Guide Complet : Sécurisation Applications et Stockage AWS pour SaaS

Version: 1.0
Date: Novembre 2025
Destiné à: Architectes Applications et Équipes Backend

Table des Matières

1. Sécurité Amazon S3
2. Sécurité Amazon RDS
3. Sécurité API Gateway
4. Sécurité DynamoDB
5. Bonnes Pratiques Multi-Services
6. Checklist Applications & Stockage

Sécurité Amazon S3

1. Chiffrement et Protection des Données

1.1 Chiffrement Par Défaut Activé

Depuis 2023: Tous les nouveaux buckets S3 ont le chiffrement activé par défaut (SSE-S3). Cependant, pour une sécurité renforcée, utilisez SSE-KMS.

```
# Activer le chiffrement par défaut avec KMS
aws s3api put-bucket-encryption \
    --bucket my-saas-bucket \
    --server-side-encryption-configuration '{
      "Rules": [{
        "ApplyServerSideEncryptionByDefault": {
          "SSEAlgorithm": "aws:kms",
          "KMSMasterKeyID": "arn:aws:kms:us-east-1:123456789012:key/xxxxx"
        },
        "BucketKeyEnabled": true
      }]
    }'
```

Avantages SSE-KMS:
- ✅ **Audit via CloudTrail** : Qui accède à quelles données
- ✅ **Contrôle granulaire** : Politiques KMS par tenant/application
- ✅ **Rotation automatique** : Rotation annuelle des clés
- ✅ **Compliance** : Requis pour PCI-DSS, HIPAA

1.2 Forcer HTTPS (TLS) Uniquement

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyInsecureConnections",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "s3:*",
      "Resource": [
        "arn:aws:s3:::my-saas-bucket",
        "arn:aws:s3:::my-saas-bucket/*"
      ],
      "Condition": {
        "Bool": {
          "aws:SecureTransport": "false"
        }
      }
    }
  ]
}
```

```
# Appliquer la politique
aws s3api put-bucket-policy \
    --bucket my-saas-bucket \
    --policy file://enforce-https-policy.json
```

1.3 Versioning et MFA Delete

```
# Activer le versioning
aws s3api put-bucket-versioning \
    --bucket my-saas-bucket \
    --versioning-configuration Status=Enabled

# Activer MFA Delete (nécessite le compte root)
aws s3api put-bucket-versioning \
    --bucket my-saas-bucket \
    --versioning-configuration Status=Enabled,MFADelete=Enabled \
    --mfa "arn:aws:iam::123456789012:mfa/root-account-mfa-device 123456"
```

MFA Delete nécessite une authentification MFA pour:
- ❌ Supprimer une version d'objet
- ❌ Désactiver le versioning

2. Contrôle d'Accès et Isolation

2.1 Block Public Access (OBLIGATOIRE)

```
# Activer Block Public Access au niveau du compte
aws s3control put-public-access-block \
    --account-id 123456789012 \
    --public-access-block-configuration \
        BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBucket

# Activer au niveau du bucket
aws s3api put-public-access-block \
    --bucket my-saas-bucket \
    --public-access-block-configuration \
        BlockPublicAcls=true,IgnorePublicAcls=true,BlockPublicPolicy=true,RestrictPublicBucket
```

Statistique: Les buckets S3 mal configurés restent l'une des principales causes d'incidents de sécurité cloud en 2025.

2.2 Politique Bucket Basée sur le Moindre Privilège

❌ Dangereux:

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": "*",
    "Action": "s3:*",
    "Resource": "arn:aws:s3:::my-bucket/*"
  }]
}
```

✅ Sécurisé - Multi-Tenant avec Préfixes:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowTenantAccess",
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/TenantApplicationRole"
      },
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": "arn:aws:s3:::my-saas-bucket/${aws:PrincipalTag/TenantID}/*"
    }
  ]
}
```

Cette politique permet aux utilisateurs d'accéder uniquement aux objets sous leur préfixe tenant.

2.3 S3 Access Points pour Multi-Tenant

```
# Créer un Access Point par tenant
aws s3control create-access-point \
    --account-id 123456789012 \
    --name tenant-a-access-point \
    --bucket my-saas-bucket \
    --policy '{
      "Version": "2012-10-17",
      "Statement": [{
        "Effect": "Allow",
        "Principal": {"AWS": "arn:aws:iam::123456789012:role/TenantARole"},
        "Action": ["s3:GetObject", "s3:PutObject"],
        "Resource": "arn:aws:s3:us-east-1:123456789012:accesspoint/tenant-a-access-point/obje
      }]
    }'
```

Avantages S3 Access Points:
- ✅ Politique d'accès dédiée par tenant
- ✅ Isolation simplifiée
- ✅ Aucune modification de la politique bucket principale

3. Journalisation et Surveillance

3.1 Activer S3 Server Access Logs

```
# Activer les logs d'accès
aws s3api put-bucket-logging \
    --bucket my-saas-bucket \
    --bucket-logging-status '{
      "LoggingEnabled": {
        "TargetBucket": "my-s3-access-logs",
        "TargetPrefix": "logs/my-saas-bucket/"
      }
    }'
```

3.2 S3 Event Notifications

```
# Configurer des notifications pour les modifications d'objets
aws s3api put-bucket-notification-configuration \
    --bucket my-saas-bucket \
    --notification-configuration '{
      "LambdaFunctionConfigurations": [{
        "Id": "ObjectCreatedAlert",
        "LambdaFunctionArn": "arn:aws:lambda:us-east-1:123456789012:function:S3SecurityAudit",
        "Events": ["s3:ObjectCreated:*", "s3:ObjectRemoved:*"]
      }]
    }'
```

3.3 CloudTrail pour Audit S3

```
# Activer les événements de données S3 dans CloudTrail
aws cloudtrail put-event-selectors \
    --trail-name MyTrail \
    --event-selectors '[{
      "ReadWriteType": "All",
      "IncludeManagementEvents": true,
      "DataResources": [{
        "Type": "AWS::S3::Object",
        "Values": ["arn:aws:s3:::my-saas-bucket/*"]
      }]
    }]'
```

4. Lifecycle et Gouvernance

4.1 Politique de Lifecycle

```
{
  "Rules": [
    {
      "Id": "TransitionToInfrequentAccess",
      "Status": "Enabled",
      "Filter": {
        "Prefix": "archives/"
      },
      "Transitions": [
        {
          "Days": 30,
          "StorageClass": "STANDARD_IA"
        },
        {
          "Days": 90,
          "StorageClass": "GLACIER"
        }
      ]
    },
    {
      "Id": "DeleteOldVersions",
      "Status": "Enabled",
      "NoncurrentVersionExpiration": {
        "NoncurrentDays": 90
      }
    }
  ]
}
```

```
aws s3api put-bucket-lifecycle-configuration \
    --bucket my-saas-bucket \
    --lifecycle-configuration file://lifecycle.json
```

Sécurité Amazon RDS

1. Chiffrement

1.1 Chiffrement au Repos

```
# Créer une instance RDS avec chiffrement
aws rds create-db-instance \
    --db-instance-identifier prod-database \
    --db-instance-class db.r6g.xlarge \
    --engine postgres \
    --master-username admin \
    --master-user-password $(aws secretsmanager get-secret-value --secret-id prod/db/password
    --allocated-storage 100 \
    --storage-encrypted \
    --kms-key-id arn:aws:kms:us-east-1:123456789012:key/xxxxx \
    --vpc-security-group-ids sg-xxxxx \
    --db-subnet-group-name private-db-subnet-group \
    --multi-az \
    --backup-retention-period 30 \
    --preferred-backup-window "03:00-04:00" \
    --preferred-maintenance-window "mon:04:00-mon:05:00"
```

Important: Le chiffrement ne peut être activé **qu'à la création**. Pour chiffrer une base existante :

1. Créer un snapshot

2. Copier le snapshot avec chiffrement
3. Restaurer une nouvelle instance depuis le snapshot chiffré

1.2 Chiffrement en Transit (SSL/TLS)

```
-- PostgreSQL: Forcer SSL
ALTER SYSTEM SET ssl = on;
ALTER SYSTEM SET ssl_min_protocol_version = 'TLSv1.2';

-- Vérifier les connexions SSL
SELECT datname, usename, client_addr, ssl, cipher
FROM pg_stat_ssl
JOIN pg_stat_activity ON pg_stat_ssl.pid = pg_stat_activity.pid;
```

```
# Télécharger le certificat RDS
wget https://truststore.pki.rds.amazonaws.com/global/global-bundle.pem

# Connexion PostgreSQL avec SSL
psql "host=mydb.xxxxx.us-east-1.rds.amazonaws.com port=5432 dbname=mydb user=admin sslmode=ve
```

2. Isolation Réseau

2.1 Sous-Réseaux Privés UNIQUEMENT

```
# Terraform - DB Subnet Group dans sous-réseaux privés
resource "aws_db_subnet_group" "private_db_subnet" {
  name       = "private-db-subnet-group"
  subnet_ids = [
    aws_subnet.private_db_1.id,
    aws_subnet.private_db_2.id
  ]

  tags = {
    Name = "Private DB Subnet Group"
  }
}

# Security Group - Accès uniquement depuis l'application tier
resource "aws_security_group" "rds_sg" {
  name        = "rds-security-group"
  description = "Allow access from application tier only"
  vpc_id      = aws_vpc.main.id

  ingress {
    description     = "PostgreSQL from App Tier"
    from_port       = 5432
    to_port         = 5432
    protocol        = "tcp"
    security_groups = [aws_security_group.app_tier.id]
  }

  egress {
    description = "No outbound required"
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

3. Sauvegardes et Récupération

3.1 Sauvegardes Automatiques

```
# Modifier la période de rétention des sauvegardes
aws rds modify-db-instance \
    --db-instance-identifier prod-database \
    --backup-retention-period 30 \
    --preferred-backup-window "03:00-04:00" \
    --apply-immediately
```

3.2 Snapshots Manuels

```
# Créer un snapshot manuel
aws rds create-db-snapshot \
    --db-instance-identifier prod-database \
    --db-snapshot-identifier prod-db-snapshot-$(date +%Y%m%d-%H%M%S)

# Copier vers une autre région (DR)
aws rds copy-db-snapshot \
    --source-db-snapshot-identifier arn:aws:rds:us-east-1:123456789012:snapshot:prod-db-snaps
    --target-db-snapshot-identifier prod-db-snapshot-20251115-dr \
    --source-region us-east-1 \
    --region eu-west-1 \
    --kms-key-id arn:aws:kms:eu-west-1:123456789012:key/yyyyy
```

3.3 Point-in-Time Recovery

```
# Restaurer à un point dans le temps spécifique
aws rds restore-db-instance-to-point-in-time \
    --source-db-instance-identifier prod-database \
    --target-db-instance-identifier prod-database-restored \
    --restore-time 2025-11-15T10:30:00Z \
    --db-subnet-group-name private-db-subnet-group \
    --vpc-security-group-ids sg-xxxxx
```

4. Gestion des Secrets avec AWS Secrets Manager

```
import boto3
import json
import psycopg2

def get_db_connection():
    """Obtenir une connexion DB en récupérant les credentials depuis Secrets Manager"""
    client = boto3.client('secretsmanager')

    try:
        response = client.get_secret_value(SecretId='prod/database/credentials')
        secret = json.loads(response['SecretString'])

        # Connexion PostgreSQL
        conn = psycopg2.connect(
            host=secret['host'],
            port=secret['port'],
            database=secret['dbname'],
            user=secret['username'],
            password=secret['password'],
            sslmode='verify-full',
            sslrootcert='/path/to/global-bundle.pem'
        )

        return conn
    except Exception as e:
        print(f"Error connecting to database: {e}")
        raise
```

4.1 Rotation Automatique des Mots de Passe

```
# Activer la rotation automatique (tous les 30 jours)
aws secretsmanager rotate-secret \
    --secret-id prod/database/credentials \
    --rotation-lambda-arn arn:aws:lambda:us-east-1:123456789012:function:RDSPasswordRotation \
    --rotation-rules AutomaticallyAfterDays=30
```

5. Monitoring et Audit

5.1 Enhanced Monitoring

```
# Activer Enhanced Monitoring
aws rds modify-db-instance \
    --db-instance-identifier prod-database \
    --monitoring-interval 60 \
    --monitoring-role-arn arn:aws:iam::123456789012:role/rds-monitoring-role
```

5.2 Performance Insights

```
# Activer Performance Insights
aws rds modify-db-instance \
    --db-instance-identifier prod-database \
    --enable-performance-insights \
    --performance-insights-retention-period 7
```

Sécurité API Gateway

1. Authentification et Autorisation

1.1 Architecture de Sécurité Multi-Couches

```
┌─────────────────────────────────────────────────────────┐
│                           Client                          │
└─────────────────────┬───────────────────────────────────┘
                      │
                      │
            ┌─────────▼─────────┐
            │      AWS WAF      │  ← Filtrage Layer 7
            │  (Rate Limiting,  │
            │   SQL Injection)  │
            └─────────┬─────────┘
                      │
                      │
            ┌─────────▼─────────┐
            │   API Gateway     │
            │                   │
            │  • API Keys       │  ← Authentification
            │  • Cognito        │
            │  • Lambda Authorizer│
            └─────────┬─────────┘
                      │
                      │
            ┌─────────▼─────────┐
            │   IAM Authorization │  ← Autorisation fine
            └─────────┬─────────┘
                      │
                      │
            ┌─────────▼─────────┐
            │   Backend         │
            │  (Lambda/ECS)     │
            └───────────────────┘
```

1.2 Cognito User Pools

```
# CloudFormation - API Gateway avec Cognito
Resources:
  MyApi:
    Type: AWS::ApiGatewayV2::Api
    Properties:
      Name: SecureAPI
      ProtocolType: HTTP

  CognitoAuthorizer:
    Type: AWS::ApiGatewayV2::Authorizer
    Properties:
      Name: CognitoAuthorizer
      ApiId: !Ref MyApi
      AuthorizerType: JWT
      IdentitySource:
        - $request.header.Authorization
      JwtConfiguration:
        Audience:
          - !Ref UserPoolClient
        Issuer: !Sub https://cognito-idp.${AWS::Region}.amazonaws.com/${UserPool}

  SecureRoute:
    Type: AWS::ApiGatewayV2::Route
    Properties:
      ApiId: !Ref MyApi
      RouteKey: "GET /secure"
      AuthorizationType: JWT
      AuthorizerId: !Ref CognitoAuthorizer
      Target: !Sub integrations/${MyIntegration}
```

1.3 Lambda Authorizer Personnalisé

```
import json

def lambda_handler(event, context):
    """Lambda Authorizer avec validation de token JWT custom"""
    token = event['authorizationToken']

    # Valider le token (ex: JWT, API key custom, etc.)
    if validate_token(token):
        principal_id = get_principal_id(token)
        tenant_id = get_tenant_id(token)

        # Générer la politique IAM
        policy = generate_policy(principal_id, 'Allow', event['methodArn'], tenant_id)

        # Ajouter le context (disponible dans le backend)
        policy['context'] = {
            'tenantId': tenant_id,
            'userId': principal_id,
            'permissions': get_user_permissions(principal_id)
        }

        return policy
    else:
        # Token invalide - refuser
        return generate_policy('user', 'Deny', event['methodArn'], None)
```

```
def generate_policy(principal_id, effect, resource, tenant_id):
    """Générer une politique IAM"""
    auth_response = {
        'principalId': principal_id
    }

    if effect and resource:
        policy_document = {
            'Version': '2012-10-17',
            'Statement': [{
                'Action': 'execute-api:Invoke',
                'Effect': effect,
                'Resource': resource
            }]
        }
        auth_response['policyDocument'] = policy_document

    return auth_response

def validate_token(token):
    """Valider le token (implémenter votre logique)"""
    # Ex: Vérifier signature JWT, expiration, issuer, etc.
    return True  # Placeholder

def get_principal_id(token):
    """Extraire l'ID utilisateur du token"""
    return "user-123"  # Placeholder

def get_tenant_id(token):
    """Extraire l'ID tenant du token"""
    return "tenant-a"  # Placeholder

def get_user_permissions(user_id):
    """Récupérer les permissions utilisateur"""
    return "read,write"  # Placeholder
```

2. AWS WAF pour API Gateway

2.1 Configuration WAF

```
# Créer un Web ACL
aws wafv2 create-web-acl \
    --name api-gateway-waf \
    --scope REGIONAL \
    --default-action Allow={} \
    --rules file://waf-rules.json \
    --visibility-config SampledRequestsEnabled=true,CloudWatchMetricsEnabled=true,MetricName=A

# Associer au API Gateway
aws wafv2 associate-web-acl \
    --web-acl-arn arn:aws:wafv2:us-east-1:123456789012:regional/webacl/api-gateway-waf/xxxxx \
    --resource-arn arn:aws:apigateway:us-east-1::/restapis/xxxxx/stages/prod
```

2.2 Règles WAF Essentielles

```
{
  "Name": "RateLimitRule",
  "Priority": 1,
  "Statement": {
    "RateBasedStatement": {
      "Limit": 2000,
      "AggregateKeyType": "IP"
    }
  },
  "Action": {
    "Block": {}
  },
  "VisibilityConfig": {
    "SampledRequestsEnabled": true,
    "CloudWatchMetricsEnabled": true,
    "MetricName": "RateLimitRule"
  }
}
```

3. Throttling et Quotas

```
# Créer un Usage Plan
aws apigateway create-usage-plan \
    --name "Standard-Plan" \
    --throttle burstLimit=500,rateLimit=1000 \
    --quota limit=100000,period=DAY \
    --api-stages apiId=xxxxx,stage=prod

# Créer une API Key
aws apigateway create-api-key \
    --name "TenantA-API-Key" \
    --enabled

# Associer l'API Key au Usage Plan
aws apigateway create-usage-plan-key \
    --usage-plan-id xxxxx \
    --key-id yyyyy \
    --key-type API_KEY
```

4. Logs et Monitoring

4.1 Activer CloudWatch Logs

```
# Activer les logs d'exécution
aws apigateway update-stage \
    --rest-api-id xxxxx \
    --stage-name prod \
    --patch-operations \
        op=replace,path=/*/logging/dataTrace,value=true \
        op=replace,path=/*/logging/loglevel,value=INFO
```

4.2 Métriques CloudWatch Personnalisées

```
import boto3
import time

cloudwatch = boto3.client('cloudwatch')

def lambda_handler(event, context):
    start_time = time.time()

    try:
        # Traiter la requête
        result = process_request(event)

        # Métrique de succès
        cloudwatch.put_metric_data(
            Namespace='MyAPI/Requests',
            MetricData=[{
                'MetricName': 'SuccessfulRequests',
                'Value': 1,
                'Unit': 'Count',
                'Dimensions': [
                    {'Name': 'Endpoint', 'Value': event['resource']},
                    {'Name': 'TenantId', 'Value': event['requestContext']['authorizer']['tenar
                ]
            }]
        )

        return result
    except Exception as e:
        # Métrique d'erreur
        cloudwatch.put_metric_data(
            Namespace='MyAPI/Requests',
            MetricData=[{
                'MetricName': 'FailedRequests',
                'Value': 1,
                'Unit': 'Count'
            }]
        )
        raise
    finally:
        # Métrique de latence
        duration = (time.time() - start_time) * 1000
        cloudwatch.put_metric_data(
            Namespace='MyAPI/Performance',
            MetricData=[{
                'MetricName': 'RequestDuration',
                'Value': duration,
                'Unit': 'Milliseconds'
            }]
        )
```

Sécurité DynamoDB

1. Chiffrement

1.1 Chiffrement au Repos

```
# Créer une table avec chiffrement KMS
aws dynamodb create-table \
    --table-name UserData \
    --attribute-definitions \
        AttributeName=UserId,AttributeType=S \
        AttributeName=TenantId,AttributeType=S \
    --key-schema \
        AttributeName=TenantId,KeyType=HASH \
        AttributeName=UserId,KeyType=RANGE \
    --billing-mode PAY_PER_REQUEST \
    --sse-specification \
        Enabled=true,SSEType=KMS,KMSMasterKeyId=arn:aws:kms:us-east-1:123456789012:key/xxxxx
```

2. Point-in-Time Recovery (PITR)

```
# Activer PITR
aws dynamodb update-continuous-backups \
    --table-name UserData \
    --point-in-time-recovery-specification PointInTimeRecoveryEnabled=true

# Restaurer à un point dans le temps
aws dynamodb restore-table-to-point-in-time \
    --source-table-name UserData \
    --target-table-name UserData-Restored \
    --restore-date-time 2025-11-15T10:00:00Z
```

3. Contrôle d'Accès Fine-Grained

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:Query",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem"
      ],
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/UserData",
      "Condition": {
        "ForAllValues:StringEquals": {
          "dynamodb:LeadingKeys": ["${aws:PrincipalTag/TenantID}"]
        }
      }
    }
  ]
}
```

Cette politique permet aux utilisateurs d'accéder uniquement aux items dont la partition key correspond à leur TenantID.

4. DynamoDB Streams pour Audit

```
import boto3
import json

def lambda_handler(event, context):
    """Auditeur DynamoDB Streams"""
    for record in event['Records']:
        if record['eventName'] == 'REMOVE':
            # Enregistrer la suppression pour audit
            log_deletion(record)
        elif record['eventName'] in ['INSERT', 'MODIFY']:
            # Détecter les modifications sensibles
            detect_sensitive_changes(record)

def log_deletion(record):
    """Logger les suppressions pour audit"""
    print(f"Item deleted: {record['dynamodb']['Keys']}")
    # Envoyer à CloudWatch Logs, S3, etc.

def detect_sensitive_changes(record):
    """Détecter les modifications de données sensibles"""
    if 'NewImage' in record['dynamodb']:
        new_image = record['dynamodb']['NewImage']
        if 'email' in new_image or 'phone' in new_image:
            print(f"Sensitive data modified: {record['dynamodb']['Keys']}")
            # Alerter l'équipe sécurité
```

Scénarios d'Attaque et Détection

Scénario 1: S3 Bucket Hijacking via Subdomain Takeover

Contexte:
Un attaquant trouve un enregistrement DNS pointant vers un bucket S3 qui n'existe plus ou est mal configuré.

Attack Vector:

```
# L'attaquant découvre un sous-domaine orphelin
dig files.example.com
# Réponse: files.example.com CNAME files-example-bucket.s3.amazonaws.com

# Le bucket n'existe pas - l'attaquant le crée
aws s3 mb s3://files-example-bucket --region us-east-1

# L'attaquant héberge du contenu malveillant
echo "<html><body>Malicious content</body></html>" > index.html
aws s3 cp index.html s3://files-example-bucket/ --acl public-read
aws s3 website s3://files-example-bucket/ --index-document index.html
```

Prévention:

1. **Supprimer les enregistrements DNS immédiatement lors de la suppression d'un bucket**
2. **Réserver les noms de buckets** même après suppression (créer un bucket vide)
3. **Monitoring CloudTrail** pour détecter les créations de buckets avec des noms similaires

```
# Query CloudWatch Logs Insights pour détecter
fields @timestamp, requestParameters.bucketName, userIdentity.arn
| filter eventName = "CreateBucket"
| filter requestParameters.bucketName like /^(files|assets|static|media)/
| stats count(*) as bucketCreations by requestParameters.bucketName, userIdentity.accountId
| filter userIdentity.accountId != "YOUR_ACCOUNT_ID"
```

Détection:

```
import boto3

def audit_bucket_dns_alignment():
    """Vérifier que tous les buckets ont des enregistrements DNS valides"""
    s3 = boto3.client('s3')
    route53 = boto3.client('route53')

    buckets = s3.list_buckets()['Buckets']

    for bucket in buckets:
        bucket_name = bucket['Name']

        # Vérifier si un enregistrement DNS pointe vers ce bucket
        # Si oui, s'assurer que le bucket est propriété du compte
        # Sinon, alerter

        try:
            bucket_acl = s3.get_bucket_acl(Bucket=bucket_name)
            # Vérifier ownership
        except Exception as e:
            print(f"Bucket {bucket_name} access issue: {e}")
```

Scénario 2: SQL Injection via RDS

Contexte:
Une application vulnérable permet l'injection SQL, exposant les données RDS.

Attack Example:

```
# ❌ CODE VULNÉRABLE
def get_user(user_id):
    query = f"SELECT * FROM users WHERE id = '{user_id}'"
    cursor.execute(query)
    return cursor.fetchone()

# Attaque: user_id = "1' OR '1'='1"
# Query résultante: SELECT * FROM users WHERE id = '1' OR '1'='1'
# Résultat: Tous les utilisateurs retournés
```

Protection:

```
# ✅ CODE SÉCURISÉ - Prepared Statements
def get_user(user_id):
    query = "SELECT * FROM users WHERE id = %s"
    cursor.execute(query, (user_id,))
    return cursor.fetchone()
```

Détection via RDS Performance Insights:

```sql
-- Détecter les requêtes SQL suspectes
SELECT
    query_text,
    calls,
    total_time,
    mean_time
FROM pg_stat_statements
WHERE query_text LIKE '%OR%=%'
   OR query_text LIKE '%UNION%SELECT%'
   OR query_text LIKE '%DROP%TABLE%'
   OR query_text LIKE '%--'
   OR query_text LIKE '%/*'
ORDER BY calls DESC
LIMIT 20;
```

Monitoring CloudWatch:

```python
import boto3
import re

cloudwatch = boto3.client('cloudwatch')
logs = boto3.client('logs')

def detect_sql_injection_attempts():
    """Détecter les tentatives d'injection SQL dans les logs RDS"""

    sql_injection_patterns = [
        r"(\%27)|(\')|(\-\-)|(\%23)|(#)",  # Single quote, comment
        r"((\%3D)|(=))[^\n]*((\%27)|(\')|(\-\-)|(\%3B)|(;))",  # SQL operators
        r"\w*((\%27)|(\'))((\%6F)|o|(\%4F))((\%72)|r|(\%52))",  # OR keyword
        r"((\%27)|(\'))union",  # UNION keyword
    ]

    response = logs.filter_log_events(
        logGroupName='/aws/rds/instance/prod-database/postgresql',
        filterPattern='ERROR',
        limit=100
    )

    for event in response.get('events', []):
        message = event['message']
        for pattern in sql_injection_patterns:
            if re.search(pattern, message, re.IGNORECASE):
                # Alerte injection SQL détectée
                print(f"SQL Injection detected: {message}")
                send_security_alert("SQL Injection Attempt", message)
```

Scénario 3: API Abuse - Credential Stuffing

Contexte:

Un attaquant utilise des credentials volés pour tester l'accès à l'API.

Attack Pattern:

```
# L'attaquant utilise un dictionnaire de credentials
for cred in credentials_list:
    curl -X POST https://api.example.com/auth/login \
      -H "Content-Type: application/json" \
      -d "{\"email\":\"${cred.email}\",\"password\":\"${cred.password}\"}"
```

Détection:

```
# Lambda Authorizer avec détection de brute force
import boto3
from datetime import datetime, timedelta

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('LoginAttempts')

def lambda_handler(event, context):
    """Détection credential stuffing"""
    source_ip = event['requestContext']['identity']['sourceIp']
    current_time = datetime.utcnow()

    # Compter les tentatives depuis cette IP dans les 5 dernières minutes
    response = table.query(
        KeyConditionExpression='SourceIP = :ip AND Timestamp > :time',
        ExpressionAttributeValues={
            ':ip': source_ip,
            ':time': (current_time - timedelta(minutes=5)).isoformat()
        }
    )

    attempts = response['Count']

    if attempts > 10:
        # Bloquer temporairement cette IP
        return {
            'principalId': 'blocked',
            'policyDocument': {
                'Version': '2012-10-17',
                'Statement': [{
                    'Action': 'execute-api:Invoke',
                    'Effect': 'Deny',
                    'Resource': event['methodArn']
                }]
            },
            'context': {
                'reason': 'Too many login attempts'
            }
        }

    # Logger la tentative
    table.put_item(Item={
        'SourceIP': source_ip,
        'Timestamp': current_time.isoformat(),
        'UserAgent': event['requestContext']['identity']['userAgent']
    })

    # Continuer avec l'authentification normale
    return validate_credentials(event)
```

WAF Rule pour Credential Stuffing:

```
{
  "Name": "CredentialStuffingProtection",
  "Priority": 2,
  "Statement": {
    "RateBasedStatement": {
      "Limit": 100,
      "AggregateKeyType": "IP",
      "ScopeDownStatement": {
        "ByteMatchStatement": {
          "SearchString": "/auth/login",
          "FieldToMatch": {
            "UriPath": {}
          },
          "TextTransformations": [{
            "Priority": 0,
            "Type": "LOWERCASE"
          }],
          "PositionalConstraint": "CONTAINS"
        }
      }
    }
  },
  "Action": {
    "Block": {
      "CustomResponse": {
        "ResponseCode": 429,
        "CustomResponseBodyKey": "TooManyRequests"
      }
    }
  }
}
```

Scénario 4: DynamoDB Data Exfiltration

Contexte:

Un utilisateur compromis tente d'exfiltrer toutes les données de la table.

Attack:

```
# L'attaquant effectue un scan complet
import boto3

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('UserData')

# Scan all items (inefficient but bypasses query restrictions)
response = table.scan()
items = response['Items']

# Continuer avec pagination
while 'LastEvaluatedKey' in response:
    response = table.scan(ExclusiveStartKey=response['LastEvaluatedKey'])
    items.extend(response['Items'])

# Exfiltrer les données
with open('exfiltrated_data.json', 'w') as f:
    json.dump(items, f)
```

Prévention:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PreventFullTableScans",
      "Effect": "Deny",
      "Action": "dynamodb:Scan",
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/UserData",
      "Condition": {
        "StringNotEquals": {
          "aws:userid": "ADMIN_USER_ID"
        }
      }
    }
  ]
}
```

Détection CloudWatch:

```
# Détecter les scans complets de tables
fields @timestamp, userIdentity.arn, requestParameters.tableName, responseElements.scannedCou
```

```
| filter eventName = "Scan"
| filter responseElements.scannedCount > 1000
| stats sum(responseElements.scannedCount) as totalScanned by userIdentity.arn, requestParame
| sort totalScanned desc
```

Alarme CloudWatch:

```
aws logs put-metric-filter \
    --log-group-name /aws/cloudtrail/logs \
    --filter-name DynamoDB-Large-Scans \
    --filter-pattern '{($.eventName = Scan) && ($.responseElements.scannedCount > 5000)}' \
    --metric-transformations \
        metricName=DynamoDBLargeScans,metricNamespace=SecurityMetrics,metricValue=1

aws cloudwatch put-metric-alarm \
    --alarm-name DynamoDB-Exfiltration-Detected \
    --alarm-description "Large DynamoDB scan detected" \
    --metric-name DynamoDBLargeScans \
    --namespace SecurityMetrics \
    --statistic Sum \
    --period 300 \
    --evaluation-periods 1 \
    --threshold 3 \
    --comparison-operator GreaterThanThreshold \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:SecurityAlerts
```

Architectures Complètes (Terraform)

1. Multi-Tenant S3 avec Access Points

```
# s3_multi_tenant.tf
resource "aws_kms_key" "s3_tenant_data" {
  description             = "KMS key for multi-tenant S3 data"
  deletion_window_in_days = 30
  enable_key_rotation     = true

  tags = {
    Name        = "s3-tenant-data-key"
    Environment = "production"
  }
}

resource "aws_s3_bucket" "multi_tenant_data" {
  bucket = "my-saas-multi-tenant-data"

  tags = {
    Name        = "Multi-Tenant Data Bucket"
    Environment = "production"
  }
}

resource "aws_s3_bucket_versioning" "multi_tenant_data" {
  bucket = aws_s3_bucket.multi_tenant_data.id

  versioning_configuration {
    status     = "Enabled"
    mfa_delete = "Disabled"  # Enable with root account + MFA
  }
}

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

  rule {
    apply_server_side_encryption_by_default {
      sse_algorithm     = "aws:kms"
      kms_master_key_id = aws_kms_key.s3_tenant_data.arn
    }
    bucket_key_enabled = true
  }
}

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

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

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

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Sid    = "DenyInsecureTransport"
        Effect = "Deny"
        Principal = "*"
        Action = "s3:*"
        Resource = [
          aws_s3_bucket.multi_tenant_data.arn,
          "${aws_s3_bucket.multi_tenant_data.arn}/*"
        ]
        Condition = {
          Bool = {
            "aws:SecureTransport" = "false"
          }
        }
      },
      {
        Sid    = "DenyUnencryptedObjectUploads"
        Effect = "Deny"
        Principal = "*"
        Action = "s3:PutObject"
        Resource = "${aws_s3_bucket.multi_tenant_data.arn}/*"
        Condition = {
          StringNotEquals = {
            "s3:x-amz-server-side-encryption" = "aws:kms"
          }
        }
      }
    ]
  })
}

# S3 Access Point per Tenant
resource "aws_s3_access_point" "tenant_a" {
  bucket = aws_s3_bucket.multi_tenant_data.id
  name   = "tenant-a-access-point"

  vpc_configuration {
    vpc_id = aws_vpc.main.id
  }

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Principal = {
        AWS = aws_iam_role.tenant_a_role.arn
      }
      Action = [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ]
      Resource = "arn:aws:s3:${data.aws_region.current.name}:${data.aws_caller_identity.curre
    }]
  })
}

resource "aws_s3_access_point" "tenant_b" {
  bucket = aws_s3_bucket.multi_tenant_data.id
  name   = "tenant-b-access-point"

  vpc_configuration {
    vpc_id = aws_vpc.main.id
  }

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Principal = {
        AWS = aws_iam_role.tenant_b_role.arn
      }
      Action = [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ]
      Resource = "arn:aws:s3:${data.aws_region.current.name}:${data.aws_caller_identity.curren
    }]
  })
}

# S3 Object Lambda for data masking
resource "aws_s3_object_lambda_access_point" "data_masking" {
  name = "data-masking-access-point"

  configuration {
    supporting_access_point = aws_s3_access_point.tenant_a.arn

    transformation_configuration {
      actions = ["GetObject"]

      content_transformation {
        aws_lambda {
          function_arn = aws_lambda_function.data_masking.arn
        }
      }
    }
  }
}

resource "aws_lambda_function" "data_masking" {
  filename         = "data_masking.zip"
  function_name    = "s3-object-lambda-data-masking"
  role             = aws_iam_role.object_lambda_role.arn
  handler          = "index.handler"
  source_code_hash = filebase64sha256("data_masking.zip")
  runtime          = "python3.11"

  environment {
    variables = {
      MASK_FIELDS = "ssn,credit_card,email"
    }
  }
}

# S3 Lifecycle Rules
resource "aws_s3_bucket_lifecycle_configuration" "multi_tenant_data" {
  bucket = aws_s3_bucket.multi_tenant_data.id

  rule {
    id     = "transition-to-ia"
    status = "Enabled"

    filter {
      prefix = "tenant-*/"
    }

    transition {
      days          = 30
      storage_class = "STANDARD_IA"
    }

    transition {
      days          = 90
      storage_class = "GLACIER_IR"
    }

    transition {
      days          = 180
      storage_class = "DEEP_ARCHIVE"
    }
  }

  rule {
    id     = "expire-old-versions"
    status = "Enabled"

    noncurrent_version_expiration {
      noncurrent_days = 90
    }

    noncurrent_version_transition {
      noncurrent_days = 30
      storage_class   = "STANDARD_IA"
    }
  }

  rule {
    id     = "delete-incomplete-multipart-uploads"
    status = "Enabled"

    abort_incomplete_multipart_upload {
      days_after_initiation = 7
    }
  }
}

# S3 Event Notifications
resource "aws_s3_bucket_notification" "multi_tenant_data" {
  bucket = aws_s3_bucket.multi_tenant_data.id

  lambda_function {
    lambda_function_arn = aws_lambda_function.s3_security_audit.arn
    events              = ["s3:ObjectCreated:*", "s3:ObjectRemoved:*"]
    filter_prefix       = "tenant-"
  }

  lambda_function {
    lambda_function_arn = aws_lambda_function.malware_scanner.arn
    events              = ["s3:ObjectCreated:Put", "s3:ObjectCreated:Post"]
    filter_suffix       = ".exe"
  }
}

resource "aws_lambda_permission" "allow_s3_security_audit" {
  statement_id  = "AllowS3Invoke"
  action        = "lambda:InvokeFunction"
  function_name = aws_lambda_function.s3_security_audit.function_name
  principal     = "s3.amazonaws.com"
  source_arn    = aws_s3_bucket.multi_tenant_data.arn
}
```

2. RDS Production Setup avec Read Replicas

```
# rds_production.tf
resource "aws_kms_key" "rds" {
  description             = "KMS key for RDS encryption"
  deletion_window_in_days = 30
  enable_key_rotation     = true

  tags = {
    Name = "rds-encryption-key"
  }
}

resource "aws_db_subnet_group" "private_db" {
  name       = "private-db-subnet-group"
  subnet_ids = [
    aws_subnet.private_db_1.id,
    aws_subnet.private_db_2.id,
    aws_subnet.private_db_3.id
  ]

  tags = {
    Name = "Private DB Subnet Group"
  }
}

resource "aws_db_parameter_group" "postgres_secure" {
  name   = "postgres-secure-params"
  family = "postgres15"

  parameter {
    name  = "ssl"
    value = "1"
  }

  parameter {
    name  = "ssl_min_protocol_version"
    value = "TLSv1.2"
  }

  parameter {
    name  = "log_connections"
    value = "1"
  }

  parameter {
    name  = "log_disconnections"
    value = "1"
  }

  parameter {
    name  = "log_duration"
    value = "1"
  }

  parameter {
    name  = "log_statement"
    value = "ddl"
  }

  parameter {
    name  = "rds.force_ssl"
    value = "1"
  }

  tags = {
    Name = "PostgreSQL Secure Parameters"
  }
}

resource "aws_db_instance" "primary" {
  identifier     = "prod-postgresql-primary"
  engine         = "postgres"
  engine_version = "15.4"
  instance_class = "db.r6g.2xlarge"

  allocated_storage     = 500
  max_allocated_storage = 2000
  storage_type          = "gp3"
  storage_encrypted     = true
  kms_key_id            = aws_kms_key.rds.arn

  db_name  = "production"
  username = "dbadmin"
  # Password should be managed via Secrets Manager
  manage_master_user_password = true

  db_subnet_group_name   = aws_db_subnet_group.private_db.name
  vpc_security_group_ids = [aws_security_group.rds.id]
  parameter_group_name   = aws_db_parameter_group.postgres_secure.name

  multi_az               = true
  publicly_accessible    = false
  deletion_protection    = true
  skip_final_snapshot    = false
  final_snapshot_identifier = "prod-postgresql-final-snapshot-${formatdate("YYYY-MM-DD-hhmm",
  backup_retention_period   = 30
  backup_window             = "03:00-04:00"
  maintenance_window        = "mon:04:00-mon:05:00"
  copy_tags_to_snapshot     = true

  enabled_cloudwatch_logs_exports = [
    "postgresql",
    "upgrade"
  ]

  performance_insights_enabled    = true
  performance_insights_kms_key_id = aws_kms_key.rds.arn
  performance_insights_retention_period = 7

  monitoring_interval = 60
  monitoring_role_arn = aws_iam_role.rds_monitoring.arn

  tags = {
    Name        = "Production PostgreSQL Primary"
    Environment = "production"
    Backup      = "required"
  }
}

# Read Replica 1
resource "aws_db_instance" "read_replica_1" {
  identifier     = "prod-postgresql-read-replica-1"
  replicate_source_db = aws_db_instance.primary.identifier
  instance_class = "db.r6g.xlarge"

  publicly_accessible = false
  skip_final_snapshot = true

  performance_insights_enabled = true
  performance_insights_kms_key_id = aws_kms_key.rds.arn

  monitoring_interval = 60
  monitoring_role_arn = aws_iam_role.rds_monitoring.arn

  tags = {
    Name        = "Production PostgreSQL Read Replica 1"
    Environment = "production"
    Role        = "read-replica"
  }
}

# Read Replica 2 (autre AZ)
resource "aws_db_instance" "read_replica_2" {
  identifier          = "prod-postgresql-read-replica-2"
  replicate_source_db = aws_db_instance.primary.identifier
  instance_class      = "db.r6g.xlarge"

  availability_zone   = "us-east-1c"
  publicly_accessible = false
  skip_final_snapshot = true

  performance_insights_enabled    = true
  performance_insights_kms_key_id = aws_kms_key.rds.arn

  monitoring_interval = 60
  monitoring_role_arn = aws_iam_role.rds_monitoring.arn

  tags = {
    Name        = "Production PostgreSQL Read Replica 2"
    Environment = "production"
    Role        = "read-replica"
  }
}

# RDS Proxy for connection pooling
resource "aws_db_proxy" "main" {
  name                   = "prod-postgresql-proxy"
  engine_family          = "POSTGRESQL"
  auth {
    auth_scheme = "SECRETS"
    iam_auth    = "REQUIRED"
    secret_arn  = aws_secretsmanager_secret.db_credentials.arn
  }

  role_arn               = aws_iam_role.rds_proxy.arn
  vpc_subnet_ids         = [
    aws_subnet.private_db_1.id,
    aws_subnet.private_db_2.id
  ]

  require_tls = true

  tags = {
    Name = "Production RDS Proxy"
  }
}

resource "aws_db_proxy_default_target_group" "main" {
  db_proxy_name = aws_db_proxy.main.name

  connection_pool_config {
    connection_borrow_timeout    = 120
    max_connections_percent      = 100
    max_idle_connections_percent = 50
  }
}

resource "aws_db_proxy_target" "primary" {
  db_proxy_name          = aws_db_proxy.main.name
  target_group_name      = aws_db_proxy_default_target_group.main.name
  db_instance_identifier = aws_db_instance.primary.identifier
}

# CloudWatch Alarms for RDS
```

```
resource "aws_cloudwatch_metric_alarm" "rds_cpu" {
  alarm_name          = "rds-primary-high-cpu"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods  = 2
  metric_name         = "CPUUtilization"
  namespace           = "AWS/RDS"
  period              = 300
  statistic           = "Average"
  threshold           = 80

  dimensions = {
    DBInstanceIdentifier = aws_db_instance.primary.identifier
  }

  alarm_actions = [aws_sns_topic.rds_alerts.arn]
}

resource "aws_cloudwatch_metric_alarm" "rds_storage" {
  alarm_name          = "rds-primary-low-storage"
  comparison_operator = "LessThanThreshold"
  evaluation_periods  = 1
  metric_name         = "FreeStorageSpace"
  namespace           = "AWS/RDS"
  period              = 300
  statistic           = "Average"
  threshold           = 10737418240  # 10 GB

  dimensions = {
    DBInstanceIdentifier = aws_db_instance.primary.identifier
  }

  alarm_actions = [aws_sns_topic.rds_alerts.arn]
}

resource "aws_cloudwatch_metric_alarm" "rds_connections" {
  alarm_name          = "rds-primary-high-connections"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods  = 2
  metric_name         = "DatabaseConnections"
  namespace           = "AWS/RDS"
  period              = 300
  statistic           = "Average"
  threshold           = 200

  dimensions = {
    DBInstanceIdentifier = aws_db_instance.primary.identifier
  }

  alarm_actions = [aws_sns_topic.rds_alerts.arn]
}
```

3. API Gateway Multi-Couches de Sécurité

```
# api_gateway_secure.tf
resource "aws_apigatewayv2_api" "main" {
  name          = "secure-saas-api"
  protocol_type = "HTTP"

  cors_configuration {
    allow_origins = ["https://app.example.com"]
    allow_methods = ["GET", "POST", "PUT", "DELETE", "OPTIONS"]
    allow_headers = ["Content-Type", "Authorization", "X-Tenant-ID"]
    max_age       = 3600
  }

  tags = {
    Name        = "Secure SaaS API"
    Environment = "production"
  }
}

# Custom Domain with ACM Certificate
resource "aws_apigatewayv2_domain_name" "main" {
  domain_name = "api.example.com"

  domain_name_configuration {
    certificate_arn = aws_acm_certificate.api.arn
    endpoint_type   = "REGIONAL"
    security_policy = "TLS_1_2"
  }
}

resource "aws_apigatewayv2_api_mapping" "main" {
  api_id      = aws_apigatewayv2_api.main.id
  domain_name = aws_apigatewayv2_domain_name.main.id
  stage       = aws_apigatewayv2_stage.prod.id
}

# WAF Web ACL
resource "aws_wafv2_web_acl" "api_gateway" {
  name  = "api-gateway-waf"
  scope = "REGIONAL"

  default_action {
    allow {}
  }

  # Rule 1: Rate Limiting
  rule {
    name     = "RateLimitRule"
    priority = 1

    action {
      block {
        custom_response {
          response_code = 429
        }
      }
    }

    statement {
      rate_based_statement {
        limit              = 2000
        aggregate_key_type = "IP"
      }
    }

    visibility_config {
      cloudwatch_metrics_enabled = true
      metric_name                = "RateLimitRule"
      sampled_requests_enabled   = true
    }
  }

  # Rule 2: SQL Injection Protection
  rule {
    name     = "SQLInjectionProtection"
    priority = 2

    override_action {
      none {}
    }

    statement {
      managed_rule_group_statement {
        name        = "AWSManagedRulesSQLiRuleSet"
        vendor_name = "AWS"
      }
    }

    visibility_config {
      cloudwatch_metrics_enabled = true
      metric_name                = "SQLInjectionProtection"
      sampled_requests_enabled   = true
    }
  }

  # Rule 3: Known Bad Inputs
  rule {
    name     = "KnownBadInputsProtection"
    priority = 3

    override_action {
      none {}
    }

    statement {
      managed_rule_group_statement {
        name        = "AWSManagedRulesKnownBadInputsRuleSet"
        vendor_name = "AWS"
      }
    }

    visibility_config {
      cloudwatch_metrics_enabled = true
      metric_name                = "KnownBadInputsProtection"
      sampled_requests_enabled   = true
    }
  }

  # Rule 4: Geographic Blocking (optional)
  rule {
    name     = "GeoBlockingRule"
    priority = 4

    action {
      block {}
    }

    statement {
      not_statement {
        statement {
          geo_match_statement {
            country_codes = ["US", "CA", "GB", "FR", "DE"]
          }
        }
      }
    }

    visibility_config {
      cloudwatch_metrics_enabled = true
      metric_name                = "GeoBlockingRule"
      sampled_requests_enabled   = true
    }
  }

  visibility_config {
    cloudwatch_metrics_enabled = true
    metric_name                = "ApiGatewayWAF"
    sampled_requests_enabled   = true
  }

  tags = {
    Name = "API Gateway WAF"
  }
}

resource "aws_wafv2_web_acl_association" "api_gateway" {
  resource_arn = aws_apigatewayv2_stage.prod.arn
  web_acl_arn  = aws_wafv2_web_acl.api_gateway.arn
}

# Cognito User Pool
resource "aws_cognito_user_pool" "main" {
  name = "saas-user-pool"
  password_policy {
    minimum_length    = 12
    require_lowercase = true
    require_uppercase = true
    require_numbers   = true
    require_symbols   = true
  }

  mfa_configuration = "OPTIONAL"

  software_token_mfa_configuration {
    enabled = true
  }

  account_recovery_setting {
    recovery_mechanism {
      name     = "verified_email"
      priority = 1
    }
  }

  user_attribute_update_settings {
    attributes_require_verification_before_update = ["email"]
  }

  schema {
    name                = "tenant_id"
    attribute_data_type = "String"
    mutable             = false
    required            = true

    string_attribute_constraints {
      min_length = 1
      max_length = 256
    }
  }

  tags = {
    Name = "SaaS User Pool"
  }
}

resource "aws_cognito_user_pool_client" "main" {
  name         = "saas-app-client"
  user_pool_id = aws_cognito_user_pool.main.id

  explicit_auth_flows = [
    "ALLOW_USER_SRP_AUTH",
    "ALLOW_REFRESH_TOKEN_AUTH"
  ]

  prevent_user_existence_errors = "ENABLED"

  access_token_validity  = 1  # 1 hour
  id_token_validity      = 1  # 1 hour
  refresh_token_validity = 30 # 30 days

  token_validity_units {
    access_token  = "hours"
    id_token      = "hours"
    refresh_token = "days"
  }
}

# Cognito Authorizer
resource "aws_apigatewayv2_authorizer" "cognito" {
  api_id           = aws_apigatewayv2_api.main.id
  authorizer_type  = "JWT"
  identity_sources = ["$request.header.Authorization"]
  name             = "cognito-authorizer"

  jwt_configuration {
    audience = [aws_cognito_user_pool_client.main.id]
    issuer   = "https://${aws_cognito_user_pool.main.endpoint}"
  }
}

# API Gateway Stage
resource "aws_apigatewayv2_stage" "prod" {
  api_id      = aws_apigatewayv2_api.main.id
  name        = "prod"
  auto_deploy = true

  access_log_settings {
    destination_arn = aws_cloudwatch_log_group.api_gateway.arn
    format = jsonencode({
      requestId      = "$context.requestId"
      ip             = "$context.identity.sourceIp"
      requestTime    = "$context.requestTime"
      httpMethod     = "$context.httpMethod"
      routeKey       = "$context.routeKey"
      status         = "$context.status"
      protocol       = "$context.protocol"
      responseLength = "$context.responseLength"
      errorMessage   = "$context.error.message"
      authorizerError = "$context.authorizer.error"
    })
  }

  default_route_settings {
    throttling_burst_limit = 5000
    throttling_rate_limit  = 10000
  }

  tags = {
    Name        = "Production Stage"
    Environment = "production"
  }
}
```

```
resource "aws_cloudwatch_log_group" "api_gateway" {
  name              = "/aws/apigateway/${aws_apigatewayv2_api.main.name}"
  retention_in_days = 30
  kms_key_id        = aws_kms_key.cloudwatch_logs.arn
}

# Lambda Integration with Authorizer
resource "aws_apigatewayv2_route" "get_users" {
  api_id    = aws_apigatewayv2_api.main.id
  route_key = "GET /users"

  authorization_type = "JWT"
  authorizer_id      = aws_apigatewayv2_authorizer.cognito.id
  target             = "integrations/${aws_apigatewayv2_integration.get_users.id}"
}

resource "aws_apigatewayv2_integration" "get_users" {
  api_id           = aws_apigatewayv2_api.main.id
  integration_type = "AWS_PROXY"

  integration_uri    = aws_lambda_function.get_users.arn
  integration_method = "POST"
}

# Usage Plan and API Keys
resource "aws_api_gateway_usage_plan" "standard" {
  name = "standard-usage-plan"

  api_stages {
    api_id = aws_apigatewayv2_api.main.id
    stage  = aws_apigatewayv2_stage.prod.name
  }

  quota_settings {
    limit  = 100000
    period = "DAY"
  }

  throttle_settings {
    burst_limit = 500
    rate_limit  = 1000
  }
}
```

Bonnes Pratiques Multi-Services

1. Defense-in-Depth

```
┌─────────────────────────────────────────────────────────┐
│  Layer 7: Application Logic Validation                  │
├─────────────────────────────────────────────────────────┤
│  Layer 6: Business Rules Authorization                  │
├─────────────────────────────────────────────────────────┤
│  Layer 5: IAM Policies (Resource-level)                 │
├─────────────────────────────────────────────────────────┤
│  Layer 4: API Gateway Authorizers                       │
├─────────────────────────────────────────────────────────┤
│  Layer 3: AWS WAF (Rate Limiting, SQL Injection)        │
├─────────────────────────────────────────────────────────┤
│  Layer 2: Network Security (Security Groups, NACLs)     │
├─────────────────────────────────────────────────────────┤
│  Layer 1: Encryption (Data at Rest & in Transit)        │
└─────────────────────────────────────────────────────────┘
```

2. Principe du Moindre Privilège

Chaque composant doit avoir uniquement les permissions nécessaires:

- **Lambda** → Accès DynamoDB spécifique (table + leading keys)
- **API Gateway** → Invocation Lambda spécifique
- **Application** → Accès S3 limité à un préfixe tenant

3. Surveillance Continue

```
# AWS Config Rules pour conformité
aws configservice put-config-rule --config-rule '{
  "ConfigRuleName": "s3-bucket-public-read-prohibited",
  "Source": {
    "Owner": "AWS",
    "SourceIdentifier": "S3_BUCKET_PUBLIC_READ_PROHIBITED"
  }
}'
```

Checklist Applications & Stockage

✅ Amazon S3 (Priorité Critique)

- [ ] **Block Public Access activé (compte + buckets)**
- [ ] **Chiffrement par défaut SSE-KMS**
- [ ] **HTTPS obligatoire (politique bucket)**
- [ ] **Versioning activé**
- [ ] **MFA Delete activé sur buckets critiques**
- [ ] **Access Logs activés**
- [ ] **CloudTrail data events activés**
- [ ] **Lifecycle policies configurées**

✅ Amazon RDS (Priorité Critique)

- [ ] **Chiffrement activé (KMS)**
- [ ] **SSL/TLS obligatoire**
- [ ] **Sous-réseaux privés uniquement**
- [ ] **Security Groups restrictifs**
- [ ] **Backup rétention ≥ 30 jours**
- [ ] **Multi-AZ activé (production)**
- [ ] **Enhanced Monitoring activé**
- [ ] **Performance Insights activé**
- [ ] **Secrets Manager pour credentials**
- [ ] **Rotation automatique des mots de passe**

✅ API Gateway (Priorité Critique)

- [ ] **Authentification configurée (Cognito/Lambda Authorizer)**
- [ ] **AWS WAF activé et configuré**
- [ ] **Rate Limiting configuré**
- [ ] **Usage Plans et Quotas**
- [ ] **CloudWatch Logs activés**
- [ ] **X-Ray tracing activé**

- [ ] **CORS configuré correctement**
- [ ] **API Keys pour clients B2B**

✅ DynamoDB (Priorité Importante)

- [ ] **Chiffrement KMS activé**
- [ ] **Point-in-Time Recovery (PITR) activé**
- [ ] **Fine-grained access control (IAM)**
- [ ] **DynamoDB Streams pour audit**
- [ ] **Auto Scaling configuré**
- [ ] **Backups automatiques**
- [ ] **CloudWatch Contributor Insights activé**

Références et Ressources

Documentation Officielle AWS

- S3 Security Best Practices
- RDS Encryption Best Practices
- API Gateway Security
- DynamoDB Security

Conclusion

La sécurisation des applications et du stockage AWS nécessite:

1. **Chiffrement systématique** au repos et en transit
2. **Isolation multi-tenant** avec politiques IAM et préfixes
3. **Authentification forte** avec Cognito ou Lambda Authorizers
4. **Protection WAF** contre les attaques Layer 7
5. **Audit continu** avec CloudTrail et DynamoDB Streams
6. **Sauvegardes automatiques** avec rétention adaptée

Ces pratiques garantissent une infrastructure d'applications et de stockage conforme, sécurisée et résiliente pour vos applications SaaS critiques.

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