# TP Cloud Mise en Prod

## Installation

Il n'y a que 2 prérequis pour que les scripts fonctionnent :

- avoir installé boto3
- avoir configuré grâce à aws configure son environnement pour avoir accès au cloud AWS de pythagore, rien de plus :)

## 1) Création d'instances

Pour cette partie, aucun problème n'a été rencontré, j'ai créé directement l'instance EC2 type t2.nano grâce à la console AWS.
Cependant, lorsque l'on créé une instance EC2, elle n'a par défaut pas de tag "Name" qui est un surnom pour cette instance, ce qui rend difficile de l'identifier hormis lorsque l'on a son id bien entendu.
C'est pourquoi j'ai rajouté un script "add_instance_name.py" qui permet comme son nom l'indique, de rajouter un nom à notre instance, ce qui facilitera les scripts suivants, qui prennent quasiment tous en paramètres un tag name.

```python
#! /usr/env python3
import boto3
import argparse
from botocore.exceptions import ClientError

# Add instance id and new instance name
parser = argparse.ArgumentParser()
parser.add_argument("--instance-id", "-id", help="Instance id")
parser.add_argument("--name", "-n", help="Instance name")

# Get args
instance_id = parser.parse_args().instance_id
instance_name = parser.parse_args().name

# Get all instances filtered by instances names
client = boto3.client("ec2")
try:
    response = client.describe_instances(InstanceIds=[instance_id])
except ClientError:
    print(f"{instance_id} does not exists")
    exit()

# Now add a name to our instance
client.create_tags(Resources=[instance_id], Tags=[{"Key": "Name", "Value": instance_name}])
print(f"Instance with id {instance_id} is now tagged as {instance_name}")

```

Comme on peut le voir, ce script prend en paramètre l'id de l'instance, et le nom que l'on veut lui attribuer. Si tout s'est bien passé, il print "Instance with id <id-de-l'instance> is now tagged as <nom-de-l'instance>".

J'ai trouvé cette solution plus simple, car il est plus simple d'écrire un nom de tag comme "my_instance_2" que l'id de l'instance.

## 2) Groupes d'instances

Le premier script qui permet d'ajouter un tag à une ou plusieurs instances données est le script "add_ec2_tags.py".

```python
#!/usr/env python3
import boto3
import argparse

# Add instances names and tags arguments
parser = argparse.ArgumentParser()
parser.add_argument("--instances", nargs="+", default=[], help="List of EC2 instances name")
parser.add_argument("--tag-name", help="Tag Name to add to instances")
parser.add_argument("--tag-value", help="Tag Value to add to instances")

# Get args
instances = parser.parse_args().instances
tag_name = parser.parse_args().tag_name
tag_value = parser.parse_args().tag_value

# Get all instances filtered by instances names
client = boto3.client("ec2")
filters = [{"Name": "tag:Name", "Values": instances}]
response = client.describe_instances(Filters=filters)

# Get instances names found
instances_names = []
instances_ids = []
for instance in response["Reservations"][0]["Instances"]:
    instances_ids.append(instance["InstanceId"])
    for tag in instance["Tags"]:
        if tag["Key"] == "Name":
            instances_names.append(tag["Value"])

# Compare them to find if an instance in arguments does not exists
# If 1 or more instances dont exists, print them and exit
instances_not_found = []
for instance in instances:
    if instance not in instances_names:
        instances_not_found.append(instance)
if len(instances_not_found) > 0:
    for i in instances_not_found:
        print(f"Instance {i} does not exists")
    exit()

# Now add tag to given instances
client.create_tags(Resources=instances_ids, Tags=[{"Key": tag_name, "Value": tag_value}])
for i in instances_names:
    print(f"Tag {tag_name}:{tag_value} added to {i}")

```

Ce script prend en paramètre une liste de noms d'instances auxquelles on veut ajouter un tag, le nom du tag, et la valeur du tag.
Ensuite il ajoute le tag pour chacune des instances, et enfin si tout se passe bien, il print que le tag a été ajouté pour chacune des instances.

Le second script consiste à lister les paramètres des instances d'un groupe donné.
J'avoue que "d'un groupe donné" m'a laissé un peu perplexe mais j'en ai déduit qu'il

s’agissait des instances qui ont le même tag en commun. Ce script se nomme “get_instances_params.py”.

```python
#! /usr/env python3
import boto3
import argparse

# Add tags arguments
parser = argparse.ArgumentParser()
parser.add_argument("--tag-name", help="Tag Name")
parser.add_argument("--tag-value", help="Tag Value")

# Get args
tag_name = parser.parse_args().tag_name
tag_value = parser.parse_args().tag_value

# Get all instances filtered by tag
client = boto3.client("ec2")
filters = [{"Name": f"tag:{tag_name}", "Values": [tag_value]}]
response = client.describe_instances(Filters=filters)

# Now filter the result to get few information about each instance and print them
print(f"Instances found with tag {tag_name}:{tag_value} \n")
for instance in response["Reservations"][0]["Instances"]:
    print({
        "image_id": instance["ImageId"],
        "instance_id": instance["InstanceId"],
        "instance_type": instance["InstanceType"],
        "availability_zone": instance["Placement"]["AvailabilityZone"],
        "tags": instance["Tags"]
    })
    print("\n")

```

Ce script prend donc en paramètres le nom du tag, ainsi que la valeur du tag, pour pouvoir retrouver toutes les instances associées à ce tag. Puis le script va lister les paramètres les plus utiles pour chacune des instances trouvées plutôt que tout afficher, AWS renvoie un dictionnaire vraiment trop gros, ce qui le rend difficile à lire. On aurait pu mettre ce que l’on veut, mais j’ai choisi ce que je trouvais de plus pertinent, c'est à dire : L’id de l’image utilisée par l’instance, l’id de l’instance, le type de l’instance (t2.nano, t2.micro …), l’availability zone dans laquelle se trouve l’instance, et enfin les tags associés à cette instance.

Enfin pour le dernier script de cette partie, nommé “stop_instances.py” permet de stopper les instances grâce à un tag donné. Ce script récupère donc le nom et la

valeur d'un tag, récupère toutes les instances qui possèdent ce tag, et stop toutes ces instances.

```python
#! /usr/env python3
import ...

# Add tags arguments
parser = argparse.ArgumentParser()
parser.add_argument("--tag-name", help="Tag Name")
parser.add_argument("--tag-value", help="Tag Value")

# Get args
tag_name = parser.parse_args().tag_name
tag_value = parser.parse_args().tag_value

# Get all instances filtered by tag
client = boto3.client("ec2")
filters = [{"Name": f"tag:{tag_name}", "Values": [tag_value]}]
response = client.describe_instances(Filters=filters)

# Get ids of instances
instances_ids = []
instances_names = []
for instance in response["Reservations"][0]["Instances"]:
    instances_ids.append(instance["InstanceId"])
    for tag in instance["Tags"]:
        if tag["Key"] == "Name":
            instances_names.append(tag["Value"])

# Stop instances by their ids
client.stop_instances(InstanceIds=instances_ids)
print(f"{' '.join(instances_names)} has been stopped")

```

Je trouvais dommage de ne pas ajouter le même script pour faire l'inverse : lancer les instances grâce à leurs tags !
C'est pour cela que j'ai ajouté le script "start_instances.py".

```python
#! /usr/env python3
import boto3
import argparse

# Add tags arguments
parser = argparse.ArgumentParser()
parser.add_argument("--tag-name", help="Tag Name")
parser.add_argument("--tag-value", help="Tag Value")

# Get args
tag_name = parser.parse_args().tag_name
tag_value = parser.parse_args().tag_value

# Get all instances filtered by tag
client = boto3.client("ec2")
filters = [{"Name": f"tag:{tag_name}", "Values": [tag_value]}]
response = client.describe_instances(Filters=filters)

# Get ids of instances
instances_ids = []
instances_names = []
for instance in response["Reservations"][0]["Instances"]:
    instances_ids.append(instance["InstanceId"])
    for tag in instance["Tags"]:
        if tag["Key"] == "Name":
            instances_names.append(tag["Value"])

# Start instances by their ids
client.start_instances(InstanceIds=instances_ids)
print(f"{' '.join(instances_names)} has been started")

```

## 3) Groupes de sécurité

Le script de cette partie se nomme “set_ec2_sg_rule.py”, il permet de modifier / ajouter une règle aux security groups liés aux instances trouvées grâce à un tag. C’est-à-dire : il prend en paramètre un tag, un paramètre authorize ou revoke, le port lié à cette règle, et 2 paramètres inbound et outbound pour définir si l’on met cette règle en entrant et/ou sortant, il faut donc en mettre au moins 1 entre les 2.

Le script récupère donc tous ces paramètres, recherche toutes les instances liées au tag donné, puis récupère les security groups liés à ces instances. Enfin il ajoute la

règle à ces security groups en fonction des paramètres donnés (inbound, outbound, revoke, authorize, le port).

```python
#! /usr/env python3
import boto3
import argparse

# Add needed arguments
parser = argparse.ArgumentParser()
parser.add_argument("--tag-name", "-tn", help="Tag Name")
parser.add_argument("--tag-value", "-tv", help="Tag Value")
parser.add_argument("--authorize", "-a", nargs="?", const=True, default=False, help="Authorize security group access")
parser.add_argument("--revoke", "-r", nargs="?", const=True, default=False, help="Revoke security group access")
parser.add_argument("--port", "-p", type=int, help="Specify Port")
parser.add_argument("--inbound", "-i", nargs="?", const=True, default=False, help="Specify inbound access")
parser.add_argument("--outbound", "-o", nargs="?", const=True, default=False, help="Specify outbound access")

# Get args
tag_name = parser.parse_args().tag_name
tag_value = parser.parse_args().tag_value
authorize = parser.parse_args().authorize
revoke = parser.parse_args().revoke
port = parser.parse_args().port
inbound = parser.parse_args().inbound
outbound = parser.parse_args().outbound

# Authorize vs revoke
if authorize is True and revoke is True:
    print("You must choose between authorize and revoke")
    exit()
elif authorize is False and revoke is False:
    print("You must choose authorize or revoke")
    exit()
```

```python
# Get all instances filtered by tag
client = boto3.client("ec2")
filters = [{"Name": f"tag:{tag_name}", "Values": [tag_value]}]
response = client.describe_instances(Filters=filters)

# Get security groups ids of each instances
security_group_ids = []
for instance in response["Reservations"][0]["Instances"]:
    for group in instance["SecurityGroups"]:
        security_group_ids.append(group["GroupId"])

# Set rules on all security groups by their ids
ip_permissions = [{"FromPort": port, "IpProtocol": "tcp", "ToPort": port, "IpRanges": [{"CidrIp": "0.0.0.0/0"}]}]
for security_group_id in security_group_ids:
    if authorize:
        if inbound:
            client.authorize_security_group_ingress(GroupId=security_group_id, IpPermissions=ip_permissions)
        if outbound:
            client.authorize_security_group_egress(GroupId=security_group_id, IpPermissions=ip_permissions)
    else:
        if inbound:
            client.revoke_security_group_ingress(GroupId=security_group_id, IpPermissions=ip_permissions)
        if outbound:
            client.revoke_security_group_egress(GroupId=security_group_id, IpPermissions=ip_permissions)

# Return print
print(f"Security groups {' '.join(security_group_ids)} now {'authorize' if authorize else 'revoke'} "
      f"traffic on port {port} with inbound={inbound} and outbound={outbound}")
```

## 4) Images

Pour cette dernière partie, il fallait écrire un script qui permet de lancer une reconnaissance sur les images d'un répertoire donné. Ayant déjà effectué ce script lors de notre cours, j'ai décidé d'utiliser textract pour faire de la reconnaissance de texte dans l'image, sur les images donné, je trouvais cela plus intéressant et un cela m'a fait un bon exercice !

Ce script nommé "extract_text_from_images.py" prend donc en paramètre un path vers un répertoire qui contient des images, ensuite il va regarder si tous les fichiers présents dans le répertoire sont bien des images, et que les formats sont bien supportés par textract (.png, .jpeg, .jpg, .pdf, .tiff). Si tout est bon, le script va lancer textract sur chacune des images et print le résultat pour chaque image !

```python
#! /usr/env python3
import boto3
import os
import argparse

# Add image path params
parser = argparse.ArgumentParser()
parser.add_argument("--folder", "-f", help="Absolute folder path which contains images")
folder = parser.parse_args().folder

# Check if folder exists
if not os.path.exists(folder):
    print(f"{folder} does not exists")
    exit()

# Check if all images format are supported by textract
images = os.listdir(folder)
supported_formats = [".pdf", ".tiff", ".jpg", ".jpeg", ".png"]
not_supported = False
for image in images:
    format = os.path.splitext(image)[1]
    if format not in supported_formats:
        not_supported = True
        print(f"Image {image} with format {format} is not supported, use {' '.join(supported_formats)} instead")
if not_supported:
    exit()
```

```python

# Extract text from images
client = boto3.client("textract")
results = []
for image in images:
    with open(f"{folder}/{image}", "rb") as file:
        img = bytearray(file.read())
    response = client.detect_document_text(Document={"Bytes": img})
    text = [block["Text"] for block in response["Blocks"] if block["BlockType"] == "LINE"]
    results.append([image, text])

# Print results
for result in results:
    print(f"Image: {result[0]}, detected text: {result[1]}")

```

Pour conclure, je trouve que certaines parties du sujet n’étaient pas très clair sur ce qui était attendu, j’espère donc avoir bien compris, ou que même si je n’ai pas tout bien compris que j’ai pu vous montrer que je suis capable d’utiliser les services AWS ainsi que le langage python.