TD1-b : IAM Central et IAM Projet avec CI/CD

Objectifs

- Comprendre le modèle IAM central / IAM projet utilisé en entreprise
- Structurer deux dépôts Git avec des responsabilités distinctes
- Configurer un pipeline GitHub Actions qui assume un rôle IAM via OIDC
- Appliquer le principe du moindre privilège à l'échelle d'un projet
- Comprendre pourquoi les clés statiques sont à éviter

Contexte

Vous intégrez l'équipe Cloud Platform d'une entreprise SaaS.

L'entreprise utilise AWS et GitHub Actions pour ses déploiements.

Deux équipes coexistent :

- **Équipe Cloud Platform** : gère les utilisateurs, groupes, et politiques globales via un dépôt centralisé
- **Équipe Dev projet-alpha** : déploie une application sur AWS via son propre pipeline

Votre mission est de mettre en place cette séparation proprement, avec un pipeline CI/CD qui n'utilise pas de clés AWS statiques.

Prérequis

- Avoir réalisé le TD1 IAM (groupes, utilisateurs, politiques de base)
- Avoir un compte AWS avec les droits IAM
- Avoir un compte GitHub
- Terraform installé en local (>= 1.5)
- AWS CLI configuré

Structure des dépôts

Vous allez travailler avec deux dépôts distincts.

Créez les deux dépôts GitHub suivants :

- `repo-iam-central`
- `repo-projet-alpha`

La structure cible est la suivante :

```
repo-iam-central/
  ├── versions.tf
```

```
  ├── providers.tf
  ├── groups.tf
  ├── users.tf
  ├── policies-base.tf
  └── .github/
        └── workflows/
              └── iam-apply.yml

repo-projet-alpha/
  ├── terraform/
  │   ├── versions.tf
  │   ├── providers.tf
  │   ├── role.tf
  │   └── policy.tf
  ├── app/
  │   └── index.html
  └── .github/
        └── workflows/
              └── deploy.yml
```

Partie 1 : Le dépôt IAM central

1.1 Configuration Terraform

Fichier : `repo-iam-central/versions.tf`

```
terraform {
  required_version = ">= 1.5"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }

  backend "s3" {
    bucket = "mon-tfstate-central"
    key    = "iam/central/terraform.tfstate"
    region = "eu-west-3"
  }
}
```

Fichier : `repo-iam-central/providers.tf`

```
provider "aws" {
  region = "eu-west-3"
}
```

1.2 Groupes globaux

Fichier : `repo-iam-central/groups.tf`

```
resource "aws_iam_group" "developers" {
  name = "developers"
}

resource "aws_iam_group" "ops" {
  name = "ops"
}
```

1.3 Politique de base lecture S3

Fichier : `repo-iam-central/policies-base.tf`

```
resource "aws_iam_policy" "read_only_s3" {
  name        = "global-read-only-s3"
  description = "Lecture seule sur tous les buckets S3 - politique globale"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect   = "Allow"
        Action   = ["s3:GetObject", "s3:ListBucket"]
        Resource = "*"
      }
    ]
  })
}

resource "aws_iam_group_policy_attachment" "dev_read_s3" {
  group      = aws_iam_group.developers.name
  policy_arn = aws_iam_policy.read_only_s3.arn
}
```

Question 1.1 : Pourquoi cette politique est-elle attachée au groupe et non à un utilisateur directement ?

Question 1.2 : Que se passe-t-il si un développeur change de projet ? Faut-il modifier cette politique ?

Question 1.3 : Qui devrait avoir le droit de modifier ce dépôt ? Pourquoi ?

1.4 Pipeline CI/CD du dépôt central

Ce pipeline s'exécute uniquement sur la branche `main`, après une Pull Request validée.

Fichier : `repo-iam-central/.github/workflows/iam-apply.yml`

```
name: IAM Central Apply

on:
  push:
    branches:
      - main

permissions:
  id-token: write
  contents: read

jobs:
  terraform:
    name: Apply IAM Central
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Configure AWS credentials via OIDC
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::${{ secrets.AWS_ACCOUNT_ID }}:role/role-iam-
central-ci
          aws-region: eu-west-1

      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v3
        with:
          terraform_version: 1.5.0

      - name: Terraform Init
        run: terraform init

      - name: Terraform Plan
        run: terraform plan

      - name: Terraform Apply
        run: terraform apply -auto-approve
```

Question 1.4 : Pourquoi le pipeline utilise `id-token: write` ?

Question 1.5 : Pourquoi le pipeline ne se déclenche que sur `main` et pas sur toutes les branches ?

Question 1.6 : Que se passe-t-il si quelqu'un pousse directement sur main sans PR ? Quelle règle GitHub pourrait empêcher ça ?

Partie 2 : Le rôle OIDC pour le pipeline central

Avant que le pipeline puisse fonctionner, il faut créer le rôle IAM que GitHub Actions va assumer.

Ce rôle est créé **une seule fois manuellement** ou via un bootstrap Terraform séparé.

2.1 Comprendre OIDC

Sans OIDC :

```
GitHub Actions
  └── utilise une clé AWS statique stockée dans les secrets GitHub
        └── risque : clé exposée = compte compromis
```

Avec OIDC :

```
GitHub Actions
  └── présente un token signé par GitHub
        └── AWS vérifie ce token
              └── AWS délivre des credentials temporaires
                    └── valables le temps du job uniquement
```

Question 2.1 : Quelle est la durée de vie des credentials délivrés par OIDC ?

Question 2.2 : Que se passe-t-il si le token GitHub est intercepté après la fin du job ?

2.2 Création du provider OIDC GitHub dans AWS

Fichier : `repo-iam-central/oidc-github.tf`

```
resource "aws_iam_openid_connect_provider" "github" {
  url = "https://token.actions.githubusercontent.com"

  client_id_list = ["sts.amazonaws.com"]

  thumbprint_list = ["6938fd4d98bab03faadb97b34396831e3780aea1"]
}
```

Question 2.3 : À quoi correspond le `thumbprint_list` ? Pourquoi ne faut-il pas le modifier sans vérification ?

2.3 Création du rôle assumé par le pipeline central

Fichier : `repo-iam-central/role-ci-central.tf`

```
data "aws_iam_policy_document" "github_assume_role_central" {
  statement {
    effect  = "Allow"
    actions = ["sts:AssumeRoleWithWebIdentity"]

    principals {
      type        = "Federated"
      identifiers = [aws_iam_openid_connect_provider.github.arn]
    }

    condition {
      test     = "StringEquals"
      variable = "token.actions.githubusercontent.com:aud"
      values   = ["sts.amazonaws.com"]
    }

    condition {
      test     = "StringLike"
      variable = "token.actions.githubusercontent.com:sub"
      values   = ["repo:mon-org/repo-iam-central:ref:refs/heads/main"]
    }
  }
}

resource "aws_iam_role" "iam_central_ci" {
  name               = "role-iam-central-ci"
  assume_role_policy = data.aws_iam_policy_document.github_assume_role_central.json
}

resource "aws_iam_role_policy_attachment" "iam_central_ci_policy" {
  role       = aws_iam_role.iam_central_ci.name
  policy_arn = "arn:aws:iam::aws:policy/IAMFullAccess"
}
```

Remplacez `mon-org/repo-iam-central` par votre organisation et nom de dépôt GitHub réels.

Question 2.4 : La condition `StringLike` sur le `sub` limite l'accès à quoi exactement ?

Question 2.5 : Pourquoi attache-t-on `IAMFullAccess` à ce rôle et pas à un utilisateur ?

Question 2.6 : Serait-il correct d'attacher `AdministratorAccess` à ce rôle ? Justifiez.

Partie 3 : Le dépôt projet-alpha

L'équipe dev du projet-alpha a besoin de déployer des fichiers sur un bucket S3.

Elle ne doit pas avoir accès aux autres ressources AWS.

3.1 Configuration Terraform du projet

Fichier : `repo-projet-alpha/terraform/versions.tf`

```
terraform {
  required_version = ">= 1.5"

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }

  backend "s3" {
    bucket = "mon-tfstate-central"
    key    = "projets/projet-alpha/terraform.tfstate"
    region = "eu-west-1"
  }
}
```

Fichier : `repo-projet-alpha/terraform/providers.tf`

```
provider "aws" {
  region = "eu-west-1"
}
```

3.2 Rôle IAM spécifique au projet

Fichier : `repo-projet-alpha/terraform/role.tf`

```
data "aws_iam_policy_document" "github_assume_role_alpha" {
  statement {
    effect  = "Allow"
    actions = ["sts:AssumeRoleWithWebIdentity"]

    principals {
      type        = "Federated"
      identifiers = ["arn:aws:iam::${var.aws_account_id}:oidc-
provider/token.actions.githubusercontent.com"]
    }
```

```
    condition {
      test     = "StringEquals"
      variable = "token.actions.githubusercontent.com:aud"
      values   = ["sts.amazonaws.com"]
    }

    condition {
      test     = "StringLike"
      variable = "token.actions.githubusercontent.com:sub"
      values   = ["repo:mon-org/repo-projet-alpha:ref:refs/heads/main"]
    }
  }
}

resource "aws_iam_role" "deploy_alpha" {
  name               = "role-deploy-projet-alpha"
  assume_role_policy = data.aws_iam_policy_document.github_assume_role_alpha.json
}
```

3.3 Politique limitée au projet

Fichier : `repo-projet-alpha/terraform/policy.tf`

```

variable "aws_account_id" {
  type        = string
  description = "ID du compte AWS"
}

variable "bucket_name" {
  type        = string
  description = "Nom du bucket S3 du projet alpha"
  default     = "projet-alpha-assets"
}

resource "aws_s3_bucket" "alpha_assets" {
  bucket = var.bucket_name
}

resource "aws_iam_policy" "deploy_alpha_policy" {
  name        = "policy-deploy-projet-alpha"
  description = "Permissions de déploiement pour projet-alpha uniquement"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "s3:PutObject",
          "s3:GetObject",
          "s3:DeleteObject",
          "s3:ListBucket"
        ]
```

```
        Resource = [
          aws_s3_bucket.alpha_assets.arn,
          "${aws_s3_bucket.alpha_assets.arn}/*"
        ]
      }
    ]
  })
}

resource "aws_iam_role_policy_attachment" "deploy_alpha_attach" {
  role       = aws_iam_role.deploy_alpha.name
  policy_arn = aws_iam_policy.deploy_alpha_policy.arn
}
```

Question 3.1 : La politique donne accès à `s3:DeleteObject`. Est-ce justifié pour un pipeline de déploiement ? Dans quel cas oui, dans quel cas non ?

Question 3.2 : Pourquoi le `Resource` pointe sur un bucket précis et pas sur `*` ?

Question 3.3 : Que se passe-t-il si le pipeline du projet-alpha tente d'accéder à un bucket appartenant au projet-beta ?

3.4 Pipeline de déploiement du projet

Fichier : `repo-projet-alpha/.github/workflows/deploy.yml`

```
name: Deploy projet-alpha

on:
  push:
    branches:
      - main

permissions:
  id-token: write
  contents: read

jobs:
  deploy:
    name: Deploy to S3
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Configure AWS credentials via OIDC
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::${{ secrets.AWS_ACCOUNT_ID }}:role/role-
```

```
deploy-projet-alpha
          aws-region: eu-west-1

      - name: Deploy fichiers vers S3
        run: |
          aws s3 sync app/ s3://projet-alpha-assets/ --delete
```

Question 3.4 : Le flag `–delete` supprime les fichiers dans S3 qui ne sont plus dans `app/`. Quel risque cela représente-t-il ? Comment le mitiger ?

Question 3.5 : Pourquoi le rôle assumé ici est différent du rôle utilisé dans le dépôt IAM central ?

Partie 4 : Erreur volontaire - le piégeage du pipeline

Appliquez la modification suivante dans `repo-projet-alpha/terraform/policy.tf` et observez ce qui se passe.

Remplacez la section `Resource` par :

Fichier : `repo-projet-alpha/terraform/policy.tf` (version modifiée)

```
Resource = ["*"]
```

Question 4.1 : Que permet cette modification ? Quel est le risque concret ?

Question 4.2 : Le pipeline CI/CD du projet peut-il appliquer cette modification seul ? Pourquoi ?

Question 4.3 : Quel mécanisme AWS permet d'empêcher cette modification même si Terraform l'applique ?

Revertez la modification avant de continuer.

Partie 5 : Séparation DEV / PROD

Vous allez maintenant adapter le dépôt projet-alpha pour supporter deux environnements.

5.1 Ajout du workspace Terraform

```
cd repo-projet-alpha/terraform

terraform workspace new dev
terraform workspace new prod
terraform workspace select dev
```

5.2 Adaptation du code avec le workspace

Fichier : `repo-projet-alpha/terraform/policy.tf` (version workspace)

```
locals {
  environment = terraform.workspace
  bucket_name = "${local.environment}-projet-alpha-assets"
}

resource "aws_s3_bucket" "alpha_assets" {
  bucket = local.bucket_name
}

resource "aws_iam_policy" "deploy_alpha_policy" {
  name        = "policy-deploy-projet-alpha-${local.environment}"
  description = "Permissions de déploiement pour projet-alpha -
${local.environment}"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "s3:PutObject",
          "s3:GetObject",
          "s3:DeleteObject",
          "s3:ListBucket"
        ]
        Resource = [
          aws_s3_bucket.alpha_assets.arn,
          "${aws_s3_bucket.alpha_assets.arn}/*"
        ]
      }
    ]
  })
}
```

5.3 Pipeline avec environnement

Fichier : `repo-projet-alpha/.github/workflows/deploy.yml` (version multi-env)

```
name: Deploy projet-alpha

on:
  push:
    branches:
      - main
      - develop

permissions:
  id-token: write
  contents: read

jobs:
  deploy:
    name: Deploy to S3
    runs-on: ubuntu-latest

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Définir l'environnement
        id: env
        run: |
          if [ "${{ github.ref }}" == "refs/heads/main" ]; then
            echo "environment=prod" >> $GITHUB_OUTPUT
          else
            echo "environment=dev" >> $GITHUB_OUTPUT
          fi

      - name: Configure AWS credentials via OIDC
        uses: aws-actions/configure-aws-credentials@v4
        with:
          role-to-assume: arn:aws:iam::${{ secrets.AWS_ACCOUNT_ID }}:role/role-
deploy-projet-alpha
          aws-region: eu-west-1

      - name: Setup Terraform
        uses: hashicorp/setup-terraform@v3

      - name: Terraform Init
        working-directory: terraform
        run: terraform init

      - name: Terraform Workspace
        working-directory: terraform
        run: terraform workspace select ${{ steps.env.outputs.environment }}

      - name: Terraform Apply
        working-directory: terraform
        run: terraform apply -auto-approve

      - name: Deploy fichiers vers S3
        run: |
          aws s3 sync app/ s3://${{ steps.env.outputs.environment }}-projet-alpha-
assets/ --delete
```

Question 5.1 : Quelle branche déclenche le déploiement en prod ? Quelle branche déclenche le déploiement en dev ?

Question 5.2 : Le même rôle IAM est utilisé pour DEV et PROD. Quel problème cela pose-t-il ? Proposez une solution.

Question 5.3 : Si un développeur pousse accidentellement sur `main`, que se passe-t-il ? Comment l'empêcher ?

Challenge final

Vous devez implémenter les éléments suivants :

1. Deux rôles IAM distincts dans `repo-projet-alpha/terraform/role.tf`

- `role-deploy-projet-alpha-dev` : assumé uniquement depuis la branche `develop`
- `role-deploy-projet-alpha-prod` : assumé uniquement depuis la branche `main`

2. Deux politiques distinctes

- La politique DEV autorise `s3:DeleteObject`
- La politique PROD interdit `s3:DeleteObject` (déploiement additif uniquement)

3. Modifier le pipeline

- Le pipeline choisit le bon rôle selon la branche
- Ajouter une étape de validation manuelle avant le déploiement en prod (`environment: production` dans GitHub Actions)

Livrable attendu : les fichiers Terraform et le fichier de pipeline complets et fonctionnels.

Extension : SCP pour bloquer les débordements

En entreprise avec AWS Organizations, on ajoute une couche supplémentaire.

Une SCP (Service Control Policy) est une politique appliquée au niveau du compte AWS. Elle s'applique même si un rôle IAM essaie de faire quelque chose d'interdit.

Exemple : interdire la création de ressources hors `eu-west-1`

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:RequestedRegion": "eu-west-1"
        }
      }
    }
  ]
}
```

Question extension 1 : Une SCP est-elle une politique IAM comme les autres ? Quelle est la différence fondamentale ?

Question extension 2 : Si une SCP interdit une action et qu'une politique IAM l'autorise, que se passe-t-il ?

Question extension 3 : Dans quel cas une SCP est-elle indispensable en entreprise ?

Récap du modèle complet

```
repo-iam-central
  └── géré par équipe Cloud Platform
  └── pipeline protégé par branch protection + review
  └── rôle CI assumé via OIDC depuis main uniquement
  └── permissions : IAMFullAccess uniquement

repo-projet-alpha
  └── géré par équipe dev
  └── pipeline DEV depuis develop
  └── pipeline PROD depuis main avec validation manuelle
  └── rôle CI assumé via OIDC - un rôle par environnement
  └── permissions : S3 sur le bucket du projet uniquement

AWS Organizations (optionnel / niveau avancé)
  └── SCP appliquées au niveau du compte
  └── garde-fous infranchissables même pour un admin IAM
```

Bilan pédagogique

À l'issue de ce TD, vous devez être capables de :

- Expliquer pourquoi un dépôt IAM central est géré séparément des dépôts projet
- Configurer OIDC entre GitHub Actions et AWS sans clé statique
- Créer un rôle IAM dont la portée est limitée à un dépôt et une branche
- Distinguer les responsabilités entre équipe Cloud Platform et équipe Dev

- Identifier les risques d'une politique trop permissive dans un pipeline CI/CD