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
- Créer et utiliser une clé KMS Customer Managed Key
- Activer le chiffrement des données au repos sur RDS et S3 avec KMS
- 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.

Quatre 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
- le bucket S3 applicatif n'est pas chiffré
- aucune trace des actions réalisées sur le compte AWS n'est conservée

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

Votre mission est de corriger ces 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, KMS, 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 quatre nouveaux modules.

Fichier : `td3/structure.txt`

```
td3/
├── main.tf
```

```
├── variables.tf
├── outputs.tf
└── modules/
    ├── secrets/
    │   └── main.tf
    ├── kms/
    │   └── main.tf
    ├── storage/
    │   └── main.tf
    └── logging/
        └── main.tf
```

Le fichier `td3/main.tf` orchestre les quatre modules et récupère les ressources du TD2 via des data sources.

1. Analyse de l'existant

Lire le fichier `td2/network/instances.tf` avant toute manipulation.

Localiser la ligne où le mot de passe de la base de données est défini.

Répondre aux questions suivantes :

- Qui peut lire ce mot de passe concrètement ?
- Ce fichier est-il souvent versionné dans Git en entreprise ?
- Quelles sont les conséquences si ce dépôt est public ou partagé avec un prestataire ?

Dans `td2/network/instances.tf`, la ressource `aws_db_instance` ne contient pas le paramètre `storage_encrypted`.

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

Comment vérifier l'état du chiffrement d'une instance RDS existante dans la console AWS ?

Aucun service de logging n'est présent dans le TD2.

Citer deux situations concrètes en production où l'absence de logs rend la gestion d'incident impossible.

2. Récupération des ressources du TD2

Le TD3 ne redéploie pas les ressources du TD2.

Il les récupère via des data sources Terraform pour les réutiliser.

Fichier : `td3/main.tf`

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

```
    }
    random = {
      source  = "hashicorp/random"
      version = "~> 3.0"
    }
  }
}

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

# Recuperation du compte AWS courant
data "aws_caller_identity" "current" {}

# Recuperation de la region courante
data "aws_region" "current" {}

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

# Recuperation du Security Group backend du TD2
data "aws_security_group" "backend_sg" {
  filter {
    name   = "tag:Name"
    values = ["backend-sg"]
  }
}

module "secrets" {
  source = "./modules/secrets"
}

module "kms" {
  source     = "./modules/kms"
  account_id = data.aws_caller_identity.current.account_id
  region     = data.aws_region.current.name
}

module "storage" {
  source  = "./modules/storage"
  kms_key_arn = module.kms.key_arn
}

module "logging" {
  source     = "./modules/logging"
  account_id = data.aws_caller_identity.current.account_id
  kms_key_arn = module.kms.key_arn
}
```

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

Que se passerait-il si on redéclarait `aws_vpc` avec le même CIDR dans un nouveau fichier Terraform ?

3. Gestion des secrets avec Secrets Manager

Objectif : supprimer le mot de passe en clair de `instances.tf`.



Créer le module secrets.

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

```
variable "db_username" {
  type        = string
  description = "Nom d'utilisateur de la base de données"
  default     = "admin"
}

variable "db_password" {
  type        = string
  description = "Mot de passe de la base de données"
  sensitive   = true
}

resource "aws_secretsmanager_secret" "db_credentials" {
  name        = "td3/db/credentials"
  description = "Identifiants de la base de donnees RDS TD3"
}

resource "aws_secretsmanager_secret_version" "db_credentials" {
  secret_id = aws_secretsmanager_secret.db_credentials.id
  secret_string = jsonencode({
    username = var.db_username
    password = var.db_password
  })
}

output "secret_arn" {
  value       = aws_secretsmanager_secret.db_credentials.arn
  description = "ARN du secret contenant les identifiants RDS"
}
```

Fichier : `td3/variables.tf`

```
variable "db_password" {
  type        = string
  description = "Mot de passe initial de la base de donnees"
  sensitive   = true
}
```

Le mot de passe est transmis via une variable sensible, jamais écrit en dur.

Pour appliquer, Terraform demandera la valeur au moment de l'exécution :

Fichier : `td3/commandes/terraform.txt`

```
# Appliquer en fournissant le mot de passe de maniere interactive
terraform apply -var="db_password=MonMotDePasse123!"

# Ou via une variable d'environnement sans valeur dans le code
export TF_VAR_db_password="MonMotDePasse123!"
terraform apply
```

Pourquoi utilise-t-on `sensitive = true` sur la variable `db_password` ?

Que change ce paramètre dans l'affichage Terraform ?

Est-ce suffisant pour garantir que le mot de passe ne sera jamais visible ?

Pourquoi utilise-t-on un JSON pour stocker le secret plutôt qu'une simple chaîne ?

Quel avantage cela apporte-t-il si l'application doit récupérer le secret programmatiquement ?

4. Problème volontaire – dépendance entre modules



Tenter d'appliquer uniquement le module secrets :

```
terraform apply -target=module.secrets
```

Puis observer ce qui se passe si on tente de lire le secret immédiatement dans le module storage sans `depends_on`.

Terraform gère les dépendances implicites entre ressources du même fichier.

Pourquoi cette dépendance implicite ne fonctionne-t-elle pas toujours entre modules différents ?

Dans quels cas faut-il utiliser `depends_on` explicitement ?

5. Création d'une clé KMS

Objectif : créer une clé de chiffrement gérée par le client (CMK).

AWS propose deux types de clés KMS :

- les clés AWS managées (aws/rds, aws/s3...)
- les clés Customer Managed Keys (CMK)

Quelle est la différence concrète entre les deux en termes de contrôle ?

Qui peut utiliser une clé AWS managée ?

Qui décide des droits d'accès sur une CMK ?



Créer le module KMS.

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

```
variable "account_id" {
  type        = string
  description = "ID du compte AWS courant"
}

variable "region" {
  type        = string
  description = "Region AWS courante"
}

resource "aws_kms_key" "td3" {
  description             = "Cle KMS TD3 — chiffrement RDS et S3"
  deletion_window_in_days = 7
  enable_key_rotation     = true

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Sid    = "Acces administrateur au compte"
        Effect = "Allow"
        Principal = {
          AWS = "arn:aws:iam::${var.account_id}:root"
        }
        Action   = "kms:*"
        Resource = "*"
      }
    ]
  })
}

resource "aws_kms_alias" "td3" {
  name          = "alias/td3-key"
  target_key_id = aws_kms_key.td3.id
}

output "key_arn" {
  value       = aws_kms_key.td3.arn
  description = "ARN de la cle KMS TD3"
}

output "key_id" {
  value       = aws_kms_key.td3.id
```

```
  description = "ID de la cle KMS TD3"
}
```

Le paramètre `deletion_window_in_days = 7` est défini sur la clé KMS.

Que se passe-t-il si on supprime cette clé alors que des données RDS ou S3 sont chiffrées avec elle ?

Pourquoi AWS impose-t-il un délai minimum avant la suppression effective d'une clé ?

Le paramètre `enable_key_rotation = true` est activé.

Que signifie la rotation d'une clé KMS concrètement ?

La rotation de la clé KMS invalide-t-elle les données déjà chiffrées avec l'ancienne version ?

La key policy autorise `kms:*` pour le root du compte.

Pourquoi cette règle est-elle nécessaire dans une key policy ?

Que se passerait-il si on supprimait cette règle et qu'aucun autre principal n'était autorisé ?

6. Chiffrement de la base de données RDS

Objectif : activer le chiffrement au repos sur RDS avec la CMK.



Modifier `td2/network/instances.tf` pour activer le chiffrement et utiliser le secret Secrets Manager.

Fichier : `td2/network/instances.tf`

```
# Recuperation du secret cree dans le module TD3
data "aws_secretsmanager_secret_version" "db_credentials" {
  secret_id = "td3/db/credentials"

  depends_on = [
    aws_secretsmanager_secret_version.db_credentials
  ]
}

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

resource "aws_instance" "backend" {
  ami                    = "ami-0f61de2873e29e866"
  instance_type          = "t2.micro"
  subnet_id              = aws_subnet.private.id
```

```
  vpc_security_group_ids = [aws_security_group.backend_sg.id]

  user_data = <<-EOF
    #!/bin/bash
    yum install -y python3
    python3 -m http.server 80 &
  EOF

  tags = {
    Name = "td2-backend"
  }
}

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

  username = local.db_secret.username
  password = local.db_secret.password

  storage_encrypted = true
  kms_key_id        = var.kms_key_arn

  db_subnet_group_name   = aws_db_subnet_group.main.name
  vpc_security_group_ids = [aws_security_group.db_sg.id]
  skip_final_snapshot    = true

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

resource "aws_db_subnet_group" "main" {
  name       = "td3-db-subnet-group"
  subnet_ids = [aws_subnet.private.id, aws_subnet.public_b.id]

  tags = {
    Name = "td3-db-subnet-group"
  }
}

variable "kms_key_arn" {
  type        = string
  description = "ARN de la cle KMS utilisee pour le chiffrement RDS"
}
```

Le chiffrement `storage_encrypted = true` protège les données au repos.

Quel mécanisme protège les données en transit entre le backend et RDS ?

Le chiffrement au repos protège-t-il contre un accès non autorisé via SQL si les credentials sont compromis ?

L'identifiant de la base est passé de `td2-db` à `td3-db`.

Que se passe-t-il si on tente de modifier `storage_encrypted` sur une instance RDS existante ?

Pourquoi AWS impose-t-il cette contrainte ?

7. Problème volontaire – chiffrement RDS immuable



Tenter de modifier une instance RDS existante non chiffrée pour activer le chiffrement :

```
terraform apply
```

Observer le message d'erreur Terraform.

Terraform indique qu'il doit détruire et recréer l'instance RDS.

Pourquoi le chiffrement d'une instance RDS ne peut-il pas être activé à chaud ?

Quelles précautions prendre en production avant de forcer cette opération ?

8. Création d'un bucket S3 sécurisé

Objectif : créer un bucket S3 avec chiffrement SSE-KMS et blocage d'accès public.



Créer le module storage.

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

```
variable "kms_key_arn" {
  type        = string
  description = "ARN de la cle KMS utilisee pour le chiffrement S3"
}

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

resource "aws_s3_bucket" "app_data" {
  bucket = "td3-app-data-${random_id.bucket_suffix.hex}"

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

resource "aws_s3_bucket_public_access_block" "app_data" {
```

```
  bucket = aws_s3_bucket.app_data.id

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

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

  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm     = "aws:kms"
      kms_master_key_id = var.kms_key_arn
    }
    bucket_key_enabled = true
  }
}

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

Dans ce TD la valeur de `sse_algorithm` est `aws:kms`.

Dans le TD précédent la valeur était `AES256`.

Quelle est la différence concrète entre ces deux modes de chiffrement S3 ?

Quel mode offre le plus de contrôle sur la clé de chiffrement ?

Le paramètre `bucket_key_enabled = true` est activé.

Rechercher ce que fait ce paramètre.

Quel impact a-t-il sur les coûts liés à KMS ?

Les quatre paramètres de `aws_s3_bucket_public_access_block` sont tous à `true`.

Expliquer ce que bloque concrètement chacun des quatre paramètres.

Pourquoi ce bloc est-il indépendant du chiffrement ?

9. Mise en place du logging avec CloudTrail

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



Créer le module logging.

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

```
variable "account_id" {
  type        = string
  description = "ID du compte AWS courant"
}

variable "kms_key_arn" {
  type        = string
  description = "ARN de la cle KMS utilisee pour le chiffrement des logs"
}

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

resource "aws_s3_bucket" "cloudtrail_logs" {
  bucket = "td3-cloudtrail-${var.account_id}-${random_id.bucket_suffix.hex}"

  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 =
"${aws_s3_bucket.cloudtrail_logs.arn}/AWSLogs/${var.account_id}/*"
        Condition = {
```

```
          StringEquals = {
            "s3:x-amz-acl" = "bucket-owner-full-control"
          }
        }
      }
    ]
  })
}

resource "aws_cloudtrail" "main" {
  name                          = "td3-cloudtrail"
  s3_bucket_name                = aws_s3_bucket.cloudtrail_logs.id
  include_global_service_events = true
  is_multi_region_trail         = false
  enable_log_file_validation    = true
  kms_key_id                    = var.kms_key_arn

  depends_on = [
    aws_s3_bucket_policy.cloudtrail_logs
  ]
}

output "cloudtrail_arn" {
  value       = aws_cloudtrail.main.arn
  description = "ARN du trail CloudTrail"
}
```

La bucket policy pour CloudTrail contient deux statements distincts.

Expliquer le rôle de chacun :

- `AWSCloudTrailAclCheck`
- `AWSCloudTrailWrite`

Que se passerait-il si cette policy était absente ?

Le paramètre `enable_log_file_validation = true` est activé.

Que permet concrètement la validation des fichiers de log ?

Dans quel scénario cette fonctionnalité est-elle particulièrement utile ?

CloudTrail enregistre les appels API réalisés sur le compte AWS.

Cocher parmi ces actions celles qui sont tracées par CloudTrail :

- création d'un Security Group
- modification d'une règle IAM
- upload d'un fichier dans S3
- requête SQL vers RDS
- connexion SSH vers une instance EC2
- suppression d'un secret Secrets Manager

Justifier les réponses non évidentes.

10. Simulation d'un incident

Objectif : utiliser CloudTrail pour retracer une action suspecte.



Réaliser une action volontairement suspecte dans AWS.

Modifier un Security Group existant du TD2 pour ouvrir temporairement le port 22 depuis `0.0.0.0/0` :

```
terraform apply -target=module.network
```

Attendre deux minutes, puis supprimer cette règle :

```
terraform apply
```



Aller dans la console AWS.

Naviguer vers : CloudTrail > Event history.

Rechercher l'événement `AuthorizeSecurityGroupIngress`.

Identifier :

- l'heure de l'action
- l'identité qui a réalisé l'action
- les paramètres de la règle ajoutée

CloudTrail enregistre l'identité qui a réalisé chaque action.

Si plusieurs développeurs partagent le même utilisateur IAM, que perd-on en termes de traçabilité ?

Quel principe de gestion des identités du TD1 est directement lié à cette problématique ?

Les logs CloudTrail sont stockés dans S3.

Par défaut, combien de temps sont-ils conservés dans CloudTrail Event History ?

Pourquoi stocker les logs dans S3 permet-il une conservation plus longue ?

11. Vérification globale



Vérifier les points suivants dans le code Terraform et dans la console AWS :

- le mot de passe n'apparaît plus dans aucun fichier `.tf`
- la variable `db_password` est marquée `sensitive = true`
- la base RDS a `storage_encrypted = true` et un `kms_key_id` renseigné
- le bucket S3 applicatif a `sse_algorithm = “aws:kms”`



- le bucket S3 applicatif a les quatre paramètres `public_access_block` à `true`
- CloudTrail est actif et écrit dans son bucket dédié
- la clé KMS a `enable_key_rotation = true`

Challenge final

Axe secrets



Répondre aux questions suivantes par écrit :

- Où est stocké physiquement le secret de la base après le TD3 ?
- Qui peut lire ce secret dans AWS ?
- Que se passe-t-il si le secret Secrets Manager est lui-même chiffré avec la clé KMS TD3 et que cette clé est supprimée ?
- Quelles seraient les trois premières actions à réaliser si le mot de passe de la base était compromis ?

Axe KMS

La clé KMS est utilisée à la fois pour RDS, S3 et CloudTrail.

Quels sont les avantages de centraliser sur une seule clé ?

Quels sont les risques de cette approche ?

Comment organiserait-on les clés KMS dans un environnement de production avec plusieurs équipes et plusieurs applications ?

Axe chiffrement

Dresser un tableau récapitulatif avec les colonnes suivantes :

- Ressource (RDS, S3, Secrets Manager, CloudTrail)
- Type de chiffrement (au repos, en transit, les deux)
- Algorithme ou mécanisme utilisé
- Clé utilisée (AWS managée ou CMK TD3)

Dans quel cas le chiffrement AES256 de S3 ne suffit-il pas à garantir la confidentialité 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 ?

Rechercher 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, KMS, 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 qu'obligatoire dans ce module ?

Après une rotation automatique, 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=1782281412

Last update: **2026/06/24 08:10**