
AWS
Solutions
Architect Associate
– Week 1 Notes
Meta AI

AWS Solutions Architect Associate - Day 1 Notes

Topic: AWS Well-Architected Framework

The AWS Well-Architected Framework solves common problems faced in cloud computing. It is a set of guidelines or a blueprint that helps to build the best architectures, which are secure, durable, resilient, scalable, etc. This helps to build trust.

Its six main pillars are:

1. Security

Goal: Protect data and assets.

Key Considerations:

- Confidentiality of data.
- Identity and Access Management (IAM).
- Detecting and mitigating threats and risks.

2. Cost Optimization

Goal: Use only necessary resources and avoid unnecessary costs/resources.

Key Considerations:

- Eliminating unneeded resources.
- Choosing the right resource types and sizes.
- Taking advantage of pricing models (e.g., Reserved Instances, Spot Instances).
- Optimizing for consumption.

3. Performance Efficiency

Goal: Use resources efficiently and maintain efficiency as changes occur.

Key Considerations:

- Selecting appropriate resource types.
- Monitoring performance.
- Adapting to demand (e.g., auto-scaling).
- Using serverless architectures where appropriate.

4. Sustainability

Goal: Reduce carbon footprint and minimize environmental impact.

Key Considerations:

- Optimizing resource utilization.
- Adopting managed services.
- Reducing data transfer.
- Choosing energy-efficient regions.

5. Operational Excellence

Goal: Ensure smooth work is done and improve operations over time.

Key Considerations:

- **Business Continuity:** Running and monitoring the system.
- The system should be efficient, reliable, and improve over time.
- **Practices for Operational Excellence:**
    - **Use operations as code:** This automates work, reduces human error, and speeds up processes (e.g., using Terraform).
    - **Make frequent, small, and reversible changes:** This minimizes risk and allows for quick rollbacks.

    - **Anticipate failures:** Design systems to be resilient to expected issues.
    - **Continuously improve:** Regularly review and refine operational procedures and architectures.

6. Reliability

Goal: Ensure the system performs its intended function correctly and consistently.

Key Considerations:

- **Backup and System Recovery:** Implementing robust backup strategies and disaster recovery plans.
- **Dynamically Meeting Demand:** Designing systems to scale up or down based on fluctuating demand.
- **Mitigate Risk:** Identifying and addressing potential failure points.
- **Practices for Reliability:**
    - **Automatically recover from failure:** Implement mechanisms like auto-healing and self-correction.
    - **Test recovery policies:** Regularly validate that disaster recovery plans work as expected.
    - **Scale horizontally:** Distribute load across multiple resources so that if one fails, others will continue to work, ensuring high availability.

AWS Solutions Architect Associate - Day 2 Notes

Topic: AWS Global Infrastructure

The AWS Global Infrastructure forms the backbone of AWS, designed to provide robust, highly available, and scalable services worldwide.

Key Components:

- **Regions:**
  - Large, geographically distinct areas (e.g., `us-east-1`).
  - Each region consists of multiple isolated Availability Zones and data centers.
  - Designed for high availability and fault tolerance.
- **Availability Zones (AZs):**
  - Located within a Region.
  - Each AZ comprises one or more discrete data centers with redundant power, networking, and connectivity.
  - They are physically separated by a meaningful distance (typically 10s to 100s of kilometers) to minimize the risk of a single event impacting multiple AZs, but close enough to enable ultra-low latency connections between them.
- **Edge Locations (Points of Presence - PoPs):**
  - Globally distributed data centers designed for low-latency content delivery.
  - They cache frequently used data closer to end-users.
  - Primarily used by Amazon CloudFront (Content Delivery Network - CDN) and AWS WAF (Web Application Firewall) to protect against DDoS attacks.
  - There are currently over 400+ edge locations worldwide.

AWS Service Types by Scope:

- **AWS Global Services:** Services that are not tied to a specific region and are accessible globally.
    - Examples: IAM (Identity and Access Management), Route 53 (DNS), CloudFront (CDN), AWS WAF (Web Application Firewall).
- **AWS Region-Specific Services:** Services that are deployed within specific AWS Regions.
    - Examples: EC2 (Elastic Compute Cloud - IaaS), Elastic Beanstalk (PaaS), Lambda (Function as a Service - FaaS).

Topic: AWS Identity and Access Management (IAM)

AWS IAM is a service that helps you securely manage identities and access to AWS resources. It focuses on **authentication** (verifying who you are) and **authorization** (determining what you can do). The core principle of IAM is **least privilege**, meaning users and resources should only have the minimum permissions required to perform their tasks.

Key IAM Components:

- **Root User:**
    - The main account created when you first set up your AWS account.
    - Possesses all permissions and cannot be restricted.
    - **Best Practice:** Never use the root user for daily tasks. Use it only for initial setup and highly sensitive account management.
- **Users:**
    - Represents a person, individual application, or service within your AWS account.
    - Can be granted programmatic access via an **Access Key ID** (public key) and a **Secret Access Key** (private key) for interacting with AWS CLI, SDKs, and APIs.

- **Groups:**
    - A collection of IAM users.
    - Permissions are attached to the group, and all users within that group inherit those permissions.
    - **Important:** There is no nesting of groups (a group cannot contain another group).
    - One user can belong to multiple groups.
- **Roles:**
    - IAM entities that define a set of permissions for making AWS service requests.
    - They are designed to grant **temporary access** to users or AWS services.
    - **Main Use Case:** Allowing an AWS resource (e.g., an EC2 instance) to securely access another AWS service (e.g., S3).
    - Example: An auditor might assume a specific role that only grants read-only access to certain resources.
- **IAM Policies:**
    - JSON documents that define permissions.
    - They specify what actions are allowed or denied on which resources.
    - Policies can be attached to users, groups, roles, or directly to resources.
    - **Key Elements of a Policy:**
        - `Effect`: Specifies whether the policy `Allow`s or `Deny`s access.
        - `Principal`: The entity (user, role, or AWS service) that is allowed or denied access.
        - `Action`: The specific API operations that are allowed or denied (e.g., `s3:GetObject`, `ec2:RunInstances`).
        - `Resource`: The AWS resource(s) on which the action can be performed (e.g., a specific S3 bucket, an EC2 instance).
        - `Sid` (Statement ID): An optional identifier for the policy statement.
    - **Types of Policies:**

    - **AWS Managed Policies:** Predefined policies created and managed by AWS (e.g., `AmazonS3ReadOnlyAccess`).
    - **Customer Managed Policies:** Custom policies created and managed by you. These can be reused and attached to multiple users, groups, or roles.
    - **Inline Policies:** Policies embedded directly within a single user, group, or role. They cannot be reused and are deleted if the associated entity is deleted.

IAM Key Features:

- **MFA (Multi-Factor Authentication):** Adds an extra layer of security by requiring more than one method of verification.
- **Access Analyzer:** Helps you identify unintended access to your external resources (e.g., S3 buckets, SQS queues) by external entities at the account level.
- **Credential Report:** A report that lists all your account's users and the status of their various credentials (passwords, access keys, MFA devices). Useful for security audits.
- **Access Advisor:** Shows the services that a user has accessed and the last time they accessed them. This helps in refining permissions to the principle of least privilege at the user level.

AWS Command Line Interface (CLI):

- An open-source tool that allows you to interact with AWS services from your command line.
- Enables automation, scripting, and managing AWS resources programmatically.
- Requires an **Access Key ID** (public key) and a **Secret Access Key** (private key) for authentication.

AWS Software Development Kit (AWS SDK):

- A set of libraries that enables you to access and manage AWS services programmatically using various programming languages (e.g., Python, Java, Node.js, .NET).

AWS Solutions Architect Associate - Day 3 Notes

Topic: Guided Lab: Exploring AWS Identity and Access Management (IAM)

Lab Summary

This lab was a hands-on practice session for AWS IAM. We learned how users get permissions from groups and tested these permissions in a real scenario.

Main Goals:

- See pre-made IAM users and groups.
- Check IAM policies attached to groups.
- Add users to groups based on their job roles.
- Find and use the IAM login page.
- Test if users could access services as expected.

Time: About 40 minutes.

Task 1: Understanding Users, Groups, and Policies

We looked at existing IAM users (`user-1`, `user-2`, `user-3`) and groups (`EC2-Admin`, `EC2-Support`, `S3-Support`).

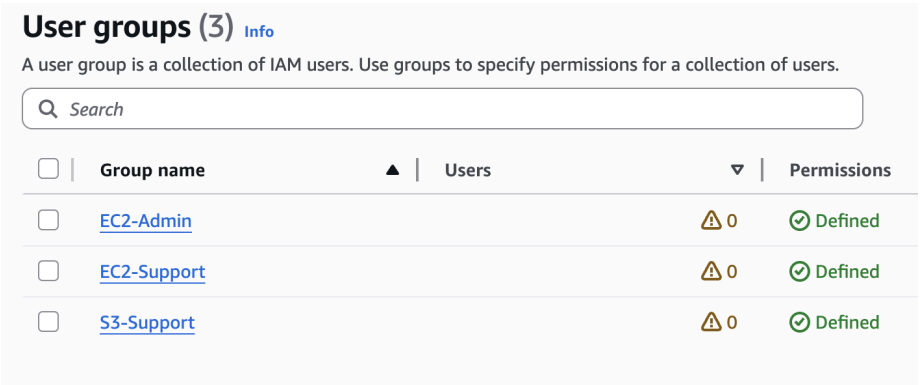
- **Policy Types:**
    - **Managed Policies:** These are AWS-made policies, like `AmazonEC2ReadOnlyAccess` for `EC2-Support` and `AmazonS3ReadOnlyAccess` for `S3-Support`. They update automatically and are reusable.

        - For example, `AmazonEC2ReadOnlyAccess` lets you *view* EC2, Load Balancers, CloudWatch, and Auto Scaling info.
        - Another example, `AmazonS3ReadOnlyAccess` lets you *view* S3 buckets and their contents.
    - **Inline Policies:** These policies are attached directly to one user or group, like `EC2-Admin-Policy` for `EC2-Admin`. They are for specific situations and aren't reusable.
        - For example, `EC2-Admin-Policy` lets you *view, start, and stop* EC2 instances.

Business Scenario

We set up access for new staff based on their job functions:

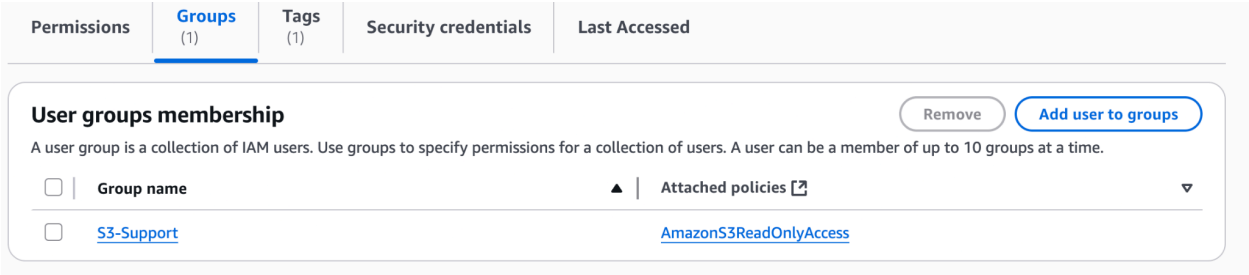
- `user-1` was assigned to the `S3-Support` group for read-only access to Amazon S3.
- `user-2` was assigned to the `EC2-Support` group for read-only access to Amazon EC2.
- `user-3` was assigned to the `EC2-Admin` group for viewing, starting, and stopping Amazon EC2 instances.

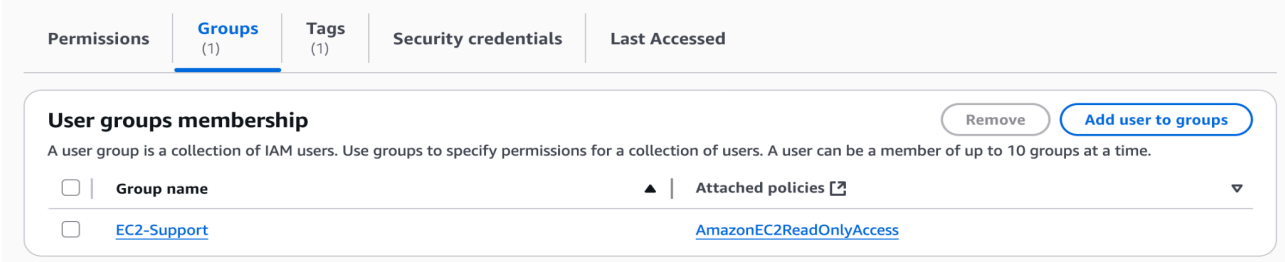


Task 2: Adding Users to Groups

We added each user to their assigned group:

- `user-1` joined `S3-Support`.
- `user-2` joined `EC2-Support`.
- `user-3` joined `EC2-Admin`. Each group then correctly showed 1 user.

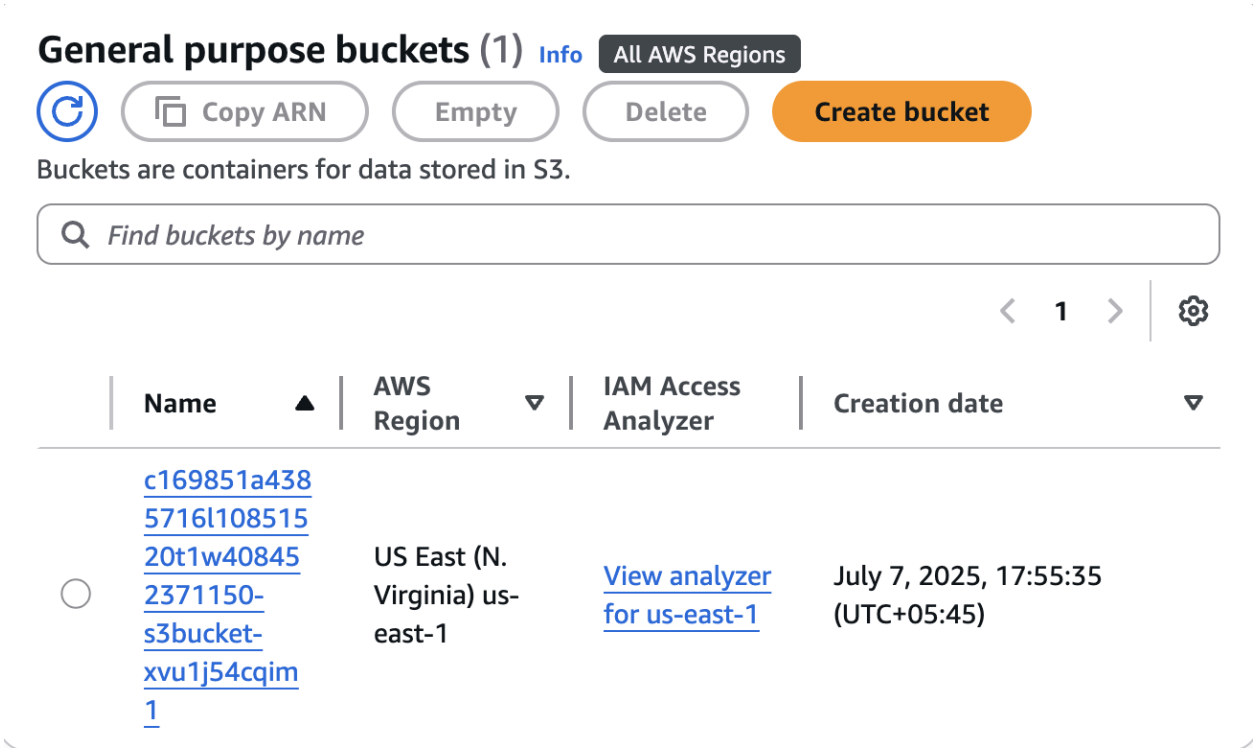




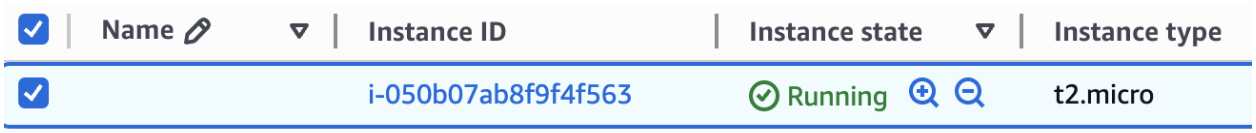
Task 3: Testing User Permissions

We logged in as each user in a private browser to check their access.

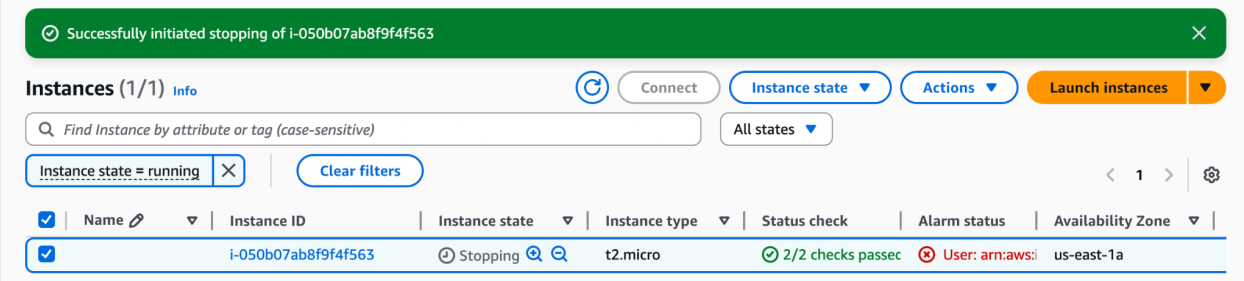
- **`user-1` (S3 Support):**
  - **S3:** Could view S3 buckets and contents (Success).
  - **EC2:** Got "not authorized" error when trying to access EC2 (Correct).



- `user-2` **(EC2 Support):**
  - **EC2:** Could view EC2 instances (Success).
  - **EC2 Action:** Got "not authorized" error when trying to stop an EC2 instance (Correct, as it's read-only).
  - **S3:** Got "don't have permissions" error when trying to access S3 (Correct).



- `user-3` **(EC2 Administrator):**
  - **EC2:** Could view EC2 instances (Success).
  - **EC2 Action:** Successfully stopped an EC2 instance (Success, confirming admin rights).



Conclusion

This lab helped us understand how AWS IAM works in practice. We saw how users get permissions from groups, how different policy types work, and how the "least privilege" rule keeps things secure.

AWS Solutions Architect Associate - Day 4 Notes

Topic: How Networking Works in AWS

Today, we learned about the basic parts of networking that help us build things in AWS.

1. CIDR (Classless Inter-Domain Routing)

- **What it is:** A way to set a range of IP addresses.
- **How it looks:** An IP address plus a number like `/24` (e.g., `10.0.0.0/24`).
- **The `/XX` number:** This number tells us how many IPs are in the range.
    - `/0` means a very big range (almost all IPs).
    - `/32` means just one single IP address.
    - **AWS Special IPs:** AWS always keeps the first five IP addresses in any network you make for its own use. So, plan your IPs knowing these will be taken.

2. Public vs. Private IP Addresses

- **Public IPs:**
    - Unique around the world on the internet.
    - Let your AWS resources be seen and reached from anywhere on the internet.
- **Private IPs:**
    - Used only inside your own private network (like your VPC).
    - Can be used again in different, separate private networks without causing problems.
    - Cannot be reached directly from the internet.

3. VPC (Virtual Private Cloud)

- **What it is:** Your own private, isolated section of the AWS Cloud. It's like having your own data center inside AWS.
- **Purpose:** Keeps your resources separate from others.
- **Limits:** You can have up to 5 VPCs in each AWS Region.
- **IP Range for VPCs:**
    - Smallest size: `/28` (e.g., `10.0.0.0/28`).
    - Biggest size: `/16` (e.g., `10.0.0.0/16`).
- **Default VPC:** AWS gives you one VPC automatically when you start. It usually has an IP range like `/16` (e.g., `172.31.0.0/16`).

4. Internet Gateway (IGW)

- **What it does:** Lets your VPC talk to the internet.
- **How many:** You can only have one Internet Gateway per VPC.
- **Strength:** It's built to handle lots of traffic and is very reliable, so it won't break easily.

5. Bastion Host

- **What it is:** A special EC2 server placed in a public part of your network. It's like a secure jump point.
- **How it helps:** You connect to this server first, and then from there, you can securely connect to your private servers that are not directly on the internet.
- **Security:** Needs specific rules to allow only certain connections to it.

6. NAT (Network Address Translation)

- **What it does:** Allows your private servers to connect to the internet (or other AWS services) without being directly exposed to the internet. It changes their private IP to a public one when they go out.
- **Two Types:**

    - **NAT Instance (Old Way):**
        - An EC2 server that does the NAT job.
        - **Speed:** Depends on how powerful the EC2 server is (a small one like `t2.micro` will be slow).
        - **Management:** You have to set it up and manage it yourself.
        - **Security:** You can put security rules on it.
        - **Cost:** Cheaper for very little traffic.
    - **NAT Gateway (Newer, Recommended Way):**
        - An AWS service that handles NAT for you.
        - **Speed:** Very fast, up to 100 Gbps, and its speed doesn't depend on an EC2 server type.
        - **Management:** AWS manages it completely, so you don't have to do anything.
        - **Security:** AWS handles its security.
        - **Cost:** Costs more than a NAT instance, but it's more reliable and performs better.
        - **Setup:** Very easy to set up.

7. Security Groups and Network Access Control Lists (NACLs)

These are two different ways to add security to your network.

- **Security Group:**
    - Acts like a **firewall for individual servers** (like your EC2 instances).
    - **Smart (Stateful):** If you allow traffic *in*, it automatically allows the reply traffic *out* for that connection. You only need to set rules for one direction.
    - By default, it blocks all incoming traffic but allows all outgoing traffic.
    - Attached directly to your servers.
- **Network Access Control List (NACL):**
    - Acts like a **firewall for an entire network section** (a subnet).
    - **Not Smart (Stateless):** If you allow traffic *in*, you *must also* set a rule to allow the reply traffic *out*. You need to set rules for both directions.

- The default NACL allows all traffic in and out. If you make your own, it blocks all traffic by default.
- Rules are followed in order, starting with the lowest rule number.
- Works with **ephemeral ports**.
    - **Ephemeral Ports:** These are temporary, random port numbers (usually high numbers like 1024-65535) that your computer uses when it starts a connection to a server. For example, when your computer connects to a website, it uses one of these random ports. NACLs need rules for these port ranges in both directions for connections to work.

AWS Solutions Architect Associate - Day 5 Notes

Topic: Vpc peering,Transit Gateway, Vpc endpoint, site to Site vpn, Direct Connect (DX), vpc flowlog, Vpc traffic Mirroring

1. VPC Peering

- Like a direct, private cable connecting two separate networks (VPCs) so they can talk to each other.
- Can connect your own networks or networks from different AWS accounts.
- You don't need the internet, a special VPN, or a Direct Connect cable for them to chat.
- Keeps your data safe and private when moving between your networks.
- **Important Things to Remember:**
    - **No Same IPs:** The IP addresses in the two networks must be different. No overlap!
    - **No Jumping Through:** If Network A talks to B, and B talks to C, Network A *cannot* talk to C *through* B. You'd need a direct link from A to C.
    - **Can Get Messy:** If you have too many networks, keeping track of all these direct links can get complicated.
    - **Cost:** Cheaper if networks are in the same AWS region. Costs more if they are far apart in different regions.
- **Better for Big Needs:** For very complex setups, there's something called Transit Gateway.

2. Transit Gateway

- A central "hub" or "router" in the cloud that connects all your networks (VPCs) and even your office networks.
- Helps you manage all your network connections from one spot.
- Unlike VPC peering, this hub *does* let networks talk through it. So, if Network A and Network C both connect to the Transit Gateway, A can talk to C.
- Can connect many networks, making your overall setup simpler and easier to grow.
- **Cost:** Costs more than simple VPC peering, but it's better for big, growing networks.

3. VPC Endpoints

- A special way to connect your network (VPC) to other AWS services (like S3 for storage or DynamoDB for databases) *without* ever touching the public internet.
- **Good Stuff:**
    - **Faster and Safer:** Your data stays private and moves quickly.
    - **Saves Money:** You don't pay for internet data charges that you would with other ways.
    - **Another Name:** Also called "PrivateLink."
- **Two Types:**
    - **Interface Endpoint:**
        - **For:** Most AWS services (like tools for monitoring or security).
        - **Cost:** You pay a small fee for this.
        - **How it Works:** It creates a **virtual network card (ENI)** in your network.
            - **What is an ENI?** Imagine a virtual plug for your computer in the cloud. It gives your server a private address and lets it connect to your private network.
    - **Gateway Endpoint:**

    - **For:** Only two specific AWS services: **Amazon S3** (for storage) and **Amazon DynamoDB** (for databases).
    - **Cost:** This one is **free** to use!
    - **How it Works:** It adds a special rule to your network's "directions map" (route table) that sends traffic for S3 or DynamoDB directly to AWS's private network, skipping the internet completely.

4. Site-to-Site VPN

- A secure, encrypted "tunnel" created over the regular internet. It connects your office network (or another cloud) to your AWS network (VPC).
- **VPN (Virtual Private Network):** A general term for making a safe connection over an unsafe network, like hiding your computer's public address.
- **Security:** Uses strong encryption (IPsec) to keep your data secret as it travels.
- **Cost:** Usually cheaper than a super-fast, dedicated cable.
- **Speed:** Can be slower because it uses the internet.
- **Security Concern:** Not as safe as a direct cable because it still uses the public internet.
- **Good For:** When you need to connect your office to AWS, but don't need the fastest speed or highest security.

5. AWS Direct Connect

- A dedicated, physical cable connection (fiber optic) that goes straight from your office or data center to an AWS location.
- **Speed:** Very, very fast.
- **Setup Time:** Can take a long time to get installed because it's a physical cable.
- **Security:** Much safer because it *never* uses the public internet.
- **Latency:** Very quick response responses (almost no delay).
- **Cost:** Expensive to set up, but cheaper if you send a lot of data all the time.
- **Requirement:** You need to be physically near an AWS Direct Connect location.
- **Two Types:**

    - **Dedicated Connection:**
        - Your own private cable straight to AWS. No sharing.
        - **Cost:** More expensive.
        - **Growth:** Not easy to quickly change its speed.
    - **Hosted Connection:**
        - You share a part of a connection that an AWS partner provides.
        - **Permission:** You need approval from the partner.
        - **Cost:** Cheaper than your own dedicated cable.
        - **Growth:** Easier to change its speed as needed.

6. VPC Flow Logs

- Like a security camera for your network traffic. It records details about the IP traffic going in and out of your VPC, subnets, or network interfaces.
- **Purpose:** Mainly used for monitoring and security.
- **What it shows:** It shows *metadata* (who, when, how much data), not the actual content of the data. For example, "10.0.0.5 sent 500 bytes to 10.0.0.8."
- **Where logs go:** The logs are stored in Amazon S3 or Amazon CloudWatch Logs.
- **Bandwidth Impact:** Has very little effect on your network speed.
- **Use Cases:** Good for auditing, general monitoring, and basic troubleshooting.
- **Filtering:** You can filter logs (e.g., to see only accepted or rejected traffic).

7. VPC Traffic Mirroring

- A more advanced tool that makes a full copy of your network traffic from an EC2 instance.
- **Purpose:** For deep inspection, security checks, and finding problems.
- **What it shows:** It shows the *full packets* (the actual data content, including headers and payloads). For example, it could show the actual text of an HTTP request.

- **Where it sends data:** The copied traffic is sent to another EC2 instance, a Network Load Balancer (NLB), or a special tool (like Wireshark) for analysis.
- **Can see passwords/files?** Yes, if the data was sent without encryption (in clear text).
- **Bandwidth Impact:** Can use a lot of network bandwidth because it copies all traffic.
- **Use Cases:** Great for deep security checks, digital forensics, and using Intrusion Detection Systems (IDS).
- **Works with:** Only works with newer EC2 instances (Nitro-based instances).

IPv4 and IPv6 (Internet Protocol Versions)

These are the rules that allow devices to talk to each other on a network, like addresses for houses.

- **IPv4 (Internet Protocol version 4):**
    - Uses 32-bit addresses (e.g., `192.168.1.1`).
    - Can create about 4.3 billion unique addresses.
    - **Why it's still preferred/used:**
        - **Works Everywhere:** Most older devices and systems still use IPv4. It's widely compatible.
        - **Simple:** The addresses are shorter and easier for humans to remember and type.
        - **Cost:** Changing everything to IPv6 costs money for new equipment and training.
        - **NAT Helps:** Network Address Translation (NAT) helps IPv4 last longer by letting many private devices share one public IP.
        - **Slow Change:** Not everyone has moved to IPv6 yet, so IPv4 is still needed.
- **IPv6 (Internet Protocol version 6):**

- Uses 128-bit addresses (e.g., `2001:0db8:85a3:0000:0000:8a2e:0370:7334`).
- Can create a *huge* number of unique addresses (trillions upon trillions), solving the problem of running out of IPs.
- **Why it's important (more than just running out of addresses):**
  - **Better Routing:** Can send data more efficiently because its structure is simpler.
  - **Built-in Security:** Has security features (like IPsec) built right in, making connections safer by default.
  - **Auto-Setup:** Devices can often set up their own IP addresses automatically, making network management easier.
  - **Good for Mobile/IoT:** Works better with mobile devices and the massive number of Internet of Things (IoT) devices.
  - **Better Quality:** Can handle different types of traffic better, which is good for things like video calls or online gaming.
  - **No NAT Needed:** Because there are so many addresses, the complex NAT system is often not needed, simplifying networks.

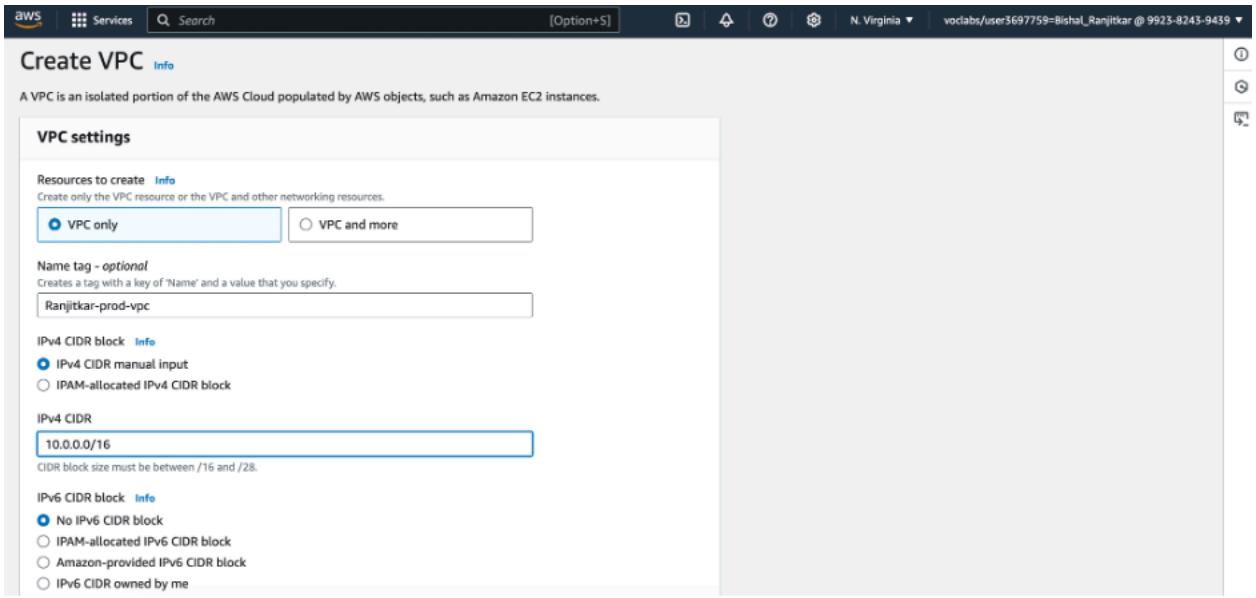
AWS Solutions Architect Associate - Day 6 Notes

Topic: Guided Lab: Creating a VPC

Task 1: Creating a VPC

We started by making a new VPC.

- **VPC Name:** `Lab VPC`
- **IP Range (CIDR):** `10.0.0.0/16` (This gives us over 65,000 IP addresses).
- We also turned on **DNS hostnames** so our servers would get easy-to-read names.

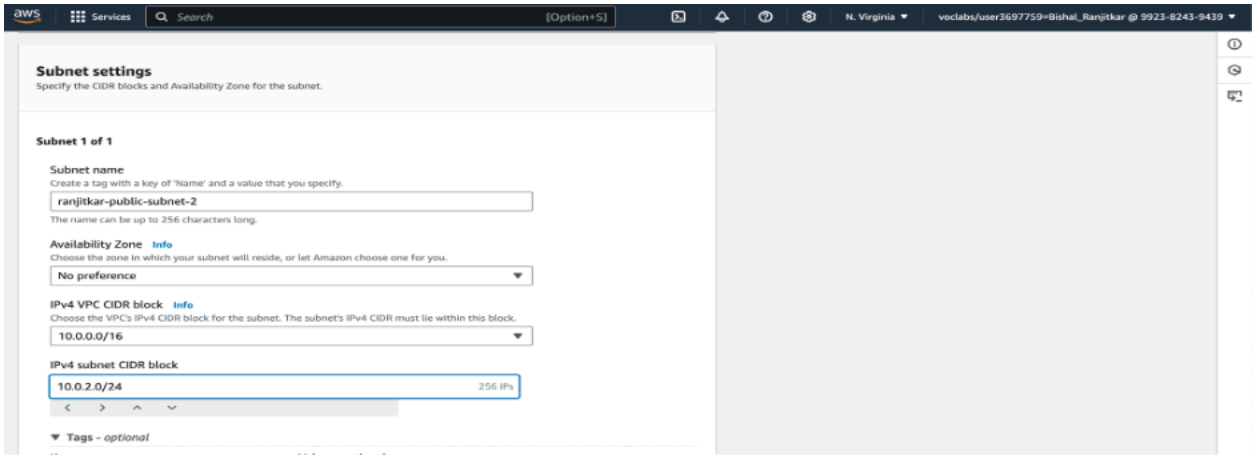


Task 2: Creating Subnets

Next, we made two smaller networks (subnets) inside our VPC.

- **Public Subnet:**

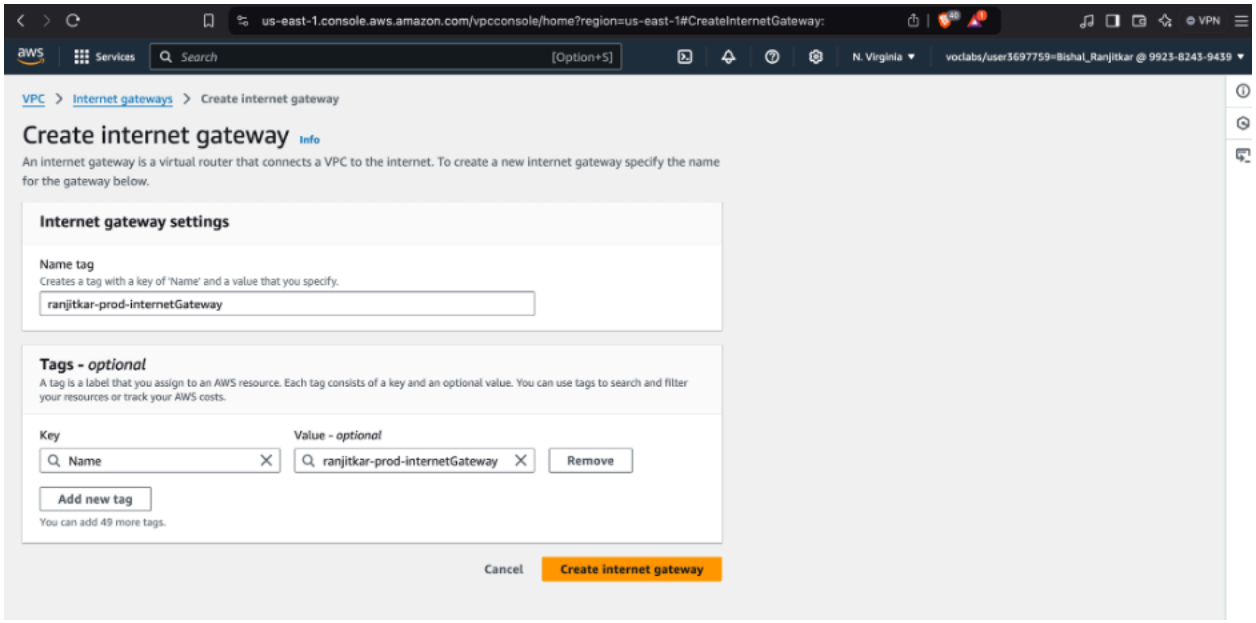
- **Name:** `Public Subnet`
- **IP Range (CIDR):** `10.0.0.0/24`
- We set it to automatically give public IP addresses to any server launched in it. This subnet is for internet-facing resources.



Task 3: Creating an Internet Gateway

We created an Internet Gateway (IGW) to let our VPC talk to the internet.

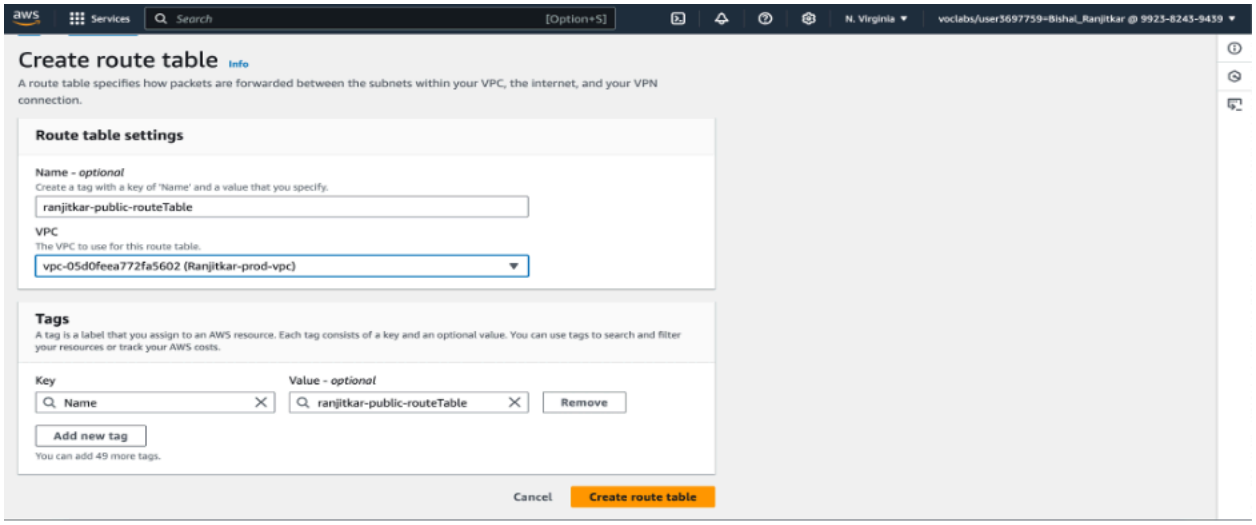
- **Name:** `Lab IGW`
- We then **attached** this IGW to our `Lab VPC`. An IGW is very reliable and handles lots of traffic.



Task 4: Configuring Route Tables

Route tables tell network traffic where to go.

- We **renamed** the default route table to `Private Route Table`.
- We created a **new route table** called `Public Route Table`.
- In the `Public Route Table`, we added a rule to send all internet-bound traffic (`0.0.0.0/0`) to our `Lab IGW`.
- Finally, we **connected** our `Public Subnet` to this `Public Route Table`. This step made the `Public Subnet` truly public.



Task 5: Creating a Security Group

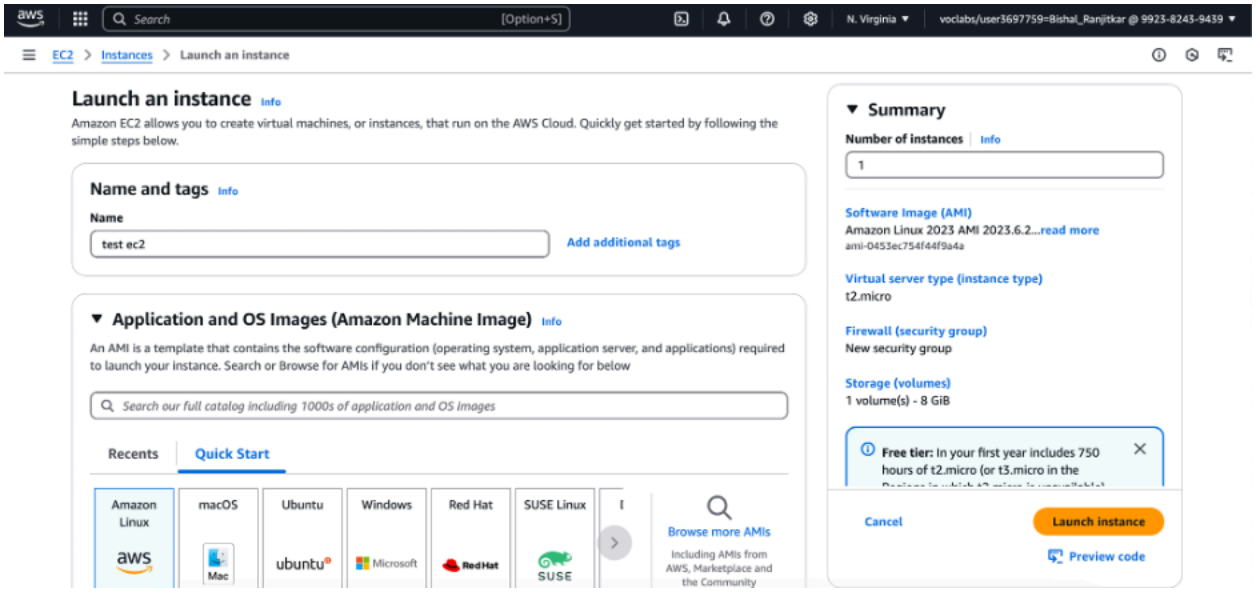
We made a security group, which acts like a firewall for individual servers.

- **Name:** `App-SG`
- **Description:** `Allow HTTP traffic`
- We added a rule to **allow incoming HTTP (web) traffic** on port 80 from anywhere on the internet. This group will protect our web server.

Task 6: Launching an Application Server

To test our VPC setup, we launched an EC2 instance (a virtual server) into our `Public Subnet`.

- **Name:** `App Server`
- We chose a basic Linux image (`Amazon Linux 2023 AMI`) and a small server type (`t2.micro`).
- We made sure it used our `Lab VPC`, `Public Subnet`, and the `App-SG` security group.
- After the server started, we copied its **Public IPv4 DNS** and pasted it into a web browser.



Conclusion

This lab successfully taught us how to:

- Build a VPC from scratch.
- Set up public subnets.
- Connect our VPC to the internet using an Internet Gateway.
- Configure network traffic rules with route tables.
- Secure our servers with security groups.
- Launch and test an application server within our custom VPC.

AWS Solutions Architect Associate - Day 7 Notes

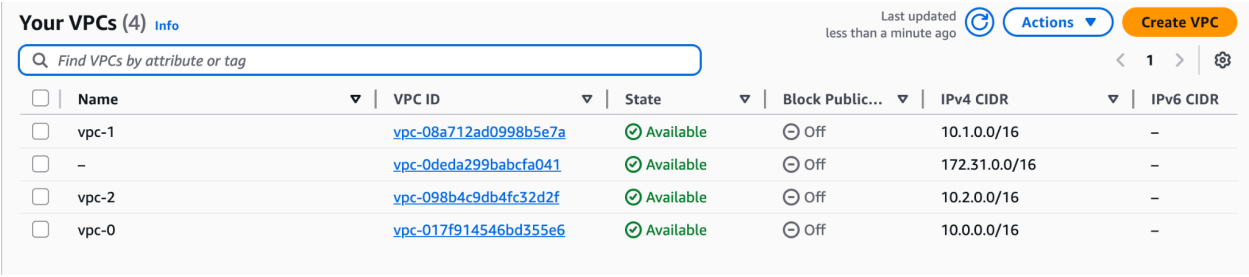
Topic: Guided Lab: VPC Peering

Lab Summary

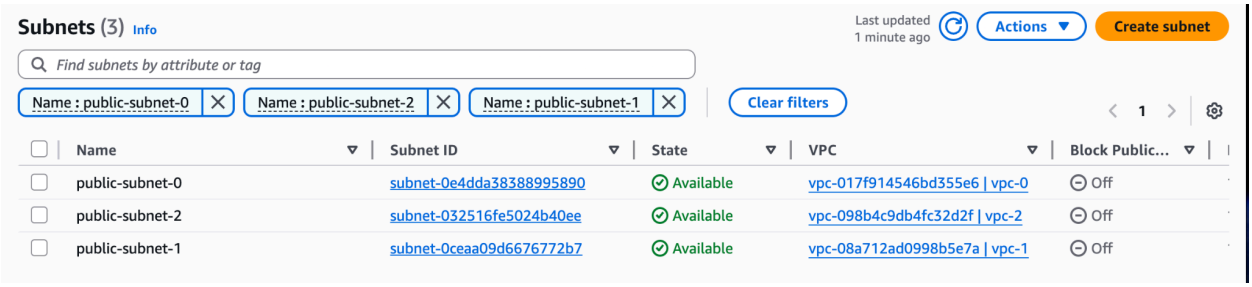
Today's lab was a hands-on session where we learned how to connect different private networks (VPCs) using VPC peering. This allows them to share data directly and securely.

What we built and connected:

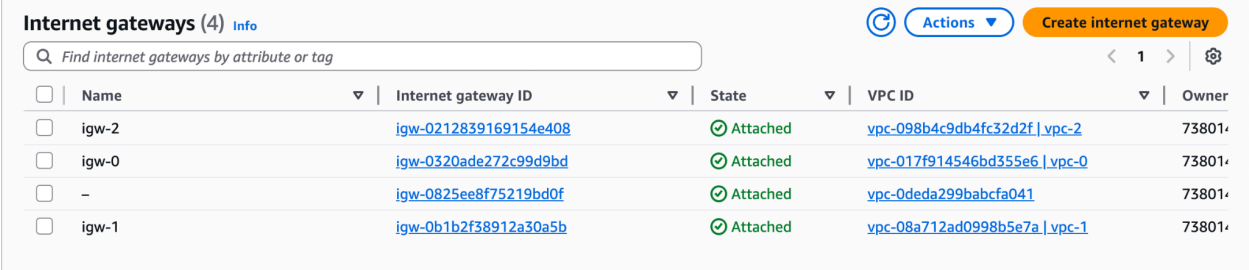
- **3 VPCs:** Our main private networks.



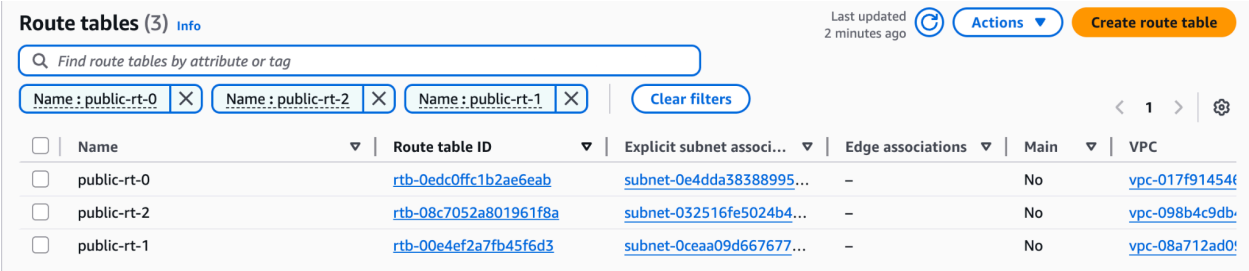
- **3 Public Subnets:** One in each VPC, for internet-facing resources.



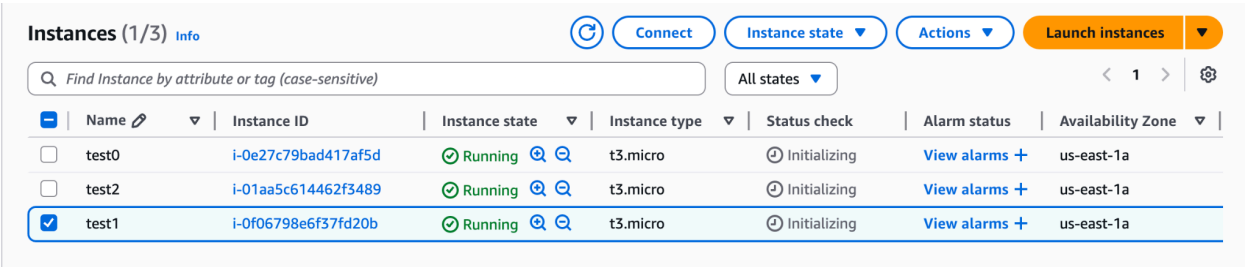
- **Internet Gateways (IGWs):** One for each VPC, to allow internet access.



- **Route Tables:** To tell network traffic where to go.



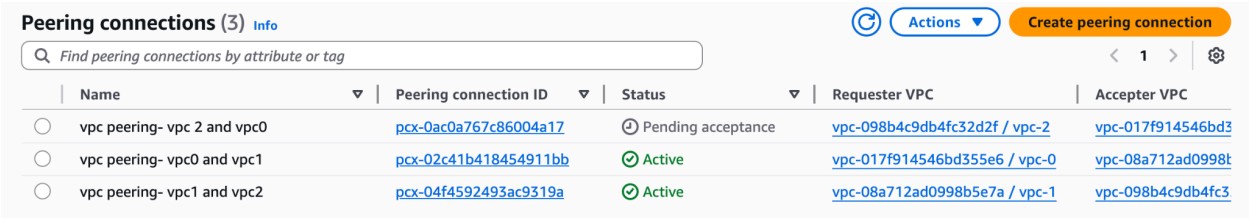
- **3 EC2 Instances:** One virtual server in each VPC's public subnet.



Key Steps: VPC Peering Setup

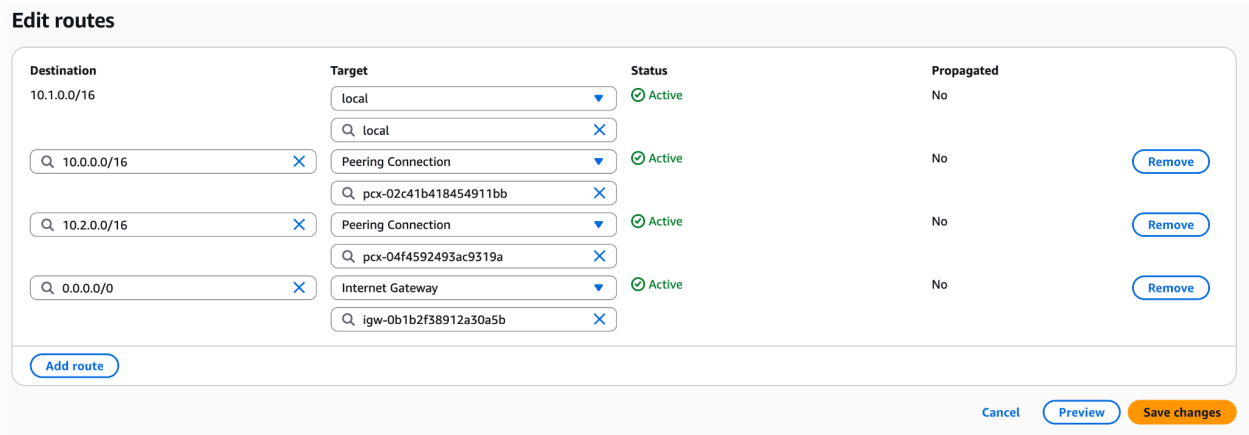
The main part of the lab was setting up the VPC peering connections:

1. **VPC Peering Connection 1:** Between `VPC 0` and `VPC 1`.
2. **VPC Peering Connection 2:** Between `VPC 1` and `VPC 2`.
3. **VPC Peering Connection 3:** Between `VPC 2` and `VPC 0`.



Route Table Configuration

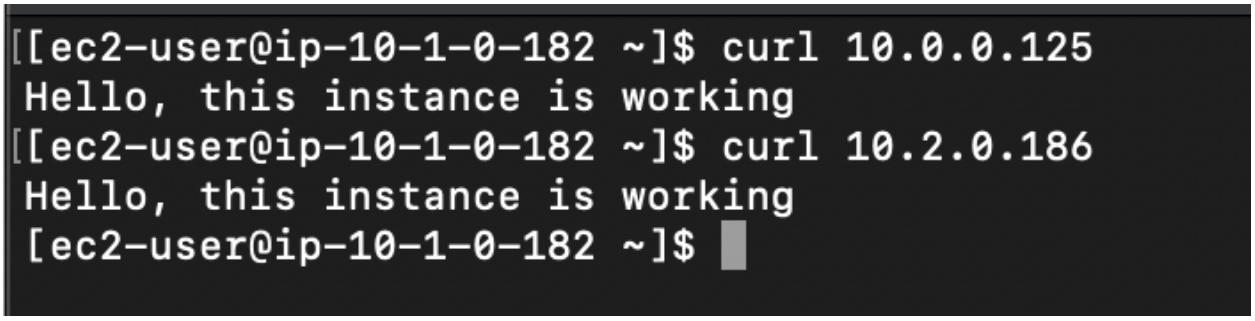
After creating the peering connections, we had to update the route tables for *each* VPC. This step is crucial because it tells each VPC how to send traffic to the other connected VPCs through the new peering links.



Testing the Connections

Finally, we tested if the connections worked as expected:

- We used SSH to log into each EC2 instance.
- From each EC2 instance, we used the `curl` command to try and connect to the *private IP addresses* of the EC2 instances in the other VPCs.
- **Result:** The `curl` commands were successful, showing that the VPC peering connections and route table setups allowed private communication between the EC2 instances in different VPCs.



Conclusion

This lab successfully taught us how to:

- Set up multiple VPCs.
- Create VPC peering connections between them.
- Configure route tables to direct traffic through peering connections.
- Verify private network communication between instances in different peered VPCs.

This hands-on practice clearly showed how VPC peering enables secure and private data sharing between isolated networks in AWS.