] # Cette règle autorise tout le trafic entrant sur le port 22 (SSH)
  }

  egress {
    from_port = 0
    to_port   = 0
    protocol  = "-1"
    cidr_blocks = ["0.0.0.0/0"] # Cette règle autorise tout le trafic sortant
  }
}
```


On prend soin de bien récupérer l'adresse de nos subnet sur AWS ainsi que de notre VPC.

Une fois cette étape terminée, nous passons à notre load balancer que nous avons défini en spécifiant que les cibles sont nos deux machines précédemment créées en récupérant l'adresse de nos subnet sur AWS. Voici comment nous avons défini cette partie:

```
resource "aws_alb_target_group_attachment" "example" {
  target_group_arn = aws_alb_target_group.example.arn
  target_id       = aws_instance.example.id
}

resource "aws_alb_target_group_attachment" "example" {}
resource "aws_alb_target_group_attachment" "example1" {}

resource "aws_alb_listener" "example" {
  load_balancer_arn = aws_alb.example.arn
  port              = "80"
  protocol          = "HTTP"

  default_action {
    type = "forward"
    target_group_arn = aws_alb_target_group.example.arn
  }
}
```

Cette partie du code nous permet de cibler nos machines pour notre équilibreur de charge.

Après l'exécution de notre code terraform, nous avons exécuté trois commandes qui sont :

- terraform init

```
ismael@Mugiwara:~/Bureau/TP_Docker/doodle/terraform$ terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Using previously-installed hashicorp/aws v5.47.0

Terraform has been successfully initialized!

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.

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.
ismael@Mugiwara:~/Bureau/TP_Docker/doodle/terraform$
```

- terraform plan

```
+ {
+   + cidr_blocks      = [
+     + "0.0.0.0/0",
+   ]
+   + from_port        = 80
+   + ipv6_cidr_blocks = []
+   + prefix_list_ids  = []
+   + protocol         = "tcp"
+   + security_groups  = []
+   + self             = false
+   + to_port          = 80
+   # (1 unchanged attribute hidden)
+ },
+ name                = "example-sg"
+ name_prefix         = (known after apply)
+ owner_id            = (known after apply)
+ revoke_rules_on_delete = false
+ tags_all            = (known after apply)
+ vpc_id              = "vpc-0e6f0d0f29ce293a9"
}

Plan: 8 to add, 0 to change, 0 to destroy.

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.
ismael@Mugiwara:~/Bureau/TP_Docker/doodle/terraform$
```

- terraform apply

```
PROBLEMS 26 OUTPUT DEBUG CONSOLE TERMINAL PORTS
group/example-target-group/febd018db7f700cd-20240429221047108200000004]
aws_alb.example: Still creating... [20s elapsed]
aws_alb.example: Still creating... [30s elapsed]
aws_alb.example: Still creating... [40s elapsed]
aws_alb.example: Still creating... [50s elapsed]
aws_alb.example: Still creating... [1m0s elapsed]
aws_alb.example: Still creating... [1m10s elapsed]
aws_alb.example: Still creating... [1m20s elapsed]
aws_alb.example: Still creating... [1m30s elapsed]
aws_alb.example: Still creating... [1m40s elapsed]
aws_alb.example: Still creating... [1m50s elapsed]
aws_alb.example: Still creating... [2m0s elapsed]
aws_alb.example: Still creating... [2m10s elapsed]
aws_alb.example: Still creating... [2m20s elapsed]
aws_alb.example: Still creating... [2m30s elapsed]
aws_alb.example: Still creating... [2m40s elapsed]
aws_alb.example: Still creating... [2m50s elapsed]
aws_alb.example: Still creating... [3m0s elapsed]
aws_alb.example: Creation complete after 3m2s [id=arn:aws:elasticloadbalancing:eu-north-1:905418283435:loadbalancer/app/6d756f96f942a469]
aws_alb_listener.example: Creating...
aws_alb_listener.example: Creation complete after 1s [id=arn:aws:elasticloadbalancing:eu-north-1:905418283435:listener/app/6d756f96f942a469/9bffa6da9762a39ac]
Apply complete! Resources: 8 added, 0 changed, 0 destroyed.
ismael@Mugiwara:~/Bureau/TP_Docker/doodle/terraform$
```

Le résultat de notre config est visible sur AWS

Instances (1/3) Informations							
<div>Rechercher Instance par attribut ou identification (case-sensitive)</div> <div>En cours d'exécution</div>							
	Name	ID d'instance	État de l'instance	Type d'instance	Contrôle des statuts	Statut d'alarme	Zone de disponibilité
<input type="checkbox"/>	MonInstance	i-0f1960a25be52ab7a	En cours d'exécution	t3.micro	2/2 vérifications réussies	Afficher les alarmes	eu-north-1
<input type="checkbox"/>	MonInstance2	i-053a768be2575fd22	En cours d'exécution	t3.micro	2/2 vérifications réussies	Afficher les alarmes	eu-north-1

Voici notre equilibreur de charge

EC2 > Équilibreurs de charge > example-alb

example-alb

Actions

Détails

Type d'équilibreurs de charge

Application

Statut

Actif

VPC

vpc-0e6f0d0f29ce293a9

Type d'adresse IP

IPv4

Méthode

Internet-facing

Zone hébergée

Z23TAZ6LKFMNIO

Zones de disponibilité

subnet-071e79594b3df367
eu-north-1b (eun1-az2)
subnet-0da98b2c2a06177a
eu-north-1a (eun1-az1)

Date de création

30 avril 2024, 00:10 (UTC+02:00)

ARN de l'équilibreur de charge

arn:aws:elasticloadbalancing:eu-north-1:905418283435:loadbalancer/app/example-alb/6d756f96f942a469

Nom du DNS

example-alb-1151822969.eu-north-1.elb.amazonaws.com (Enregistrement A)

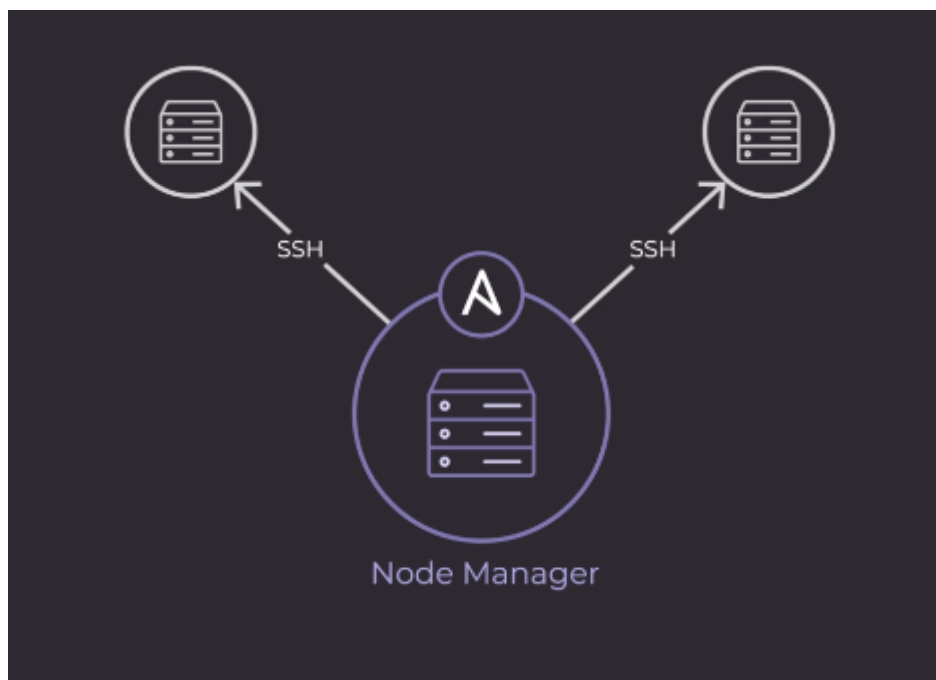
II) Projet 2: Utilisation de Ansible dans le déploiement d'une application web sur AWS

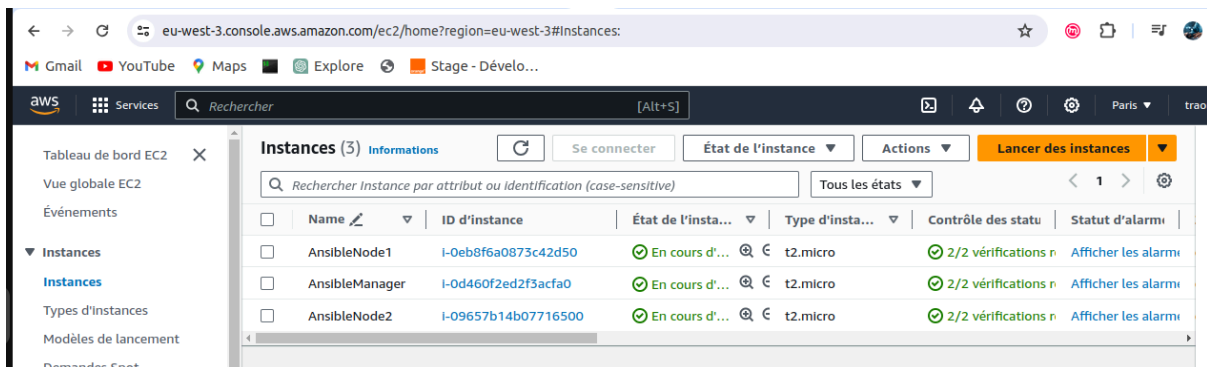
Ici; l'objectif principal du projet est de déployer une application web sur des instances EC2 de manière automatisée à l'aide d'Ansible. Les sous-objectifs du projet comprennent :

- Provisionner et configurer les instances EC2 sur AWS.
- Installer et configurer les dépendances logicielles nécessaires sur les instances.
- Déployer l'application web sur les instances EC2.

1. Création des instances et mise en place de la connexion ssh

Nous avons créé 3 instances à savoir : Ansiblemanager, AnsibleNode1 et AnsibleNode2.





Un node (ou *managed node*, ou *host*) est un poste connecté au node manager en SSH, et sur lequel Ansible viendra pousser les tâches d'automatisation. Ansible n'est pas installé sur les nodes.

Un node manager, ou *control node*, est un poste qui contrôle les nodes grâce à sa connexion SSH. Il dispose d'une version Ansible d'installé pour leur pousser les tâches d'automatisation grâce aux commandes ansible et ansible-playbook.

Pour faciliter la connexion ssh entre les différentes instances, nous allons créer des clés SSH avec la commande `ssh-keygen` sur chaque instance. Ensuite, on copie la clé publique du AnsibleManager et colle dans le fichier des clés d'autorisation des 2 Nodes.

Ce qui nous permet de se connecter par ssh depuis le Nodemanager en utilisant `ssh @ip` de la machine.

```
[root@ip-172-31-45-178 ec2-user]# cd .ssh/
[root@ip-172-31-45-178 .ssh]# ls
authorized_keys  id_rsa  id_rsa.pub  known_hosts  known_hosts.old
[root@ip-172-31-45-178 .ssh]# cat known_hosts
172.31.40.114 ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAINhqn0TvEeh5gkZEFm4tN4CLzyyyqIeDDmzbihVtjFt
172.31.40.114 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBHmY/Q8JRVixPK1h6v/Tp+61PEUJ15DCFLybvq3fDd20hUf41M
TFY8Xs2sF1lGhsdBKNECbZfM0Jo08GnoEtg=
172.31.35.202 ssh-ed25519 AAAAC3NzaC1lZDI1NTE5AAAAAILi6yCit1CFFjq+kxFLmH452TrU39cgfg2RwXq7g6m0c
172.31.35.202 ecdsa-sha2-nistp256 AAAAE2VjZHNhLXNoYTItbmlzdHAyNTYAAAAIbmlzdHAyNTYAAABBBKaJaClvbHacYg98AynYmIS12hZ+5ZC7c0TuzcRQbDjx19pMA4M
n1Fm2wXFD9HxfSqMum/eDwk5q12hE45ZMzkE=
[root@ip-172-31-45-178 .ssh]# nano authorized_keys
```

2. Installation de Ansible sur le NodeManager

On va maintenant installer sur le NodeManager (sur linux) comme suit:

`sudo apt-get install python3 python3-pip` (pour les dépendances)

`sudo pip3 install ansible`

Enfin, on peut vérifier que Ansible est correctement installé grâce à la commande :

`ansible --version`

```

bash: $ ansible: command not found
[root@ip-172-31-45-178 src]# ansible --version
ansible [core 2.15.3]
  config file = None
  configured module search path = ['/root/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.9/site-packages/ansible
  ansible collection location = /root/.ansible/ansible_collections:/usr/share/ansible/ansible_collections
  executable location = /usr/bin/ansible
  python version = 3.9.16 (main, Sep  8 2023, 00:00:00) [GCC 11.4.1 20230605 (Red Hat 11.4.1-2)] (/usr/bin/python3.9)
  jinja version = 3.1.2
  libyaml = True
[root@ip-172-31-45-178 src]#

```

Quelques commandes accessibles après l'installation de Ansible:

- **ansible** : cette commande permet de lancer des actions Ansible en mode ad-hoc (en ligne de commande) ;
- **ansible-config** : cette commande permet de manager la configuration de Ansible :
 - si vous lancez la commande **\$ ansible-config list**, vous allez lister la configuration de Ansible. Toutes ces variables sont contenues dans **./lib/pythonX.Y/site-packages/ansible/constants.py** ;
- **ansible-doc** : cette commande permet d'obtenir de l'aide pour utiliser Ansible; la documentation est très bien faite.

3. Configuration des fichiers

☐ *Inventaire des nodes*

Pour faire nous avons créer un fichier nommé **hosts** dans lequel nous avons défini les adresse IP et donné un nom à nos 2 nodes.

```

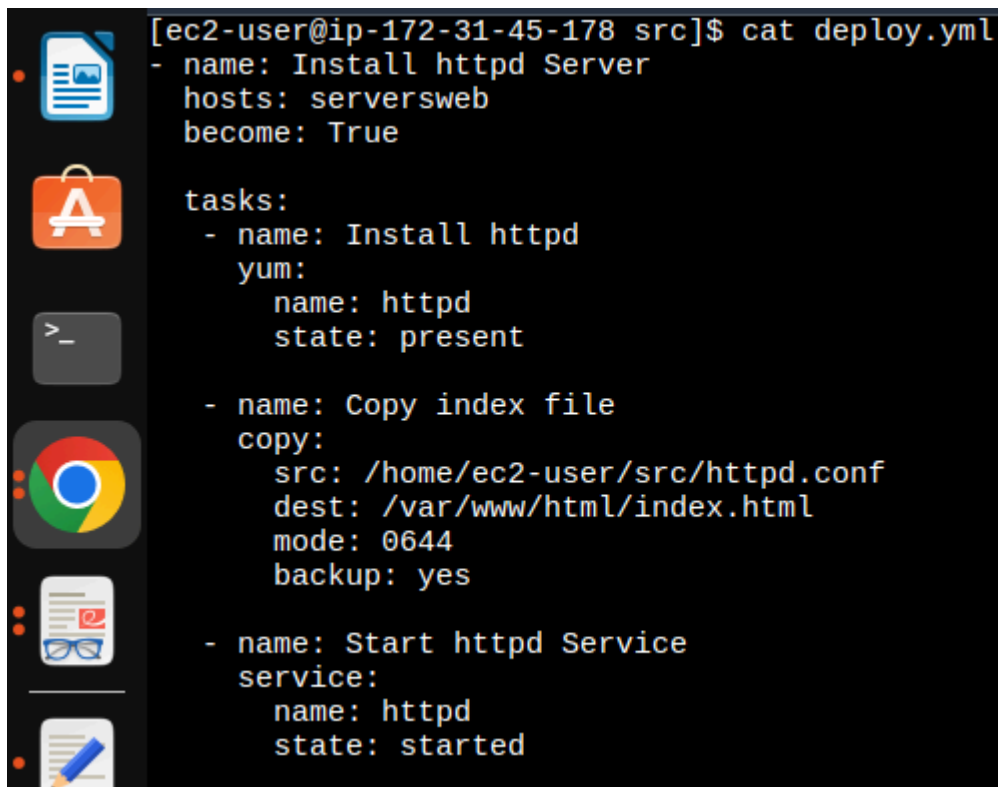
[ec2-user@ip-172-31-45-178 ~]$ cd src/
[ec2-user@ip-172-31-45-178 src]$ ls
deploy.yml  doodle_deployment  hosts  httpd.conf
[ec2-user@ip-172-31-45-178 src]$ cat hosts
[serversweb]
172.31.35.202
172.31.40.114
[ec2-user@ip-172-31-45-178 src]$

```

Ainsi on utilise la commande suivant pour vérifier la connectivité:

```
[root@ip-172-31-45-178 ~]# cd /home/ec2-user/src/doodle_deployment/ansible/
[root@ip-172-31-45-178 ansible]# ansible all -i ./inventories/production/hosts -m ping
[WARNING]: Platform linux on host 172.31.40.114 is using the discovered Python interpreter at /usr/bin/python3.9, but future
installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-
core/2.15/reference_appendices/interpreter_discovery.html for more information.
172.31.40.114 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3.9"
  },
  "changed": false,
  "ping": "pong"
}
[WARNING]: Platform linux on host 172.31.35.202 is using the discovered Python interpreter at /usr/bin/python3.9, but future
installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-
core/2.15/reference_appendices/interpreter_discovery.html for more information.
172.31.35.202 | SUCCESS => {
  "ansible_facts": {
    "discovered_interpreter_python": "/usr/bin/python3.9"
  },
  "changed": false,
  "ping": "pong"
}
[root@ip-172-31-45-178 ansible]#
```

☐ Mise en place du playbook

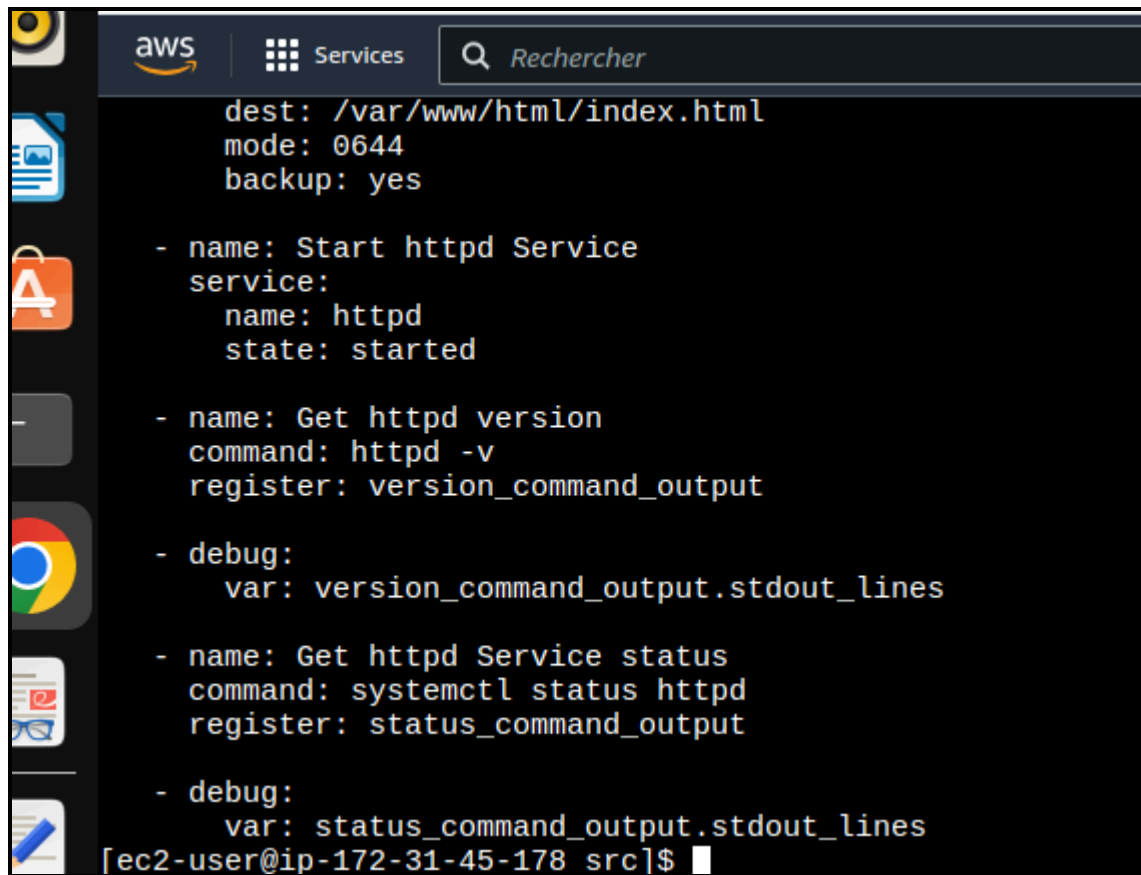
A terminal window with a dark background and a sidebar on the left containing icons for a file manager, an application store, a terminal, a web browser, a document, and a notepad. The terminal displays the command 'cat deploy.yml' and its output, which is an Ansible playbook named 'deploy.yml'. The playbook has a 'hosts' list of 'serversweb' and 'become: True'. It contains three tasks: 'Install httpd Server' using the 'yum' module, 'Copy index file' using the 'copy' module, and 'Start httpd Service' using the 'service' module.

```
[ec2-user@ip-172-31-45-178 src]$ cat deploy.yml
- name: Install httpd Server
  hosts: serversweb
  become: True

  tasks:
    - name: Install httpd
      yum:
        name: httpd
        state: present

    - name: Copy index file
      copy:
        src: /home/ec2-user/src/httpd.conf
        dest: /var/www/html/index.html
        mode: 0644
        backup: yes

    - name: Start httpd Service
      service:
        name: httpd
        state: started
```

The image shows a screenshot of an AWS Management Console terminal window. The top bar includes the AWS logo, a 'Services' menu, and a search bar labeled 'Rechercher'. The terminal output displays the results of an Ansible playbook. It starts with file configuration for /var/www/html/index.html, followed by starting the httpd service, getting its version, and checking its status. Debug messages show the stdout_lines for the version and status commands. The prompt at the bottom is [ec2-user@ip-172-31-45-178 src]\$.

```
dest: /var/www/html/index.html
mode: 0644
backup: yes

- name: Start httpd Service
  service:
    name: httpd
    state: started

- name: Get httpd version
  command: httpd -v
  register: version_command_output

- debug:
  var: version_command_output.stdout_lines

- name: Get httpd Service status
  command: systemctl status httpd
  register: status_command_output

- debug:
  var: status_command_output.stdout_lines
[ec2-user@ip-172-31-45-178 src]$
```

Ce playbook Ansible est conçu pour installer et configurer le serveur HTTP Apache (httpd) sur les hôtes spécifiés dans le groupe serversweb. Voici ce que chaque tâche du playbook fait :

-Install httpd : Cette tâche utilise le module yum pour installer le package httpd sur les hôtes cibles.

-Copy index file : Cette tâche utilise le module copy pour copier un fichier index.html depuis le chemin local /home/ec2-user/src/httpd.conf vers le répertoire de base du serveur web /var/www/html/ sur les hôtes cibles. Le fichier est copié avec les autorisations de lecture pour tout le monde (0644).

-Start httpd Service : Cette tâche utilise le module service pour démarrer le service httpd sur les hôtes cibles.

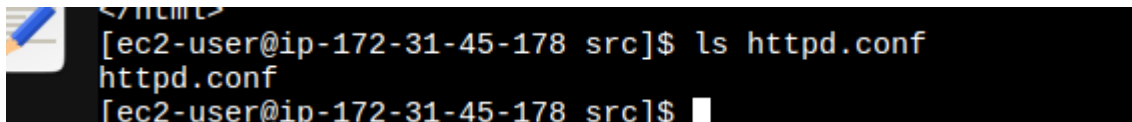
-Get httpd version : Cette tâche utilise la commande httpd -v pour obtenir la version du serveur HTTP Apache installée sur les hôtes cibles. Le résultat de la commande est stocké dans la variable version_command_output.

-Debug : Cette tâche utilise le module debug pour afficher les lignes de sortie de la commande httpd -v, stockées dans la variable version_command_output.

-Get httpd Service status : Cette tâche utilise la commande `systemctl status httpd` pour obtenir le statut du service httpd sur les hôtes cibles. Le résultat de la commande est stocké dans la variable `status_command_output`.

-Debug : Cette tâche utilise le module `debug` pour afficher les lignes de sortie de la commande `systemctl status httpd`, stockées dans la variable `status_command_output`.

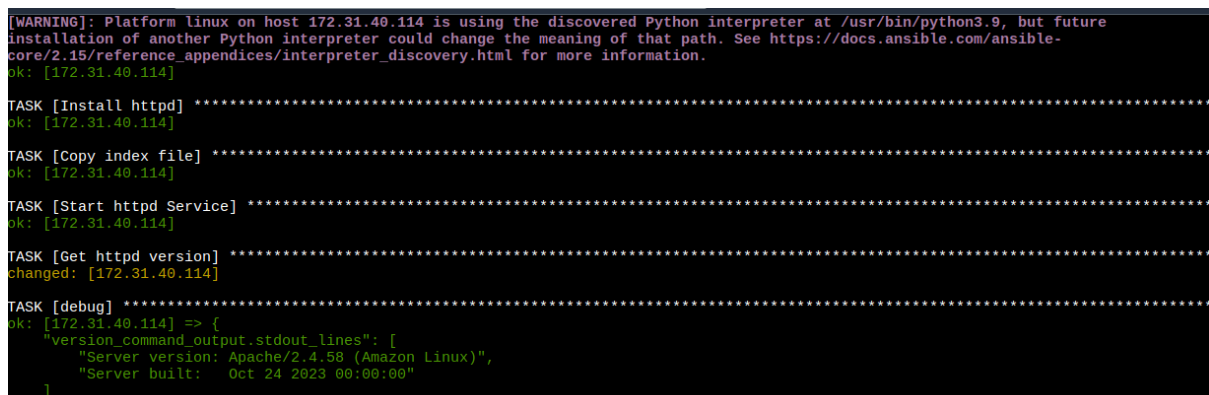
Ainsi , le fichier `httpd.conf` contient notre site web sur le node manager.



```
</html>
[ec2-user@ip-172-31-45-178 src]$ ls httpd.conf
httpd.conf
[ec2-user@ip-172-31-45-178 src]$
```

4. Deploiement automatique

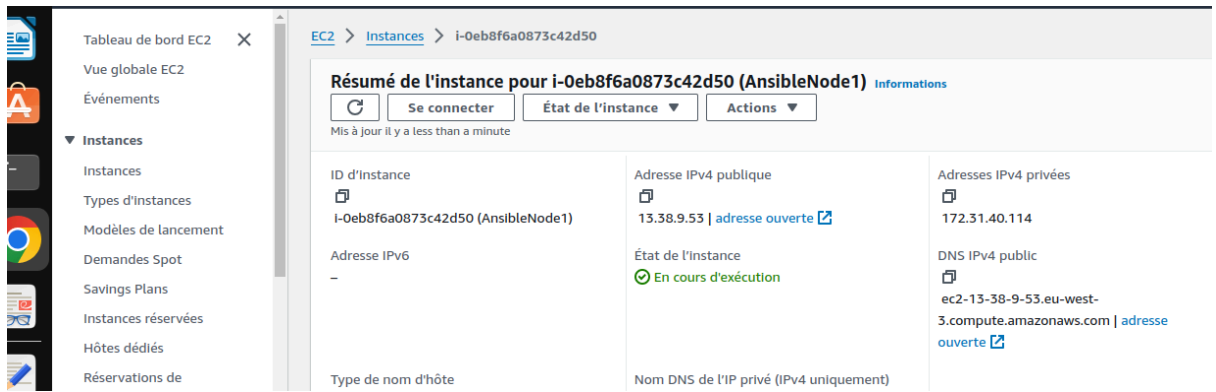
Pour faire le depoiement ,nous allons utilis   la commande suivant :
ansible-playbook deploy.yml -i hosts



```
[WARNING]: Platform linux on host 172.31.40.114 is using the discovered Python interpreter at /usr/bin/python3.9, but future
installation of another Python interpreter could change the meaning of that path. See https://docs.ansible.com/ansible-
core/2.15/reference_appendices/interpreter_discovery.html for more information.
ok: [172.31.40.114]
TASK [Install httpd] *****
ok: [172.31.40.114]
TASK [Copy index file] *****
ok: [172.31.40.114]
TASK [Start httpd Service] *****
ok: [172.31.40.114]
TASK [Get httpd version] *****
changed: [172.31.40.114]
TASK [debug] *****
ok: [172.31.40.114] => {
  "version_command_output.stdout_lines": [
    "Server version: Apache/2.4.58 (Amazon Linux)",
    "Server built:   Oct 24 2023 00:00:00"
  ]
}
```

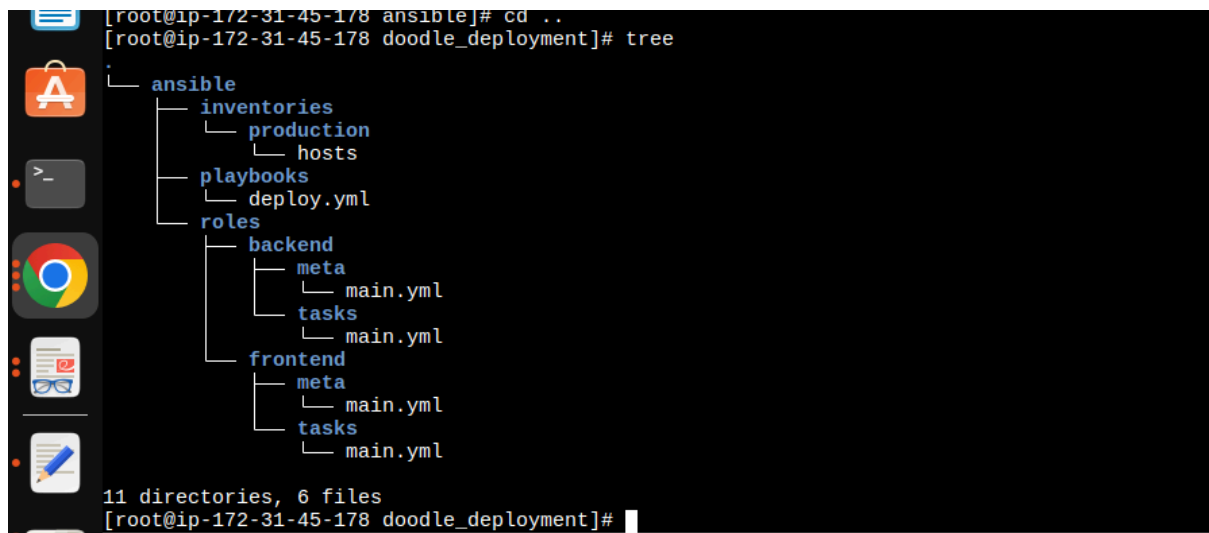
La figure pr  c  dente permet de confirmer que le d  ploiement est fait avec succ  s.

Pour tester nos serveurs, on se connecte avec l'adresse publique d'un serveur node et la plage doit s'afficher .



On voit bien que notre site s'affiche ainsi le déploiement s'est bien passé.

Après nous avons aussi fait le playbook du projet doodle(voir ci-dessous)



Description du déploiement avec Ansible pour le projet Doodle :

-Inventaires :

Le dossier inventories/ contient les fichiers d'inventaire pour les différents environnements de déploiement, par exemple production/ pour les instances de production.

Le fichier hosts répertorie les adresses IP ou les noms d'hôtes des machines cibles.

-Rôles :

Le dossier roles/ contient des rôles Ansible pour organiser les tâches spécifiques au backend et au frontend.

Le rôle backend contient les tâches pour le backend Quarkus, telles que la construction du projet et le déploiement.

Le rôle frontend contient les tâches pour le frontend Angular, telles que la construction du projet et le déploiement.

-Playbooks :

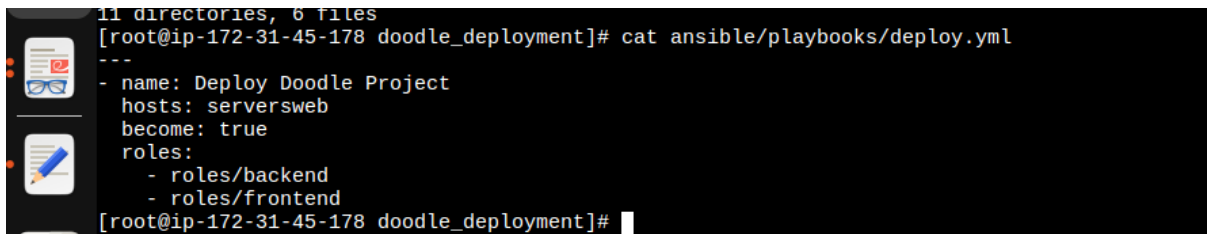
Le dossier playbooks/ contient le playbook principal deploy.yml pour orchestrer le déploiement de l'ensemble du projet Doodle.

Ce playbook utilise les rôles définis dans roles/ pour effectuer les tâches nécessaires pour déployer le backend et le frontend.

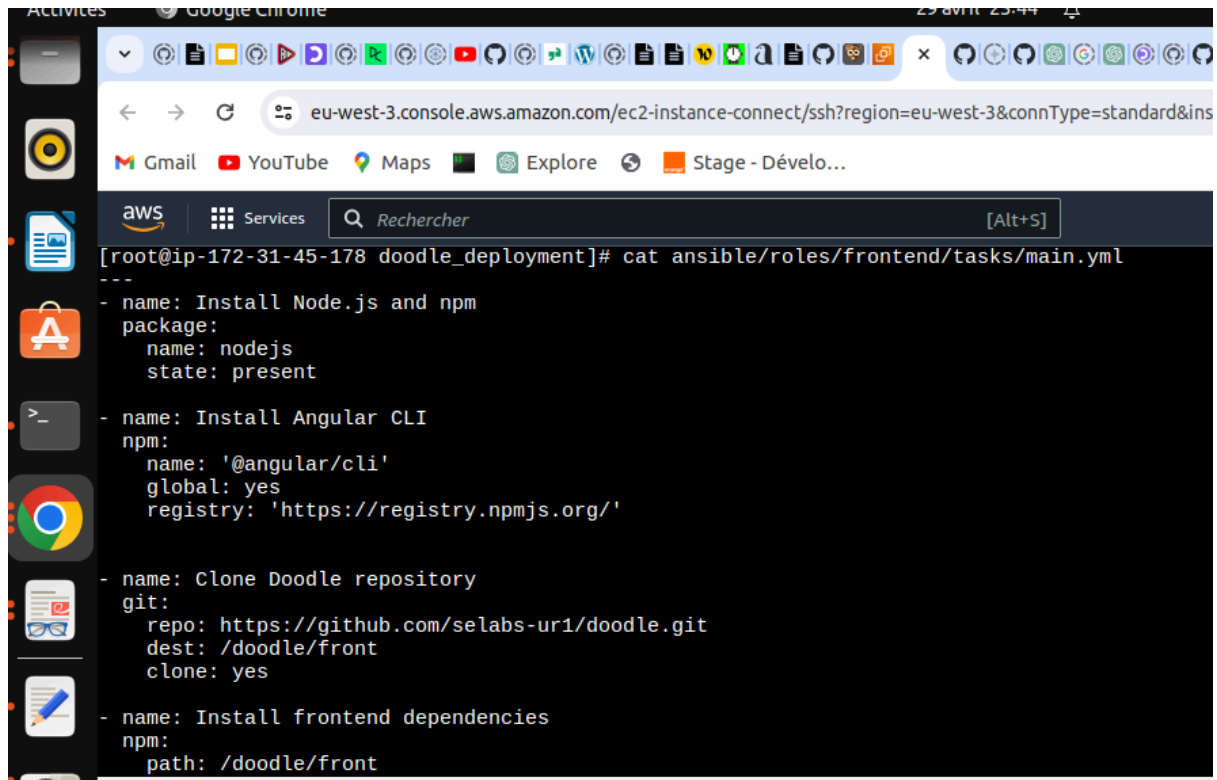
-Dossier du projet Doodle :

Le dossier doodle/ contient les fichiers source du projet Doodle, y compris les dossiers api/ et front/ pour le backend et le frontend respectivement.

Ces dossiers contiennent les fichiers de code source spécifiques à chaque partie du projet, tels que les fichiers source Java pour le backend Quarkus et les fichiers TypeScript et HTML pour le frontend Angular.

A terminal window with a dark background. On the left, there is a vertical sidebar with three icons: a document with a red 'X', a document with a blue pencil, and a document with a magnifying glass. The terminal text shows the command to view the content of the 'deploy.yml' file in the 'ansible/playbooks/' directory. The output shows the file's metadata and its content, which defines a playbook named 'Deploy Doodle Project' targeting 'serversweb' hosts, becoming the user 'true', and including 'backend' and 'frontend' roles.

```
11 directories, 6 files
[root@ip-172-31-45-178 doodle_deployment]# cat ansible/playbooks/deploy.yml
---
- name: Deploy Doodle Project
  hosts: serversweb
  become: true
  roles:
    - roles/backend
    - roles/frontend
[root@ip-172-31-45-178 doodle_deployment]#
```



The screenshot shows a terminal window with the following content:

```
[root@ip-172-31-45-178 doodle_deployment]# cat ansible/roles/frontend/tasks/main.yml
---
- name: Install Node.js and npm
  package:
    name: nodejs
    state: present

- name: Install Angular CLI
  npm:
    name: '@angular/cli'
    global: yes
    registry: 'https://registry.npmjs.org/'

- name: Clone Doodle repository
  git:
    repo: https://github.com/selabs-ur1/doodle.git
    dest: /doodle/front
    clone: yes

- name: Install frontend dependencies
  npm:
    path: /doodle/front
```

En conclusion, Ansible s'est avéré être un outil puissant et polyvalent pour l'automatisation des tâches de déploiement, offrant des avantages significatifs en termes d'efficacité, de cohérence et de fiabilité. Son utilisation a grandement facilité la gestion et le déploiement de nos projets (Notre projet choisi et le **projet Doodle**), et nous envisageons de continuer à l'exploiter dans nos futurs projets pour maximiser notre productivité et notre efficacité opérationnelle.

Conclusion

Cette réalisation pratique nous a permis de mettre en lumière les aspects techniques de la mise en place d'outils DevOps, en l'occurrence Terraform et Ansible. Nous avons examiné en détail les étapes nécessaires pour déployer un projet sur AWS en utilisant ces deux outils, ainsi que les difficultés rencontrées et les solutions mises en œuvre. Cette expérience nous a offert un aperçu précieux des pratiques DevOps et des outils associés, renforçant ainsi notre compréhension des meilleures pratiques pour la gestion des infrastructures et des configurations dans un environnement de développement moderne.