**Index**

1. VPC Introduction
2. VPC Design & Components
3. VPC Setup Details
4. Default VPC
5. Create VPC
6. Subnets
7. Internet Gateway
8. Route table
9. NAT Gateway
10. Bastion Host
11. Website in VPC
12. Peering

**Understanding AWS VPC: Virtual Private Cloud & Networking Fundamentals**

**1. Introduction to VPC**

What is a Virtual Private Cloud (VPC)?

A VPC (Virtual Private Cloud) is a logically isolated network within AWS that allows users to:

* Define their own networking schemes, including subnets, security controls, and routing tables.
* Control inbound and outbound traffic using Network ACLs and Security Groups.
* Enable scalable architectures by segmenting resources across multiple Availability Zones.

Why was VPC Introduced?

Before VPC, AWS had EC2 Classic, where networking was less flexible:

* Users had little control over IP addressing schemes.
* Security mechanisms were not as customizable.
* Networking relied on default public configurations, limiting scalability.

Evolution of AWS Networking

1. EC2 Classic: Early AWS networking allowed users to launch instances but with limited control over networking.
2. VPC Introduction: AWS provided users full control over subnetting, routing, and access management.
3. Modern VPC Implementations: AWS VPC supports VPN connections, NAT gateways, inter-VPC peering, and transit gateways, making cloud networking more advanced.

**2. Networking Basics: Revisiting Key Concepts**

Since VPC is deeply rooted in networking, understanding the following basics is crucial.

What is an IP Address?

* An IPv4 address is a 32-bit binary number, commonly represented in decimal format, such as 192.168.1.1.
* It consists of four octets, each separated by a dot:
* 192.168.1.1 → 11000000.10101000.00000001.00000001
  + - Each octet represents an 8-bit binary number, making IPv4 a total of 32 bits.
    - Understanding Subnetting & CIDR Notation
    - Subnetting helps divide larger networks into smaller, manageable subnets, improving efficiency and security.
* CIDR notation (Classless Inter-Domain Routing) is used for IP address allocation, commonly written as:192.168.1.0/24

Here:

* 192.168.1.0 is the network address.
* /24 means 24 bits are reserved for the network, leaving 8 bits for host allocation.

Subnet Mask Examples

| CIDR | Subnet Mask | Number of Usable Hosts |
| --- | --- | --- |
| /16 | 255.255.0.0 | 65,534 hosts |
| /24 | 255.255.255.0 | 254 hosts |
| /30 | 255.255.255.252 | 2 hosts |

AWS requires proper subnet planning when designing a VPC for optimal networking.

**3. AWS VPC Architecture**

Key Components of a VPC

1. VPC – The overall private network environment.
2. Subnets – Segments within the VPC for organizing resources.
3. Route Tables – Determines how traffic is routed between subnets.
4. Internet Gateway (IGW) – Enables public internet access for instances.
5. NAT Gateway – Allows private subnet instances to access the internet without being publicly exposed.
6. Security Groups & NACLs – Firewall-like controls for traffic management.

Subnet Segmentation: Public vs. Private

A well-architected VPC design requires separating subnets:

* Public Subnet: Direct internet access enabled.
* Private Subnet: Restricted access; resources can only communicate internally or via NAT Gateway**.**

**4. Security Considerations in VPC**

Security Groups vs. NACLs

Security Groups (Instance Level)

* Stateful (tracks connections automatically).
* Applied at the instance level.
* Default denies all inbound traffic but allows all outbound traffic.

Network ACLs (Subnet Level)

* Stateless (each request needs explicit inbound/outbound rules).
* Applied at the subnet level.
* Rules must be configured for both inbound and outbound directions.

Controlling Access with Route Tables

Route tables determine how traffic moves within a VPC. Example:

| Destination | Target |
| --- | --- |
| 0.0.0.0/0 | Internet Gateway (IGW) |
| 10.0.1.0/24 | Local Subnet |
| 172.16.0.0/16 | Peered VPC |

By configuring custom route tables, users can establish secure internal networking.

**5. Advanced VPC Features**

VPC Peering

* Enables private communication between two VPCs.
* No internet gateway is required for cross-VPC traffic.
* Useful for connecting database VPCs to application VPCs securely.

VPC Endpoints

* Allows direct access to AWS services like S3, DynamoDB without needing an internet gateway.
* Reduces data transfer costs while improving security.

Transit Gateway

* Connects multiple VPCs and on-premise networks in a hub-and-spoke model.
* More scalable than VPC Peering, suited for multi-region enterprise networks.

**6. Summary & Best Practices**

**Key Takeaways**

✅ AWS VPC provides full control over networking, enabling scalable architectures.

✅ Subnet segmentation improves security, ensuring private resources remain protected.

✅ NACLs and Security Groups must be configured correctly to manage traffic.

✅ VPC Peering and Endpoints help optimize secure inter-VPC communication.

✅ Proper CIDR planning prevents IP conflicts and ensures efficient resource allocation.

**Best Practices**

**🔹** Use least privilege access with strict Security Group rules.

🔹 Plan CIDR ranges carefully to prevent overlap in multi-VPC setups.

🔹 Enable VPC Flow Logs for auditing network traffic.

🔹 Utilize NAT Gateways for outbound internet access instead of public IPs.

🔹 Use VPC Endpoints for direct AWS service access, avoiding unnecessary internet exposure.

**7. Next Steps**

**For further learning:**

* **Explore AWS Documentation on VPC & Networking.**
* **Practice setting up a custom VPC architecture in AWS.**
* **Experiment with VPC Peering & Transit Gateway for multi-VPC connectivity.**

**Conclusion**

By understanding VPC, networking basics, subnetting, and security, cloud architects and DevOps engineers can design highly scalable and secure infrastructures. Whether building a basic web server or a complex multi-region architecture, AWS VPC is the backbone of networking in the cloud.

Would you like additional refinements or specific diagrams to accompany this document?

**Understanding IPv4 Addressing, Subnetting & CIDR Notation**

**1. Introduction to IPv4 Addressing**

What is an IPv4 Address?

An IPv4 address is a 32-bit binary number, commonly represented in decimal format, and divided into four octets, separated by dots. For example:

192.168.1.1 → Binary Equivalent: 11000000.10101000.00000001.00000001

Each octet consists of 8 bits, making IPv4 addresses 32 bits long. The decimal representation is derived from binary numbers.

Binary Number Conversion to Decimal

* 192 → 11000000
* 168 → 10101000
* 1 → 00000001
* 1 → 00000001

Every IPv4 address is represented both numerically and in binary.

**2. IPv4 Range & Public vs. Private IPs**

Full IPv4 Range

IPv4 addresses range from 0.0.0.0 to 255.255.255.255.

Each octet ranges from 0 to 255 due to binary constraints:

* 0.0.0.0 is the lowest address.
* 255.255.255.255 is the highest address.

Public vs. Private IPs

IPv4 addresses are categorized as public or private.

* Public IPs: Used on the internet, must be globally unique.
* Private IPs: Used in local networks (LANs) and not routable over the internet.

Private IP Address Ranges

| Class | Private IP Range | Number of Hosts |
| --- | --- | --- |
| Class A | 10.0.0.0 – 10.255.255.255 | 16,777,216 |
| Class B | 172.16.0.0 – 172.31.255.255 | 1,048,576 |
| Class C | 192.168.0.0 – 192.168.255.255 | 65,536 |

Only private IP ranges can be used for local networking without internet exposure.

**3. Subnetting & CIDR Notation**

Subnet Masks Explained

Subnet masks define network boundaries by separating the network portion from the host portion.

Common subnet masks:

* 255.255.255.0 → /24 → 256 total IPs
* 255.255.0.0 → /16 → 65,536 total IPs
* 255.0.0.0 → /8 → 16,777,216 total IPs

Subnet Mask Breakdown

| CIDR | Subnet Mask | Number of Usable Hosts |
| --- | --- | --- |
| /30 | 255.255.255.252 | 2 Hosts |
| /24 | 255.255.255.0 | 254 Hosts |
| /16 | 255.255.0.0 | 65,534 Hosts |
| /8 | 255.0.0.0 | 16,777,216 Hosts |

CIDR notation /24 means 24 bits are reserved for the network, leaving 8 bits for host allocation.

**4. Network Address & Broadcast Address**

Understanding Network & Broadcast Addresses

* Network Address: The first address in a subnet (reserved for identification).
* Broadcast Address: The last address in a subnet (used for sending messages to all hosts).

Example of a Subnet: 192.168.1.0/24

| Address Type | Value |
| --- | --- |
| Network Address | 192.168.1.0 |
| First Usable IP | 192.168.1.1 |
| Last Usable IP | 192.168.1.254 |
| Broadcast Address | 192.168.1.255 |

🔹 The network address (192.168.1.0) and broadcast address (192.168.1.255) cannot be assigned to hosts.

**5. Practical Application: Subnet Planning**

When designing an AWS VPC or corporate network, consider the following:

1. Choose a Class A/B/C Private IP Range.
2. Determine the Subnet Mask (/24, /16, etc.).
3. Plan for Network & Broadcast Addresses.
4. Ensure IP Allocation Accounts for Scaling.

Example: If an organization needs 500 hosts, a /24 subnet won’t be sufficient. A larger subnet like /22 or /23 should be chosen.

6. Summary & Best Practices

Key Takeaways

✅ Subnet masks define IP allocation within a network.

✅ CIDR notation helps identify usable IP ranges.

✅ Network addresses and broadcast addresses cannot be assigned to hosts.

✅ Private IPs improve security by preventing direct internet exposure.

✅ Careful subnet planning ensures efficient network operations.

Subnetting Best Practices

🔹 Use smaller subnets to avoid unnecessary IP waste.

🔹 Prevent overlapping IP ranges in multi-VPC environments.

🔹 Utilize CIDR tables to plan network architectures efficiently.

🔹 Consider future scalability when choosing subnet sizes.

**Conclusion**

By understanding IPv4 addressing, subnetting, CIDR notation, and public/private IP schemes, network engineers can design secure and scalable infrastructures. Whether working in AWS, on-premise networks, or hybrid setups, proper subnet planning ensures efficient resource allocation.

**AWS VPC Architecture: Subnet Segmentation, Route Tables, and Gateway Configuration**

**1. Introduction to AWS Virtual Private Cloud (VPC)**

AWS VPC (Virtual Private Cloud) enables users to create a logically isolated networking environment where they can define:

* **Public** and **private** subnets for resource segmentation.
* Security controls such as Network ACLs and Security Groups.
* Routing policies using Route Tables and Gateways for internet access.

A properly designed VPC architecture ensures security, availability, and efficient traffic management for cloud workloads.

**2. Components of a Secure VPC Setup**

Public vs. Private Subnets

* Public Subnets: Directly connected to the Internet Gateway (IGW).
* Private Subnets: No direct internet access, relying on NAT Gateway for outbound traffic.

Each subnet belongs to an Availability Zone (AZ), ensuring high availability in multi-AZ deployments.

Internet Gateway (IGW)

* Enables public internet connectivity for instances in public subnets.
* AWS manages IGW for redundancy, ensuring high availability.

NAT Gateway

* Located in public subnets but used by private subnet instances.
* Outbound traffic from private subnets passes through NAT Gateway before reaching the internet.
* Inbound connections from the internet are blocked, enhancing security.

Route Tables

* Define traffic routing rules for subnets.
* Public subnet route tables direct traffic to Internet Gateway.
* Private subnet route tables route internet-bound traffic via NAT Gateway.

**3. High Availability in VPC Design**

To improve fault tolerance, VPC architectures should:

* Distribute subnets across multiple Availability Zones.
* Deploy redundant NAT Gateways in different public subnets.
* Implement failover mechanisms for uninterrupted network operations.

Multi-AZ Subnet Deployment

Every AWS region has at least two Availability Zones (some have up to six). Best practices include:

* Creating public subnets in at least two AZs.
* Creating private subnets in corresponding two AZs.
* Deploying EC2 instances across AZs for high availability.

**4. Accessing Private Subnet Instances**

Jump Server (Bastion Host)

* A publicly accessible EC2 instance allowing SSH access to private subnet instances.
* Jump servers prevent direct exposure of private resources.

Access Flow:

1. Connect via SSH to Bastion Host in public subnet.
2. From Bastion Host, SSH into private subnet instances.

⚠ Security Best Practice: Restrict Bastion Host access only to corporate IPs.

**5. Network Security Considerations**

Security Groups

* Stateful firewall rules applied at instance level.
* Restrict inbound and outbound traffic with specific rules.
* Example:
  + Allow SSH (22) only from trusted IPs.
  + Allow HTTP/S traffic (80/443) from Internet Gateway.

Network ACLs

* Stateless rules applied at subnet level.
* Must explicitly define both inbound and outbound rules.

Best Practices for Secure VPC Design

✅ Use least privilege access in Security Groups

✅ Deploy jump servers instead of exposing private instances.

✅ Enable VPC Flow Logs for traffic monitoring.

✅ Plan CIDR blocks carefully to prevent IP conflicts.

**6. Cleaning Up VPC Resources**

If resources are no longer needed, follow these steps:

1. Delete the NAT Gateway.
2. Release the Elastic IP.
3. Remove Route Table entries for NAT Gateway (blackhole routes).
4. Terminate EC2 instances.
5. Delete custom subnets and VPC.

⚠ AWS Charges for NAT Gateway Usage, so ensure it is removed if not needed.

**7. Summary & Next Steps**

This guide ensures a secure and efficient AWS VPC architecture, covering:

✅ Subnet segmentation for controlled network access.

✅ Gateways for internet and outbound traffic management.

✅ Route table configuration for directing traffic.

✅ High availability strategies using multi-AZ deployment.

✅ Security hardening best practices for resource protection.

For practical implementation, follow AWS VPC Documentation and test configurations using AWS VPC Wizard.

**Conclusion**

A well-planned AWS VPC setup ensures scalable, secure, and high-availability infrastructure. By implementing subnet segmentation, NAT Gateway routing, security group restrictions, and multi-AZ deployments, cloud networks remain efficient and protected.

**AWS VPC Detailed Configuration Guide**

**Internet Gateway Setup**

1. **Creating the Gateway**:
   * Navigate to the "Internet Gateways" section in the AWS Management Console.
   * Locate the default Internet Gateway and rename it to something identifiable, such as default internet gateway IGW.
   * To create a new Internet Gateway:
     + Click on “Create Internet Gateway.”
     + Give the gateway a name (e.g., vprofile-IGW), ensuring clarity in identification.
   * Once the gateway is created, its state will display as "detached."
   * To attach it to a VPC:
     + Click on “Action” → “Attach to VPC.”
     + Select your VPC from the dropdown (e.g., vprofile VPC) and confirm.
   * You can also use the AWS CLI to achieve this step via commands like attach internet gateway.
2. **Outcome**:
   * You will now have two Internet Gateways: one for the default VPC and another for the customized VPC (vprofile-VPC).

**Connecting Route Table to Public Subnet**

1. **Understanding Route Tables**:
   * Each VPC automatically comes with a default route table.
   * Rename the default route table for clarity (e.g., Default-PubRT) and ensure it represents the public route table for the default VPC.
   * Similarly, rename the customized route table to something like vprofile-default-RT. While this route table won’t be used, naming helps with easy identification.
2. **Creating Route Table for Public Subnets**:
   * In the AWS Management Console, click on "Create Route Table."
   * Name the route table appropriately (e.g., vpro-public-RT) to signify it corresponds to the public subnet for your customized VPC (vprofile-VPC).
   * Select your VPC (e.g., vprofile-VPC) during the creation process and save.
3. **Associating Subnets**:
   * Click on "Subnet Associations."
   * Select "Edit Subnet Associations" and choose the public subnets created earlier.
   * Save the association, ensuring the public subnets are now connected to the new route table.
4. **Defining Routes**:
   * Click on "Edit Routes."
   * Add the following entry: 0.0.0.0/0 → points to the Internet Gateway (e.g., vprofile Internet Gateway).
   * Save changes.
5. **Outcome**:
   * The public subnets are now functional, and instances launched within them can access the internet.

**Connecting Route Table to Public Subnet**

1. **Understanding Route Tables:**
   * Each VPC automatically comes with a default route table.
   * Rename the default route table for clarity (e.g., Default-PubRT) and ensure it represents the public route table for the default VPC.
   * Similarly, rename the customized route table to something like vprofile-default-RT. While this route table won’t be used, naming helps with easy identification.
2. **Creating Route Table for Public Subnets:**
   * In the AWS Management Console, click on "Create Route Table."
   * Name the route table appropriately (e.g., vpro-public-RT) to signify it corresponds to the public subnet for your customized VPC (vprofile-VPC).
   * Select your VPC (e.g., vprofile-VPC) during the creation process and save.
3. **Associating Subnets:**
   * Click on "Subnet Associations."
   * Select "Edit Subnet Associations" and choose the public subnets created earlier.
   * Save the association, ensuring the public subnets are now connected to the new route table.
4. **Defining Routes:**
   * Click on "Edit Routes."
   * Add the following entry: 0.0.0.0/0 → points to the Internet Gateway (e.g., vprofile Internet Gateway).
   * Save changes.
5. **Outcome:**
   * The public subnets are now functional, and instances launched within them can access the internet.

**Enabling Auto-Assign IP Settings**

1. **Steps to Enable Auto-Assign IPs:**
   * Navigate to "Subnets" in the AWS Management Console.
   * Select the public subnets and click on "Edit Subnet Settings."
   * Enable the option "Auto Assign IP Setting."
   * Save changes to ensure instances launched in these public subnets automatically receive public IPs.
   * Repeat the same for all public subnets.
2. **Outcome:**
   * Instances launched in the public subnets will now automatically be assigned public IPs, enabling direct internet access.

**Setting Up NAT Gateway for Private Subnets**

1. **Creating Elastic IP**:
   * Navigate to the "Elastic IPs" section.
   * Allocate a new Elastic IP.
   * Tag the Elastic IP (e.g., vprofile-NAT-Elastic IP) for easy identification.
2. **Configuring NAT Gateway**:
   * Navigate to the "NAT Gateways" section in the AWS Management Console.
   * Click on “Create NAT Gateway.”
   * Name it appropriately (e.g., vpro-NAT-Gateway).
   * Select the subnet where the NAT Gateway will reside.

**Important**: The NAT Gateway must be placed in a public subnet (e.g., vpro public subnet 1) to connect to the internet through the Internet Gateway.

1. **Connecting Private Subnets**:
   * Create a route table specifically for private subnets.
   * Associate private subnets with the newly created route table.
   * Add routes directing traffic through the NAT Gateway to access external resources.
2. **Outcome**:
   * Private subnets can now access the internet indirectly via the NAT Gateway.

**AWS VPC Setup: NAT Gateway and Bastion Host Configuration**

**1. Creating a NAT Gateway and Configuring Route Tables**

**Placing the NAT Gateway in the Right Subnet**

Ensure that the NAT Gateway is placed in either **Public Subnet 1 or 2**—it doesn’t matter which, since we are only creating **one** NAT Gateway.

**Associating an Elastic IP**

Before creating the NAT Gateway, we need an **Elastic IP**. If you haven’t allocated one yet, follow these steps:

1. Navigate to **AWS Console** → **Elastic IPs**.
2. Click on **Allocate Elastic IP** to create a new one.
3. Select the newly created Elastic IP for the NAT Gateway.

**Creating the NAT Gateway**

Once the Elastic IP is allocated:

1. Go to **NAT Gateway** in AWS Console.
2. Select the correct subnet (Public Subnet 1 or 2).
3. Attach the allocated **Elastic IP**.
4. Click on **Create NAT Gateway**.
5. The NAT Gateway’s state will initially be **pending**—wait a few minutes until it becomes **available**.

**Setting Up Route Tables**

While waiting for the NAT Gateway to become available, we configure **Route Tables** for private subnets:

1. Navigate to **Route Tables**.
2. Click on **Create Route Table**.
3. Name it "vpro-private-route-table" for clarity.
4. Associate it with your custom **VPC**.

**Subnet Association for Private Subnets**

1. Go to **Subnet Associations**.
2. Click on **Edit Subnet Associations**.
3. Select **both private subnets**.
4. Click **Save**.

**Adding Routes for Outbound Traffic**

1. Go to the **Routes** section.
2. Click on **Add Route**.
3. Configure:
   * **Destination:** 0.0.0.0/0 (all non-local traffic).
   * **Target:** Select the **NAT Gateway**.
4. Save changes.

At this point, the **private subnets** are now fully functional, with outbound internet access routed through the **NAT Gateway** instead of a public internet gateway.

**2. Configuring Public and Private Subnet Networking**

For instances launched within these subnets:

* **Public Subnet Instances** get both **Public and Private IPs**.
* **Private Subnet Instances** only get **Private IPs**.

**Enabling DNS Host Names**

Instances launched in either subnet **don’t get DNS hostnames by default**. To enable:

1. Navigate to **VPC Settings**.
2. Click on **Edit VPC Settings**.
3. Scroll down and select **Enable DNS Host Names**.
4. Click **Save**.

Now, instances will have **EC2-style DNS hostnames** along with their assigned IP addresses.

**3. Setting Up a Bastion Host**

**What is a Bastion Host?**

A Bastion Host (or **Jump Server**) is a secure entry point for accessing private subnet instances that do not have public IPs.

**Key Properties:**

* Provides **SSH access** to private subnet resources.
* Acts as a **single point of entry**, ensuring tighter security.

**Why is Bastion Host Security Critical?**

Think of a Bastion Host as **the fortress gate**—if it is weak, the entire infrastructure is at risk. It must have:

1. **Restricted SSH access** (only from trusted IPs).
2. **Strong key pair authentication**.
3. **A hardened AMI (Amazon Machine Image)** to reduce vulnerabilities.

**Creating the Bastion Host**

1. Go to **AWS EC2**.
2. Click **Launch Instance**.
3. Choose an AMI:
   * Default AWS AMIs work, but for production use a **security-tested AMI** (e.g., CIS-hardened images from AWS Marketplace).
4. Select **Instance Type** (t2.micro for basic use).
5. Configure **Network Settings**:
   * Place it in the **Public Subnet** of your VPC.

**Creating a Security Group for SSH Access**

1. Navigate to **Security Groups**.
2. Click **Create Security Group**.
3. Name it **vpro-bastion-sg**.
4. Attach it to **your custom VPC**.
5. Set inbound rules:
   * Allow **SSH** from **My IP** only (corporate IP for production setups).

**Generating a Key Pair**

1. Go to **EC2 → Key Pairs**.
2. Click **Create Key Pair**.
3. Name it vpro-bastion-key.pem.
4. **Download** and store the key securely.

**Launching the Instance**

1. Go to **EC2 → Instances**.
2. Click **Launch Instance**.
3. Select the **Bastion Host Security Group**.
4. Attach the **Key Pair**.
5. Launch the instance.

**Accessing Private Subnet Instances via Bastion Host**

1. SSH into the **Bastion Host** (public subnet):

bash

ssh -i vpro-bastion-key.pem ec2-user@<Bastion Public IP>

1. From the Bastion Host, SSH into a private subnet instance:

bash

ssh -i vpro-private-key.pem ec2-user@<Private Instance IP>

**3. Creating a NAT Gateway and Configuring Route Tables**

**Placing the NAT Gateway in the Right Subnet**

Ensure that the **NAT Gateway** is placed in either **Public Subnet 1 or Public Subnet 2**. The specific public subnet doesn’t matter because we are creating **only one NAT Gateway**.

**Associating an Elastic IP**

Before creating the NAT Gateway, we need an **Elastic IP**:

1. Navigate to **AWS Console** → **Elastic IPs**.
2. Click on **Allocate Elastic IP** to create a new one.
3. Select the newly created Elastic IP to associate it with the NAT Gateway.

**Creating the NAT Gateway**

Once the **Elastic IP** is allocated:

1. Go to **NAT Gateway** in the AWS Console.
2. Select the correct **Public Subnet**.
3. Attach the allocated **Elastic IP**.
4. Click on **Create NAT Gateway**.
5. The NAT Gateway’s state will initially be **pending**—wait a few minutes until it becomes **available**.

**Setting Up Route Tables**

While waiting for the NAT Gateway to become available, we configure **Route Tables** for private subnets:

1. Navigate to **Route Tables** in AWS.
2. Click on **Create Route Table**.
3. Name it **vpro-private-route-table** to maintain clarity.
4. Associate it with your custom **VPC**.

**Subnet Association for Private Subnets**

1. Go to **Subnet Associations**.
2. Click on **Edit Subnet Associations**.
3. Select **both private subnets**.
4. Click **Save**.

**Adding Routes for Outbound Traffic**

1. Navigate to **Routes**.
2. Click on **Add Route**.
3. Configure:
   * **Destination:** 0.0.0.0/0 (all non-local traffic).
   * **Target:** Select the **NAT Gateway**.
4. Click **Save Changes**.

At this point, the **private subnets** are now fully functional with outbound internet access routed through the **NAT Gateway** instead of an **internet gateway**.

**4. Configuring Public & Private Subnets for Instance Networking**

For instances launched within these subnets:

* **Public Subnet Instances** receive **both Public and Private IPs**.
* **Private Subnet Instances** receive **only Private IPs**.

**Enabling DNS Host Names**

Instances launched in **either subnet** **don’t get DNS hostnames by default**. To enable:

1. Navigate to **VPC Settings**.
2. Click **Edit VPC Settings**.
3. Scroll down and select **Enable DNS Host Names**.
4. Click **Save**.

Now, instances will have **EC2-style DNS hostnames** along with their assigned IP addresses.

**5. Setting Up a Bastion Host**

**What is a Bastion Host?**

A Bastion Host (**or Jump Server**) is a **secure entry point** for accessing private subnet instances that do not have public IPs.

**Why is Bastion Host Security Critical?**

Think of a Bastion Host as **the fortress gate**—if it is weak, the entire infrastructure is at risk. It must have:

1. **Restricted SSH access** (only from trusted IPs).
2. **Strong key pair authentication**.
3. **A hardened AMI (Amazon Machine Image)** to mitigate security vulnerabilities.

**Creating the Bastion Host**

1. Navigate to **AWS EC2**.
2. Click **Launch Instance**.
3. Choose an AMI:
   * Default AWS AMIs work, but for production use a **security-tested AMI** (e.g., CIS-hardened images from AWS Marketplace).
4. Select **Instance Type** (t2.micro for basic setups).
5. Configure **Network Settings**:
   * Place it in the **Public Subnet** of your VPC.

**Creating a Security Group for SSH Access**

1. Navigate to **Security Groups**.
2. Click **Create Security Group**.
3. Name it **vpro-bastion-sg**.
4. Associate it with your **custom VPC**.
5. Set inbound rules:
   * Allow **SSH** only from **trusted IP addresses** (corporate IP for production use).

**Generating a Key Pair**

1. Navigate to **AWS Console → Key Pairs**.
2. Click **Create Key Pair**.
3. Name it vpro-bastion-key.pem.
4. **Download** and store the key securely.

**Launching the Instance**

1. Go to **EC2 → Instances**.
2. Click **Launch Instance**.
3. Select the **Bastion Host Security Group**.
4. Attach the **Key Pair**.
5. Launch the instance.

**Accessing Private Subnet Instances via Bastion Host**

1. SSH into the **Bastion Host** (public subnet):

bash

ssh -i vpro-bastion-key.pem ec2-user@<Bastion Public IP>

1. From the Bastion Host, SSH into a private subnet instance:

bash

ssh -i vpro-private-key.pem ec2-user@<Private Instance IP>

**6. Creating a Load Balancer for Secure Website Access**

**Setting Up a Web Server in the Private Subnet**

1. Launch a new EC2 instance in **Private Subnet**.
2. Install necessary packages:

bash

yum install -y httpd wget unzip

1. Download a website template:

bash

wget <template-download-link>

1. Extract files and move them:

bash

unzip <filename.zip>

cp -r <extracted-folder> /var/www/html

1. Restart and enable the Apache server:

bash

systemctl restart httpd

systemctl enable httpd

**Creating a Target Group**

1. Navigate to **AWS EC2** → **Target Groups**.
2. Create a new target group for instances.
3. Associate it with the **VPC**.
4. Register the **Private Subnet Instance** in this group.

**Configuring Load Balancer Security Groups**

1. Create a security group **web-elb-sg**.
2. Allow **Port 80** from anywhere (public access).
3. Allow **Port 80** traffic from the Load Balancer to the target group.

**Creating the Load Balancer**

1. Navigate to **AWS EC2 → Load Balancers**.
2. Click **Create Load Balancer** → **Application Load Balancer**.
3. Place it in **Public Subnet 1 & 2**.
4. Associate it with the security group created for the **Load Balancer**.
5. Configure listeners and attach the **Target Group**.
6. Deploy the Load Balancer.

At this point, traffic from the internet is routed **securely** through the load balancer to the web server in the **private subnet**.

**Conclusion**

This architecture ensures:

* **Controlled network segmentation** with NAT Gateway.
* **Secure SSH access** with a Bastion Host.
* **Reliable inbound traffic handling** through a Load Balancer.

**AWS VPC Setup: NAT Gateway, Bastion Host & Load Balancer Configuration**

**1. Configuring the VPC, Subnets, and Routing**

To establish a **secure and scalable** cloud networking setup in AWS, we begin by creating the **VPC architecture** and configuring public and private subnets.

**Creating a VPC**

1. Go to the **AWS Console → VPC**.
2. Click **Create VPC**.
3. Name it "vprofile-VPC".
4. Use a **CIDR block** like 10.0.0.0/16.
5. Click **Create**.

**Creating Subnets**

To properly **segregate network traffic**, we need **two public and two private subnets**:

1. Navigate to **Subnets** → **Create Subnet**.
2. Select "vprofile-VPC".
3. Create:
   * **Public Subnet 1** (10.0.1.0/24)
   * **Public Subnet 2** (10.0.2.0/24)
   * **Private Subnet 1** (10.0.3.0/24)
   * **Private Subnet 2** (10.0.4.0/24)
4. Ensure public subnets have **auto-assign public IP enabled**.

**Creating and Associating an Internet Gateway**

Public subnet instances require an **Internet Gateway** to access external resources.

1. Go to **Internet Gateways → Create Internet Gateway**.
2. Name it "vprofile-IGW", attach it to "vprofile-VPC".
3. Update the **Public Route Table**:
   * Add a route: 0.0.0.0/0 → Internet Gateway.

**2. Setting Up NAT Gateway for Private Subnet Internet Access**

Since private subnets **cannot access the internet directly**, we use a **NAT Gateway** to allow outbound traffic.

**Allocating an Elastic IP**

Before creating the NAT Gateway, allocate an Elastic IP:

1. Go to **AWS Console → Elastic IPs**.
2. Click **Allocate Elastic IP**.

**Creating a NAT Gateway**

1. Navigate to **NAT Gateway → Create NAT Gateway**.
2. Place it in **Public Subnet 1 or 2**.
3. Attach the **Elastic IP**.
4. Click **Create NAT Gateway**.
5. Wait until its state changes to **available**.

**Updating Private Route Tables**

To route private subnet traffic through the NAT Gateway:

1. Navigate to **Route Tables**.
2. Click **Create Route Table**, name it "vprofile-private-route-table", associate with "vprofile-VPC".
3. Go to **Subnet Associations**, associate **both private subnets**.
4. Configure the **Routes**:
   * **Destination:** 0.0.0.0/0
   * **Target:** NAT Gateway.

**3. Setting Up a Bastion Host for Secure SSH Access**

**What is a Bastion Host?**

A **Bastion Host (Jump Server)** enables secure **SSH access** to private subnet resources, preventing direct exposure to the internet.

**Creating the Bastion Host**

1. Go to **AWS EC2** → **Launch Instance**.
2. Choose a suitable AMI:
   * **For production:** Use **CIS-hardened images** from AWS Marketplace.
   * **For learning:** Use **Ubuntu Server 22** (Free Tier).
3. Select **t2.micro** instance type.
4. Configure **Network Settings**:
   * Place it in **Public Subnet 1 or 2**.

**Creating a Security Group for SSH Access**

1. Go to **Security Groups → Create Security Group**.
2. Name it "vpro-bastion-sg", associate it with "vprofile-VPC".
3. Set inbound rules:
   * **Allow SSH only from trusted IPs (My IP / Corporate IP)**.

**Generating a Key Pair**

1. Navigate to **AWS Console → Key Pairs**.
2. Click **Create Key Pair**.
3. Name it "vpro-bastion-key.pem".
4. **Download** and store the key securely.

**Connecting to the Bastion Host**

Once the instance is running:

1. SSH into it using:

bash

ssh -i vpro-bastion-key.pem ubuntu@<Bastion Public IP>

**4. Creating a Private Instance and Accessing it via Bastion Host**

**Setting Up the Private Instance**

1. Launch a new **EC2 Instance** in **Private Subnet**.
2. Select **Amazon Linux 2 or Ubuntu**.
3. Use **Key Pair:** "web-key.pem" (to allow SSH from Bastion).
4. Set **Security Group Rules**:
   * **Allow SSH only from** "vpro-bastion-sg".

**Transferring the Web Instance Key to Bastion Host**

Since private instances **cannot be directly accessed**, we must copy the key into the Bastion Host:

1. Use SCP to transfer the private key:

bash

scp -i vpro-bastion-key.pem web-key.pem ubuntu@<Bastion Public IP>:~/

**Accessing Private Instance via Bastion**

1. SSH into **Bastion Host**.
2. SSH into **Private Instance**:

bash

ssh -i web-key.pem ec2-user@<Private Instance IP>

**5. Setting Up a Load Balancer for Web Access**

**Configuring the Web Server**

1. Install necessary packages:

bash

yum install -y httpd wget unzip

1. Download a website template:

bash

wget <template-download-link>

Move files to the web directory:

bash

mv <extracted-folder> /var/www/html

systemctl restart httpd

**Creating a Target Group**

1. Go to **AWS EC2 → Target Groups**.
2. Create a **Target Group** (web-tg) for instances.
3. Associate it with **Private Subnet EC2 Instance**.

**Configuring Load Balancer Security Groups**

1. Create **web-elb-sg**:
   * Allow **Port 80** traffic from **anywhere**.
   * Allow **Port 80** traffic to **Target Group**.

**Creating the Load Balancer**

1. Go to **AWS EC2 → Load Balancers**.
2. Click **Create Load Balancer** → **Application Load Balancer**.
3. **Network Mapping**:
   * Place it in **Public Subnet 1 & 2**.
4. Associate **Load Balancer Security Group**.
5. Attach **Target Group**.
6. Deploy **Load Balancer**.

**Updating Web Instance Security Group**

1. Navigate to **Web Instance Security Group**.
2. Update inbound rules:
   * **Allow Port 80** from **Load Balancer Security Group**.

**Testing Load Balancer**

1. Wait until **Load Balancer** status is available and **Target Group** instance is healthy.
2. Copy the **Load Balancer DNS name**.
3. Open a browser and paste the **Load Balancer URL**.

At this point, **traffic is securely routed** through the Load Balancer to the web server in the private subnet.

**6. Cleanup Procedures**

To **avoid unnecessary charges**, delete the following after testing:

1. **Load Balancer**

bash

aws elbv2 delete-load-balancer --load-balancer-arn <LoadBalancerARN>

1. **Terminate Instances**
2. **Delete NAT Gateway**
3. **Release Elastic IPs**

**Conclusion**

This setup provides:

* **Secure network segmentation** via NAT Gateway.
* **Controlled SSH access** through a Bastion Host.
* **Efficient inbound traffic routing** using Load Balancer.

**AWS VPC Peering Setup: Connecting Multiple VPCs Across Regions**

**1. Overview of VPC Peering**

**Why VPC Peering?**

In large-scale cloud architectures, it is common to have **multiple VPCs**:

* One VPC for **web infrastructure**.
* Another for **APIs or microservices**.
* A separate **database VPC** for enhanced security.

Since VPCs are **isolated by default**, peering allows **direct communication** between VPCs **without using public internet**.

**Key Benefits of VPC Peering**

* **Secure private communication** between instances in different VPCs.
* **Eliminates the need for VPN or public endpoints** for inter-VPC traffic.
* **Lower latency** compared to routing through external networks.
* **Direct network-level access** without NAT or proxies.

**2. Creating a New VPC for Peering**

We have an existing VPC, "vprofile-VPC" in **North California**. Now, we create a second VPC, "vpro-db", in **Oregon region**.

**Steps to Create a New VPC**

1. **Navigate to AWS Console → VPC**.
2. Click **Create VPC**.
3. Name the VPC "vpro-db".
4. Choose a **CIDR block** that does not overlap:
   * "vprofile-VPC" uses 172.20.0.0/16.
   * "vpro-db" must use 172.21.0.0/16 or any non-overlapping range.
5. Click **Create VPC**.

**Creating a Route Table for the New VPC**

1. Go to **Route Tables**.
2. Name the default route table "vprodb-rt" for clarity.

**3. Establishing VPC Peering Connection**

**Step-by-Step Peering Configuration**

1. **Go to VPC Console** → **Peering Connections**.
2. Click **Create Peering Connection**.
3. Name it "vprofile-db-peering".
4. Configure the requester and accepter VPCs:
   * **Requester VPC:** "vprofile-VPC" (North California).
   * **Accepter VPC:** "vpro-db" (Oregon).
5. Click **Create Peering Connection**.
6. Wait for the connection state to become **Pending Acceptance**.

**Accepting the Peering Request**

Since "vpro-db" is the **accepter VPC**, follow these steps:

1. **Go to VPC Console** in **Oregon**.
2. Navigate to **Peering Connections**.
3. Locate "vprofile-db-peering".
4. Click **Accept Request**.

Now, both VPCs are **connected**, but **routes must be added** for traffic flow.

**4. Updating Route Tables for Peering**

Without proper routing, instances in "vprofile-VPC" cannot reach those in "vpro-db".

**Modifying Routes in "vprofile-VPC"**

1. Navigate to **Route Tables**.
2. Select "vprofile-private-route-table".
3. Click **Edit Routes**.
4. Add the following route:
   * **Destination:** 172.21.0.0/16 (CIDR of "vpro-db").
   * **Target:** "vprofile-db-peering".

**Modifying Routes in "vpro-db"**

1. Select "vprodb-rt" from **Route Tables**.
2. Click **Edit Routes**.
3. Add the following route:
   * **Destination:** 172.20.0.0/16 (CIDR of "vprofile-VPC").
   * **Target:** "vprofile-db-peering".

**5. Security Group Configuration for Inter-VPC Traffic**

Even after peering and route updates, **security groups block all traffic by default**.

**Allowing Traffic Between Peered VPCs**

1. Go to **Security Groups** in both "vprofile-VPC" and "vpro-db".
2. Edit inbound rules:
   * Allow **SSH (Port 22)** from the **peered VPC CIDR**.
   * Allow **application-specific ports** (e.g., 3306 for databases).
3. Save changes.

Now, instances can securely **communicate across VPCs**.

**6. Validating VPC Peering Setup**

**Testing Connectivity**

1. SSH into an instance in "vprofile-VPC":

bash

ssh -i vprofile-key.pem ec2-user@<instance-in-vprofile-VPC>

1. **Ping a private IP** of an instance in "vpro-db":

bash

ping <private-ip-of-instance-in-vpro-db>

1. If successful, **VPC Peering is working!**

**Troubleshooting Steps**

* **Peering connection state:** Ensure it is **active**.
* **Route table updates:** Check correct CIDR blocks in both VPCs.
* **Security group rules:** Validate inbound/outbound permissions.

**7. Cleanup Steps (If Required)**

If VPC peering is no longer needed:

1. **Delete the Peering Connection**.
2. **Remove Peering Routes** from both VPCs.
3. **Revert Security Group Rules**.

**Conclusion**

With this **VPC Peering configuration**, we achieve:

✅ **Secure private communication between VPCs**.

✅ **Seamless data exchange without using public endpoints**.

✅ **Optimized cross-region networking with minimal latency**.