Setup Secure Web Application Infrastructure on AWS using IAM, EC2 and VPC

Project Overview

This comprehensive guide provides a complete implementation plan for building a secure, production-ready web application infrastructure on AWS. The architecture leverages VPC isolation, public-private subnet segmentation, and role-based access control to create a robust, scalable environment for hosting web applications.

Project Title: Setup Secure Web Application Infrastructure on AWS using IAM, EC2 and VPC

Architecture Components:

- Custom VPC with public and private subnets
- Internet Gateway for public subnet connectivity
- NAT Gateway for private subnet outbound access
- EC2 instances (nginx in public, application in private)
- Security Groups and Network ACLs for multi-layer security
- IAM users and groups with role-based access control

Architecture Diagram

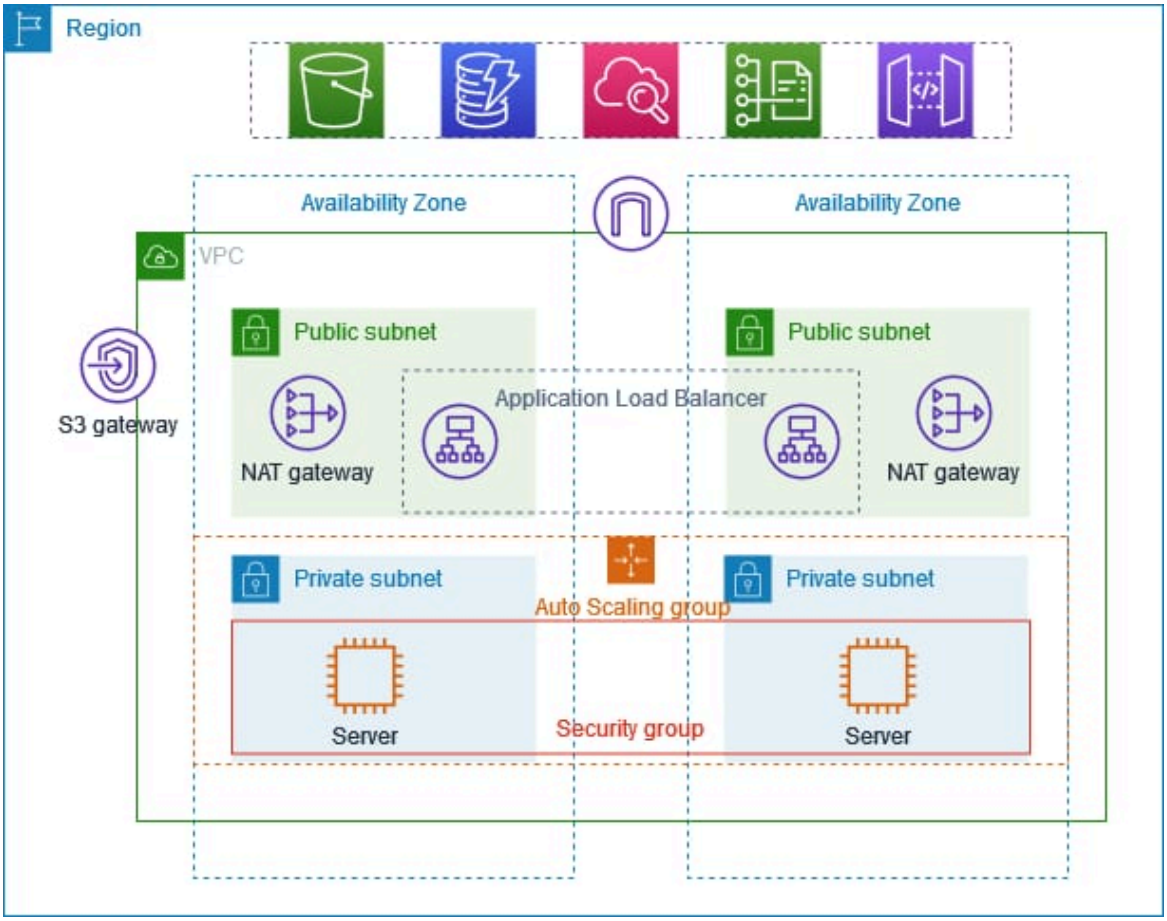

Figure 1: AWS VPC Architecture: Public and Private Subnets with Security Controls

The diagram illustrates the complete network topology showing:

- VPC CIDR block 10.0.0.0/16
- Public Subnet (10.0.1.0/24) with Internet Gateway access
- Private Subnet (10.0.2.0/24) with NAT Gateway for outbound connectivity
- Security Groups and NACLs at multiple layers
- Nginx web server in public subnet acting as reverse proxy
- Application server in private subnet with restricted access

---

Phase 1: VPC and Subnet Configuration

Step 1: Create the VPC

Begin by establishing the foundational network infrastructure through the AWS VPC console. Navigate to the VPC Dashboard and select "Create VPC", choosing "VPC and more" for a guided setup that automatically provisions related resources.

Configuration Parameters:

Parameter	Value
Name	Custom-VPC
IPv4 CIDR block	10.0.0.0/16
IPv6 CIDR block	None
Tenancy	Default
Enable DNS hostnames	Yes
Enable DNS resolution	Yes

Table 1: VPC Configuration Parameters

The VPC CIDR block determines the total IP address space available for all subnets and resources. The /16 CIDR provides 65,536 addresses, offering flexibility for creating multiple subnets while avoiding overlap with typical on-premises networks. DNS hostname enablement ensures that EC2 instances receive DNS names, facilitating internal communication and service discovery.

Step 2: Configure Internet Gateway

The Internet Gateway (IGW) serves as the VPC's connection point to the public internet, enabling communication between instances in public subnets and external networks.

Setup Steps:

1. Navigate to VPC Dashboard → Internet Gateways → Create Internet Gateway
2. Name the gateway: "Custom-IGW"
3. Select Actions → Attach to VPC
4. Choose your Custom-VPC
5. Verify attachment status shows "Attached"

The IGW is horizontally scaled, redundant, and highly available by default. AWS automatically handles its availability across multiple Availability Zones, requiring no manual configuration for redundancy.

Step 3: Create Public Subnet

Public subnets host resources that require direct internet access, including load balancers, bastion hosts, and NAT Gateways.

Configuration Parameters:

Parameter	Value
Name	Public-Subnet
VPC	Custom-VPC
Availability Zone	us-east-1a
IPv4 CIDR block	10.0.1.0/24
Auto-assign public IPv4	Enable

Table 2: Public Subnet Configuration

Auto-assigning public IPs ensures that instances launched in this subnet automatically receive public IP addresses, making them accessible from the internet when security groups permit. This subnet will eventually host the nginx web server and NAT Gateway.

Step 4: Create Private Subnet

Private subnets isolate backend resources from direct internet access, providing an additional security layer.

Configuration Parameters:

Parameter	Value
Name	Private-Subnet
VPC	Custom-VPC
Availability Zone	us-east-1a
IPv4 CIDR block	10.0.2.0/24
Auto-assign public IPv4	Disable

Table 3: Private Subnet Configuration

Disabling public IP assignment prevents instances in this subnet from receiving internet-routable addresses, ensuring they remain isolated. These instances can only be accessed through controlled pathways such as the bastion host in the public subnet or through the nginx reverse proxy.

Phase 2: Routing and NAT Configuration

Step 5: Allocate Elastic IP for NAT Gateway

NAT Gateways require an Elastic IP (EIP) address to provide a consistent public IP for outbound internet connections from private subnets.

Allocation Process:

1. Navigate to VPC Dashboard → Elastic IPs → Allocate Elastic IP address
2. Select "Amazon's pool of IPv4 addresses"
3. Add name tag: "NAT-Gateway-EIP"
4. Click "Allocate"

The EIP remains associated with your account until explicitly released. Charges apply when allocated but not associated with a running resource.

Step 6: Create NAT Gateway

The NAT Gateway enables instances in private subnets to initiate outbound connections to the internet while preventing inbound connections.

Configuration Parameters:

Parameter	Value
Name	Custom-NAT-Gateway
Subnet	Public-Subnet
Connectivity type	Public
Elastic IP allocation	(Previously created EIP)

Table 4: NAT Gateway Configuration

The NAT Gateway takes a few minutes to reach "Available" state. It automatically scales bandwidth up to 45 Gbps and is managed entirely by AWS, eliminating operational overhead.

Step 7: Configure Public Route Table

Route tables contain rules that determine where network traffic is directed based on destination IP addresses.

Route Table Setup:

1. Navigate to VPC Dashboard → Route Tables → Create Route Table
2. Name: "Public-Route-Table"
3. VPC: Custom-VPC
4. Click "Create route table"

Add Routes:

1. Click route table → Routes tab → Edit routes
2. Add route:
    - Destination: 0.0.0.0/0
    - Target: Internet Gateway (Custom-IGW)
3. Save changes

Associate Subnet:

1. Click Subnet Associations tab
2. Edit subnet associations
3. Select Public-Subnet
4. Save associations

The 0.0.0.0/0 route represents a default route, directing all traffic not matching other routes to the Internet Gateway.

Step 8: Configure Private Route Table

The private route table directs internet-bound traffic through the NAT Gateway while keeping local VPC traffic internal.

Route Table Setup:

1. Navigate to Route Tables → Create Route Table
2. Name: "Private-Route-Table"
3. VPC: Custom-VPC

Add Routes:

1. Routes tab → Edit routes
2. Add route:
    - Destination: 0.0.0.0/0
    - Target: NAT Gateway (Custom-NAT-Gateway)
3. Save changes

Associate Subnet:

1. Subnet Associations tab → Edit subnet associations
2. Select Private-Subnet
3. Save associations

This configuration allows private instances to reach the internet for updates and API calls while remaining inaccessible from the internet.

Phase 3: Security Configuration

Security Groups Configuration

Security Groups act as virtual firewalls at the instance level, controlling inbound and outbound traffic with stateful filtering. When you allow inbound traffic, return traffic is automatically allowed regardless of outbound rules.

Web Server Security Group (Nginx - Public Subnet):

Rule Type	Protocol	Port	Source
Inbound Rules			
HTTP	TCP	80	0.0.0.0/0
HTTPS	TCP	443	0.0.0.0/0
SSH	TCP	22	Your-IP/32
Outbound Rules			
All traffic	All	All	0.0.0.0/0

Table 5: Web Server Security Group Rules

Application Server Security Group (Private Subnet):

Rule Type	Protocol	Port	Source
Inbound Rules			
Custom TCP	TCP	8080	WebServer-SG
SSH	TCP	22	WebServer-SG
Outbound Rules			
All traffic	All	All	0.0.0.0/0

Table 6: Application Server Security Group Rules

By referencing security groups as sources, rules automatically apply to all instances in that group, simplifying management and improving security.

Network Access Control Lists (NACLs)

NACLs provide an additional security layer at the subnet level with stateless filtering. Unlike Security Groups, NACLs require explicit rules for both inbound and outbound traffic, including return traffic.

Public Subnet NACL Inbound Rules:

Rule #	Type	Port	Source	Action
100	HTTP	80	0.0.0.0/0	ALLOW
110	HTTPS	443	0.0.0.0/0	ALLOW
120	SSH	22	Your-IP/32	ALLOW
130	Ephemeral	1024-65535	0.0.0.0/0	ALLOW
*	All	All	All	DENY

Table 7: Public Subnet NACL Inbound Rules

Private Subnet NACL Inbound Rules:

Rule #	Type	Port	Source	Action
100	Custom TCP	8080	10.0.1.0/24	ALLOW
110	Ephemeral	1024-65535	0.0.0.0/0	ALLOW
*	All	All	All	DENY

Table 8: Private Subnet NACL Inbound Rules

NACL rules are evaluated in numerical order, with the lowest number processed first. Once a rule matches, processing stops, making rule ordering critical for security.

Phase 4: EC2 Instance Deployment

Deploy Nginx Web Server (Public Subnet)

Launch the public-facing nginx instance:

Launch Configuration:

Parameter	Value
Name	Nginx-WebServer
AMI	Amazon Linux 2023
Instance Type	t2.micro
VPC	Custom-VPC
Subnet	Public-Subnet
Auto-assign Public IP	Enable
Security Group	WebServer-SG

Table 9: Nginx Instance Configuration

User Data Script:

```
#!/bin/bash
yum update -y
yum install -y nginx
systemctl start nginx
systemctl enable nginx
```

Create a simple health check page

```
echo "
# Nginx Web Server Running

" > /usr/share/nginx/html/index.html
```

Deploy Application Server (Private Subnet)

Launch the private application instance:

Launch Configuration:

Parameter	Value
Name	App-Server-Private
AMI	Amazon Linux 2023
Instance Type	t2.micro
VPC	Custom-VPC
Subnet	Private-Subnet
Auto-assign Public IP	Disable
Security Group	AppServer-SG

Table 10: Application Server Configuration

User Data Script:

```
#!/bin/bash
yum update -y
yum install -y python3 python3-pip
```

Install Flask application

```
pip3 install flask
```

Create Flask application

```
cat > /home/ec2-user/app.py << 'EOF'
from flask import Flask, jsonify
app = Flask(name)

@app.route('/')
def index():
return jsonify({
"message": "Application Server Running",
"status": "healthy"
})

@app.route('/health')
def health():
return jsonify({"status": "ok"}), 200

if name == 'main':
app.run(host='0.0.0.0', port=8080)
EOF
```

Start Flask application

python3 /home/ec2-user/app.py &

Configure Nginx Reverse Proxy

After both instances are running, update the nginx configuration:

1. SSH into the nginx instance: `ssh -i your-key.pem ec2-user@public-ip`
2. Get the **private IP address** of the application server from EC2 console
3. Edit nginx configuration:

sudo nano /etc/nginx/nginx.conf

Add to the http block:

upstream app_server {
server <PRIVATE_IP_OF_APP_SERVER>:8080;
}

server {
listen 80 default_server;
listen [::]:80 default_server;

```
location / {
    proxy_pass http://app_server;
    proxy_set_header Host \$host;
    proxy_set_header X-Real-IP \$remote_addr;
    proxy_set_header X-Forwarded-For \$proxy_add_x_forwarded_for;
    proxy_set_header X-Forwarded-Proto \$scheme;
}
```

}

4. Test configuration: `sudo nginx -t`
5. Reload nginx: `sudo systemctl reload nginx`

Phase 5: IAM Configuration

Create IAM Groups

IAM groups simplify permission management by allowing you to assign permissions to groups rather than individual users.

AdminGroup - Full Administrative Access:

Group	Policies Attached
AdminGroup	AmazonEC2FullAccess AmazonVPCFullAccess IAMFullAccess

Table 11: AdminGroup Permissions

DevGroup - Read-Only EC2 Access:

Group	Policies Attached
DevGroup	AmazonEC2ReadOnlyAccess

Table 12: DevGroup Permissions

Create IAM Users

Admin User:

1. Navigate to IAM Dashboard → Users → Add Users
2. Username: admin-user1
3. AWS credential type: Password and Access key
4. Add to group: AdminGroup
5. Create user

Developer User:

1. IAM Dashboard → Users → Add Users
2. Username: dev-user1
3. AWS credential type: Password and Access key
4. Add to group: DevGroup
5. Create user

IAM Best Practices

- **Principle of Least Privilege**: Grant only the minimum permissions required for users to perform their jobs
- **Regular Audits**: Periodically review IAM users, groups, and policies to ensure they align with organizational needs
- **Use MFA**: Require multi-factor authentication for users with elevated privileges
- **Monitor with CloudTrail**: Enable AWS CloudTrail to log all IAM actions for security analysis
- **Rotate Credentials**: Regularly rotate access keys and passwords every 90 days

---

Phase 6: Testing and Validation

VPC and Routing Tests

Test 1: VPC Validation

- Console: VPC Dashboard → Your VPCs
- Verify: VPC with CIDR 10.0.0.0/16 exists
- Check: DNS hostnames and resolution enabled

Test 2: Route Table Validation

- Public Route Table: Contains route 0.0.0.0/0 → IGW
- Private Route Table: Contains route 0.0.0.0/0 → NAT-Gateway
- Verify subnet associations are correct

Test 3: NAT Gateway Status

- Console: VPC Dashboard → NAT Gateways
- Verify: State is "Available"
- Verify: Elastic IP is associated

Security and Connectivity Tests

Test 4: Public Instance Internet Access:

ssh -i your-key.pem ec2-user@nginx-public-ip
curl http://google.com

This confirms the public instance can reach the internet through the Internet Gateway.

Test 5: Private Instance Internet Access via NAT:

ssh ec2-user@private-instance-ip
curl http://google.com

This confirms the NAT Gateway is correctly routing private subnet traffic to the internet.

Test 6: Nginx Reverse Proxy:

curl http://nginx-public-ip/

This validates the reverse proxy configuration is forwarding traffic correctly.

Test 7: IAM Permissions:

- Log in as AdminGroup user
- Attempt to launch/terminate EC2 instance → Should succeed
- Log in as DevGroup user
- Attempt to terminate EC2 instance → Should fail with access denied

---

Common Troubleshooting Scenarios

Cannot SSH to Public Instance

Problem: Connection timeout when trying to SSH

Solutions:

- Check Security Group allows port 22 from your IP
- Verify instance has public IP assigned
- Check NACL allows inbound SSH (port 22)
- Verify Internet Gateway is attached
- Check route table has route to IGW

Cannot Reach Private Instance from Public

Problem: Cannot SSH from public to private instance

Solutions:

- Verify both instances in same VPC
- Check private instance security group allows SSH from public SG
- Use SSH agent forwarding: `ssh -A`
- Verify private instance has private IP in correct subnet
- Check NACL rules on both subnets

Private Instance Cannot Access Internet

Problem: curl fails from private instance

Solutions:

- Verify NAT Gateway is in 'available' state
- Check NAT Gateway has Elastic IP
- Verify private route table has route to NAT Gateway
- Check private subnet is associated with correct route table
- Verify security group allows outbound traffic
- Check NACL allows required outbound/inbound traffic

Nginx Reverse Proxy Not Working

Problem: Public IP returns error or timeout

Solutions:

- Check nginx is running: `systemctl status nginx`
- Verify nginx config: `nginx -t`
- Check private instance IP in nginx config is correct
- Verify private instance app is running on port 8080
- Check security groups allow traffic between instances
- Review nginx logs: `/var/log/nginx/error.log`

IAM Permission Issues

Problem: User cannot perform expected actions

Solutions:

- Verify user is in correct group
- Check group has correct policies attached
- Test with AWS CLI: `aws iam get-user`
- Review policy permissions
- Check for explicit deny rules
- Verify account access

Best Practices and Security Recommendations

Network Security

Implement Defense in Depth: Use multiple security layers including Security Groups, NACLs, and VPC isolation. Each layer provides redundancy if another is misconfigured.

Use Multi-AZ Deployments: For production workloads, deploy resources across multiple Availability Zones to ensure high availability. Create duplicate subnets in different AZs and use Auto Scaling groups.

Minimize Public Subnet Resources: Only place resources that require direct internet access in public subnets. Keep databases, application servers, and sensitive workloads in private subnets.

Enable VPC Flow Logs: Monitor network traffic to detect anomalies, troubleshoot connectivity issues, and meet compliance requirements. Publish logs to CloudWatch Logs or S3 for analysis.

Bastion Host Alternative

For production environments requiring frequent SSH access to private instances, consider deploying a dedicated bastion host instead of using the nginx instance for SSH:

1. Launch a dedicated t2.micro instance in the public subnet
2. Create a restrictive security group allowing SSH only from your office IP
3. Configure private instance security groups to allow SSH only from the bastion security group
4. Use SSH agent forwarding to connect through the bastion to private instances

Alternatively, use **AWS Systems Manager Session Manager** to access private instances without opening SSH ports.

Cost Optimization

NAT Gateway Costs: NAT Gateways charge $0.045 per hour plus data processing fees. For development environments, consider using a NAT instance instead to reduce costs.

Elastic IP Management: Release unused Elastic IPs to avoid hourly charges. EIPs are free when associated with running instances but incur charges when idle.

Instance Right-Sizing: Monitor instance utilization with CloudWatch and adjust instance types accordingly. Use t3/t4g instance types for burstable performance at lower costs.

Monitoring and Logging

Enable CloudWatch Monitoring: Track instance metrics like CPU utilization, network traffic, and disk I/O. Set up alarms for critical thresholds.

Centralized Logging: Configure application logs to stream to CloudWatch Logs or a centralized logging solution for easier troubleshooting and audit trails.

AWS Config: Enable AWS Config to track configuration changes across VPC resources and maintain compliance.

Production Deployment Checklist

Before deploying to production, ensure the following:

- Multi-AZ deployment with redundant resources across Availability Zones
- Application Load Balancer (ALB) for distributing traffic across instances
- Auto Scaling groups for dynamic capacity management
- AWS Web Application Firewall (WAF) for application-layer protection
- CloudWatch alarms for CPU, memory, network, and disk metrics
- CloudTrail enabled for audit logging of all API calls
- VPC Flow Logs enabled for network traffic analysis
- Automated backups configured for all stateful resources
- Disaster recovery plan with Recovery Time Objective (RTO) and Recovery Point Objective (RPO)
- Regular security assessments and vulnerability scanning
- Compliance validation with relevant standards (PCI-DSS, HIPAA, SOC 2)
- Documentation of architecture, runbooks, and disaster recovery procedures
- Team training on operational procedures and troubleshooting
- Cost monitoring and optimization implemented

References

[1] Amazon Web Services. (2024). Amazon VPC Documentation. Retrieved from https://docs.aws.amazon.com/vpc/

[2] Amazon Web Services. (2024). EC2 Security Documentation. Retrieved from https://docs.aws.amazon.com/ec2/

[3] Amazon Web Services. (2024). IAM Best Practices. Retrieved from https://docs.aws.amazon.com/iam/

[4] Amazon Web Services. (2024). VPC Security Best Practices. Retrieved from https://docs.aws.amazon.com/vpc/latest/userguide/vpc-security.html

[5] Amazon Web Services. (2024). NAT Gateways Documentation. Retrieved from https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html

[6] Amazon Web Services. (2024). Security Groups Documentation. Retrieved from https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html

[7] Amazon Web Services. (2024). Network ACLs Documentation. Retrieved from https://docs.aws.amazon.com/vpc/latest/userguide/vpc-network-acls.html

[8] Amazon Web Services. (2024). AWS Well-Architected Framework. Retrieved from https://aws.amazon.com/architecture/well-architected/