Guide Complet : Sécurisation de l'Hébergement AWS pour Applications SaaS

Version: 1.0
Date: Novembre 2025
Destiné à: Équipes DevOps et Ingénieurs Cloud

Table des Matières

Sécurité EC2

1. IMDSv2 (Instance Metadata Service v2)

1.1 Pourquoi IMDSv2 est Critique

IMDSv2 fournit une **protection renforcée contre l'exploitation** à travers une authentification orientée session, nécessitant un token de session pour les requêtes de métadonnées et limitant la durée de session.

Différences clés:

Caractéristique	IMDSv1	IMDSv2
Authentication	Aucune	Token requis (PUT)
Protection SSRF	❌ Non	✅ Oui
Hop Limit	Illimité	Configurable (1-64)
TTL Restriction	Non	Oui (1 hop par défaut)

1.2 Activer IMDSv2 sur Instances Existantes

```
# Forcer IMDSv2 sur toutes les instances
aws ec2 modify-instance-metadata-options \
    --instance-id i-1234567890abcdef0 \
    --http-tokens required \
    --http-put-response-hop-limit 1

# Vérifier la configuration
aws ec2 describe-instances \
    --instance-ids i-1234567890abcdef0 \
    --query 'Reservations[].Instances[].[InstanceId,MetadataOptions.HttpTokens]' \
    --output table
```

1.3 Appliquer IMDSv2 par Défaut avec Launch Template

```
{
  "LaunchTemplateName": "secure-ec2-template",
  "LaunchTemplateData": {
    "MetadataOptions": {
      "HttpTokens": "required",
      "HttpPutResponseHopLimit": 1,
      "HttpEndpoint": "enabled"
    },
    "InstanceType": "t3.medium",
    "SecurityGroupIds": ["sg-xxxxx"],
    "IamInstanceProfile": {
      "Arn": "arn:aws:iam::123456789012:instance-profile/MyInstanceProfile"
    }
  }
}
```

1.4 Utiliser IMDSv2 depuis une Application

```
import requests

# IMDSv1 (NON SÉCURISÉ)
response = requests.get('http://169.254.169.254/latest/meta-data/iam/security-credentials/MyRo
```

```
# IMDSv2 (SÉCURISÉ)
# Étape 1: Obtenir un token
token_response = requests.put(
    'http://169.254.169.254/latest/api/token',
    headers={'X-aws-ec2-metadata-token-ttl-seconds': '21600'}
)
token = token_response.text

# Étape 2: Utiliser le token
response = requests.get(
    'http://169.254.169.254/latest/meta-data/iam/security-credentials/MyRole',
    headers={'X-aws-ec2-metadata-token': token}
)
```

2. Chiffrement EBS

2.1 Activer le Chiffrement par Défaut

```
# Activer le chiffrement EBS par défaut pour la région
aws ec2 enable-ebs-encryption-by-default --region us-east-1

# Vérifier le statut
aws ec2 get-ebs-encryption-by-default --region us-east-1
```

Important: Cette configuration s'applique uniquement aux **nouveaux volumes**. Les volumes existants doivent être migrés.

2.2 Chiffrer un Volume Existant

```
# 1. Créer un snapshot du volume
aws ec2 create-snapshot \
    --volume-id vol-xxxxx \
    --description "Snapshot before encryption"

# 2. Copier le snapshot avec chiffrement
aws ec2 copy-snapshot \
    --source-region us-east-1 \
    --source-snapshot-id snap-xxxxx \
    --destination-region us-east-1 \
    --encrypted \
    --kms-key-id arn:aws:kms:us-east-1:123456789012:key/xxxxx

# 3. Créer un nouveau volume chiffré depuis le snapshot
aws ec2 create-volume \
    --snapshot-id snap-yyyyy \
    --availability-zone us-east-1a \
    --encrypted \
    --kms-key-id arn:aws:kms:us-east-1:123456789012:key/xxxxx
```

```
# 4. Attacher le nouveau volume à l'instance
aws ec2 attach-volume \
    --volume-id vol-yyyyy \
    --instance-id i-xxxxx \
    --device /dev/sdf
```

2.3 Politique AWS Config pour Conformité

```
# Règle AWS Config: Vérifier que tous les volumes EBS sont chiffrés
Resources:
  EBSEncryptionRule:
    Type: AWS::Config::ConfigRule
    Properties:
      ConfigRuleName: encrypted-volumes
      Source:
        Owner: AWS
        SourceIdentifier: ENCRYPTED_VOLUMES
      Scope:
        ComplianceResourceTypes:
          - AWS::EC2::Volume
```

3. Security Groups et Isolation

3.1 Pas d'Adresses IP Publiques

✅ Architecture Recommandée:

```
┌───────────────────────────────────────────────────────┐
│                 Application Load Balancer             │
│               (Public Subnet avec IP publique)        │
└──────────────────────────────────────────────────────┘
                     │
                     │
          ┌──────────┴───────────┐
          │                      │
┌─────────▼───────┐      ┌───────▼─────────┐
│  EC2 Instance   │      │  EC2 Instance   │
│ (Private Subnet│      │ (Private Subnet│
│  Pas d'IP pub.)│      │  Pas d'IP pub.)│
└─────────────────┘      └─────────────────┘
```

```
# Vérifier les instances avec IP publiques
aws ec2 describe-instances \
    --filters "Name=instance-state-name,Values=running" \
    --query 'Reservations[].Instances[?PublicIpAddress!=`null`].[InstanceId,PublicIpAddress,Ta
    --output table
```

❌ Instances EC2 avec IP publique = Surface d'attaque accrue

3.2 Principe du Moindre Privilège - Security Groups

```
# Terraform - Security Group pour instances d'application
resource "aws_security_group" "app_instances" {
  name        = "app-instances-sg"
  description = "Security group for application instances"
  vpc_id      = aws_vpc.main.id

  # Autoriser uniquement le trafic depuis le load balancer
  ingress {
    description     = "HTTP from ALB"
    from_port       = 8080
    to_port         = 8080
    protocol        = "tcp"
    security_groups = [aws_security_group.alb.id]
  }

  # Autoriser le trafic sortant vers Internet (via NAT Gateway)
  egress {
    description = "Allow all outbound"
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "app-instances-sg"
  }
}
```

4. Gestion des Clés SSH

4.1 AWS Systems Manager Session Manager (Recommandé)

Avantages:
- ✅ Aucun port SSH ouvert (port 22)
- ✅ Accès audité via CloudTrail
- ✅ Pas besoin de gérer des clés SSH
- ✅ Accès basé sur IAM

```
# Se connecter à une instance via Session Manager
aws ssm start-session --target i-1234567890abcdef0

# Transférer un port local (ex: pour accéder à une base de données)
aws ssm start-session \
    --target i-1234567890abcdef0 \
    --document-name AWS-StartPortForwardingSession \
    --parameters "portNumber=3306,localPortNumber=3306"
```

4.2 EC2 Instance Connect (Alternative)

```
# Envoyer une clé SSH publique temporaire (60 secondes)
aws ec2-instance-connect send-ssh-public-key \
    --instance-id i-1234567890abcdef0 \
    --availability-zone us-east-1a \
    --instance-os-user ec2-user \
    --ssh-public-key file://my-key.pub

# Se connecter immédiatement
ssh ec2-user@ec2-xxx-xxx-xxx-xxx.compute-1.amazonaws.com
```

Sécurité Serverless (Lambda)

1. Configuration VPC pour Lambda

1.1 Quand utiliser un VPC pour Lambda ?

Cas d'Usage	VPC Requis ?
Accès RDS dans VPC privé	✅ Oui
Accès ElastiCache	✅ Oui
Accès services AWS publics (S3, DynamoDB)	❌ Non (utiliser VPC Endpoints)
Appels API externes (Internet)	❌ Non

1.2 Configuration VPC avec Interface Endpoints

```
# CloudFormation - Lambda dans VPC avec accès S3 privé
Resources:
  LambdaFunction:
    Type: AWS::Lambda::Function
    Properties:
      FunctionName: secure-lambda-function
      Runtime: python3.11
      VpcConfig:
        SecurityGroupIds:
          - !Ref LambdaSecurityGroup
        SubnetIds:
          - !Ref PrivateSubnet1
          - !Ref PrivateSubnet2

  # VPC Endpoint pour S3 (pas besoin NAT Gateway)
  S3VPCEndpoint:
```

```
    Type: AWS::EC2::VPCEndpoint
    Properties:
      VpcId: !Ref VPC
      ServiceName: !Sub com.amazonaws.${AWS::Region}.s3
      RouteTableIds:
        - !Ref PrivateRouteTable
```

Important: Lambda dans un VPC perd l'accès Internet par défaut. Utilisez des VPC Endpoints pour les services AWS ou un NAT Gateway pour Internet.

2. Gestion des Secrets

2.1 ❌ JAMAIS faire ceci:

```
import os

# DANGEREUX: Secrets en dur dans le code
DB_PASSWORD = "MyP@ssw0rd123"
API_KEY = "sk-1234567890abcdef"

# DANGEREUX: Secrets en variables d'environnement (visibles en clair)
DB_PASSWORD = os.environ['DB_PASSWORD']  # Visible dans la console Lambda
```

2.2 ✅ Bonne Pratique: AWS Secrets Manager

```
import boto3
import json
from botocore.exceptions import ClientError

# Solution 1: Récupérer durant le init (une fois par cold start)
secrets_client = boto3.client('secretsmanager')

try:
    response = secrets_client.get_secret_value(SecretId='prod/myapp/database')
    secret = json.loads(response['SecretString'])
    DB_HOST = secret['host']
    DB_PASSWORD = secret['password']
except ClientError as e:
    raise e

def lambda_handler(event, context):
    # Utiliser DB_HOST et DB_PASSWORD
    pass
```

2.3 Optimisation: Extension Lambda pour Secrets Manager

```
# L'extension Lambda cache les secrets et rafraîchit automatiquement
import os
import urllib.request
```

```
import json

def get_secret(secret_name):
    """Récupérer secret via l'extension Lambda (avec cache)"""
    secrets_extension_endpoint = f"http://localhost:2773/secretsmanager/get?secretId={secret_n
    headers = {"X-Aws-Parameters-Secrets-Token": os.environ['AWS_SESSION_TOKEN']}

    req = urllib.request.Request(secrets_extension_endpoint, headers=headers)
    response = urllib.request.urlopen(req)
    secret = json.loads(response.read())

    return json.loads(secret['SecretString'])

# Récupération avec cache
db_creds = get_secret('prod/myapp/database')
```

Avantages de l'Extension:
- ✅ Cache local des secrets
- ✅ Rafraîchissement automatique
- ✅ Réduction des appels API (coût)
- ✅ Latence < 10ms

3. Principe du Moindre Privilège - IAM

3.1 Une Fonction = Un Rôle IAM

❌ **Mauvaise Pratique:**

```
# Rôle partagé par toutes les Lambda
LambdaExecutionRole:
  Type: AWS::IAM::Role
  Properties:
    Policies:
      - PolicyDocument:
          Statement:
            - Effect: Allow
              Action: "*"
              Resource: "*"
```

✅ **Bonne Pratique:**

```
# Rôle dédié avec permissions minimales
ProcessOrderLambdaRole:
  Type: AWS::IAM::Role
  Properties:
    AssumeRolePolicyDocument:
      Statement:
        - Effect: Allow
          Principal:
```

```
          Service: lambda.amazonaws.com
        Action: sts:AssumeRole
  ManagedPolicyArns:
    - arn:aws:iam::aws:policy/service-role/AWSLambdaVPCAccessExecutionRole
  Policies:
    - PolicyName: OrderProcessingPolicy
      PolicyDocument:
        Statement:
          - Effect: Allow
            Action:
              - dynamodb:GetItem
              - dynamodb:PutItem
              - dynamodb:Query
            Resource: !GetAtt OrdersTable.Arn
          - Effect: Allow
            Action:
              - sqs:SendMessage
            Resource: !GetAtt OrderQueue.Arn
```

4. Sécurité du Code Lambda

4.1 Validation des Entrées

```
import json
from jsonschema import validate, ValidationError

# Schéma pour valider les événements
ORDER_SCHEMA = {
    "type": "object",
    "properties": {
        "orderId": {"type": "string", "pattern": "^ORD-[0-9]{10}$"},
        "amount": {"type": "number", "minimum": 0, "maximum": 100000},
        "email": {"type": "string", "format": "email"}
    },
    "required": ["orderId", "amount", "email"]
}

def lambda_handler(event, context):
    try:
        # Valider l'événement
        validate(instance=event, schema=ORDER_SCHEMA)
    except ValidationError as e:
        return {
            'statusCode': 400,
            'body': json.dumps({'error': f'Invalid input: {e.message}'})
        }

    # Traiter l'événement validé
    order_id = event['orderId']
    # ...
```

4.2 Ne Jamais Logger des Informations Sensibles

❌ **Dangereux:**

```
import logging
logger = logging.getLogger()

def lambda_handler(event, context):
    # DANGEREUX: Log l'événement complet (peut contenir des secrets)
    logger.info(f"Processing event: {event}")

    # DANGEREUX: Log des données sensibles
    logger.info(f"User password: {event['password']}")
```

✅ **Sécurisé:**

```
import logging
logger = logging.getLogger()

def lambda_handler(event, context):
    # Log uniquement les champs nécessaires
    logger.info(f"Processing order: {event.get('orderId')}")

    # Sanitize les logs
    safe_event = {k: v for k, v in event.items() if k not in ['password', 'apiKey', 'token']}
    logger.debug(f"Event details: {safe_event}")
```

5. Chiffrement et Protection des Données

```
# Chiffrer les variables d'environnement avec KMS
Resources:
  MyLambda:
    Type: AWS::Lambda::Function
    Properties:
      KmsKeyArn: !GetAtt LambdaKMSKey.Arn
      Environment:
        Variables:
          DB_HOST: encrypted-value  # Chiffré au repos avec KMS
```

Sécurité Containers (ECS/EKS)

1. Scan d'Images avec Amazon ECR

1.1 Activer le Scan Automatique

```
# Activer le scan automatique au push
aws ecr put-image-scanning-configuration \
    --repository-name my-app \
    --image-scanning-configuration scanOnPush=true

# Utiliser le scan amélioré (Enhanced Scanning avec Inspector)
aws ecr put-registry-scanning-configuration \
    --scan-type ENHANCED \
    --rules '[{"repositoryFilters":[{"filter":"*","filterType":"WILDCARD"}],"scanFrequency":"S
```

1.2 Analyser les Résultats de Scan

```
# Obtenir les résultats de scan pour une image
aws ecr describe-image-scan-findings \
    --repository-name my-app \
    --image-id imageTag=v1.2.3 \
    --query 'imageScanFindings.findings[?severity==`CRITICAL` || severity==`HIGH`]'
```

1.3 Bloquer les Images Vulnérables dans la CI/CD

```
#!/bin/bash
# Pipeline script pour bloquer les déploiements d'images vulnérables

REPO_NAME="my-app"
IMAGE_TAG="$CI_COMMIT_SHA"

# Attendre que le scan soit terminé
aws ecr wait image-scan-complete \
    --repository-name $REPO_NAME \
    --image-id imageTag=$IMAGE_TAG

# Récupérer les vulnérabilités critiques
CRITICAL_COUNT=$(aws ecr describe-image-scan-findings \
    --repository-name $REPO_NAME \
    --image-id imageTag=$IMAGE_TAG \
    --query 'length(imageScanFindings.findings[?severity==`CRITICAL`])')

if [ "$CRITICAL_COUNT" -gt 0 ]; then
    echo "ERROR: $CRITICAL_COUNT critical vulnerabilities found. Blocking deployment."
    exit 1
fi
```

```
echo "Image scan passed. Proceeding with deployment."
```

2. Images Distroless et Minimales

2.1 Pourquoi les Images Distroless ?

Images traditionnelles:
- Shell, package managers, outils de debug
- Surface d'attaque large
- Bruit dans les scans de vulnérabilités

Images distroless:
- Uniquement l'application + runtime
- Pas de shell, pas de package manager
- 🔒 Surface d'attaque minimale

2.2 Exemple Dockerfile Distroless

```
# Build stage
FROM python:3.11-slim AS builder

WORKDIR /app
COPY requirements.txt .
RUN pip install --user --no-cache-dir -r requirements.txt

COPY . .

# Production stage - Distroless
FROM gcr.io/distroless/python3-debian11

# Copier seulement les dépendances et l'application
COPY --from=builder /root/.local /root/.local
COPY --from=builder /app /app

WORKDIR /app

# Pas de shell disponible !
# USER nonroot

CMD ["main.py"]
```

3. Sécurité Runtime avec Amazon Inspector

Amazon Inspector surveille en continu les images ECR en cours d'exécution sur les containers ECS et EKS.

```
# Activer Inspector pour containers
aws inspector2 enable \
    --resource-types ECR ECS

# Voir les vulnérabilités actives
aws inspector2 list-findings \
    --filter-criteria '{
      "resourceType": [{"comparison": "EQUALS", "value": "AWS_ECR_CONTAINER_IMAGE"}],
      "findingStatus": [{"comparison": "EQUALS", "value": "ACTIVE"}]
    }'
```

Informations fournies par Inspector:
- `lastInUseAt`: Dernière fois que l'image était active
- `InUseCount`: Nombre de pods EKS / tasks ECS utilisant l'image
- Mapping: Image ECR → Containers en cours d'exécution

4. Ne Pas Exécuter en Mode Privilégié

4.1 ECS Task Definition

```
{
  "family": "my-secure-task",
  "containerDefinitions": [
    {
      "name": "app",
      "image": "my-app:latest",
      "privileged": false,
      "readonlyRootFilesystem": true,
      "user": "1000:1000",
      "linuxParameters": {
        "capabilities": {
          "drop": ["ALL"]
        }
      }
    }
  ]
}
```

4.2 Kubernetes Pod Security

```
apiVersion: v1
kind: Pod
metadata:
  name: secure-pod
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    fsGroup: 2000
```

```yaml
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: app
    image: my-app:latest
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      capabilities:
        drop:
        - ALL
```

4.3 Désactiver le Mode Privilégié sur ECS

```
# Variable d'environnement ECS Agent
ECS_DISABLE_PRIVILEGED=true
```

5. IAM Roles for Service Accounts (EKS)

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-app-sa
  namespace: production
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::123456789012:role/MyAppRole

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  template:
    spec:
      serviceAccountName: my-app-sa  # Utilise le rôle IAM
      containers:
      - name: app
        image: my-app:latest
```

Avantages:
- ✅ Permissions IAM granulaires par pod
- ✅ Auditabilité via CloudTrail
- ✅ Isolation multi-tenant

Systems Manager et Automatisation

1. Patch Management avec Patch Manager

1.1 Configuration Automatique des Patches

```
# Créer une baseline de patches (approuver automatiquement après 7 jours)
aws ssm create-patch-baseline \
    --name "Production-Baseline" \
    --operating-system "AMAZON_LINUX_2" \
    --approval-rules "PatchRules=[{PatchFilterGroup={PatchFilters=[{Key=CLASSIFICATION,Values=

# Enregistrer la baseline comme default
aws ssm register-default-patch-baseline \
    --baseline-id pb-xxxxx
```

1.2 Maintenance Window pour Patching

```
# Créer une fenêtre de maintenance (tous les dimanches à 2h00 UTC)
aws ssm create-maintenance-window \
    --name "Weekly-Patching" \
    --schedule "cron(0 2 ? * SUN *)" \
    --duration 4 \
    --cutoff 1 \
    --allow-unassociated-targets

# Enregistrer les targets (toutes les instances avec tag Environment=Production)
aws ssm register-target-with-maintenance-window \
    --window-id mw-xxxxx \
    --target-type "INSTANCE" \
    --owner-information "Production Instances" \
    --resource-type "RESOURCE_GROUP" \
    --targets "Key=tag:Environment,Values=Production"

# Ajouter une tâche de patching
aws ssm register-task-with-maintenance-window \
    --window-id mw-xxxxx \
    --task-type "RUN_COMMAND" \
    --task-arn "AWS-RunPatchBaseline" \
    --priority 1 \
    --max-concurrency "50%" \
    --max-errors "25%" \
    --targets "Key=WindowTargetIds,Values=xxxxx"
```

2. Session Manager pour Accès Sécurisé

2.1 Configuration avec Logs et Chiffrement

```
{
  "schemaVersion": "1.0",
  "description": "Document to hold regional settings for Session Manager",
  "sessionType": "Standard_Stream",
  "inputs": {
    "s3BucketName": "my-session-logs-bucket",
    "s3KeyPrefix": "session-logs/",
    "s3EncryptionEnabled": true,
    "cloudWatchLogGroupName": "/aws/ssm/session-logs",
    "cloudWatchEncryptionEnabled": true,
    "kmsKeyId": "alias/session-manager-key",
    "runAsEnabled": true,
    "runAsDefaultUser": "ssm-user",
    "idleSessionTimeout": "20"
  }
}
```

2.2 Restreindre les Commandes avec Session Documents

```
{
  "schemaVersion": "1.0",
  "description": "Limited command session - read-only",
  "sessionType": "InteractiveCommands",
  "inputs": {
    "commands": [
      "ls",
      "cat",
      "grep",
      "tail",
      "head"
    ]
  }
}
```

3. Automatisation avec Run Command

```
# Exécuter une commande sur toutes les instances d'un groupe
aws ssm send-command \
    --document-name "AWS-RunShellScript" \
    --targets "Key=tag:Environment,Values=Production" \
    --parameters 'commands=["sudo systemctl restart nginx"]' \
    --max-concurrency "10" \
    --max-errors "5" \
    --timeout-seconds 600
```

Gestion des Secrets

1. AWS Secrets Manager vs Parameter Store

Fonctionnalité	Secrets Manager	Parameter Store
Rotation automatique	✅ Oui	❌ Non
Versioning	✅ Oui	✅ Oui
Chiffrement KMS	✅ Par défaut	✅ Optionnel
Coût	€€ (0.40$/secret/mois)	€ (gratuit ou 0.05$/param)
Cas d'usage	Passwords DB, API keys	Configuration, non-secrets

2. Rotation Automatique des Secrets

```
# Lambda de rotation pour RDS MySQL
import boto3
import pymysql

def lambda_handler(event, context):
    service_client = boto3.client('secretsmanager')

    arn = event['SecretId']
    token = event['ClientRequestToken']
    step = event['Step']

    if step == "createSecret":
        # Générer un nouveau mot de passe
        current_dict = get_secret_dict(service_client, arn, "AWSCURRENT")
        new_password = service_client.get_random_password(
            PasswordLength=32,
            ExcludeCharacters='/@"\'\\'
        )['RandomPassword']

        current_dict['password'] = new_password
        service_client.put_secret_value(
            SecretId=arn,
            ClientRequestToken=token,
            SecretString=json.dumps(current_dict),
            VersionStages=['AWSPENDING']
        )

    elif step == "setSecret":
        # Mettre à jour le mot de passe dans la base de données
        pending_dict = get_secret_dict(service_client, arn, "AWSPENDING")
```

```
        conn = pymysql.connect(
            host=pending_dict['host'],
            user=pending_dict['username'],
            password=current_dict['password']
        )

        with conn.cursor() as cursor:
            cursor.execute(f"ALTER USER '{pending_dict['username']}' IDENTIFIED BY '{pending_d
        conn.commit()

    elif step == "testSecret":
        # Tester la nouvelle connexion
        pending_dict = get_secret_dict(service_client, arn, "AWSPENDING")
        conn = pymysql.connect(
            host=pending_dict['host'],
            user=pending_dict['username'],
            password=pending_dict['password']
        )
        conn.close()

    elif step == "finishSecret":
        # Promouvoir AWSPENDING à AWSCURRENT
        service_client.update_secret_version_stage(
            SecretId=arn,
            VersionStage="AWSCURRENT",
            MoveToVersionId=token
        )
```

Scénarios d'Attaque et Mitigation Avancée

Attaque 1: SSRF via IMDSv1 pour Vol de Credentials IAM

Scénario:
Exploit d'une vulnérabilité SSRF (Server-Side Request Forgery) dans une application web pour accéder aux credentials IAM via IMDSv1.

Attack Chain:

```
1. Découverte de la vulnérabilité SSRF
   └─> Endpoint: /api/fetch?url=http://example.com

2. Exploitation IMDSv1
   └─> Payload: /api/fetch?url=http://169.254.169.254/latest/meta-data/iam/security-credential
   └─> Réponse: MyEC2Role

3. Vol de credentials
```

```
    └─> Payload: /api/fetch?url=http://169.254.169.254/latest/meta-data/iam/security-credential
    └─> Réponse:
    {
      "AccessKeyId": "ASIA...",
      "SecretAccessKey": "...",
      "Token": "...",
      "Expiration": "2025-11-08T12:00:00Z"
    }

4. Utilisation des credentials volés
    └─> aws s3 ls (avec credentials volés)
    └─> Exfiltration de données
```

Exemple réel de code vulnérable:

```
# ❌ VULNÉRABLE - SSRF possible
@app.route('/api/fetch')
def fetch_url():
    url = request.args.get('url')
    response = requests.get(url)  # Pas de validation
    return response.text

# Exploitation:
# GET /api/fetch?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/MyRole
```

Mitigation complète:

1. Forcer IMDSv2 (bloque SSRF):

```
# IMDSv2 requiert une requête PUT pour obtenir un token
# Impossible via SSRF simple (GET only)

# Activer IMDSv2 sur instance existante
aws ec2 modify-instance-metadata-options \
    --instance-id i-xxxxx \
    --http-tokens required \
    --http-put-response-hop-limit 1

# Terraform pour nouvelles instances
resource "aws_launch_template" "secure" {
  name = "secure-launch-template"

  metadata_options {
    http_endpoint               = "enabled"
    http_tokens                 = "required"  # Force IMDSv2
    http_put_response_hop_limit = 1
    instance_metadata_tags      = "enabled"
  }
}

# Auto Scaling Group
```

```
resource "aws_autoscaling_group" "main" {
  launch_template {
    id      = aws_launch_template.secure.id
    version = "$Latest"
  }
}
```

2. Validation URL côté application:

```
# ✅ SÉCURISÉ - Validation stricte
from urllib.parse import urlparse
import ipaddress

BLOCKED_RANGES = [
    ipaddress.ip_network('169.254.0.0/16'),   # IMDS
    ipaddress.ip_network('10.0.0.0/8'),       # Private
    ipaddress.ip_network('172.16.0.0/12'),    # Private
    ipaddress.ip_network('192.168.0.0/16'),   # Private
    ipaddress.ip_network('127.0.0.0/8'),      # Loopback
]

ALLOWED_DOMAINS = ['example.com', 'api.example.com']

def is_url_safe(url):
    """Valide qu'une URL n'est pas malveillante"""
    try:
        parsed = urlparse(url)

        # Vérifier le schéma
        if parsed.scheme not in ['http', 'https']:
            return False

        # Vérifier le domaine (whitelist)
        if parsed.hostname not in ALLOWED_DOMAINS:
            return False

        # Résoudre l'IP
        import socket
        ip = ipaddress.ip_address(socket.gethostbyname(parsed.hostname))

        # Vérifier que l'IP n'est pas dans une plage bloquée
        for blocked_range in BLOCKED_RANGES:
            if ip in blocked_range:
                return False

        return True
    except Exception as e:
        return False

@app.route('/api/fetch')
def fetch_url():
    url = request.args.get('url')
```

```
    if not is_url_safe(url):
        return {'error': 'Invalid URL'}, 400

    response = requests.get(url, timeout=5)
    return response.text
```

3. Alarmes CloudWatch pour détection:

```
# Filtre CloudTrail pour utilisation suspecte de credentials
aws logs put-metric-filter \
    --log-group-name /aws/cloudtrail/logs \
    --filter-name suspicious-credential-usage \
    --filter-pattern '[..., userIdentity.type = "AssumedRole", userIdentity.principalId != "i-
    --metric-transformations \
        metricName=SuspiciousCredentialUsage,\
        metricNamespace=Security,\
        metricValue=1

# Alarme
aws cloudwatch put-metric-alarm \
    --alarm-name ssrf-credential-theft-detected \
    --metric-name SuspiciousCredentialUsage \
    --namespace Security \
    --statistic Sum \
    --period 300 \
    --threshold 1 \
    --comparison-operator GreaterThanOrEqualToThreshold \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:SecurityAlerts
```

4. GuardDuty détection:

GuardDuty détecte automatiquement:
- `UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.OutsideAWS`
- `UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.InsideAWS`

Attaque 2: Container Escape via Kernel Exploit

Scénario:
Attaquant avec accès à un container privilégié exploite une vulnérabilité kernel pour s'échapper et accéder à l'hôte EC2.

Indicateurs:
- Container en mode `privileged: true`
- Capabilities Linux non restreintes
- `hostPath` volumes montés
- `securityContext.allowPrivilegeEscalation: true`

Exploit exemple (CVE-2022-0847 "Dirty Pipe"):

```
# Depuis un container privilégié
# 1. Vérifier les capabilities
capsh --print

# 2. Exploiter la vulnérabilité kernel
./dirty_pipe_exploit

# 3. Escape vers l'hôte
# Accès complet au système hôte
cat /host/etc/shadow
```

Mitigation complète:

1. Désactiver mode privilégié (ECS):

```
{
  "family": "secure-task",
  "containerDefinitions": [
    {
      "name": "app",
      "image": "myapp:latest",
      "privileged": false,  // ✅ JAMAIS privileged
      "readonlyRootFilesystem": true,
      "user": "1000:1000",
      "linuxParameters": {
        "capabilities": {
          "drop": ["ALL"],  // Drop toutes les capabilities
          "add": []         // N'en ajouter aucune (sauf si absolument nécessaire)
        }
      }
    }
  ],
  "taskRoleArn": "arn:aws:iam::123456789012:role/TaskRole",
  "executionRoleArn": "arn:aws:iam::123456789012:role/ExecutionRole"
}
```

2. Pod Security Standards (Kubernetes/EKS):

```
# Baseline Policy - Minimum sécurité
apiVersion: v1
kind: Namespace
metadata:
  name: production
  labels:
    pod-security.kubernetes.io/enforce: baseline
    pod-security.kubernetes.io/audit: restricted
    pod-security.kubernetes.io/warn: restricted
```

```yaml
---
# Restricted Pod (maximum sécurité)
apiVersion: v1
kind: Pod
metadata:
  name: secure-app
  namespace: production
spec:
  securityContext:
    runAsNonRoot: true
    runAsUser: 1000
    fsGroup: 2000
    seccompProfile:
      type: RuntimeDefault
    supplementalGroups: [3000]

  containers:
  - name: app
    image: myapp:latest
    securityContext:
      allowPrivilegeEscalation: false
      readOnlyRootFilesystem: true
      runAsNonRoot: true
      runAsUser: 1000
      capabilities:
        drop:
        - ALL
      seccompProfile:
        type: RuntimeDefault

    # Volume temporaire pour /tmp (readOnly filesystem)
    volumeMounts:
    - name: tmp
      mountPath: /tmp

  volumes:
  - name: tmp
    emptyDir: {}
```

3. OPA Gatekeeper pour enforcement (EKS):

```yaml
# Constraint Template: Bloquer containers privilégiés
apiVersion: templates.gatekeeper.sh/v1
kind: ConstraintTemplate
metadata:
  name: k8spspprivileged
spec:
  crd:
    spec:
      names:
        kind: K8sPSPPrivileged
  targets:
    - target: admission.k8s.gatekeeper.sh
```

```
      rego: |
        package k8spspprivileged

        violation[{"msg": msg}] {
          c := input.review.object.spec.containers[_]
          c.securityContext.privileged
          msg := sprintf("Privileged container not allowed: %v", [c.name])
        }

        violation[{"msg": msg}] {
          c := input.review.object.spec.containers[_]
          c.securityContext.allowPrivilegeEscalation
          msg := sprintf("Privilege escalation not allowed: %v", [c.name])
        }

---
# Constraint: Appliquer la politique
apiVersion: constraints.gatekeeper.sh/v1beta1
kind: K8sPSPPrivileged
metadata:
  name: psp-privileged-constraint
spec:
  match:
    kinds:
      - apiGroups: [""]
        kinds: ["Pod"]
    namespaces:
      - production
      - staging
```

4. Scan runtime avec Falco:

```
# Règle Falco pour détecter container escape
- rule: Container Drift Detected
  desc: Detect file modifications in running containers
  condition: >
    container and
    not container.privileged and
    (open_write or rename or unlink) and
    container.image.repository in (production_images)
  output: "File modified in container (user=%user.name command=%proc.cmdline file=%fd.name co
  priority: WARNING

- rule: Privilege Escalation Attempt
  desc: Detect attempts to gain privilege
  condition: >
    spawned_process and
    proc.name in (sudo, su, doas) and
    container and
    not user.name = root
  output: "Privilege escalation attempt (user=%user.name command=%proc.cmdline container=%con
  priority: CRITICAL
```

5. Inspector Runtime Monitoring:

```
# Activer GuardDuty Runtime Monitoring pour ECS/EKS
aws guardduty update-detector \
    --detector-id <id> \
    --features '[{
        "Name": "RUNTIME_MONITORING",
        "Status": "ENABLED",
        "AdditionalConfiguration": [{
            "Name": "ECS_FARGATE_AGENT_MANAGEMENT",
            "Status": "ENABLED"
        }, {
            "Name": "EKS_ADDON_MANAGEMENT",
            "Status": "ENABLED"
        }]
    }]'

# Détections automatiques:
# - Execution:Runtime/NewBinaryExecuted
# - PrivilegeEscalation:Runtime/ContainerMountsHostDirectory
# - DefenseEvasion:Runtime/FilelessExecution
```

Attaque 3: Lambda Code Injection via Event Poisoning

Scénario:
Injection de code malveillant dans une fonction Lambda via des événements non validés (S3, SQS, API Gateway).

Exemple d'attaque:

```
# ❌ Code Lambda VULNÉRABLE
import subprocess

def lambda_handler(event, context):
    # Event depuis S3: {"Records": [{"s3": {"object": {"key": "file.txt"}}}]}
    file_key = event['Records'][0]['s3']['object']['key']

    # DANGEREUX: Injection de commande
    result = subprocess.run(f"aws s3 cp s3://my-bucket/{file_key} /tmp/", shell=True)

# Exploitation:
# Upload fichier avec key: "file.txt; rm -rf / #"
# Commande exécutée: aws s3 cp s3://my-bucket/file.txt; rm -rf / # /tmp/
```

Mitigation:

1. Validation stricte des événements:

```
# ✅ Code Lambda SÉCURISÉ
import json
import re
import boto3
from jsonschema import validate, ValidationError

# Schéma JSON pour validation
S3_EVENT_SCHEMA = {
    "type": "object",
    "properties": {
        "Records": {
            "type": "array",
            "items": {
                "type": "object",
                "properties": {
                    "s3": {
                        "type": "object",
                        "properties": {
                            "object": {
                                "type": "object",
                                "properties": {
                                    "key": {"type": "string", "pattern": "^[a-zA-Z0-9_\\-\\.]+
                                },
                                "required": ["key"]
                            }
                        },
                        "required": ["object"]
                    }
                },
                "required": ["s3"]
            }
        }
    },
    "required": ["Records"]
}

def lambda_handler(event, context):
    try:
        # Valider le schéma
        validate(instance=event, schema=S3_EVENT_SCHEMA)
    except ValidationError as e:
        print(f"Invalid event: {e.message}")
        raise

    # Extraire et valider le key
    file_key = event['Records'][0]['s3']['object']['key']

    # Validation additionnelle
    if not re.match(r'^[a-zA-Z0-9_\-\.]+$', file_key):
        raise ValueError(f"Invalid file key: {file_key}")

    # Utiliser boto3 au lieu de subprocess
    s3 = boto3.client('s3')
    s3.download_file('my-bucket', file_key, f'/tmp/{file_key}')
```

2. Sandbox Lambda avec Resource Policies:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "lambda:InvokeFunction",
      "Resource": "arn:aws:lambda:us-east-1:123456789012:function:MyFunction",
      "Condition": {
        "StringEquals": {
          "AWS:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "AWS:SourceArn": "arn:aws:s3:::my-trusted-bucket"
        }
      }
    }
  ]
}
```

3. Limits et timeouts:

```
# SAM Template avec sécurité renforcée
Resources:
  ProcessFileFunction:
    Type: AWS::Serverless::Function
    Properties:
      CodeUri: src/
      Handler: app.lambda_handler
      Runtime: python3.11
      Timeout: 30  # Max 30 secondes
      MemorySize: 256
      ReservedConcurrentExecutions: 10  # Limite concurrence

      Environment:
        Variables:
          ALLOWED_EXTENSIONS: ".txt,.csv,.json"

      Policies:
        - S3ReadPolicy:
            BucketName: my-bucket
        - Statement:
            - Effect: Allow
              Action:
                - logs:CreateLogGroup
                - logs:CreateLogStream
                - logs:PutLogEvents
              Resource: "*"
```

```
      Events:
        S3Event:
          Type: S3
          Properties:
            Bucket: !Ref SourceBucket
            Events: s3:ObjectCreated:*
            Filter:
              S3Key:
                Rules:
                  - Name: suffix
                    Value: .txt
```

Architecture de Référence Complète: EKS Production

1. Cluster EKS Sécurisé avec Terraform

```
# eks-cluster.tf - Production-ready EKS

terraform {
  required_version = ">= 1.0"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
    kubernetes = {
      source  = "hashicorp/kubernetes"
      version = "~> 2.20"
    }
  }
}

# KMS Key pour chiffrement secrets EKS
resource "aws_kms_key" "eks" {
  description             = "EKS Secret Encryption Key"
  deletion_window_in_days = 30
  enable_key_rotation     = true

  tags = {
    Name = "eks-secret-encryption-key"
  }
}

resource "aws_kms_alias" "eks" {
  name          = "alias/eks-secret-key"
  target_key_id = aws_kms_key.eks.key_id
}
```

```
# EKS Cluster
resource "aws_eks_cluster" "main" {
  name     = "production-eks"
  role_arn = aws_iam_role.eks_cluster.arn
  version  = "1.28"

  vpc_config {
    subnet_ids              = concat(var.private_subnet_ids, var.public_subnet_ids)
    endpoint_private_access = true
    endpoint_public_access  = true  # Restreindre avec public_access_cidrs en prod
    public_access_cidrs     = ["203.0.113.0/24"]  # IP bureau uniquement

    security_group_ids = [aws_security_group.eks_cluster.id]
  }

  # Chiffrement des secrets Kubernetes avec KMS
  encryption_config {
    provider {
      key_arn = aws_kms_key.eks.arn
    }
    resources = ["secrets"]
  }

  # Logging activé
  enabled_cluster_log_types = ["api", "audit", "authenticator", "controllerManager", "schedule

  depends_on = [
    aws_iam_role_policy_attachment.eks_cluster_policy,
    aws_cloudwatch_log_group.eks
  ]

  tags = {
    Environment = "production"
  }
}

# CloudWatch Log Group pour logs EKS
resource "aws_cloudwatch_log_group" "eks" {
  name              = "/aws/eks/production-eks/cluster"
  retention_in_days = 90

  tags = {
    Name = "eks-cluster-logs"
  }
}

# Managed Node Group (Graviton pour coût/performance)
resource "aws_eks_node_group" "main" {
  cluster_name    = aws_eks_cluster.main.name
  node_group_name = "production-nodes"
  node_role_arn   = aws_iam_role.eks_node_group.arn
  subnet_ids      = var.private_subnet_ids

  scaling_config {
```

```
    desired_size = 3
    max_size     = 10
    min_size     = 2
  }

  update_config {
    max_unavailable = 1
  }

  ami_type       = "AL2_ARM_64"  # Graviton
  capacity_type  = "ON_DEMAND"
  instance_types = ["t4g.medium"]

  # Launch template pour sécurité
  launch_template {
    id      = aws_launch_template.eks_nodes.id
    version = "$Latest"
  }

  labels = {
    Environment = "production"
  }

  tags = {
    Name = "eks-production-nodes"
  }

  depends_on = [
    aws_iam_role_policy_attachment.eks_worker_node_policy,
    aws_iam_role_policy_attachment.eks_cni_policy,
    aws_iam_role_policy_attachment.eks_container_registry_policy,
  ]
}

# Launch Template sécurisé pour nodes
resource "aws_launch_template" "eks_nodes" {
  name = "eks-node-launch-template"

  metadata_options {
    http_endpoint               = "enabled"
    http_tokens                 = "required"  # IMDSv2
    http_put_response_hop_limit = 1
  }

  block_device_mappings {
    device_name = "/dev/xvda"

    ebs {
      volume_size           = 50
      volume_type           = "gp3"
      encrypted             = true
      kms_key_id            = aws_kms_key.ebs.arn
      delete_on_termination = true
    }
  }
```

```
  monitoring {
    enabled = true
  }

  tag_specifications {
    resource_type = "instance"
    tags = {
      Name = "eks-worker-node"
    }
  }
}

# Security Group Cluster
resource "aws_security_group" "eks_cluster" {
  name        = "eks-cluster-sg"
  description = "Security group for EKS cluster"
  vpc_id      = var.vpc_id

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "eks-cluster-security-group"
  }
}

# Security Group règles
resource "aws_security_group_rule" "cluster_ingress_workstation_https" {
  description       = "Allow workstation to communicate with the cluster API Server"
  type              = "ingress"
  from_port         = 443
  to_port           = 443
  protocol          = "tcp"
  cidr_blocks       = ["203.0.113.0/24"]  # IP bureau
  security_group_id = aws_security_group.eks_cluster.id
}

# IAM Role pour Cluster
resource "aws_iam_role" "eks_cluster" {
  name = "eks-cluster-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Principal = {
        Service = "eks.amazonaws.com"
      }
      Action = "sts:AssumeRole"
    }]
```

```
  })
}

resource "aws_iam_role_policy_attachment" "eks_cluster_policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
  role       = aws_iam_role.eks_cluster.name
}

# IAM Role pour Node Group
resource "aws_iam_role" "eks_node_group" {
  name = "eks-node-group-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Principal = {
        Service = "ec2.amazonaws.com"
      }
      Action = "sts:AssumeRole"
    }]
  })
}

resource "aws_iam_role_policy_attachment" "eks_worker_node_policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKSWorkerNodePolicy"
  role       = aws_iam_role.eks_node_group.name
}

resource "aws_iam_role_policy_attachment" "eks_cni_policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEKS_CNI_Policy"
  role       = aws_iam_role.eks_node_group.name
}

resource "aws_iam_role_policy_attachment" "eks_container_registry_policy" {
  policy_arn = "arn:aws:iam::aws:policy/AmazonEC2ContainerRegistryReadOnly"
  role       = aws_iam_role.eks_node_group.name
}

# OIDC Provider pour IRSA (IAM Roles for Service Accounts)
data "tls_certificate" "eks" {
  url = aws_eks_cluster.main.identity[0].oidc[0].issuer
}

resource "aws_iam_openid_connect_provider" "eks" {
  client_id_list  = ["sts.amazonaws.com"]
  thumbprint_list = [data.tls_certificate.eks.certificates[0].sha1_fingerprint]
  url             = aws_eks_cluster.main.identity[0].oidc[0].issuer

  tags = {
    Name = "eks-oidc-provider"
  }
}

# Outputs
```

```
output "cluster_endpoint" {
  value = aws_eks_cluster.main.endpoint
}

output "cluster_certificate_authority_data" {
  value = aws_eks_cluster.main.certificate_authority[0].data
}

output "cluster_oidc_issuer_url" {
  value = aws_eks_cluster.main.identity[0].oidc[0].issuer
}
```

2. Add-ons Sécurité EKS

```
# AWS Load Balancer Controller (Ingress)
resource "aws_eks_addon" "aws_load_balancer_controller" {
  cluster_name = aws_eks_cluster.main.name
  addon_name   = "aws-load-balancer-controller"
  addon_version = "v2.6.0-eksbuild.1"
}

# EBS CSI Driver (chiffrement volumes)
resource "aws_eks_addon" "ebs_csi_driver" {
  cluster_name = aws_eks_cluster.main.name
  addon_name   = "aws-ebs-csi-driver"
  addon_version = "v1.24.0-eksbuild.1"
}

# CoreDNS
resource "aws_eks_addon" "coredns" {
  cluster_name = aws_eks_cluster.main.name
  addon_name   = "coredns"
  addon_version = "v1.10.1-eksbuild.2"
}

# VPC CNI
resource "aws_eks_addon" "vpc_cni" {
  cluster_name = aws_eks_cluster.main.name
  addon_name   = "vpc-cni"
  addon_version = "v1.15.0-eksbuild.1"

  configuration_values = jsonencode({
    env = {
      ENABLE_POD_ENI                     = "true"
      ENABLE_PREFIX_DELEGATION           = "true"
      WARM_PREFIX_TARGET                 = "1"
      AWS_VPC_K8S_CNI_EXTERNALSNAT       = "true"
      AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG = "true"
    }
  })
}

# GuardDuty pour EKS
```

```
resource "aws_guardduty_detector" "main" {
  enable = true

  datasources {
    kubernetes {
      audit_logs {
        enable = true
      }
    }
  }

  tags = {
    Name = "eks-guardduty-detector"
  }
}
```

3. Network Policies Kubernetes

```yaml
# default-deny-all.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny-all
  namespace: production
spec:
  podSelector: {}
  policyTypes:
  - Ingress
  - Egress

---
# allow-frontend-to-backend.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-frontend-to-backend
  namespace: production
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Ingress
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: frontend
    ports:
    - protocol: TCP
      port: 8080

---
```

```
# allow-backend-to-database.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-backend-to-db
  namespace: production
spec:
  podSelector:
    matchLabels:
      app: backend
  policyTypes:
  - Egress
  egress:
  - to:
    - podSelector:
        matchLabels:
          app: database
    ports:
    - protocol: TCP
      port: 3306

---
# allow-dns.yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-dns-access
  namespace: production
spec:
  podSelector: {}
  policyTypes:
  - Egress
  egress:
  - to:
    - namespaceSelector:
        matchLabels:
          name: kube-system
    - podSelector:
        matchLabels:
          k8s-app: kube-dns
    ports:
    - protocol: UDP
      port: 53
```

Monitoring et Détection Avancée

1. CloudWatch Container Insights

```
# Activer Container Insights pour ECS
aws ecs update-cluster-settings \
```

```
    --cluster production-cluster \
    --settings name=containerInsights,value=enabled

# Installer CloudWatch Agent dans EKS
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-ins
```

Métriques clés à surveiller:

```
# CloudWatch Logs Insights - Pods crashant fréquemment
fields @timestamp, kubernetes.pod_name, kubernetes.namespace_name, log
| filter kubernetes.namespace_name = "production"
| filter log like /error|exception|fatal/i
| stats count(*) as error_count by kubernetes.pod_name
| sort error_count desc
| limit 20

# Containers avec CPU throttling
fields @timestamp, ContainerName, CpuUtilized, CpuReserved
| filter CpuUtilized / CpuReserved > 0.8
| stats avg(CpuUtilized / CpuReserved) as avg_cpu_util by ContainerName
| sort avg_cpu_util desc

# Détection OOMKilled
fields @timestamp, kubernetes.pod_name, kubernetes.container_name, reason
| filter reason = "OOMKilled"
| stats count(*) as oom_count by kubernetes.pod_name
| sort oom_count desc
```

2. Alarmes CloudWatch Critiques

```
# CPU Utilization élevé (EC2)
aws cloudwatch put-metric-alarm \
    --alarm-name high-cpu-ec2 \
    --metric-name CPUUtilization \
    --namespace AWS/EC2 \
    --statistic Average \
    --period 300 \
    --threshold 80 \
    --comparison-operator GreaterThanThreshold \
    --evaluation-periods 2 \
    --dimensions Name=InstanceId,Value=i-xxxxx \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:Alerts

# Lambda errors élevé
aws cloudwatch put-metric-alarm \
    --alarm-name lambda-high-errors \
    --metric-name Errors \
    --namespace AWS/Lambda \
    --statistic Sum \
    --period 60 \
    --threshold 10 \
```

```
    --comparison-operator GreaterThanThreshold \
    --evaluation-periods 1 \
    --dimensions Name=FunctionName,Value=MyFunction \
    --treat-missing-data notBreaching \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:Alerts

# ECS Service healthy tasks < 2
aws cloudwatch put-metric-alarm \
    --alarm-name ecs-unhealthy-tasks \
    --metric-name HealthyTaskCount \
    --namespace ECS/ContainerInsights \
    --statistic Average \
    --period 60 \
    --threshold 2 \
    --comparison-operator LessThanThreshold \
    --evaluation-periods 2 \
    --dimensions Name=ServiceName,Value=my-service Name=ClusterName,Value=production \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:Alerts
```

3. Détection d'Anomalies avec CloudWatch Anomaly Detection

```
# Créer une alarme avec détection d'anomalies
aws cloudwatch put-metric-alarm \
    --alarm-name lambda-invocations-anomaly \
    --comparison-operator LessThanLowerOrGreaterThanUpperThreshold \
    --evaluation-periods 2 \
    --metrics file://anomaly-detection-config.json \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:Alerts
```

anomaly-detection-config.json:

```
[
  {
    "Id": "m1",
    "ReturnData": true,
    "MetricStat": {
      "Metric": {
        "Namespace": "AWS/Lambda",
        "MetricName": "Invocations",
        "Dimensions": [
          {
            "Name": "FunctionName",
            "Value": "MyFunction"
          }
        ]
      },
      "Period": 300,
      "Stat": "Sum"
    }
  },
```

```
  {
    "Id": "ad1",
    "Expression": "ANOMALY_DETECTION_BAND(m1, 2)",
    "Label": "Invocations (expected)"
  }
]
```

Best Practices Avancées

1. Immutable Infrastructure

Principe: Ne jamais modifier une instance en production, toujours déployer une nouvelle version.

```
# Auto Scaling avec Launch Template versionné
resource "aws_launch_template" "app" {
  name_prefix   = "app-lt-"
  image_id      = data.aws_ami.app_ami.id  # AMI depuis pipeline CI/CD
  instance_type = "t3.medium"

  # Chaque changement crée une nouvelle version
  lifecycle {
    create_before_destroy = true
  }
}

resource "aws_autoscaling_group" "app" {
  desired_capacity = 3
  max_size         = 10
  min_size         = 2

  launch_template {
    id      = aws_launch_template.app.id
    version = "$Latest"  # Utilise toujours la dernière version
  }

  # Rolling update: remplace progressivement les instances
  instance_refresh {
    strategy = "Rolling"
    preferences {
      min_healthy_percentage = 90
      instance_warmup        = 300
    }
  }
}
```

Pipeline CI/CD pour AMI:

```bash
#!/bin/bash
# build-ami.sh

# 1. Build application
docker build -t myapp:${GIT_COMMIT} .

# 2. Run security scan
trivy image myapp:${GIT_COMMIT} --severity CRITICAL,HIGH --exit-code 1

# 3. Créer AMI avec Packer
packer build \
    -var "app_version=${GIT_COMMIT}" \
    -var "base_ami=$(aws ec2 describe-images --owners amazon --filters "Name=name,Values=amzn2
    packer-template.json

# 4. Mettre à jour Launch Template
NEW_AMI_ID=$(aws ec2 describe-images --filters "Name=tag:Version,Values=${GIT_COMMIT}" --query

aws ec2 create-launch-template-version \
    --launch-template-id lt-xxxxx \
    --source-version '$Latest' \
    --launch-template-data "{\"ImageId\":\"${NEW_AMI_ID}\"}"

# 5. Déclench

er instance refresh
aws autoscaling start-instance-refresh \
    --auto-scaling-group-name production-asg \
    --preferences MinHealthyPercentage=90,InstanceWarmup=300
```

2. Blue/Green Deployments avec CodeDeploy

```yaml
# appspec.yml pour CodeDeploy
version: 0.0
Resources:
  - TargetService:
      Type: AWS::ECS::Service
      Properties:
        TaskDefinition: "arn:aws:ecs:us-east-1:123456789012:task-definition/my-task:5"
        LoadBalancerInfo:
          ContainerName: "app"
          ContainerPort: 8080
        PlatformVersion: "LATEST"

Hooks:
  - BeforeInstall: "LambdaFunctionToValidateBeforeInstall"
  - AfterInstall: "LambdaFunctionToValidateAfterInstall"
  - AfterAllowTestTraffic: "LambdaFunctionToValidateAfterTestTrafficStarts"
  - BeforeAllowTraffic: "LambdaFunctionToValidateBeforeAllowingProductionTraffic"
  - AfterAllowTraffic: "LambdaFunctionToValidateAfterAllowingProductionTraffic"
```

Runtime Security et Détection de Menaces

1. Falco pour Kubernetes (EKS)

Falco est un outil open-source de **détection d'anomalies runtime** pour containers, capable de détecter comportements suspects au niveau du kernel.

1.1 Déploiement Falco sur EKS

```yaml
# falco-daemonset.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: falco
  namespace: security
spec:
  selector:
    matchLabels:
      app: falco
  template:
    metadata:
      labels:
        app: falco
    spec:
      serviceAccountName: falco
      hostNetwork: true
      hostPID: true
      containers:
      - name: falco
        image: falcosecurity/falco:0.36.2
        securityContext:
          privileged: true  # Requis pour accéder au kernel
        volumeMounts:
        - name: dev
          mountPath: /host/dev
        - name: proc
          mountPath: /host/proc
          readOnly: true
        - name: boot
          mountPath: /host/boot
          readOnly: true
        - name: lib-modules
          mountPath: /host/lib/modules
          readOnly: true
        - name: usr
          mountPath: /host/usr
          readOnly: true
        - name: etc
          mountPath: /host/etc
          readOnly: true
        - name: config
```

```yaml
          mountPath: /etc/falco
        env:
        - name: FALCO_K8S_AUDIT_ENDPOINT
          value: "http://localhost:8765/k8s-audit"
      volumes:
      - name: dev
        hostPath:
          path: /dev
      - name: proc
        hostPath:
          path: /proc
      - name: boot
        hostPath:
          path: /boot
      - name: lib-modules
        hostPath:
          path: /lib/modules
      - name: usr
        hostPath:
          path: /usr
      - name: etc
        hostPath:
          path: /etc
      - name: config
        configMap:
          name: falco-config
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: falco
  namespace: security
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: falco
rules:
- apiGroups: [""]
  resources:
    - pods
    - namespaces
    - nodes
  verbs: ["get", "list", "watch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: falco
subjects:
- kind: ServiceAccount
  name: falco
  namespace: security
roleRef:
  kind: ClusterRole
```

```
  name: falco
  apiGroup: rbac.authorization.k8s.io
```

1.2 Règles Falco Personnalisées

```yaml
# falco-rules.yaml
customRules:
  custom-rules.yaml: |-
    # Détection de shell interactif dans container
    - rule: Terminal Shell in Container
      desc: A shell was spawned in a container
      condition: >
        spawned_process and
        container and
        proc.name in (bash, sh, zsh, fish) and
        proc.tty != 0
      output: >
        Shell spawned in container (user=%user.name command=%proc.cmdline
        container_id=%container.id container_name=%container.name
        image=%container.image.repository:%container.image.tag)
      priority: WARNING
      tags: [container, shell, mitre_execution]

    # Détection de reverse shell
    - rule: Reverse Shell Detected
      desc: Reverse shell connection detected
      condition: >
        spawned_process and
        container and
        ((proc.name in (bash, sh, zsh) and
          proc.args contains "-i" and
          (proc.args contains "/dev/tcp" or proc.args contains "/dev/udp")) or
         (proc.name = nc and proc.args contains "-e"))
      output: >
        Reverse shell detected (user=%user.name command=%proc.cmdline
        container_id=%container.id container_name=%container.name
        image=%container.image.repository:%container.image.tag)
      priority: CRITICAL
      tags: [container, reverse_shell, mitre_execution]

    # Modification de fichiers sensibles
    - rule: Sensitive File Modification
      desc: Sensitive file was modified in container
      condition: >
        open_write and
        container and
        fd.name in (/etc/passwd, /etc/shadow, /etc/sudoers,
                    /root/.ssh/authorized_keys, /home/*/.ssh/authorized_keys)
      output: >
        Sensitive file modified (user=%user.name file=%fd.name
        command=%proc.cmdline container_id=%container.id
        container_name=%container.name image=%container.image.repository)
      priority: CRITICAL
```

```
    tags: [container, filesystem, mitre_persistence]

  # Execution de binaires suspects
  - rule: Suspicious Binary Execution
    desc: Execution of suspicious binary in container
    condition: >
      spawned_process and
      container and
      proc.name in (nmap, masscan, nc, netcat, socat, curl, wget) and
      proc.pname != package_manager
    output: >
      Suspicious binary executed (user=%user.name binary=%proc.name
      args=%proc.args container_id=%container.id
      container_name=%container.name image=%container.image.repository)
    priority: WARNING
    tags: [container, network, mitre_discovery]

  # Privilege escalation
  - rule: Privilege Escalation Attempt
    desc: Attempt to escalate privileges detected
    condition: >
      spawned_process and
      container and
      proc.name in (sudo, su) and
      not user.name in (root)
    output: >
      Privilege escalation attempt (user=%user.name command=%proc.cmdline
      container_id=%container.id container_name=%container.name)
    priority: CRITICAL
    tags: [container, privilege_escalation]

  # Crypto mining
  - rule: Cryptocurrency Mining Detected
    desc: Cryptocurrency mining activity detected
    condition: >
      spawned_process and
      container and
      (proc.name in (xmrig, ccminer, ethminer, minerd) or
       proc.cmdline contains "stratum+tcp" or
       proc.cmdline contains "mining.pool")
    output: >
      Cryptocurrency mining detected (command=%proc.cmdline
      container_id=%container.id container_name=%container.name
      image=%container.image.repository)
    priority: CRITICAL
    tags: [container, cryptomining, mitre_impact]

  # Container running as root
  - rule: Container Running as Root
    desc: Container is running as root user
    condition: >
      container_started and
      container and
      user.uid = 0
    output: >
```

```
        Container running as root (container_id=%container.id
        container_name=%container.name image=%container.image.repository:%container.image.tag
        user=%user.name)
      priority: WARNING
      tags: [container, users]

    # Outbound connection to suspicious port
    - rule: Outbound Connection to Suspicious Port
      desc: Outbound connection to suspicious port detected
      condition: >
        outbound and
        container and
        fd.sport in (4444, 5555, 6666, 7777, 8888, 9999)
      output: >
        Outbound connection to suspicious port (user=%user.name
        connection=%fd.name sport=%fd.sport dport=%fd.dport
        container_id=%container.id container_name=%container.name)
      priority: WARNING
      tags: [container, network]
```

1.3 Intégration Falco avec CloudWatch

Falco Sidekick pour router alertes vers CloudWatch:

```
# falcosidekick-deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: falcosidekick
  namespace: security
spec:
  replicas: 2
  selector:
    matchLabels:
      app: falcosidekick
  template:
    metadata:
      labels:
        app: falcosidekick
    spec:
      containers:
      - name: falcosidekick
        image: falcosecurity/falcosidekick:2.28.0
        env:
        - name: AWS_CLOUDWATCHLOGS_LOGGROUP
          value: "/aws/eks/falco-alerts"
        - name: AWS_CLOUDWATCHLOGS_LOGSTREAM
          value: "security-alerts"
        - name: AWS_REGION
          value: "us-east-1"
        - name: AWS_CLOUDWATCHLOGS_MINIMUMPRIORITY
          value: "warning"
        - name: SLACK_WEBHOOKURL
```

```
          valueFrom:
            secretKeyRef:
              name: falco-secrets
              key: slack-webhook
        ports:
        - containerPort: 2801
      serviceAccountName: falcosidekick
---
apiVersion: v1
kind: Service
metadata:
  name: falcosidekick
  namespace: security
spec:
  selector:
    app: falcosidekick
  ports:
  - port: 2801
    targetPort: 2801
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: falcosidekick
  namespace: security
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::123456789012:role/FalcoCloudWatchRole
```

IAM Role pour Falco:

```
# Terraform configuration pour IAM Role
resource "aws_iam_role" "falco_cloudwatch" {
  name = "FalcoCloudWatchRole"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Principal = {
        Federated = "arn:aws:iam::${data.aws_caller_identity.current.account_id}:oidc-provider
      }
      Action = "sts:AssumeRoleWithWebIdentity"
      Condition = {
        StringEquals = {
          "${replace(data.aws_eks_cluster.main.identity[0].oidc[0].issuer, "https://", "")}:su
        }
      }
    }]
  })
}

resource "aws_iam_role_policy" "falco_cloudwatch" {
  name = "CloudWatchLogsAccess"
```

```
  role = aws_iam_role.falco_cloudwatch.id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "logs:CreateLogGroup",
          "logs:CreateLogStream",
          "logs:PutLogEvents",
          "logs:DescribeLogStreams"
        ]
        Resource = "arn:aws:logs:*:*:log-group:/aws/eks/falco-alerts:*"
      }
    ]
  })
}
```

1.4 Analyse des Alertes Falco

CloudWatch Logs Insights queries:

```
# Top 10 règles Falco déclenchées
fields @timestamp, rule, priority, output
| filter priority = "Critical" or priority = "Warning"
| stats count(*) as alertCount by rule
| sort alertCount desc
| limit 10

# Containers suspects (shells, reverse shells)
fields @timestamp, output, container_name, container_image
| filter rule like /Shell|Reverse/
| sort @timestamp desc

# Timeline des tentatives d'escalade de privilèges
fields @timestamp, output, user_name, container_name
| filter rule = "Privilege Escalation Attempt"
| sort @timestamp desc

# Crypto mining detection
fields @timestamp, output, container_id, container_name
| filter rule = "Cryptocurrency Mining Detected"
| stats count(*) as instances by container_image
```

2. Amazon GuardDuty Runtime Monitoring

GuardDuty peut désormais monitorer le runtime des containers ECS et EKS pour détecter des menaces.

2.1 Activation GuardDuty Runtime Monitoring

```
# Activer GuardDuty pour EKS
aws guardduty update-detector \
    --detector-id <detector-id> \
    --features '[{
        "Name": "EKS_RUNTIME_MONITORING",
        "Status": "ENABLED",
        "AdditionalConfiguration": [{
            "Name": "EKS_ADDON_MANAGEMENT",
            "Status": "ENABLED"
        }]
    }]'

# GuardDuty déploiera automatiquement l'agent sur les nodes EKS

# Activer pour ECS (Fargate)
aws guardduty update-detector \
    --detector-id <detector-id> \
    --features '[{
        "Name": "ECS_FARGATE_RUNTIME_MONITORING",
        "Status": "ENABLED"
    }]'
```

2.2 Findings GuardDuty Runtime

GuardDuty détectera:

Finding	Description	Severity
`Runtime:Container/SuspiciousProcess`	Processus suspect dans container	High
`Runtime:Container/ReverseShell`	Connexion reverse shell détectée	Critical
`Runtime:Container/PrivilegeEscalation`	Tentative d'escalade de privilèges	High
`Runtime:Container/NewBinaryExecuted`	Binaire inconnu exécuté	Medium
`Runtime:Container/FileSystemModification`	Modification fichiers système	High

Réponse automatique aux findings:

```
import boto3
import json
```

```python
ecs = boto3.client('ecs')
ec2 = boto3.client('ec2')
sns = boto3.client('sns')

def lambda_handler(event, context):
    """
    Réponse automatique aux findings GuardDuty Runtime
    """

    detail = event['detail']
    finding_type = detail['type']
    severity = detail['severity']

    # Extraire infos container
    resource = detail['resource']
    container_details = resource.get('containerDetails', {})
    container_id = container_details.get('id', 'unknown')
    container_image = container_details.get('imagePrefix', 'unknown')

    # Actions basées sur sévérité
    if severity >= 7.0:  # High ou Critical
        if 'ECS' in finding_type:
            # Isoler task ECS
            task_arn = resource['ecsTaskDetails']['taskArn']
            cluster_arn = resource['ecsTaskDetails']['clusterArn']

            # Arrêter task
            ecs.stop_task(
                cluster=cluster_arn,
                task=task_arn,
                reason='GuardDuty security finding: ' + finding_type
            )

            print(f"Stopped ECS task {task_arn} due to {finding_type}")

        elif 'EKS' in finding_type:
            # Pour EKS, quarantiner pod via Network Policy
            # (nécessite intégration avec API Kubernetes)
            pass

        # Notification SNS
        sns.publish(
            TopicArn='arn:aws:sns:us-east-1:123456789012:SecurityIncidents',
            Subject=f'CRITICAL: GuardDuty Runtime Alert - {finding_type}',
            Message=json.dumps(detail, indent=2)
        )

    return {'statusCode': 200}
```

3. AWS Inspector pour Container Scanning

Inspector scanne les images ECR et les instances EC2 pour vulnérabilités.

```
# Activer Inspector v2
resource "aws_inspector2_enabler" "main" {
  account_ids    = [data.aws_caller_identity.current.account_id]
  resource_types = ["ECR", "EC2", "LAMBDA"]
}

# Alerte sur vulnérabilités critiques
resource "aws_cloudwatch_event_rule" "inspector_findings" {
  name        = "inspector-critical-vulnerabilities"
  description = "Alert on critical Inspector findings"

  event_pattern = jsonencode({
    source      = ["aws.inspector2"]
    detail-type = ["Inspector2 Finding"]
    detail = {
      severity = ["CRITICAL", "HIGH"]
    }
  })
}

resource "aws_cloudwatch_event_target" "sns" {
  rule      = aws_cloudwatch_event_rule.inspector_findings.name
  target_id = "SendToSNS"
  arn       = aws_sns_topic.security_alerts.arn
}
```

Sécurité Lambda Layers

1. Risques de Sécurité Lambda Layers

Lambda Layers peuvent introduire des vulnérabilités si mal gérés:

Risque	Impact	Mitigation
Dépendances vulnérables	Exploitation CVE	Scanner layers avec Snyk/Trivy
Layers partagés publiquement	Exposition de code	Layers privés uniquement
Layers non versionnés	Incompatibilités	Versionning strict
Code malveillant	Backdoor	Audit code tiers

2. Best Practices Lambda Layers

2.1 Scanner les Layers pour Vulnérabilités

```
# Créer layer localement
mkdir python
pip install -t python/ requests boto3

# Scanner avec Trivy avant déploiement
trivy fs python/ --severity CRITICAL,HIGH

# Si clean, créer layer
zip -r layer.zip python/
aws lambda publish-layer-version \
    --layer-name secure-dependencies \
    --zip-file fileb://layer.zip \
    --compatible-runtimes python3.11 \
    --description "Scanned dependencies - no CVEs" \
    --license-info "MIT"
```

2.2 Layers Privés et Permissions

```
# Layer privé (pas de permissions publiques)
resource "aws_lambda_layer_version" "dependencies" {
  layer_name          = "app-dependencies"
  filename            = "layer.zip"
  source_code_hash    = filebase64sha256("layer.zip")
  compatible_runtimes = ["python3.11"]

  description = "Application dependencies - scanned for vulnerabilities"
}

# NE PAS faire ceci (layer public)
# resource "aws_lambda_layer_version_permission" "public" {
#   layer_name     = aws_lambda_layer_version.dependencies.layer_name
#   version_number = aws_lambda_layer_version.dependencies.version
#   principal      = "*"  # ❌ DANGEREUX
#   action         = "lambda:GetLayerVersion"
# }

# Permissions spécifiques par account/OU
resource "aws_lambda_layer_version_permission" "specific_account" {
  layer_name     = aws_lambda_layer_version.dependencies.layer_name
  version_number = aws_lambda_layer_version.dependencies.version
  principal      = "123456789012"  # Account ID spécifique
  action         = "lambda:GetLayerVersion"
  statement_id   = "AllowAccount123456789012"
}
```

2.3 Versionning et Rotation des Layers

```python
# lambda_layer_updater.py
import boto3
import hashlib
import os

lambda_client = boto3.client('lambda')

def update_layer_if_changed(layer_name, zip_path, compatible_runtimes):
    """
    Met à jour layer uniquement si contenu a changé
    """

    # Calculer hash du nouveau layer
    with open(zip_path, 'rb') as f:
        new_hash = hashlib.sha256(f.read()).hexdigest()

    # Récupérer dernière version
    try:
        response = lambda_client.list_layer_versions(
            LayerName=layer_name,
            MaxItems=1
        )

        if response['LayerVersions']:
            latest_version = response['LayerVersions'][0]
            latest_hash = latest_version.get('CodeSha256')

            if latest_hash == new_hash:
                print(f"Layer {layer_name} unchanged, skipping update")
                return latest_version['Version']
    except lambda_client.exceptions.ResourceNotFoundException:
        pass

    # Publier nouvelle version
    with open(zip_path, 'rb') as f:
        response = lambda_client.publish_layer_version(
            LayerName=layer_name,
            Content={'ZipFile': f.read()},
            CompatibleRuntimes=compatible_runtimes,
            Description=f'SHA256: {new_hash}'
        )

    new_version = response['Version']
    print(f"Published new layer version: {layer_name}:{ new_version}")

    # Mettre à jour toutes les fonctions utilisant ce layer
    update_functions_with_layer(layer_name, new_version)

    # Supprimer anciennes versions (garder 3 dernières)
    cleanup_old_layer_versions(layer_name, keep=3)

    return new_version
```

```
def update_functions_with_layer(layer_name, new_version):
    """
    Met à jour toutes les fonctions utilisant ce layer
    """
    paginator = lambda_client.get_paginator('list_functions')

    for page in paginator.paginate():
        for function in page['Functions']:
            function_name = function['FunctionName']
            layers = function.get('Layers', [])

            # Vérifier si fonction utilise ce layer
            updated_layers = []
            layer_found = False

            for layer in layers:
                layer_arn = layer['Arn']
                if layer_name in layer_arn:
                    # Mettre à jour vers nouvelle version
                    base_arn = layer_arn.rsplit(':', 1)[0]
                    updated_layers.append(f"{base_arn}:{new_version}")
                    layer_found = True
                else:
                    updated_layers.append(layer_arn)

            if layer_found:
                lambda_client.update_function_configuration(
                    FunctionName=function_name,
                    Layers=updated_layers
                )
                print(f"Updated function {function_name} to layer version {new_version}")

def cleanup_old_layer_versions(layer_name, keep=3):
    """
    Supprime les anciennes versions de layer (garde les N dernières)
    """
    response = lambda_client.list_layer_versions(LayerName=layer_name)
    versions = response['LayerVersions']

    # Garder les N dernières versions
    if len(versions) > keep:
        for version in versions[keep:]:
            lambda_client.delete_layer_version(
                LayerName=layer_name,
                VersionNumber=version['Version']
            )
            print(f"Deleted old layer version: {layer_name}:{version['Version']}")
```

2.4 Audit et Monitoring des Layers

```
# CloudWatch Logs Insights - Quelles fonctions utilisent quels layers
aws lambda list-functions --query 'Functions[?Layers].{Name:FunctionName, Layers:Layers[].Arn}
```

```
# Trouver layers publics (risque sécurité)
aws lambda list-layers --query 'Layers[?contains(Arn, `public`)]'

# Audit permissions layer
aws lambda get-layer-version-policy --layer-name my-layer --version-number 1
```

Gestion Avancée des Secrets pour ECS/EKS

1. Secrets Manager pour ECS

1.1 Injection Sécurisée dans Task Definition

```
{
  "family": "secure-app",
  "taskRoleArn": "arn:aws:iam::123456789012:role/ecsTaskRole",
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecsExecutionRole",
  "containerDefinitions": [{
    "name": "app",
    "image": "myapp:latest",
    "secrets": [
      {
        "name": "DB_PASSWORD",
        "valueFrom": "arn:aws:secretsmanager:us-east-1:123456789012:secret:prod/db/password-Ab
      },
      {
        "name": "API_KEY",
        "valueFrom": "arn:aws:secretsmanager:us-east-1:123456789012:secret:prod/api/key-AbCdEf
      },
      {
        "name": "JWT_SECRET",
        "valueFrom": "arn:aws:secretsmanager:us-east-1:123456789012:secret:prod/jwt-AbCdEf:sec
      }
    ],
    "logConfiguration": {
      "logDriver": "awslogs",
      "secretOptions": [
        {
          "name": "SPLUNK_TOKEN",
          "valueFrom": "arn:aws:secretsmanager:us-east-1:123456789012:secret:monitoring/splunk
        }
      ]
    }
  }]
}
```

IAM Execution Role:

```
resource "aws_iam_role" "ecs_execution_role" {
  name = "ecsExecutionRole"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Principal = {
        Service = "ecs-tasks.amazonaws.com"
      }
      Action = "sts:AssumeRole"
    }]
  })
}

resource "aws_iam_role_policy" "ecs_secrets_access" {
  name = "SecretsManagerAccess"
  role = aws_iam_role.ecs_execution_role.id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "secretsmanager:GetSecretValue",
          "secretsmanager:DescribeSecret"
        ]
        Resource = [
          "arn:aws:secretsmanager:us-east-1:123456789012:secret:prod/*"
        ]
      },
      {
        Effect = "Allow"
        Action = [
          "kms:Decrypt"
        ]
        Resource = "arn:aws:kms:us-east-1:123456789012:key/*"
        Condition = {
          StringEquals = {
            "kms:ViaService" = "secretsmanager.us-east-1.amazonaws.com"
          }
        }
      }
    ]
  })
}
```

2. External Secrets Operator pour EKS

External Secrets Operator synchronise secrets depuis AWS Secrets Manager vers Kubernetes Secrets.

2.1 Installation External Secrets Operator

```
# Installer via Helm
helm repo add external-secrets https://charts.external-secrets.io
helm install external-secrets \
    external-secrets/external-secrets \
    -n external-secrets-system \
    --create-namespace
```

2.2 Configuration SecretStore

```
# secretstore.yaml
apiVersion: external-secrets.io/v1beta1
kind: SecretStore
metadata:
  name: aws-secretsmanager
  namespace: production
spec:
  provider:
    aws:
      service: SecretsManager
      region: us-east-1
      auth:
        jwt:
          serviceAccountRef:
            name: external-secrets-sa
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: external-secrets-sa
  namespace: production
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::123456789012:role/ExternalSecretsRole
```

IAM Role pour External Secrets:

```
resource "aws_iam_role" "external_secrets" {
  name = "ExternalSecretsRole"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Principal = {
        Federated = "arn:aws:iam::${data.aws_caller_identity.current.account_id}:oidc-provider
      }
      Action = "sts:AssumeRoleWithWebIdentity"
      Condition = {
        StringEquals = {
          "${replace(data.aws_eks_cluster.main.identity[0].oidc[0].issuer, "https://", "")}:su
```

```
        }
      }
    }]
  })
}

resource "aws_iam_role_policy" "external_secrets_policy" {
  name = "SecretsManagerReadAccess"
  role = aws_iam_role.external_secrets.id

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "secretsmanager:GetSecretValue",
          "secretsmanager:DescribeSecret",
          "secretsmanager:ListSecrets"
        ]
        Resource = "arn:aws:secretsmanager:us-east-1:123456789012:secret:production/*"
      },
      {
        Effect = "Allow"
        Action = ["kms:Decrypt"]
        Resource = "*"
        Condition = {
          StringEquals = {
            "kms:ViaService" = "secretsmanager.us-east-1.amazonaws.com"
          }
        }
      }
    ]
  })
}
```

2.3 ExternalSecret CR

```
# externalsecret.yaml
apiVersion: external-secrets.io/v1beta1
kind: ExternalSecret
metadata:
  name: app-database-credentials
  namespace: production
spec:
  refreshInterval: 1h  # Synchroniser toutes les heures
  secretStoreRef:
    name: aws-secretsmanager
    kind: SecretStore
  target:
    name: database-credentials  # Nom du Secret Kubernetes créé
    creationPolicy: Owner
    template:
```

```yaml
      engineVersion: v2
      data:
        # Template pour formater le secret
        config.yaml: |
          database:
            host: {{ .host }}
            port: {{ .port }}
            username: {{ .username }}
            password: {{ .password }}
            database: {{ .database }}
  data:
  - secretKey: host
    remoteRef:
      key: production/database/main
      property: host
  - secretKey: port
    remoteRef:
      key: production/database/main
      property: port
  - secretKey: username
    remoteRef:
      key: production/database/main
      property: username
  - secretKey: password
    remoteRef:
      key: production/database/main
      property: password
  - secretKey: database
    remoteRef:
      key: production/database/main
      property: database
```

Utilisation dans Pod:

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: app
  namespace: production
spec:
  template:
    spec:
      containers:
      - name: app
        image: myapp:latest
        env:
        - name: DB_HOST
          valueFrom:
            secretKeyRef:
              name: database-credentials
              key: host
        - name: DB_PASSWORD
          valueFrom:
```

```
            secretKeyRef:
              name: database-credentials
              key: password
        # Ou monter comme fichier
        volumeMounts:
        - name: db-config
          mountPath: /etc/config
          readOnly: true
      volumes:
      - name: db-config
        secret:
          secretName: database-credentials
          items:
          - key: config.yaml
            path: database.yaml
```

3. Rotation Automatique des Secrets

3.1 Lambda de Rotation pour RDS

```
import boto3
import json
import os
import pymysql

secretsmanager = boto3.client('secretsmanager')
rds = boto3.client('rds')

def lambda_handler(event, context):
    """
    Rotation automatique des credentials RDS
    """

    secret_arn = event['SecretId']
    token = event['ClientRequestToken']
    step = event['Step']

    # Récupérer secret actuel
    current_secret = secretsmanager.get_secret_value(SecretId=secret_arn)
    current_dict = json.loads(current_secret['SecretString'])

    if step == "createSecret":
        # Générer nouveau password
        new_password = generate_secure_password()

        # Créer version AWSPENDING
        pending_dict = current_dict.copy()
        pending_dict['password'] = new_password

        secretsmanager.put_secret_value(
            SecretId=secret_arn,
            ClientRequestToken=token,
            SecretString=json.dumps(pending_dict),
```

```
            VersionStages=['AWSPENDING']
        )

    elif step == "setSecret":
        # Mettre à jour password dans RDS
        pending_secret = secretsmanager.get_secret_value(
            SecretId=secret_arn,
            VersionId=token,
            VersionStage='AWSPENDING'
        )
        pending_dict = json.loads(pending_secret['SecretString'])

        # Connexion avec ancien password
        conn = pymysql.connect(
            host=current_dict['host'],
            user=current_dict['username'],
            password=current_dict['password'],
            database='mysql'
        )

        try:
            with conn.cursor() as cursor:
                # Changer password
                cursor.execute(
                    f"ALTER USER '{current_dict['username']}'@'%' IDENTIFIED BY '{pending_dic
                )
                cursor.execute("FLUSH PRIVILEGES")
            conn.commit()
        finally:
            conn.close()

    elif step == "testSecret":
        # Tester nouveau password
        pending_secret = secretsmanager.get_secret_value(
            SecretId=secret_arn,
            VersionId=token,
            VersionStage='AWSPENDING'
        )
        pending_dict = json.loads(pending_secret['SecretString'])

        # Test connexion
        conn = pymysql.connect(
            host=pending_dict['host'],
            user=pending_dict['username'],
            password=pending_dict['password']
        )
        conn.close()

    elif step == "finishSecret":
        # Promouvoir AWSPENDING vers AWSCURRENT
        secretsmanager.update_secret_version_stage(
            SecretId=secret_arn,
            VersionStage='AWSCURRENT',
            MoveToVersionId=token,
            RemoveFromVersionId=current_secret['VersionId']
```

```
        )

    return {'statusCode': 200}

def generate_secure_password(length=32):
    import secrets
    import string
    alphabet = string.ascii_letters + string.digits + "!@#$%^&*"
    return ''.join(secrets.choice(alphabet) for _ in range(length))
```

3.2 Configuration Rotation dans Terraform

```
# Secret avec rotation automatique
resource "aws_secretsmanager_secret" "db_credentials" {
  name                    = "production/database/credentials"
  description             = "RDS database credentials with auto-rotation"
  recovery_window_in_days = 7

  tags = {
    Environment = "production"
    Rotation    = "enabled"
  }
}

resource "aws_secretsmanager_secret_version" "db_credentials" {
  secret_id = aws_secretsmanager_secret.db_credentials.id
  secret_string = jsonencode({
    username = "app_user"
    password = random_password.db_password.result
    host     = aws_db_instance.main.endpoint
    port     = 3306
    database = "production_db"
  })
}

# Lambda de rotation
resource "aws_lambda_function" "rotate_secret" {
  filename      = "rotation_lambda.zip"
  function_name = "RotateRDSSecret"
  role          = aws_iam_role.lambda_rotation.arn
  handler       = "index.lambda_handler"
  runtime       = "python3.11"
  timeout       = 30

  vpc_config {
    subnet_ids         = aws_subnet.private_app[*].id
    security_group_ids = [aws_security_group.lambda_rotation.id]
  }

  environment {
    variables = {
      SECRETS_MANAGER_ENDPOINT = "https://secretsmanager.us-east-1.amazonaws.com"
    }
```

```
  }
}

# Permission pour Secrets Manager d'invoquer Lambda
resource "aws_lambda_permission" "allow_secretsmanager" {
  statement_id  = "AllowExecutionFromSecretsManager"
  action        = "lambda:InvokeFunction"
  function_name = aws_lambda_function.rotate_secret.function_name
  principal     = "secretsmanager.amazonaws.com"
}

# Configuration rotation
resource "aws_secretsmanager_secret_rotation" "db_credentials" {
  secret_id           = aws_secretsmanager_secret.db_credentials.id
  rotation_lambda_arn = aws_lambda_function.rotate_secret.arn

  rotation_rules {
    automatically_after_days = 30  # Rotation mensuelle
  }
}
```

Sidecar Security Patterns

1. Envoy Proxy comme Security Sidecar

Envoy peut servir de proxy sidecar pour:
- **Chiffrement mTLS automatique**
- **Rate limiting**
- **Authentication/Authorization**
- **Observabilité**

1.1 Configuration Envoy Sidecar pour ECS

```
{
  "family": "app-with-envoy",
  "networkMode": "awsvpc",
  "containerDefinitions": [
    {
      "name": "app",
      "image": "myapp:latest",
      "portMappings": [{
        "containerPort": 8080,
        "protocol": "tcp"
      }],
      "dependsOn": [{
        "containerName": "envoy",
        "condition": "HEALTHY"
      }]
```

```
    },
    {
      "name": "envoy",
      "image": "envoyproxy/envoy:v1.28-latest",
      "essential": true,
      "portMappings": [{
        "containerPort": 9901,
        "protocol": "tcp"
      }, {
        "containerPort": 15000,
        "protocol": "tcp"
      }],
      "healthCheck": {
        "command": [
          "CMD-SHELL",
          "curl -f http://localhost:9901/ready || exit 1"
        ],
        "interval": 10,
        "timeout": 5,
        "retries": 3
      },
      "user": "1337",
      "mountPoints": [{
        "sourceVolume": "envoy-config",
        "containerPath": "/etc/envoy",
        "readOnly": true
      }]
    }
  ],
  "volumes": [{
    "name": "envoy-config",
    "host": {
      "sourcePath": "/ecs/envoy-config"
    }
  }]
}
```

Envoy Configuration (envoy.yaml):

```
admin:
  address:
    socket_address:
      address: 0.0.0.0
      port_value: 9901

static_resources:
  listeners:
  - name: listener_0
    address:
      socket_address:
        address: 0.0.0.0
        port_value: 15000
    filter_chains:
```

```yaml
    - filters:
      - name: envoy.filters.network.http_connection_manager
        typed_config:
          "@type": type.googleapis.com/envoy.extensions.filters.network.http_connection_manage
          stat_prefix: ingress_http
          route_config:
            name: local_route
            virtual_hosts:
            - name: backend
              domains: ["*"]
              routes:
              - match:
                  prefix: "/"
                route:
                  cluster: local_app
                # Rate limiting
                rate_limits:
                - actions:
                  - request_headers:
                      header_name: "x-user-id"
                      descriptor_key: "user_id"
          http_filters:
          # JWT Authentication
          - name: envoy.filters.http.jwt_authn
            typed_config:
              "@type": type.googleapis.com/envoy.extensions.filters.http.jwt_authn.v3.JwtAuthe
              providers:
                auth0:
                  issuer: "https://myapp.auth0.com/"
                  audiences:
                  - "https://api.myapp.com"
                  remote_jwks:
                    http_uri:
                      uri: "https://myapp.auth0.com/.well-known/jwks.json"
                      cluster: auth0_jwks
                      timeout: 5s
              rules:
              - match:
                  prefix: "/api"
                requires:
                  provider_name: "auth0"
          # Rate limiting filter
          - name: envoy.filters.http.ratelimit
            typed_config:
              "@type": type.googleapis.com/envoy.extensions.filters.http.ratelimit.v3.RateLimi
              domain: api_ratelimit
              rate_limit_service:
                grpc_service:
                  envoy_grpc:
                    cluster_name: ratelimit
          # Router filter
          - name: envoy.filters.http.router
            typed_config:
              "@type": type.googleapis.com/envoy.extensions.filters.http.router.v3.Router
      # mTLS configuration
```

```
      transport_socket:
        name: envoy.transport_sockets.tls
        typed_config:
          "@type": type.googleapis.com/envoy.extensions.transport_sockets.tls.v3.DownstreamTls
          common_tls_context:
            tls_certificates:
            - certificate_chain:
                filename: "/etc/envoy/certs/server-cert.pem"
              private_key:
                filename: "/etc/envoy/certs/server-key.pem"
            validation_context:
              trusted_ca:
                filename: "/etc/envoy/certs/ca-cert.pem"
          require_client_certificate: true

  clusters:
  - name: local_app
    connect_timeout: 0.25s
    type: STRICT_DNS
    lb_policy: ROUND_ROBIN
    load_assignment:
      cluster_name: local_app
      endpoints:
      - lb_endpoints:
        - endpoint:
            address:
              socket_address:
                address: 127.0.0.1
                port_value: 8080
```

2. AWS App Mesh pour Service Mesh

App Mesh fournit communication sécurisée entre microservices avec mTLS automatique.

```
# Virtual Gateway (entry point)
resource "aws_appmesh_virtual_gateway" "main" {
  name      = "api-gateway"
  mesh_name = aws_appmesh_mesh.main.id

  spec {
    listener {
      port_mapping {
        port     = 443
        protocol = "http"
      }

      tls {
        mode = "STRICT"
        certificate {
          acm {
            certificate_arn = aws_acm_certificate.api.arn
```

```
          }
        }
      }
    }
  }
}

# Virtual Service
resource "aws_appmesh_virtual_service" "app" {
  name      = "app.local"
  mesh_name = aws_appmesh_mesh.main.id

  spec {
    provider {
      virtual_router {
        virtual_router_name = aws_appmesh_virtual_router.app.name
      }
    }
  }
}

# Virtual Node avec mTLS
resource "aws_appmesh_virtual_node" "app" {
  name      = "app-node"
  mesh_name = aws_appmesh_mesh.main.id

  spec {
    listener {
      port_mapping {
        port     = 8080
        protocol = "http"
      }

      # mTLS Backend
      tls {
        mode = "STRICT"
        certificate {
          file {
            certificate_chain = "/etc/envoy/certs/cert-chain.pem"
            private_key       = "/etc/envoy/certs/private-key.pem"
          }
        }
        validation {
          trust {
            file {
              certificate_chain = "/etc/envoy/certs/ca-chain.pem"
            }
          }
        }
      }

      health_check {
        protocol            = "http"
        path                = "/health"
        healthy_threshold   = 2
```

```
        unhealthy_threshold = 2
        timeout_millis      = 2000
        interval_millis     = 5000
      }
    }

    service_discovery {
      aws_cloud_map {
        namespace_name = aws_service_discovery_private_dns_namespace.main.name
        service_name   = "app"
      }
    }

    # Backend virtual service avec mTLS client
    backend {
      virtual_service {
        virtual_service_name = aws_appmesh_virtual_service.database.name
        client_policy {
          tls {
            enforce = true
            validation {
              trust {
                file {
                  certificate_chain = "/etc/envoy/certs/ca-chain.pem"
                }
              }
            }
          }
        }
      }
    }
  }
}
```

Checklist de Sécurité Hébergement

✅ EC2 (Priorité Critique)

- [ ] **IMDSv2 activé et obligatoire sur toutes les instances**
- [ ] **Chiffrement EBS activé par défaut**
- [ ] **Aucune instance avec IP publique (utiliser ALB)**
- [ ] **Security Groups: aucun 0.0.0.0/0 sur SSH (22)**
- [ ] **IAM Instance Profiles (pas d'access keys)**
- [ ] **Systems Manager Session Manager pour accès (pas SSH)**
- [ ] **Patch Manager configuré avec maintenance windows**

- [ ] **CloudWatch Agent installé pour métriques et logs**

✅ Lambda (Priorité Critique)

- [ ] **Secrets dans Secrets Manager (pas env variables)**
- [ ] **Un rôle IAM par fonction (moindre privilège)**
- [ ] **VPC configuration uniquement si nécessaire**
- [ ] **VPC Endpoints pour services AWS (S3, DynamoDB)**
- [ ] **Validation des entrées avec schémas**
- [ ] **Pas de logs de données sensibles**
- [ ] **Timeout < 15 minutes**
- [ ] **Réservé Concurrency configuré**

✅ Containers ECS/EKS (Priorité Critique)

- [ ] **Scan automatique des images ECR activé (Enhanced)**
- [ ] **Images distroless en production**
- [ ] **Pas de containers en mode privilégié**
- [ ] **ReadOnlyRootFilesystem activé**
- [ ] **Capabilities Linux drop ALL**
- [ ] **IAM Roles for Service Accounts (EKS)**
- [ ] **Amazon Inspector activé pour runtime security**
- [ ] **Network policies Kubernetes configurées**

✅ Systems Manager (Priorité Importante)

- [ ] **Session Manager configuré avec logs S3 + CloudWatch**
- [ ] **Patch baselines définies par OS**
- [ ] **Maintenance windows configurées**
- [ ] **Compliance reporting activé**
- [ ] **Automation runbooks pour incidents**

Références et Ressources

Documentation Officielle AWS

- EC2 IMDSv2 Best Practices
- Lambda Security Best Practices
- ECS Security Best Practices
- EKS Best Practices Guide
- Systems Manager Best Practices

Conclusion

La sécurisation de l'hébergement AWS repose sur trois piliers:

1. **Protection des instances** avec IMDSv2, chiffrement et isolation réseau
2. **Sécurité des déploiements** avec scan d'images, validation et moindre privilège
3. **Gestion proactive** avec patch management, monitoring et automatisation

L'implémentation de ces meilleures pratiques garantit une infrastructure d'hébergement sécurisée, conforme et résiliente.

Document préparé pour: [Nom du Client]
Contact support: [Email de l'équipe DevOps]
Dernière mise à jour: Novembre 2025