TD3 – Gestion des secrets, chiffrement et traçabilité AWS

Objectifs

- Supprimer les secrets en clair dans une infrastructure Terraform
- Utiliser AWS Secrets Manager pour stocker des identifiants sensibles
- Activer le chiffrement des données au repos sur RDS et S3
- Comprendre la différence entre chiffrement au repos et en transit
- Mettre en place une traçabilité avec CloudTrail
- Identifier une action suspecte via les logs

Contexte

Suite au TD2, l'infrastructure réseau est opérationnelle.

Le VPC est segmenté, l'ALB filtre le trafic entrant, les Security Groups isolent chaque couche.

Un nouvel audit de sécurité interne est réalisé.

Trois failles critiques sont identifiées malgré les corrections du TD2 :

- le mot de passe de la base de données est stocké en clair dans `instances.tf`
- la base de données RDS n'est pas chiffrée au repos
- aucune trace des actions réalisées sur le compte AWS n'est conservée

Ces trois points constituent des non-conformités bloquantes pour toute certification de sécurité (ISO 27001, SOC 2, RGPD).

Votre mission est de corriger ces trois failles en prolongeant l'infrastructure du TD2.

Pré-requis

- Projet Terraform du TD2 déployé et fonctionnel
- ALB opérationnel, Security Groups en place
- Accès AWS avec droits suffisants : IAM, Secrets Manager, S3, RDS, CloudTrail
- Terraform en version 1.5 ou supérieure

Structure du projet

Le TD3 s'appuie sur les ressources existantes du TD2 et ajoute trois nouveaux modules.

```
td3/
├── main.tf
├── variables.tf
├── outputs.tf
├── providers.tf
```

```
└── modules/
    ├── secrets/
    │   └── main.tf
    ├── database/
    │   └── main.tf
    ├── storage/
    │   └── main.tf
    └── logging/
        └── main.tf
```

Le dossier `td2/` n'est pas modifié directement.

Les ressources existantes du TD2 sont récupérées via des data sources Terraform.



Pourquoi utilise-t-on des data sources plutôt que de copier les ressources du TD2 dans TD3 ?

Que se passerait-il si on déclarait une deuxième fois la même ressource avec le même nom ?

1. Analyse de l'existant



Ouvrir le fichier `td2/network/instances.tf`.

Localiser la ressource `aws_db_instance.db`.



Relever les trois problèmes de sécurité présents dans cette ressource.

Pour chaque problème, indiquer :

- ce qui est en cause dans le code
- ce qu'un attaquant ou un développeur malveillant pourrait faire
- si ce problème est visible dans Git



Le fichier `instances.tf` est versionné dans Git.

Un développeur a poussé ce fichier il y a trois mois sur le dépôt de l'équipe.

Le mot de passe a été changé depuis, mais il reste dans l'historique Git.

Comment retrouver ce mot de passe dans l'historique Git ?

Pourquoi changer le mot de passe ne suffit-il pas si le fichier a déjà été commité ?



La ressource `aws_db_instance.db` dans le TD2 ne contient pas le paramètre

`storage_encrypted`.



Quelle est la valeur par défaut de ce paramètre dans AWS ?

Que signifie concrètement une base non chiffrée au repos ?

2. Declaration des providers

Le module `storage` utilisera le provider `random` pour générer des suffixes uniques sur les noms de buckets S3.

Les noms de buckets S3 sont globaux sur AWS : deux comptes ne peuvent pas avoir le même nom.



Créer le fichier de déclaration des providers.

Fichier : `td3/providers.tf`

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
    random = {
      source  = "hashicorp/random"
      version = "~> 3.0"
    }
  }
}

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



Pourquoi faut-il déclarer explicitement le provider `random` alors qu'il ne communique pas avec AWS ?

Que se passe-t-il si on l'utilise sans le déclarer dans le bloc `required_providers` ?

3. Declaration des variables

Le mot de passe initial doit être injecté au moment de l'exécution.

Il ne doit jamais apparaître dans un fichier `.tf`.



Créer le fichier de variables.

Fichier : `td3/variables.tf`

```
variable "db_initial_password" {
  description = "Mot de passe initial de la base de donnees. Injecte via tfvars ou
variable d'environnement."
  type        = string
  sensitive   = true
}
```



Créer le fichier de valeurs local pour les tests.

Fichier : `td3/terraform.tfvars`

```
db_initial_password = "InitialPass123!"
```



Créer le fichier `.gitignore` pour protéger les fichiers sensibles.

Fichier : `td3/.gitignore`

```
*.tfvars
*.tfstate
*.tfstate.backup
.terraform/
.terraform.lock.hcl
```



Le paramètre `sensitive = true` est déclaré sur la variable.

Est-ce que cela empêche le mot de passe d'apparaître dans le fichier `terraform.tfstate` ?

Pourquoi le state file est-il également considéré comme un fichier sensible ?



En production, on n'utilise pas de fichier `terraform.tfvars`.

Comment injecter une valeur sensible sans fichier, uniquement via le terminal ?

Quelle variable d'environnement Terraform lit-il automatiquement pour la variable `db_initial_password` ?

4. Recuperation des ressources existantes du TD2

Plutôt que de recréer les ressources, on les référence via des data sources.



Créer le fichier principal du TD3.

Fichier : `td3/main.tf`

```
# Recuperation du VPC existant du TD2
data "aws_vpc" "td2" {
  filter {
    name   = "tag:Name"
    values = ["td2-vpc"]
  }
}

# Recuperation du Security Group du backend TD2
data "aws_security_group" "backend" {
  filter {
    name   = "tag:Name"
    values = ["backend-sg"]
  }
  vpc_id = data.aws_vpc.td2.id
}

# Recuperation du subnet group RDS existant
data "aws_db_subnet_group" "main" {
  name = "td2-db-subnet-group"
}
```



Une data source ne crée rien dans AWS.

Que fait Terraform quand il rencontre un bloc `data` lors d'un `terraform plan` ?

Que se passe-t-il si la ressource référencée n'existe pas ?

5. Module secrets

Objectif : stocker le mot de passe dans AWS Secrets Manager, pas dans le code.



Créer le module de gestion des secrets.

Fichier : `td3/modules/secrets/main.tf`

```
variable "db_password" {
  type      = string
  sensitive = true
}

resource "aws_secretsmanager_secret" "db_credentials" {
  name                    = "td3/database/credentials"
  description             = "Identifiants de la base de donnees TD3"
  recovery_window_in_days = 0
}

resource "aws_secretsmanager_secret_version" "db_credentials" {
  secret_id = aws_secretsmanager_secret.db_credentials.id

  secret_string = jsonencode({
    username = "td3admin"
    password = var.db_password
  })
}

output "secret_arn" {
  description = "ARN du secret a transmettre au module database"
  value       = aws_secretsmanager_secret.db_credentials.arn
}
```



Le secret est stocké au format JSON avec deux champs : `username` et `password`.

Pourquoi utilise-t-on un objet JSON plutôt qu'une simple chaîne de caractères ?

Quel avantage si on doit ajouter le port ou le nom de la base plus tard ?



Le paramètre `recovery_window_in_days = 0` supprime le secret immédiatement lors d'un `terraform destroy`.

Quelle est la valeur par défaut d'AWS pour ce paramètre ?

Pourquoi cette valeur par défaut existe-t-elle en production ?



Ajouter l'appel au module dans le fichier principal.

Fichier : `td3/main.tf` — ajouter après les data sources

```
module "secrets" {
  source      = "./modules/secrets"
  db_password = var.db_initial_password
}
```

6. Probleme volontaire : ordre d'execution



Avant d'aller plus loin, tenter d'écrire à la main un bloc `data` qui lirait le secret qu'on vient de créer :

```
data "aws_secretsmanager_secret_version" "test" {
  secret_id = "td3/database/credentials"
}
```

Ajouter ce bloc temporairement dans `main.tf` et lancer :

```
terraform plan
```



Que se passe-t-il lors du plan ?

Terraform signale-t-il une erreur, et si oui, laquelle ?

Pourquoi Terraform ne peut-il pas garantir que le secret existe au moment où la data source est évaluée ?



Quelle est la différence fondamentale entre un bloc `resource` et un bloc `data` dans le cycle d'exécution Terraform ?

Lequel est évalué lors de la phase de lecture, avant toute création ?



Supprimer ce bloc temporaire avant de continuer.

7. Module database

Objectif : remplacer la ressource RDS du TD2 par une version sécurisée.

La dépendance entre le secret et la base est gérée implicitement via le passage de l'ARN en variable.

Terraform comprend qu'il doit créer le secret avant de lire son contenu car la valeur vient d'un output du module `secrets`.



Créer le module base de données.

Fichier : `td3/modules/database/main.tf`

```
variable "secret_arn" {
  type        = string
  description = "ARN du secret contenant les identifiants de la base"
}

variable "db_subnet_group_name" {
  type        = string
  description = "Nom du subnet group RDS existant"
}

variable "vpc_security_group_ids" {
  type        = list(string)
  description = "Liste des Security Group IDs autorises a acceder a la base"
}

data "aws_secretsmanager_secret_version" "db_credentials" {
  secret_id = var.secret_arn
}

locals {
  db_creds =
jsondecode(data.aws_secretsmanager_secret_version.db_credentials.secret_string)
}

resource "aws_db_instance" "main" {
  identifier        = "td3-database"
  engine            = "postgres"
  engine_version    = "15"
  instance_class    = "db.t3.micro"
  allocated_storage = 20

  db_name  = "td3app"
  username = local.db_creds["username"]
  password = local.db_creds["password"]

  db_subnet_group_name   = var.db_subnet_group_name
  vpc_security_group_ids = var.vpc_security_group_ids

  storage_encrypted   = true
  skip_final_snapshot = true

  tags = {
    Name = "td3-database"
  }
}

output "db_endpoint" {
  description = "Endpoint de connexion a la base de donnees"
```

```
  value       = aws_db_instance.main.endpoint
}
```



Ajouter l'appel au module database dans le fichier principal.

Fichier : `td3/main.tf` — ajouter après le module secrets

```
module "database" {
  source      = "./modules/database"
  secret_arn  = module.secrets.secret_arn

  db_subnet_group_name   = data.aws_db_subnet_group.main.name
  vpc_security_group_ids = [data.aws_security_group.backend.id]
}
```



La valeur `module.secrets.secret_arn` est transmise en variable au module database.

Terraform en déduit-il automatiquement que le module `database` dépend du module `secrets` ?

Dans quel cas faudrait-il ajouter un `depends_on` explicite malgré tout ?



Le paramètre `storage_encrypted = true` a été ajouté.

Que protège exactement ce chiffrement ?

Ce chiffrement protège-t-il les données qui transitent entre le backend et la base de données ?

Quel mécanisme complémentaire protège les données en transit vers RDS ?

8. Module storage

Objectif : créer un bucket S3 avec chiffrement et blocage des accès publics.



Créer le module de stockage.

Fichier : `td3/modules/storage/main.tf`

```
resource "random_id" "bucket_suffix" {
  byte_length = 4
}

resource "aws_s3_bucket" "secure_data" {
```

```
  bucket = "td3-secure-data-${random_id.bucket_suffix.hex}"

  tags = {
    Name = "td3-secure-data"
  }
}

resource "aws_s3_bucket_server_side_encryption_configuration" "secure_data" {
  bucket = aws_s3_bucket.secure_data.id

  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm = "AES256"
    }
    bucket_key_enabled = true
  }
}

resource "aws_s3_bucket_public_access_block" "secure_data" {
  bucket = aws_s3_bucket.secure_data.id

  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}

output "bucket_name" {
  description = "Nom du bucket S3 securise"
  value       = aws_s3_bucket.secure_data.id
}
```



Ajouter l'appel au module storage dans le fichier principal.

Fichier : `td3/main.tf` — ajouter après le module database

```
module "storage" {
  source = "./modules/storage"
}
```



La configuration `apply_server_side_encryption_by_default` est activée.

Que se passe-t-il si un développeur uploade un fichier sans spécifier de chiffrement dans son appel SDK ou CLI ?

La configuration par défaut s'applique-t-elle automatiquement ?



Les quatre paramètres `block_public_acls`, `block_public_policy`, `ignore_public_acls` et



`restrict_public_buckets` sont tous activés.

Les Security Groups du TD2 contrôlent déjà le trafic réseau vers les instances.

Pourquoi bloquer également les accès publics au niveau du bucket S3 ?

S3 est-il soumis aux Security Groups ?

9. Module logging

Objectif : tracer toutes les actions réalisées sur le compte AWS via CloudTrail.

CloudTrail écrit ses logs dans un bucket S3 dédié.

AWS exige une policy bucket explicite pour autoriser CloudTrail à écrire.

Sans cette policy, la ressource `aws_cloudtrail` échoue au déploiement.



Créer le module de logging.

Fichier : `td3/modules/logging/main.tf`

```
data "aws_caller_identity" "current" {}

resource "aws_s3_bucket" "cloudtrail_logs" {
  bucket = "td3-cloudtrail-${data.aws_caller_identity.current.account_id}"

  tags = {
    Name = "td3-cloudtrail-logs"
  }
}

resource "aws_s3_bucket_public_access_block" "cloudtrail_logs" {
  bucket = aws_s3_bucket.cloudtrail_logs.id

  block_public_acls       = true
  block_public_policy     = true
  ignore_public_acls      = true
  restrict_public_buckets = true
}

resource "aws_s3_bucket_policy" "cloudtrail_logs" {
  bucket = aws_s3_bucket.cloudtrail_logs.id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Sid    = "AWSCloudTrailAclCheck"
        Effect = "Allow"
```

```
        Principal = {
          Service = "cloudtrail.amazonaws.com"
        }
        Action   = "s3:GetBucketAcl"
        Resource = aws_s3_bucket.cloudtrail_logs.arn
      },
      {
        Sid    = "AWSCloudTrailWrite"
        Effect = "Allow"
        Principal = {
          Service = "cloudtrail.amazonaws.com"
        }
        Action   = "s3:PutObject"
        Resource =
"${aws_s3_bucket.cloudtrail_logs.arn}/AWSLogs/${data.aws_caller_identity.current.ac
count_id}/*"
        Condition = {
          StringEquals = {
            "s3:x-amz-acl" = "bucket-owner-full-control"
          }
        }
      }
    ]
  })

  depends_on = [aws_s3_bucket_public_access_block.cloudtrail_logs]
}

resource "aws_cloudtrail" "main" {
  name                          = "td3-audit-trail"
  s3_bucket_name                = aws_s3_bucket.cloudtrail_logs.id
  include_global_service_events = true
  is_multi_region_trail         = false
  enable_log_file_validation    = true

  tags = {
    Name = "td3-audit-trail"
  }

  depends_on = [aws_s3_bucket_policy.cloudtrail_logs]
}

output "cloudtrail_name" {
  description = "Nom du trail CloudTrail actif"
  value       = aws_cloudtrail.main.name
}
```



Ajouter l'appel au module logging dans le fichier principal.

Fichier : `td3/main.tf` — ajouter en dernier

```
module "logging" {
```

```
  source = "./modules/logging"
}
```



La policy S3 contient deux instructions distinctes : `AWSCloudTrailAclCheck` et `AWSCloudTrailWrite`.

Pourquoi CloudTrail a-t-il besoin de lire l'ACL du bucket avant d'y écrire ?

Que se serait-il passé si on avait appliqué `aws_cloudtrail` sans cette policy ?



Le paramètre `enable_log_file_validation = true` est activé.

Que génère AWS pour permettre cette validation ?

Comment un administrateur peut-il vérifier qu'un fichier de log n'a pas été modifié ou supprimé ?



CloudTrail enregistre les appels API réalisés sur le compte AWS.

Les modifications de Security Groups réalisées dans le TD2 via Terraform auraient-elles été enregistrées si CloudTrail avait été actif à ce moment ?

Quelle différence entre CloudTrail et VPC Flow Logs ?

10. Fichier outputs global



Créer le fichier d'outputs du projet TD3.

Fichier : `td3/outputs.tf`

```
output "db_endpoint" {
  description = "Endpoint de la base de donnees TD3"
  value       = module.database.db_endpoint
}

output "secure_bucket_name" {
  description = "Nom du bucket S3 securise"
  value       = module.storage.bucket_name
}

output "cloudtrail_name" {
  description = "Nom du trail CloudTrail actif"
  value       = module.logging.cloudtrail_name
}
```

11. Deploiement complet



Initialiser le projet et vérifier le plan avant d'appliquer.

```
cd td3/
terraform init
terraform plan
terraform apply
```

Relever les outputs après application.



Lors du `terraform plan`, combien de ressources sont listées comme a créer ?

Les data sources vers le TD2 apparaissent-elles dans le plan ?

Que se passe-t-il si le VPC `td2-vpc` n'existe pas au moment du plan ?

12. Simulation d'un incident et lecture des logs

Objectif : vérifier que CloudTrail trace bien les actions réalisées sur le compte.



Modifier temporairement un Security Group du TD2 via Terraform.

Par exemple, rouvrir le port 5432 sur `0.0.0.0/0` dans `td2/network/security_groups.tf`.

Appliquer ce changement :

```
cd td2/
terraform apply
```

Attendre deux minutes.

Se connecter à la console AWS et naviguer vers :

```
CloudTrail > Event history
```

Filtrer par nom d'événement : `AuthorizeSecurityGroupIngress`

L'événement est-il visible dans CloudTrail ?

Quelles informations sont disponibles dans le détail de l'événement :

- quel utilisateur ou rôle IAM a réalisé l'action ?
- depuis quelle adresse IP ?
- à quelle heure exactement ?
- quelle ressource a été modifiée ?

Un Security Group du TD2 a été ouvert sur Internet par erreur un vendredi soir.

L'équipe le découvre le lundi matin.

Comment CloudTrail permet-il de répondre aux questions suivantes sans CloudTrail et avec CloudTrail :

- depuis combien de temps la faille est-elle ouverte ?
- qui a fait la modification ?
- s'agit-il d'une erreur ou d'un acte malveillant ?

Remettre le Security Group du TD2 dans son état sécurisé après l'exercice.

```
cd td2/
terraform apply
```

Pourquoi est-il important de conserver les logs CloudTrail pendant au moins un an ?

Quelles obligations légales ou réglementaires peuvent imposer cette durée en France et en Europe ?

13. Vérification globale

Vérifier les points suivants avant de passer au challenge :

- aucun mot de passe n'apparait dans les fichiers `.tf`
- le fichier `terraform.tfvars` est présent dans `.gitignore`
- le secret est visible dans la console AWS Secrets Manager sous le nom `td3/database/credentials`
- la ressource `aws_db_instance` dans le module database contient `storage_encrypted = true`



- le bucket S3 applicatif a le chiffrement AES256 actif
- le bucket S3 applicatif a les quatre blocages d'accès public actifs
- CloudTrail est actif dans la console AWS
- la validation des fichiers de log est activée sur le trail

Challenge final

Axe secrets



Où est physiquement stocké le mot de passe de la base de données maintenant ?

Comparer avec la situation du TD2 : qui pouvait y accéder avant, qui peut y accéder maintenant ?

Comment restreindre l'accès au secret uniquement au rôle IAM du backend, en vous appuyant sur ce que vous avez appris au TD1 ?

Axe chiffrement



Le paramètre `storage_encrypted = true` sur RDS protège-t-il les données pendant leur transfert entre le backend et la base ?

Quel mécanisme complémentaire faut-il activer pour protéger les données en transit ?

Dans quel cas le chiffrement AES256 de S3 ne suffit-il pas a garantir la confidentialite des données ?

Axe traçabilité



Un stagiaire supprime accidentellement la ressource CloudTrail via `terraform destroy` sur le module logging.

Quelles sont les conséquences immédiates pour la traçabilité du compte ?

Comment aurait-on pu protéger cette ressource contre une suppression accidentelle ?

Chercher le paramètre Terraform qui permet de protéger une ressource contre la destruction.

Axe architecture



Produire un schéma de l'architecture complète en prolongement du schéma réalisé au TD2.

Le schéma doit inclure :

- les composants du TD2 : ALB, backend, Security Groups, subnets
- les ajouts du TD3 : Secrets Manager, RDS chiffré, S3 applicatif, CloudTrail et son bucket



- les flux de données entre chaque composant avec les ports concernés
- une légende distinguant les flux chiffrés et les flux non chiffrés

Bonus

Rotation automatique des secrets



Rechercher dans la documentation AWS comment activer la rotation automatique d'un secret Secrets Manager pour une base PostgreSQL.

Identifier l'ARN de la Lambda de rotation fournie par AWS pour PostgreSQL dans la région `eu-west-3`.

Ajouter la rotation dans le module secrets.

Fichier : `td3/modules/secrets/main.tf` — ajouter après la ressource `aws_secretsmanager_secret_version`

```
variable "rotation_lambda_arn" {
  type        = string
  description = "ARN de la Lambda de rotation fournie par AWS pour PostgreSQL"
  default     = ""
}

resource "aws_secretsmanager_secret_rotation" "db_credentials" {
  count               = var.rotation_lambda_arn != "" ? 1 : 0
  secret_id           = aws_secretsmanager_secret.db_credentials.id
  rotation_lambda_arn = var.rotation_lambda_arn

  rotation_rules {
    automatically_after_days = 30
  }
}
```



Le bloc `count = var.rotation_lambda_arn != “” ? 1 : 0` est utilisé ici.

Que signifie ce pattern dans Terraform ?

Pourquoi est-il utile de rendre la rotation optionnelle plutôt que obligatoire dans ce module ?



Après une rotation automatique du secret, l'application backend doit récupérer le nouveau mot de passe.

Si l'application a mis le mot de passe en cache au démarrage, que se passe-t-il après une rotation ?

Comment une application bien conçue doit-elle récupérer les secrets pour éviter ce problème ?

From:
http://slamwiki2.kobject.net/ - **SlamWiki 2.1**

Permanent link:
http://slamwiki2.kobject.net/eadl/bloc4/fm4/td3?rev=1782280447

Last update: **2026/06/24 07:54**