

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

1. [Sécurité EC2](#)
 2. [Sécurité Serverless \(Lambda\)](#)
 3. [Sécurité Containers \(ECS/EKS\)](#)
 4. [Systems Manager et Automatisation](#)
 5. [Gestion des Secrets](#)
 6. [Checklist de Sécurité Hébergement](#)
-

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/MyR...
```

```
# 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
```

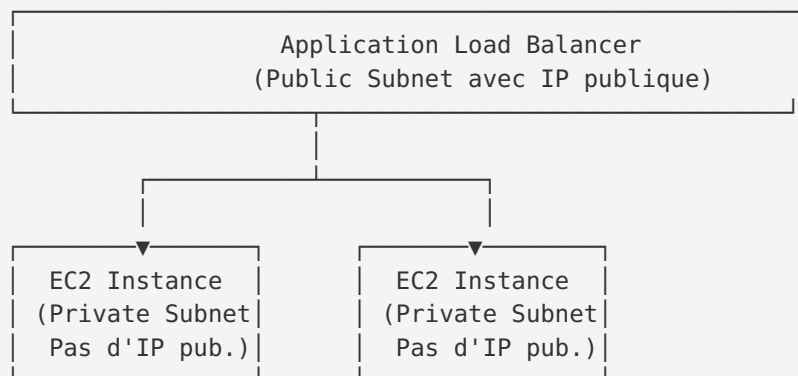
2.3 Politique AWS Config pour Conformité

```
# 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:



```
# Vérifier les instances avec IP publiques
aws ec2 describe-instances \
  --filters "Name=instance-state-name,Values=running" \
  --query 'Reservations[].Instances[?PublicIpAddress!=`null`].[InstanceId,PublicIpAddress,Tags]' \
  --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_name}"
    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/AWSLambdaVPCLAccessExecutionRole
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": "SCHEDULED"}]
```

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
```

```

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)




```

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_c
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
    )

```

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