Secure and Scalable AWS Network Architecture Using Public and Private Subnets with NAT Instance

What is a NAT Gateway in AWS?

Imagine you're building a secure cloud environment. You want some of your servers to access the internet (e.g., for downloading security patches or updates) **but not be accessible from the internet**. This is where a **NAT Gateway (Network Address Translation Gateway)** comes in.

A **NAT Gateway** allows instances in **private subnets** to connect to the internet **outbound only**—meaning they can *send* traffic to the internet but **cannot receive unsolicited traffic from it**.

Why Use NAT Gateway?

To keep backend services like **databases and app servers hidden** from the public internet

To allow software and OS **updates without direct exposure**

To **enforce security** in VPC architectures

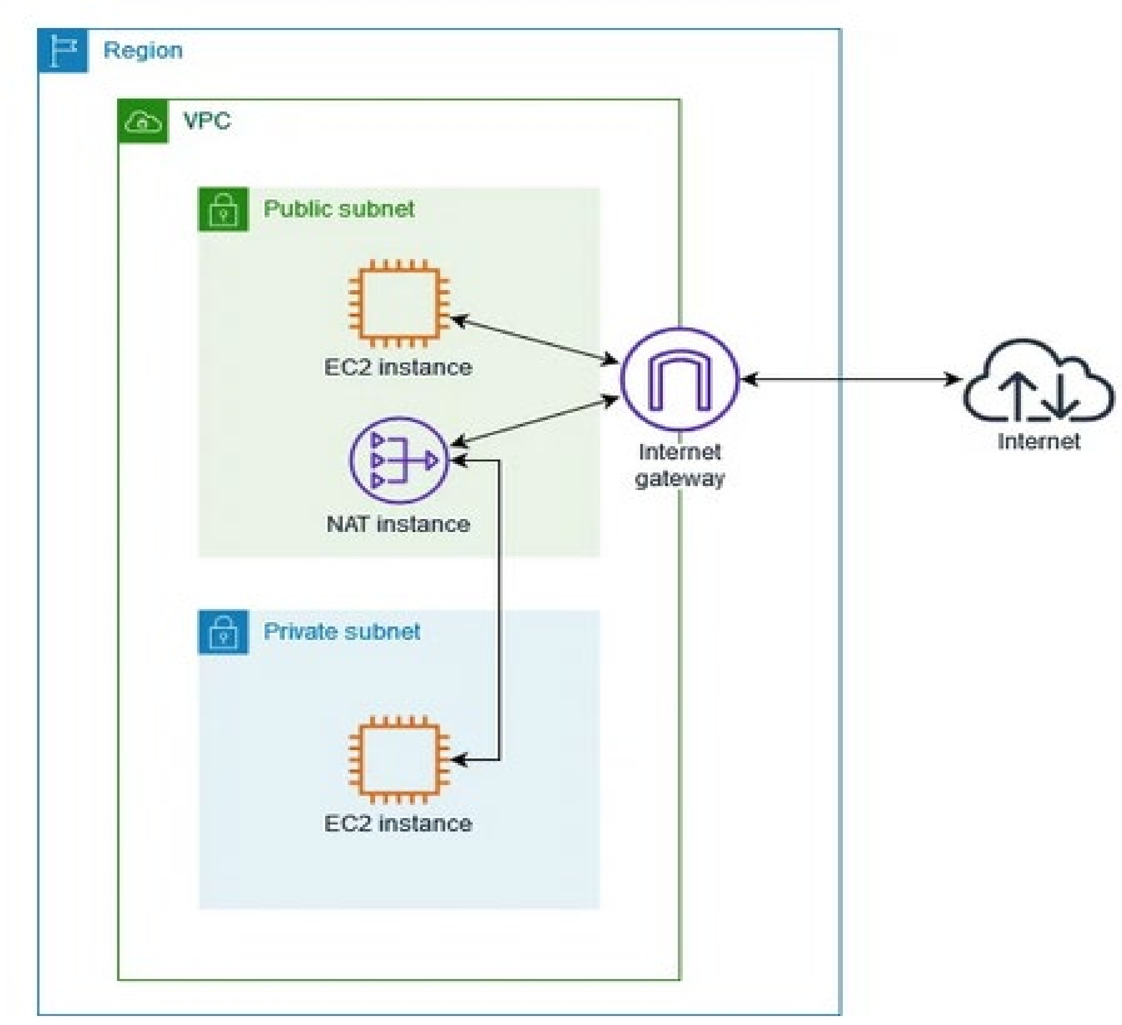
To comply with **regulatory and best practice standards**

Real-World Use Cases

1. **Secure Web App Hosting**
   - Frontend (React app) in a public subnet behind an ALB
   - Backend (Node.js/Flask) in a private subnet
   - Backend servers access the internet via NAT Gateway
2. **Private RDS with Update Access**
   - RDS hosted in private subnet
   - Periodic update scripts run from EC2 in private subnet using internet via NAT
3. **CI/CD Pipelines**

    - EC2 runners in private subnet fetch packages or update GitHub code using NAT

4. **ECS/EKS Services in Private Subnet**
    - Containerized workloads access AWS services or external APIs securely



Prerequisites:

- AWS Account
- VPC with CIDR block (e.g., 10.0.0.0/16)
- IAM permissions to create resources
- Key pair for EC2 login

Step-by-Step Setup (Console-Based)

Step 1: Create a VPC

- CIDR Block: 10.0.0.0/16
- Enable DNS support and DNS hostnames

Step 2: Create Two Subnets

1. **Public Subnet**
    - CIDR Block: 10.0.1.0/24
    - Enable "Auto-assign public IPv4"
2. **Private Subnet**
    - CIDR Block: 10.0.2.0/24

Step 3: Create and Attach an Internet Gateway

- Create an Internet Gateway
- Attach it to your VPC

Step 4: Create a Route Table for the Public Subnet

- Add a route: 0.0.0.0/0 → Internet Gateway
- Associate this route table with the public subnet

Step 5: Launch the NAT Instance

- AMI: Use **Amazon Linux 2** or a pre-built **NAT AMI** (amzn-ami-vpc-nat)
- Instance Type: t3.micro (free tier eligible)
- Network: Launch in the **public subnet**
- Assign a **public IP**
- Attach a security group allowing:
    - Inbound: SSH (port 22), ICMP (for testing)
    - Outbound: All traffic (default)

After Launch:

- SSH into the instance
- Run the following:

```
sudo sysctl -w net.ipv4.ip_forward=1

sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
```

Step 6: Disable Source/Destination Check

- Go to the NAT EC2 instance
- Actions → Networking → Change Source/Dest. Check → **Disable**

Step 7: Create a Route Table for Private Subnet

- Add a route: 0.0.0.0/0 → NAT Instance ID
- Associate this route table with the **private subnet**

Step 8: Launch EC2 Instance in Private Subnet

- Instance Type: any
- AMI: Amazon Linux 2
- Subnet: Private
- No public IP
- Attach a security group allowing outbound internet access (e.g., to port 443)

Step 9: Test the Setup

- SSH into the NAT instance (public)
- From NAT, connect to the private EC2 using its private IP (as a bastion host)
- From private EC2:
- curl https://amazon.com
- yum update -y

Step 10: Secure the Setup

- Apply least privilege security groups
- Enable logging (e.g., VPC Flow Logs)
- Restrict SSH access (only to trusted IPs)
- Configure CloudWatch alarms for instance health

Summary

- Public subnet = NAT instance + Bastion Host
- Private subnet = EC2 instances without public IP
- NAT instance = bridge for outbound traffic from private instances
- Scalable via AMI baking and Auto Recovery alarms (for basic resilience)

Advantages of NAT Instance:

- Lower cost than NAT Gateway (in low-traffic environments)
- Customizable (can run iptables, monitoring tools, logs)
- Ideal for learning and academic use

Limitations:

- Not highly available by default (no autoscaling)
- Manual configuration (e.g., routing, IP forwarding)
- Needs monitoring and patching

Implemented Using:

- AWS CloudShell (AWS CLI)
- AWS CloudFormation (YAML template)

Concept (Before Automation)

- **VPC CIDR:** 10.0.0.0/16
- **Public Subnet:** 10.0.1.0/24 → contains NAT instance
- **Private Subnet:** 10.0.2.0/24 → contains app/DB EC2s
- **NAT Instance:** Allows outbound internet access from private EC2s
- **IGW:** Internet Gateway for NAT to connect out
- **Routing:**
    - Public subnet → route to IGW
    - Private subnet → route to NAT instance

1. Using AWS CloudShell (AWS CLI)

Launch CloudShell in AWS Console and run:

```
# Create VPC

aws ec2 create-vpc --cidr-block 10.0.0.0/16 --tag-specifications
ResourceType=vpc,Tags=[{Key=Name,Value=NAT-VPC}]


# Create Subnets

aws ec2 create-subnet --vpc-id <vpc-id> --cidr-block 10.0.1.0/24 --availability-zone ap-south-1a
# Public

aws ec2 create-subnet --vpc-id <vpc-id> --cidr-block 10.0.2.0/24 --availability-zone ap-south-1a
# Private


# Create Internet Gateway

aws ec2 create-internet-gateway
aws ec2 attach-internet-gateway --internet-gateway-id <igw-id> --vpc-id <vpc-id>

# Create Route Table for Public Subnet
aws ec2 create-route-table --vpc-id <vpc-id>
aws ec2 create-route --route-table-id <rtb-id> --destination-cidr-block 0.0.0.0/0 --gateway-id <igw-id>
aws ec2 associate-route-table --subnet-id <public-subnet-id> --route-table-id <rtb-id>

# Launch NAT Instance (Amazon Linux 2 AMI or NAT AMI)
# Allocate Elastic IP for it, attach to instance manually or via CLI

# Disable source/destination check on NAT instance
aws ec2 modify-instance-attribute --instance-id <nat-instance-id> --no-source-dest-check

# Private route table → NAT instance
aws ec2 create-route-table --vpc-id <vpc-id>
aws ec2 create-route --route-table-id <private-rtb-id> --destination-cidr-block 0.0.0.0/0 --instance-id <nat-instance-id>
aws ec2 associate-route-table --subnet-id <private-subnet-id> --route-table-id <private-rtb-id>
```

Using AWS CloudFormation:

Create & deploy : nat-instance-setup.yaml

```
Description: VPC with NAT Instance in Public Subnet
Resources:
  VPC:
    Type: AWS::EC2::VPC
    Properties:
      CidrBlock: 10.0.0.0/16
      EnableDnsSupport: true
      EnableDnsHostnames: true
      Tags: [{ Key: Name, Value: NAT-VPC }]

  InternetGateway:
    Type: AWS::EC2::InternetGateway

  VPCGatewayAttachment:
    Type: AWS::EC2::VPCGatewayAttachment
    Properties:
      VpcId: !Ref VPC
      InternetGatewayId: !Ref InternetGateway

  PublicSubnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      CidrBlock: 10.0.1.0/24
      AvailabilityZone: ap-south-1a
      MapPublicIpOnLaunch: true

  PrivateSubnet:
    Type: AWS::EC2::Subnet
    Properties:
      VpcId: !Ref VPC
      CidrBlock: 10.0.2.0/24
      AvailabilityZone: ap-south-1a

  PublicRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC

  PublicRoute:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PublicRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      GatewayId: !Ref InternetGateway

  PublicSubnetRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PublicSubnet
      RouteTableId: !Ref PublicRouteTable

  NATInstance:
    Type: AWS::EC2::Instance
    Properties:
      InstanceType: t3.micro
      ImageId: ami-0dfcb1ef8550277af    # Amazon Linux 2 NAT AMI (verify region)
      SubnetId: !Ref PublicSubnet
      KeyName: your-key-pair
      SourceDestCheck: false
      Tags:
        - Key: Name
          Value: NATInstance

  PrivateRouteTable:
    Type: AWS::EC2::RouteTable
    Properties:
      VpcId: !Ref VPC

  NATRoute:
    Type: AWS::EC2::Route
    Properties:
      RouteTableId: !Ref PrivateRouteTable
      DestinationCidrBlock: 0.0.0.0/0
      InstanceId: !Ref NATInstance

  PrivateSubnetRouteTableAssociation:
    Type: AWS::EC2::SubnetRouteTableAssociation
    Properties:
      SubnetId: !Ref PrivateSubnet
      RouteTableId: !Ref PrivateRouteTable
```