

# AWS Certified Advanced Networking Specialty

By Stéphane Maarek & Chetan Agrawal



COURSE →



EXTRA PRACTICE EXAMS →

# Disclaimer: These slides are copyrighted and strictly for personal use only

- This document is reserved for people enrolled into the [Ultimate AWS Certified Advanced Networking Specialty](#) course
- Please do not share this document, it is intended for personal use and exam preparation only, thank you.
- If you've obtained these slides for free on a website that is not the course's website, please reach out to [piracy@datacumulus.com](mailto:piracy@datacumulus.com). Thanks!
- Best of luck for the exam and happy learning!

# AWS Certified Networking Specialty Course

# Welcome! We're starting in 5 minutes



- Let's prepare for the AWS Certified Advanced Networking exam
- Networking experience is preferred although we'll revisit the basics
- A previous associate-level certification is a must

# Your instructors – Chetan & Stephane

- **Chetan Agrawal:**

- AWS Networking expert
- Over 15 years of experience
- Covers most AWS services

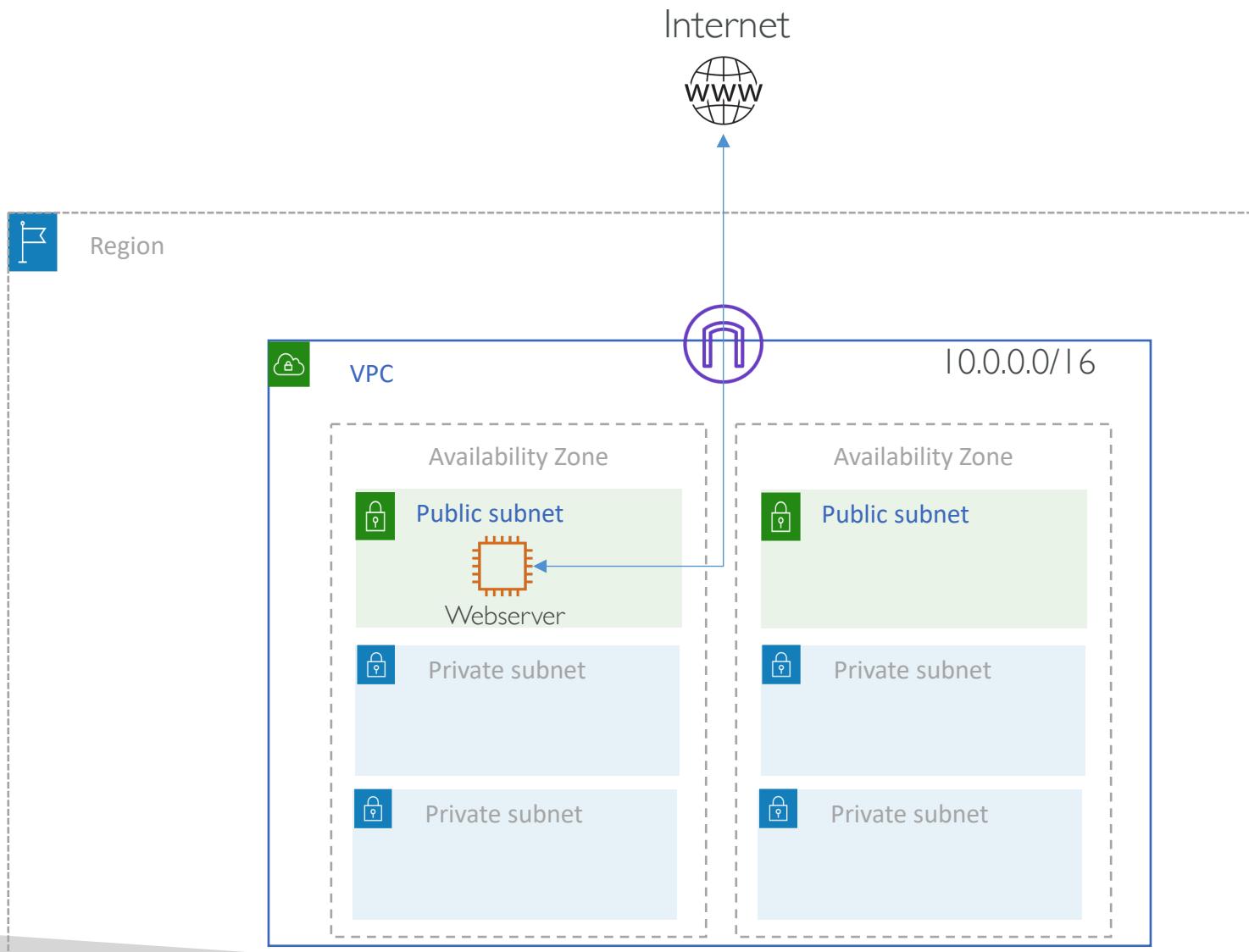


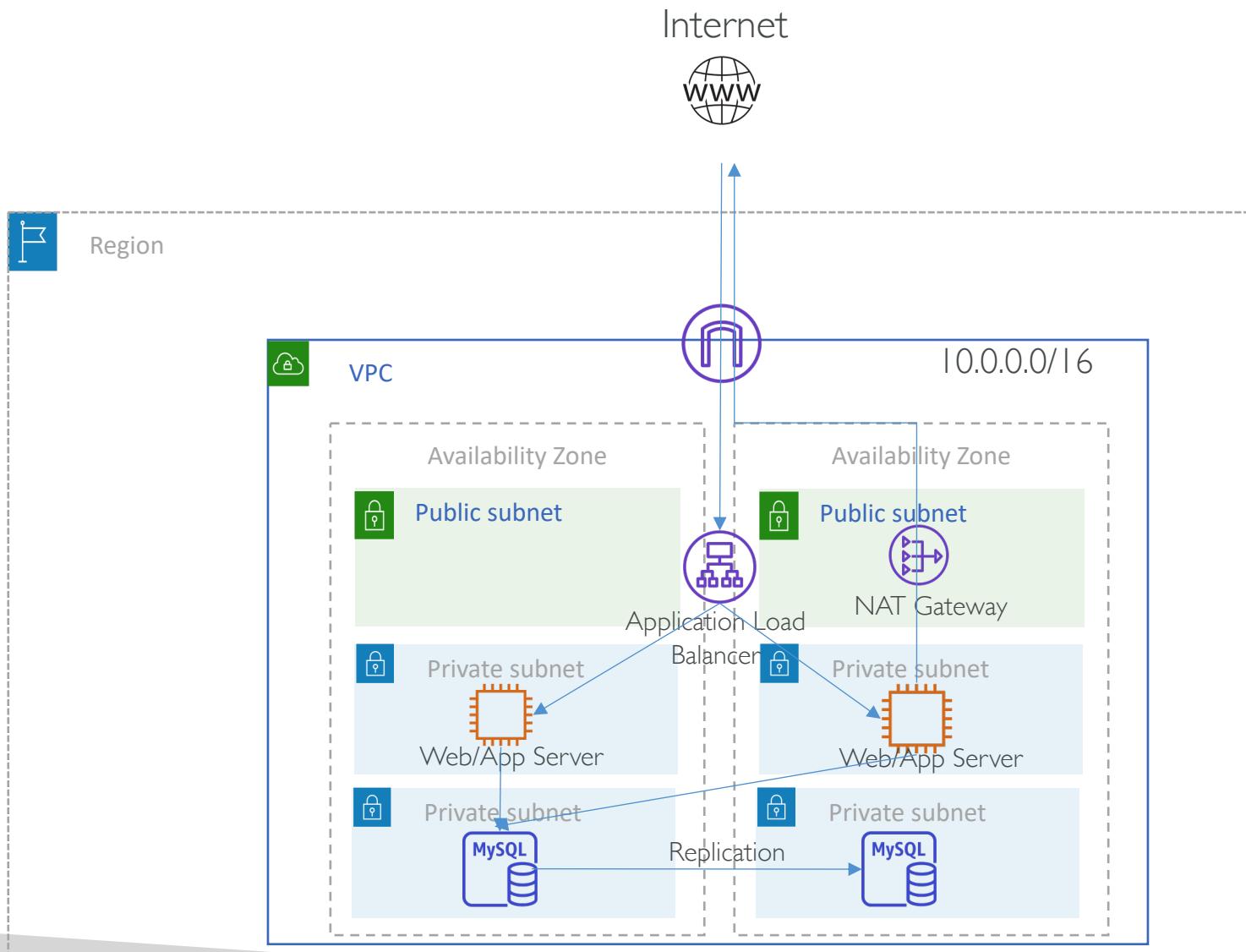
- **Stephane Maarek:**

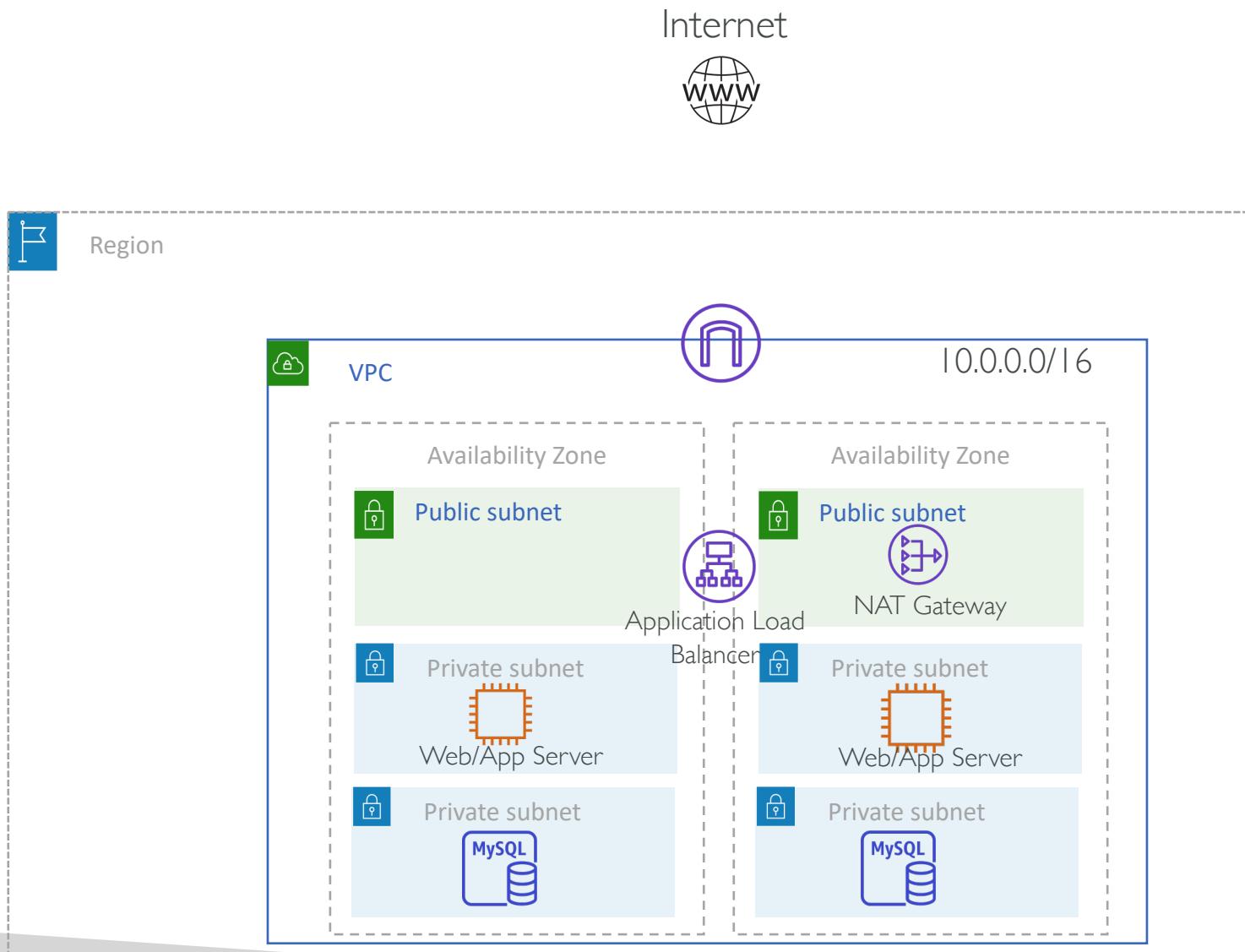
- AWS expert
- Over 8 years of experience
- Teaches ELB, CloudFront, Route 53

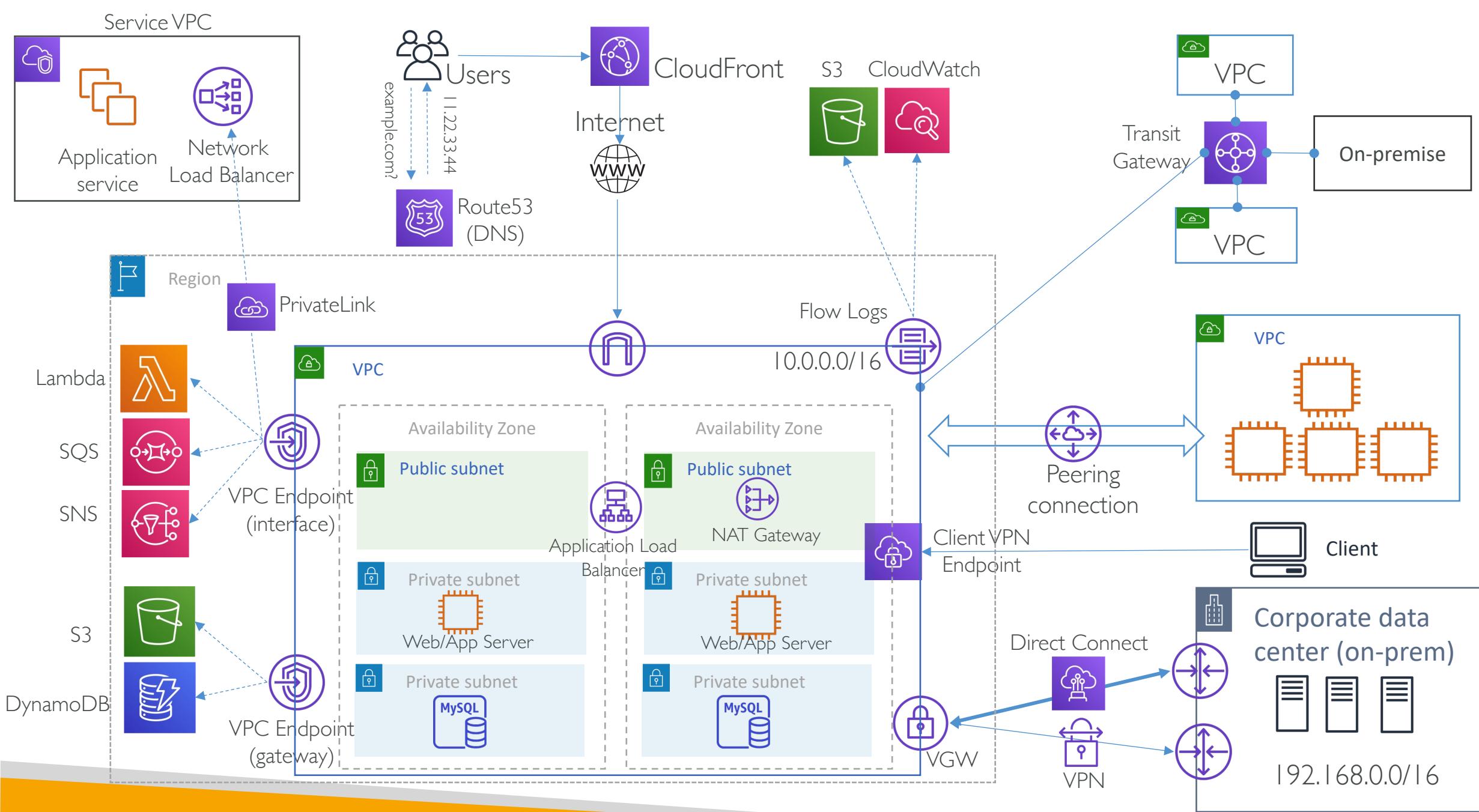


# AWS Networking Overview









# AWS VPC Fundamentals

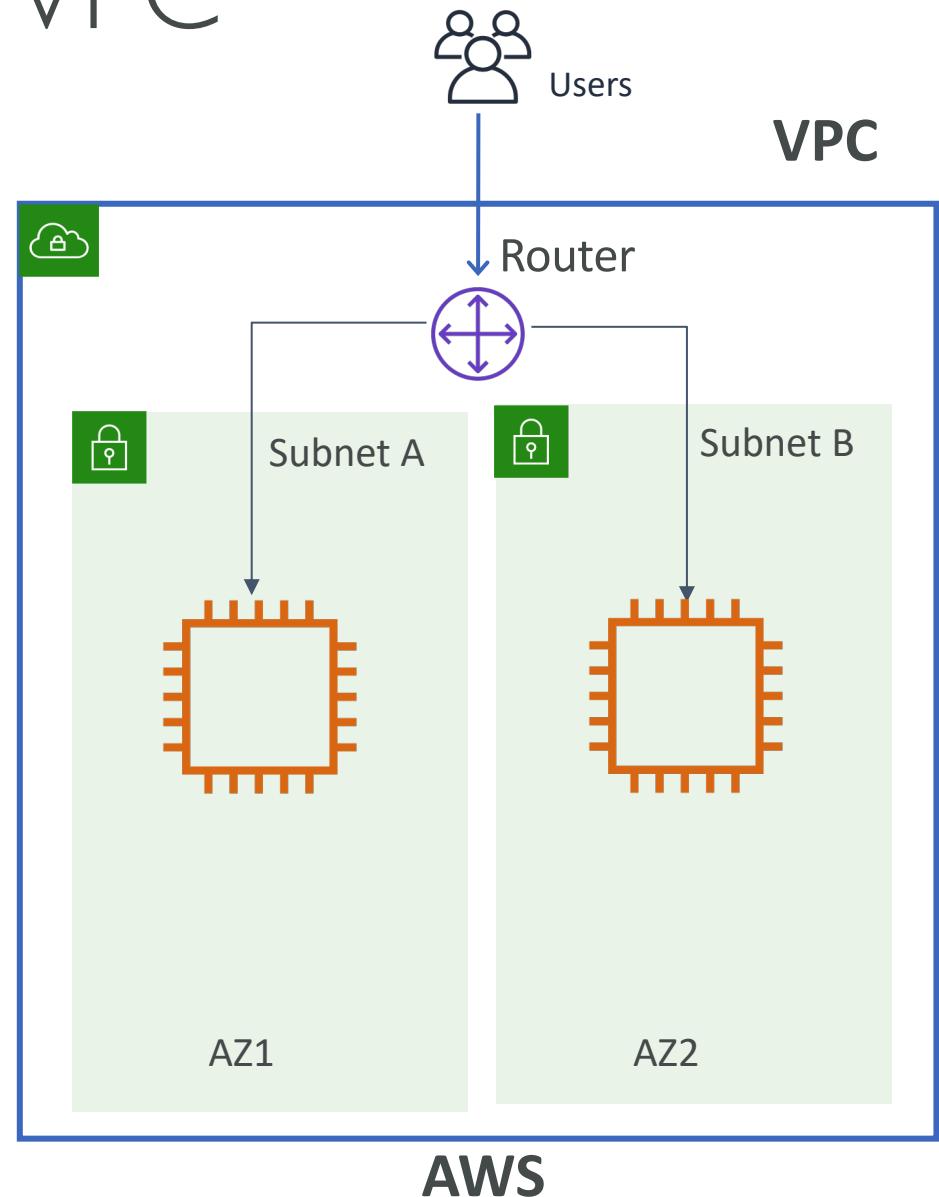
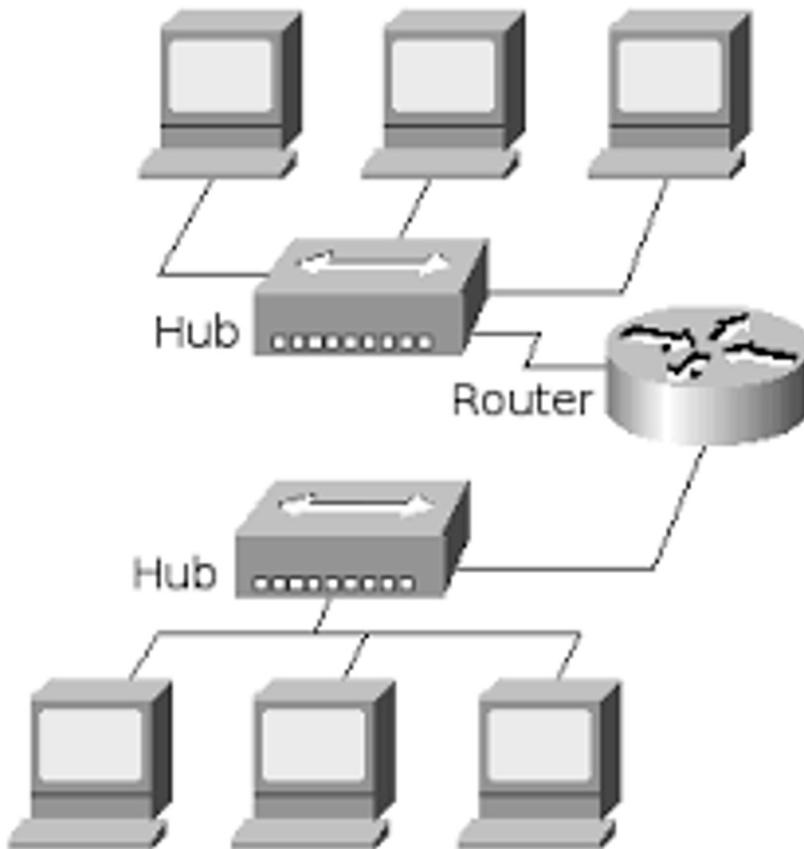
# Topics

- What is VPC?
- AWS Services inside and outside of VPC
- Typical 3 tier application VPC
- VPC Addressing (CIDR)
- VPC Subnets and Route Tables (Public/Private)
- IP Addresses (IPv4, IPv6, Private/Public/Elastic)
- Elastic Network Interface
- Security Groups and Network ACL
- NAT gateway and NAT instance

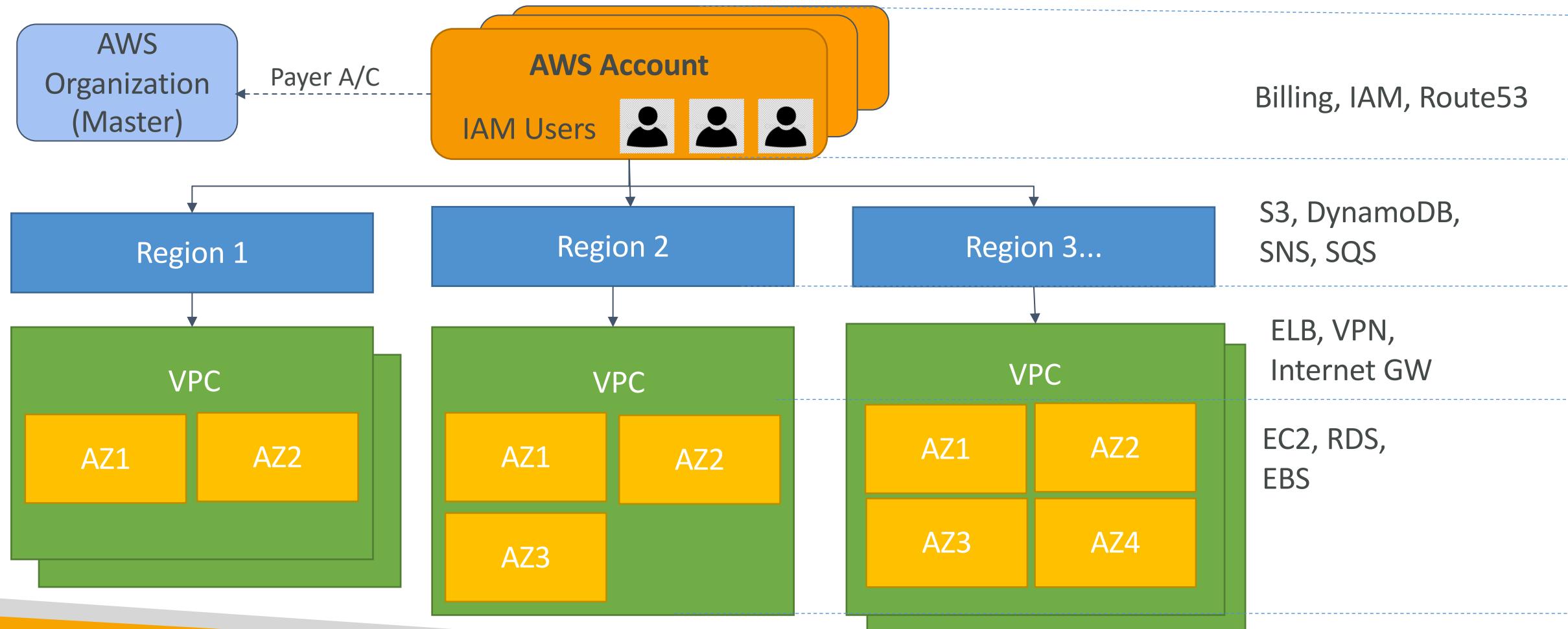
# Virtual Private Cloud (VPC)

- Amazon VPC = Amazon Virtual Private Cloud
- Launch AWS resources into a virtual network that you've defined.
- VPC closely resembles a traditional on-premises network
- VPC benefits of using the scalable infrastructure of AWS

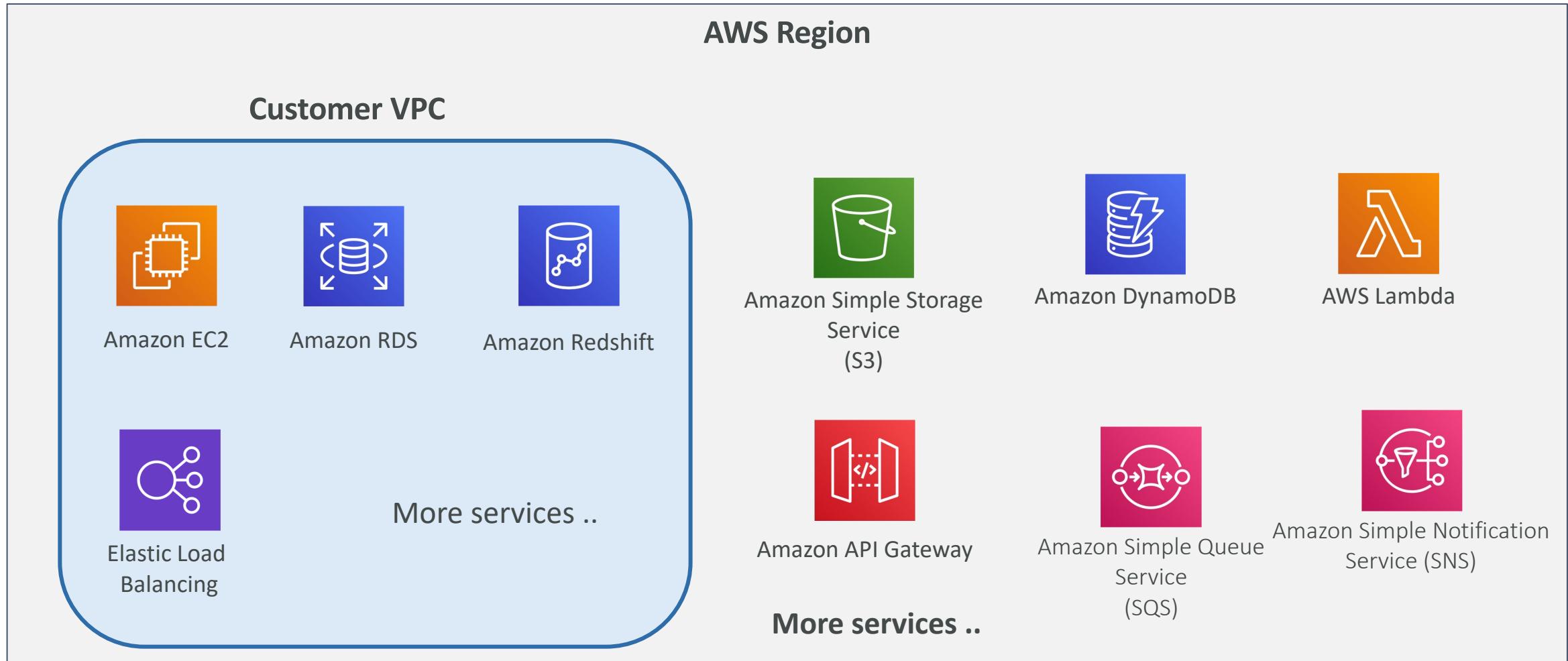
# Traditional IT network vs VPC



# Account, Users, Region hierarchy



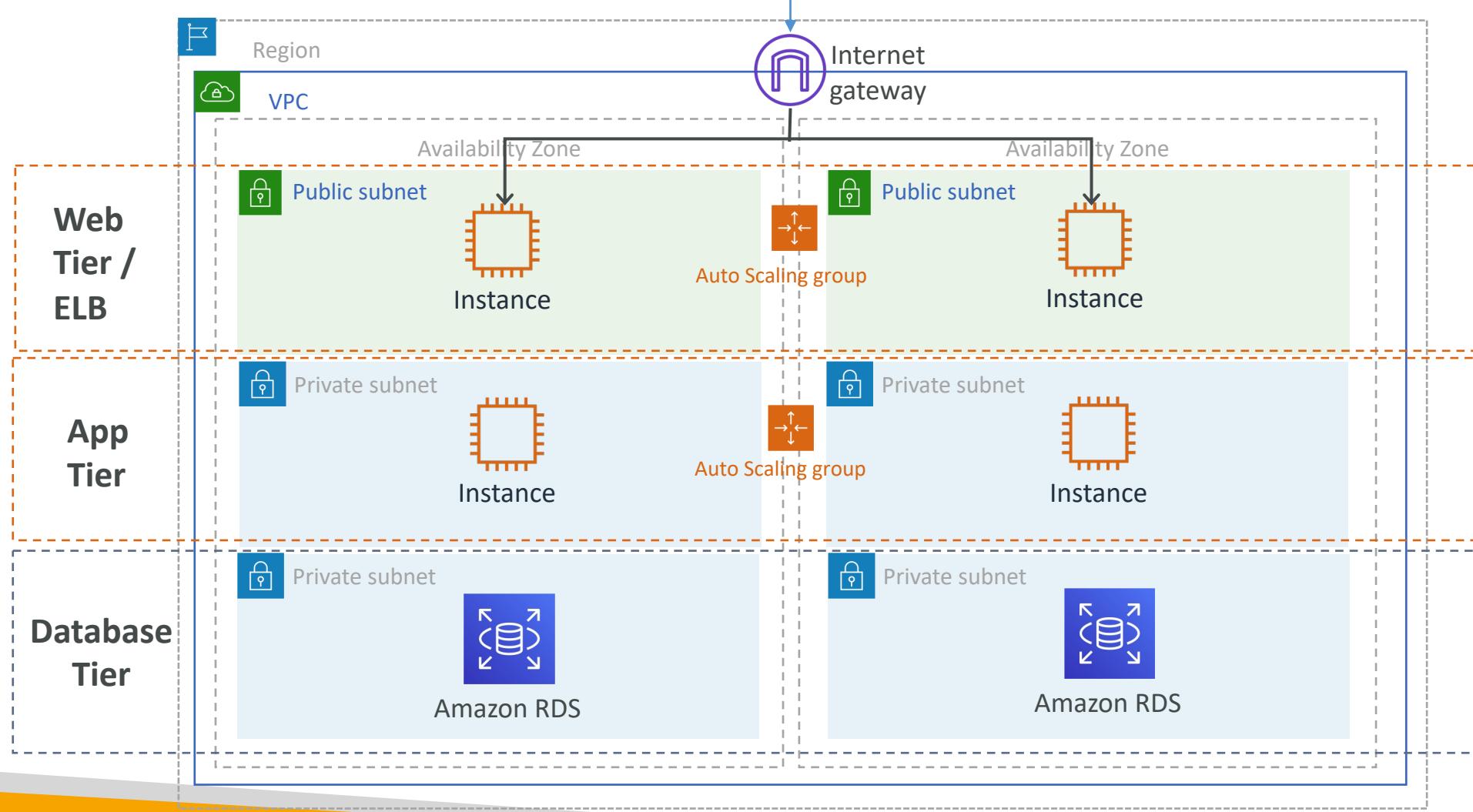
# AWS Services inside and outside VPC



# Example VPC for 3-Tier architecture



Users



# VPC Addressing - Understanding CIDR

- A CIDR has two components:
  - The base IP (XX.XX.XX.XX)
  - The Subnet Mask or Prefix (/26)
- The base IP represents an IP contained in the range
- The subnet mask defines how many bits can change in the IP

**10.10.0.0/16**

IP Range      Prefix

Total Number of host addresses (IPs) in given Network =  $2^{(32 - \text{Prefix})}$

Hence in given example: The total number of host addresses =  $2^{(32 - 16)} = 2^16 = 65536$

# CIDRs / Subnet Masks sizing

- The subnet masks basically allows part of the underlying IP to get additional next values from the base IP
- /32 allows for 1 IP =  $2^0$
- /31 allows for 2 IP =  $2^1$
- /30 allows for 4 IP =  $2^2$
- /29 allows for 8 IP =  $2^3$
- /28 allows for 16 IP =  $2^4$
- /27 allows for 32 IP =  $2^5$
- /26 allows for 64 IP =  $2^6$
- /25 allows for 128 IP =  $2^7$
- /24 allows for 256 IP =  $2^8$
- /16 allows for 65,536 IP =  $2^{16}$
- /0 allows for all IPs =  $2^{32}$

## Quick memo:

- /32 – no IP number can change
- /24 - last IP number can change
- /16 – last IP two numbers can change
- /8 – last IP three numbers can change
- /0 – all IP numbers can change

# Understanding CIDRs - Little exercise

- $192.168.0.0/24 = \dots ?$ 
  - $192.168.0.0 - 192.168.0.255$  (256 IP)
- $192.168.0.0/16 = \dots ?$ 
  - $192.168.0.0 - 192.168.255.255$  (65,536 IP)
- $134.56.78.123/32 = \dots ?$ 
  - Just 134.56.78.123
- $0.0.0.0/0$ 
  - All IP!
- When in doubt, use this website: <https://www.ipaddressguide.com/cidr>

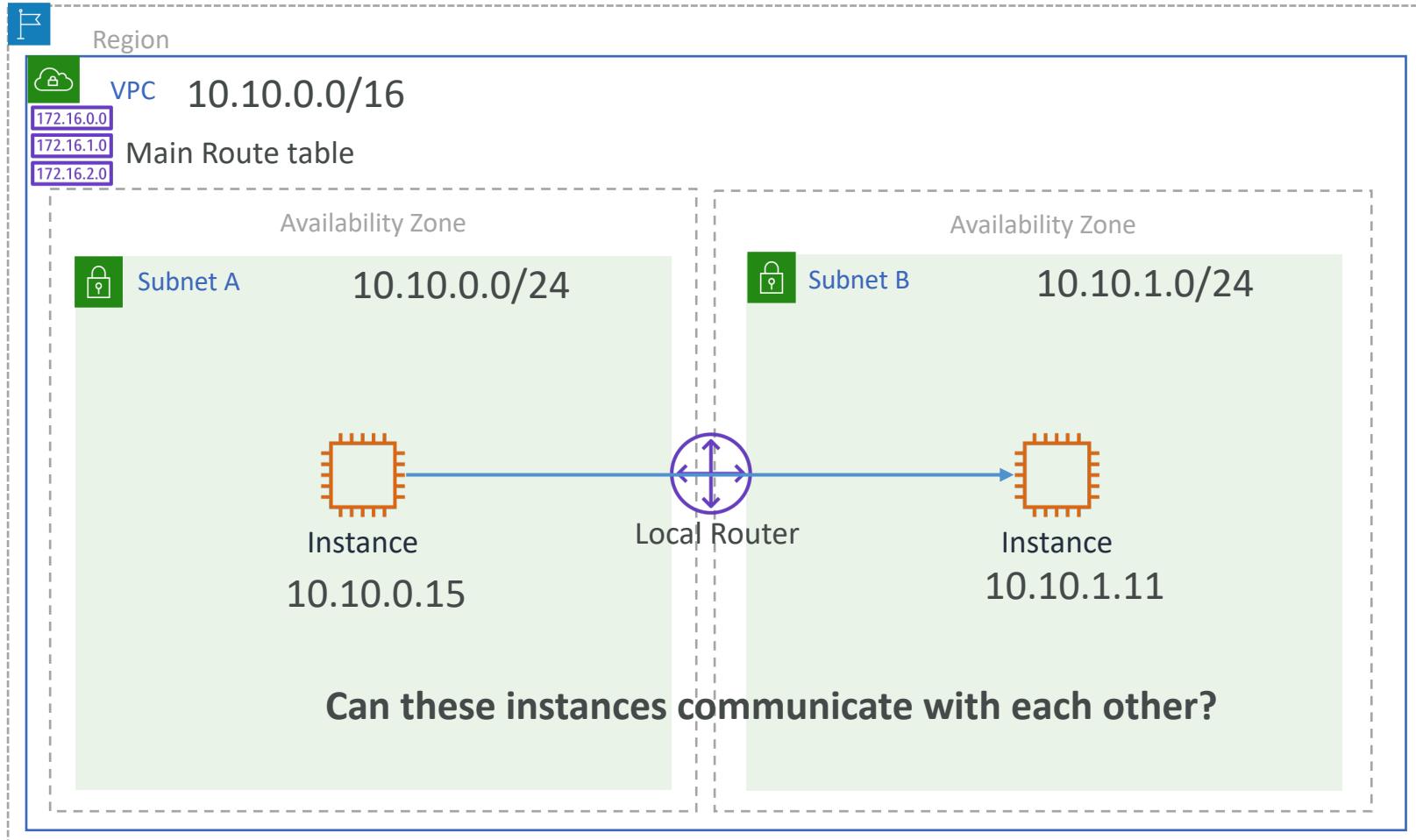
# How to calculate CIDR?

# Private vs Public IP (IPv4) - Allowed ranges

- The Internet Assigned Numbers Authority (IANA) established certain blocks of IPV4 addresses for the use of private (LAN) and public (Internet) addresses.
- Private IP can only allow certain values
  - 10.0.0 – 10.255.255.255 (10.0.0/8) <= in big networks
  - 172.16.0.0 – 172.31.255.255 (172.16.0.0/12) <= default VPC included in this range
  - 192.168.0.0 – 192.168.255.255 (192.168.0.0/16) <= example: home networks
- All the rest of the IP on the internet are public IP

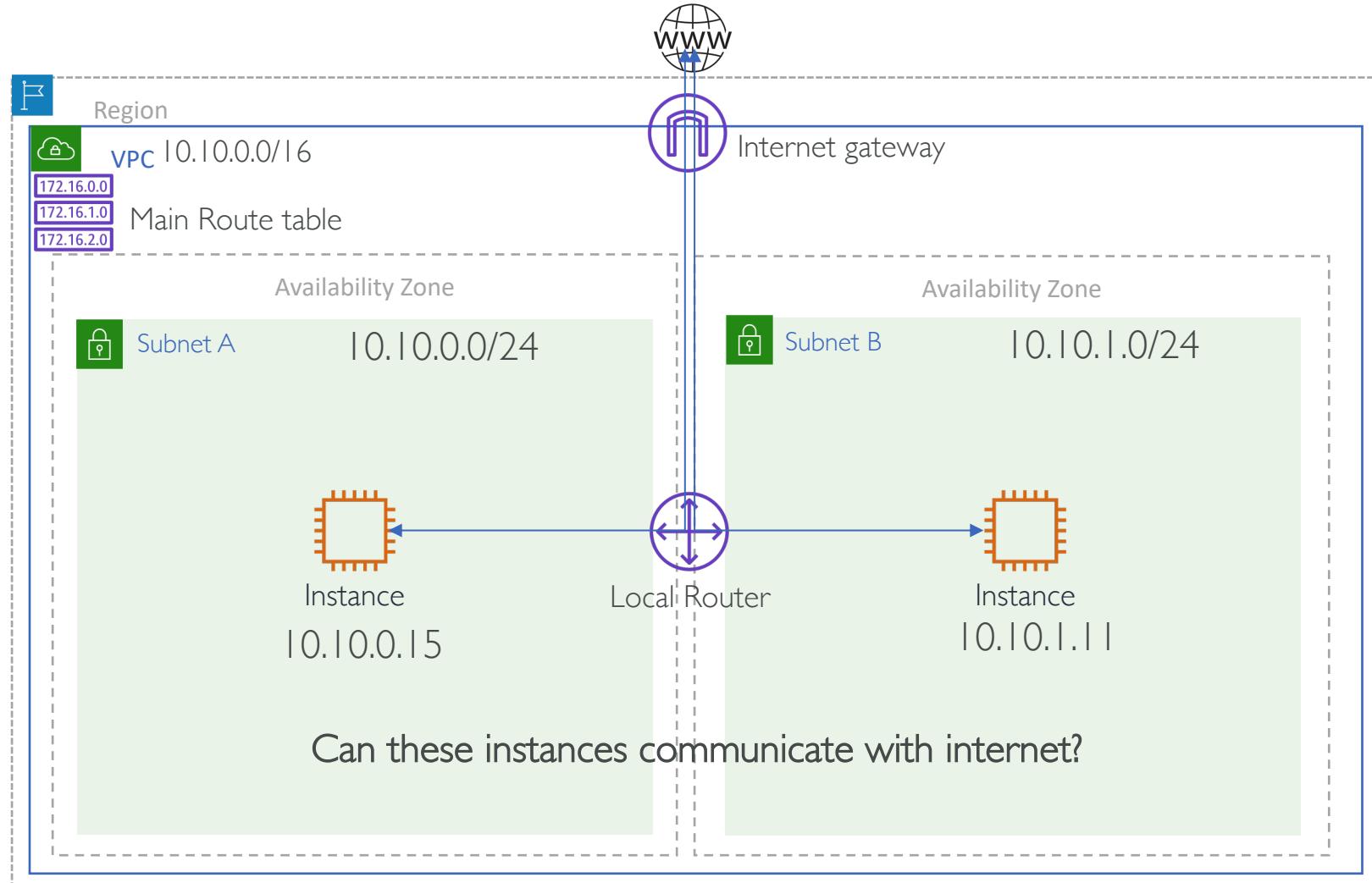
# Subnets, Route Tables and Internet Gateway

# Subnets, Route Tables and Internet Gateway



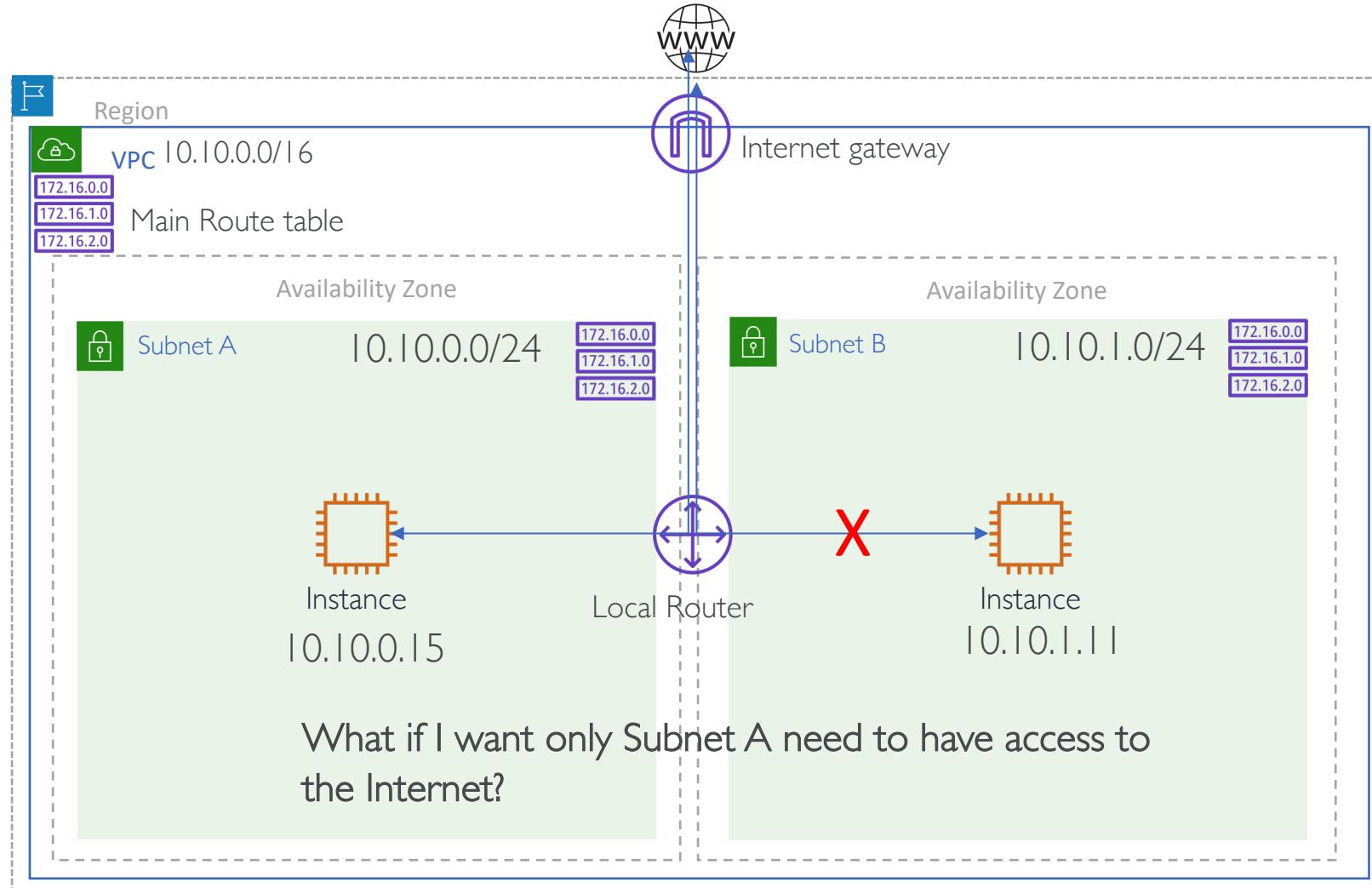
| Destination  | Target |
|--------------|--------|
| 10.10.0.0/16 | Local  |

# Subnets, Route Tables and Internet Gateway



| Destination  | Target           |
|--------------|------------------|
| 10.10.0.0/16 | Local            |
| 0.0.0.0/0    | Internet Gateway |

# Subnets, Route Tables and Internet Gateway



Subnet A Route Table

| Destination         | Target                  |
|---------------------|-------------------------|
| <b>10.10.0.0/16</b> | Local                   |
| <b>0.0.0.0/0</b>    | <b>Internet Gateway</b> |

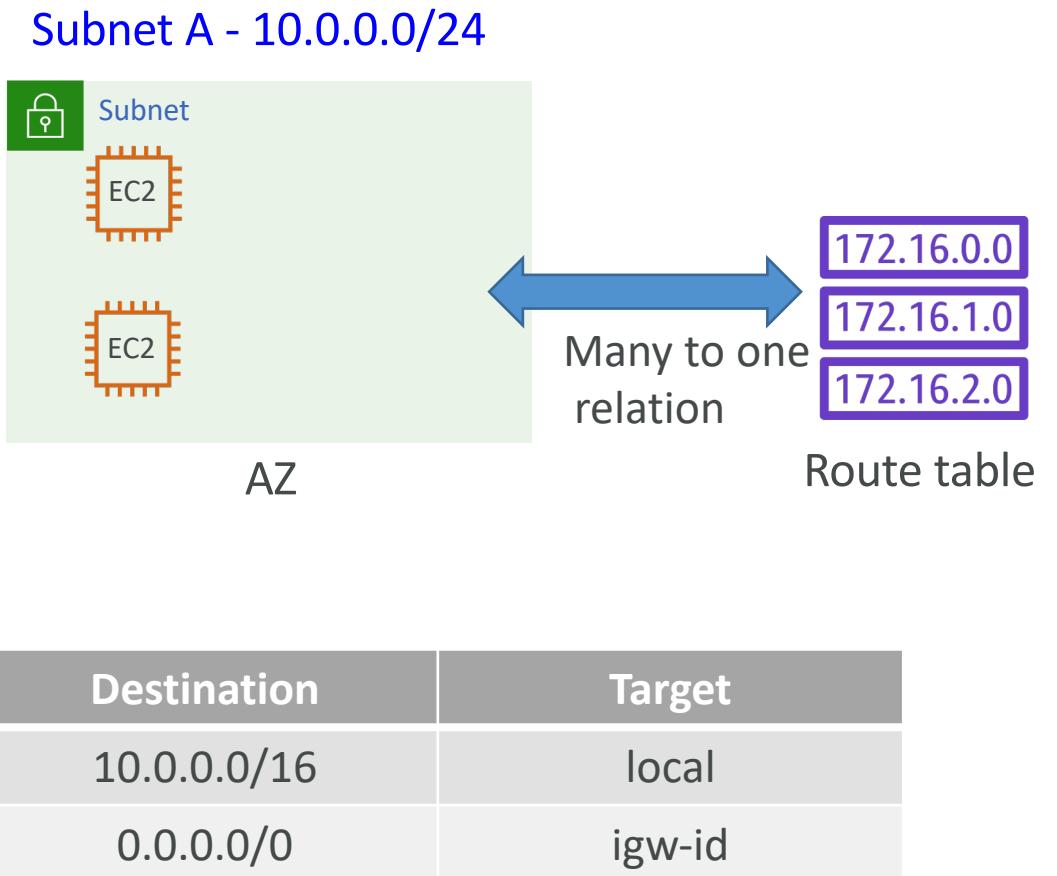
Subnet B Route Table

| Destination         | Target |
|---------------------|--------|
| <b>10.10.0.0/16</b> | Local  |

Now subnets are not following Main Route Table

# Route Table

- Contains rules to route the traffic in/out of Subnets/VPC
- Main route table at VPC level
- Custom route table at Subnet level
- Each route table contains default immutable local route for VPC
- If no custom route table is defined, then new subnets are associated with Main route table
- We can modify main route table

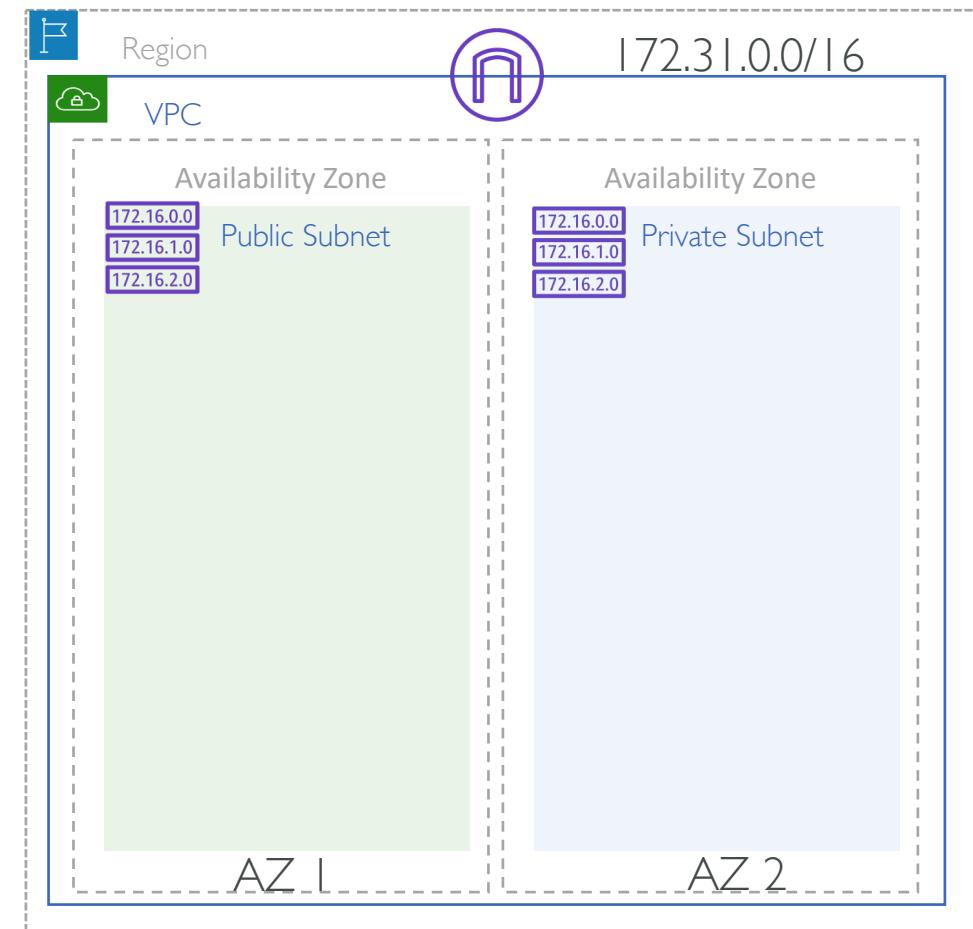


# Subnets

- Public Subnet
  - Has route for Internet
  - Instances with Public IP can communicate to internet
  - Ex: NAT, Web servers, Load balancer

| Destination   | Target  |
|---------------|---------|
| 172.31.0.0/16 | local   |
| 0.0.0.0/0     | igw-xxx |

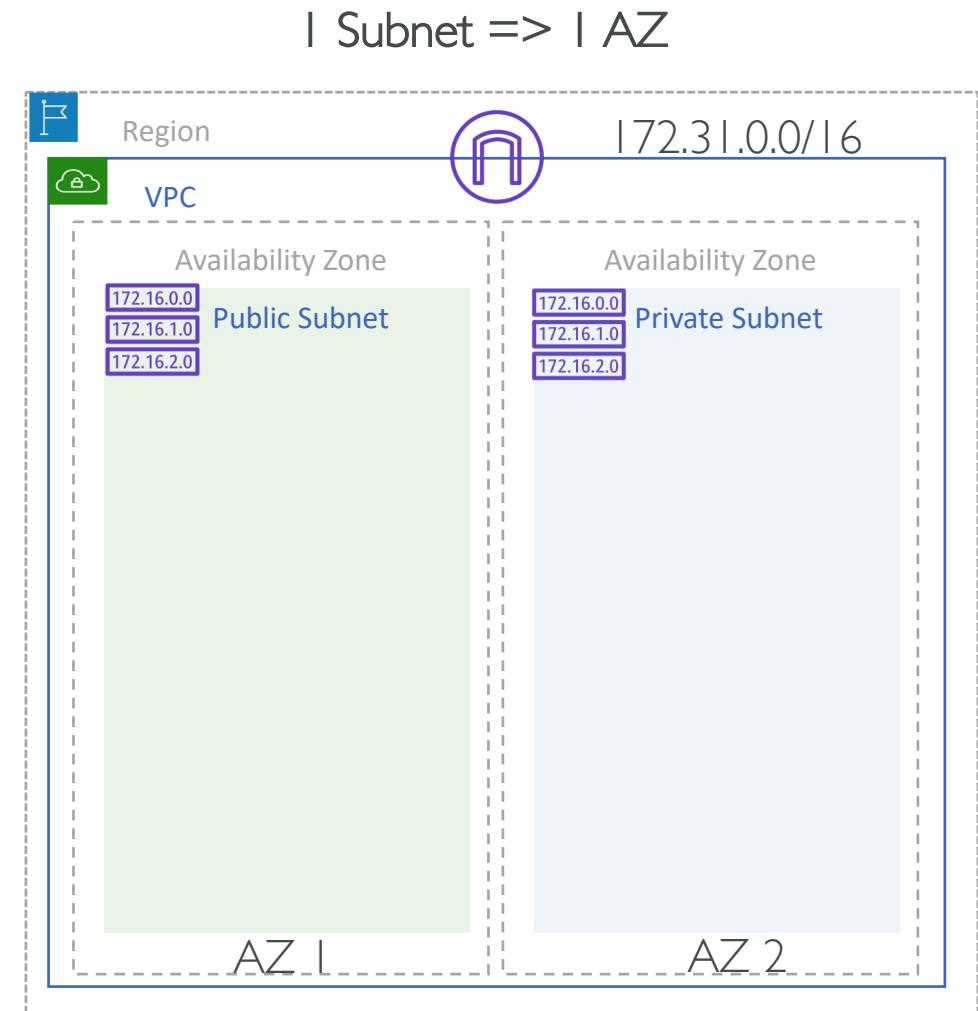
1 Subnet => 1 AZ



# Subnets

- Private Subnet
  - No route to Internet
  - Instances receive private IPs
  - Typically uses NAT for instances to have internet access
  - Ex: Database, App server

| Destination   | Target |
|---------------|--------|
| 172.31.0.0/16 | local  |

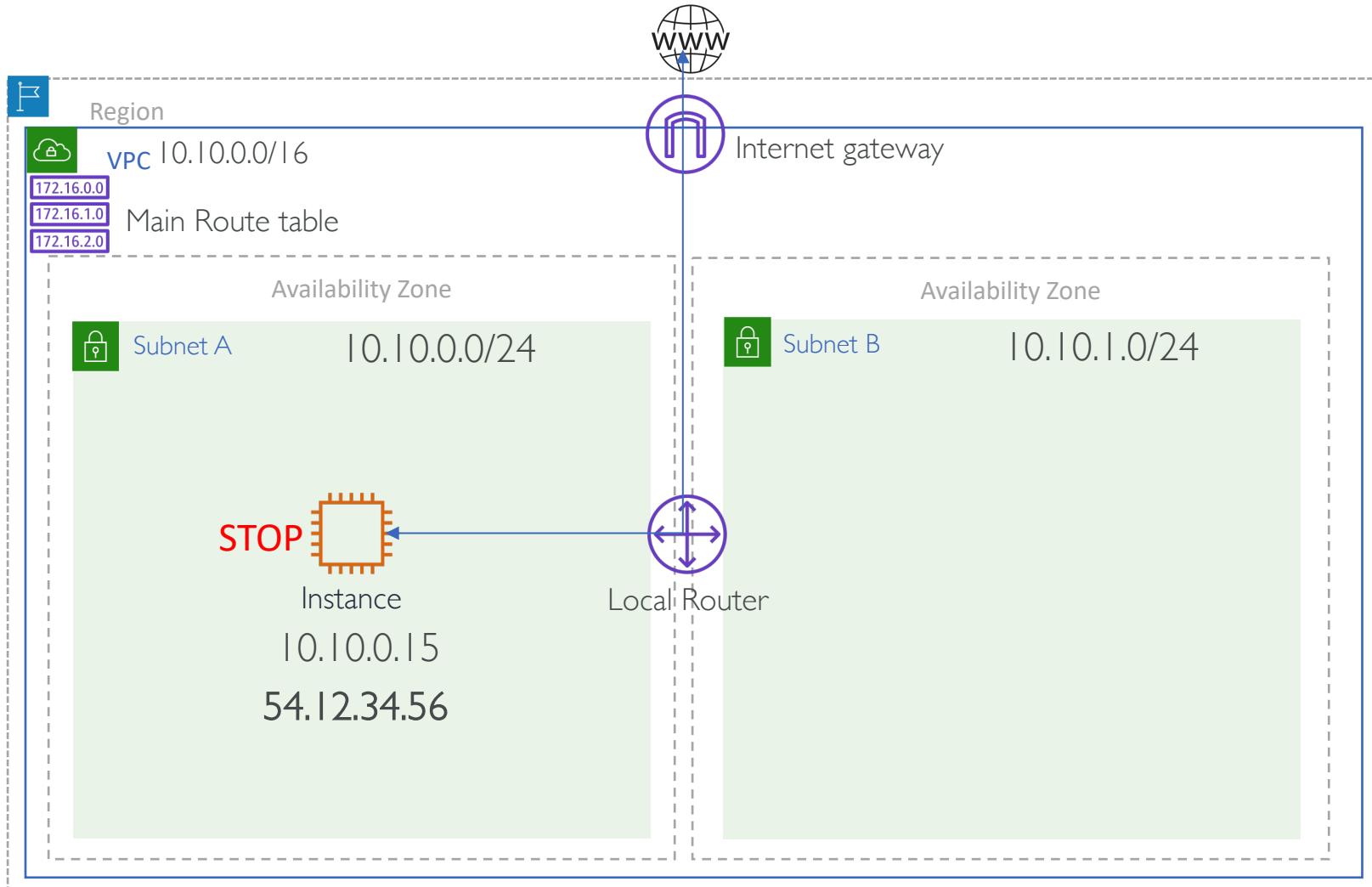


# Subnets - IPv4

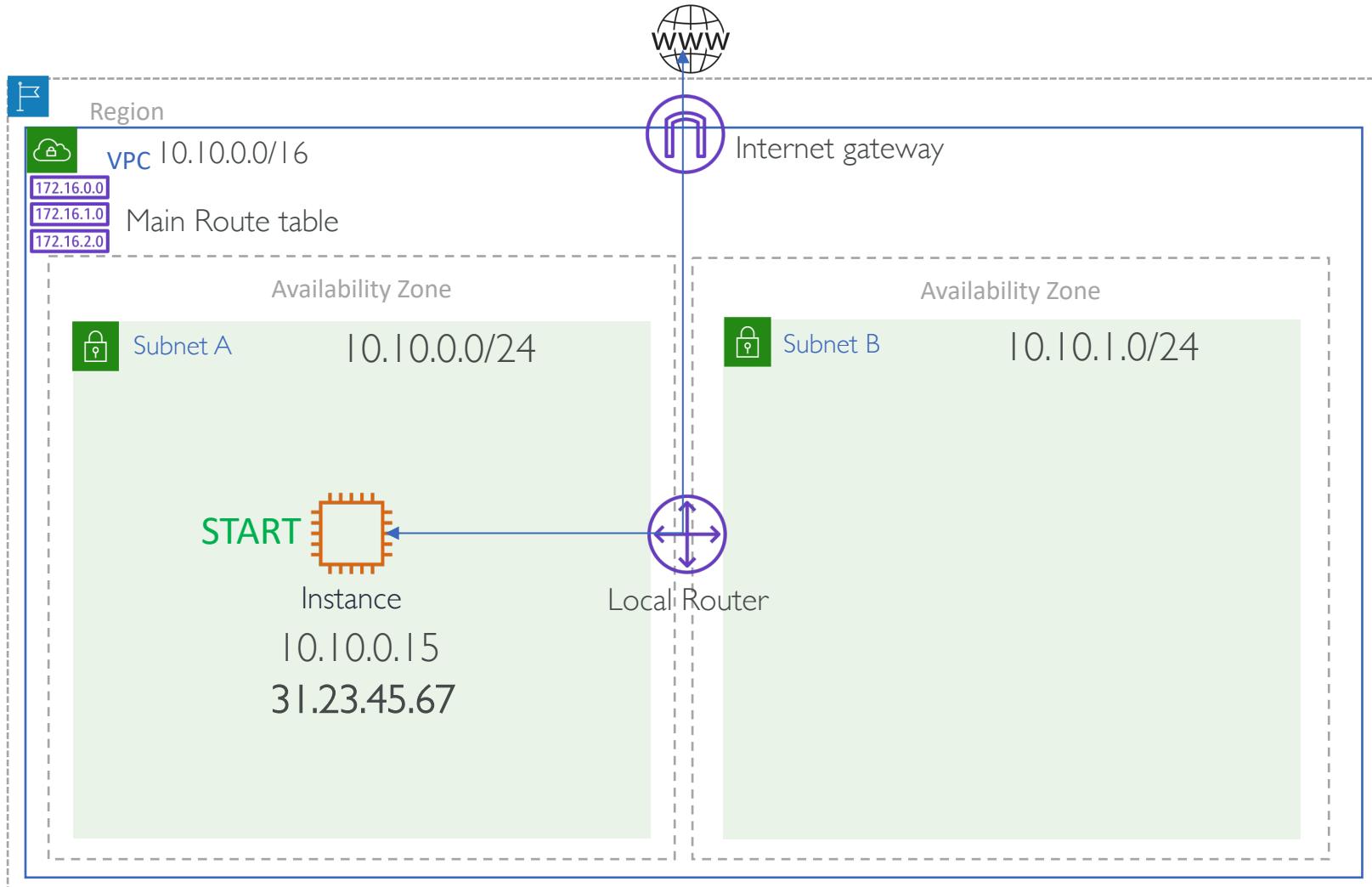
- AWS reserves 5 IPs address (first 4 and last 1 IP address) in each Subnet
- These 5 IPs are not available for use and cannot be assigned to an instance
- Ex, if CIDR block 10.0.0.0/24, reserved IP are:
  - 10.0.0.0: Network address
  - 10.0.0.1: Reserved by AWS for the VPC router
  - 10.0.0.2: Reserved by AWS for mapping to Amazon-provided DNS
  - 10.0.0.3: Reserved by AWS for future use
  - 10.0.0.255: Network broadcast address. AWS does not support broadcast in a VPC, therefore the address is reserved
- Exam Tip:
  - If you need 29 IP addresses for EC2 instances, you can't choose a Subnet of size /27 (32 IP)
  - You need at least 64 IP, Subnet size /26 ( $64-5 = 59 > 29$ , but  $32-5 = 27 < 29$ )

# IP Addresses in VPC

# IP Address



# IP Address



| Destination  | Target           |
|--------------|------------------|
| 10.10.0.0/16 | Local            |
| 0.0.0.0/0    | Internet Gateway |

# Private, Public and Elastic IP (EIP)

| Feature                   | Private   | Public  | Elastic  |
|---------------------------|---|---|--|
| Communication             | Communication within VPC                          | Can communicate over internet                           | Can communicate over internet                                |
| Address range             | Gets IP address from subnet range. Ex: 10.200.0.1 | Gets IP address from Amazon Pool within region          | Gets IP address from Amazon Pool within region               |
| Instance restart behavior | Once assigned cannot be changed                   | Changes over instance restart                           | Do not change over instance restart. Can be removed anytime. |
| Releasing IP              | Released when instance is terminated              | Released to POOL when instance is stopped or terminated | Not released. Remains in your account. (Billed)              |

Continued

# Private, Public and Elastic IP (EIP)

| Feature              | Private                                       | Public   | Elastic   |
|----------------------|---|--|---|
| Automatic Assignment | Receives private ip on launch on EC2 instance | Receives public ip on launch on EC2 instance if “Public ip addressing attribute” is set to true for subnet | Have to explicitly allocate and attach EIP to EC2 instance.<br>Can be reattached to other EC2 |
| Examples             | Application servers, databases                | Web servers, Load Balancers, Websites  | Web servers, Load Balancers, Websites   |

# IPv6 Addresses

- AWS VPC also supports IPv6 addresses
- IPv6 address is 128 bits in size with 8 blocks of 16 bits each
  - Example: 2001:0db8:85a3:0000:0000:8a2e:0370:7334
- IPv6 addresses are public and globally unique, and allows resources to communicate with the internet
- VPC can operate in dual-stack mode where VPC resources can communicate over IPv4, or IPv6, or both
- IPv6 address persists when you stop and start your instance, and is released when you terminate your instance

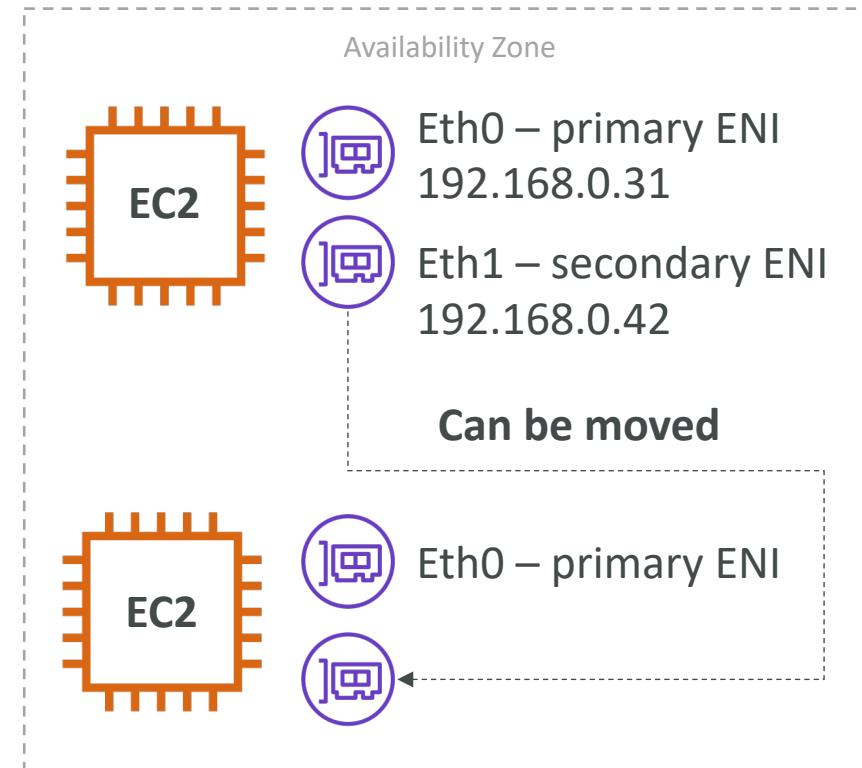
# IPv4 vs IPv6 Addresses

| IPv4   | IPv6  |
|--|---|
| Default and required for all VPCs; cannot be removed.  | Opt-in only   |
| The VPC/Subnet CIDR block size can be from /16 to /28.   | The VPC CIDR block size is fixed at /56. Subnet block is fixed at /64                                     |
| You can choose the private IPv4 CIDR block for your VPC  | IPv6 CIDR block is allocated to VPC from Amazon's pool of IPv6 addresses. We cannot select the range.     |
| Supports both Private and Public IPs   | No distinction between public and private IP addresses. IPv6 addresses are public.                        |
| An instance receives an Amazon-provided private DNS hostname that corresponds to its private/Public IPv4 address | Amazon-provided DNS hostnames are not supported.  |
| Supported for AWS Site-to-Site VPN connections and customer gateways, NAT devices, and VPC endpoints             | Not supported for AWS Site-to-Site VPN connections and customer gateways, NAT devices, and VPC endpoints. |

# Elastic Network Interface (ENI)

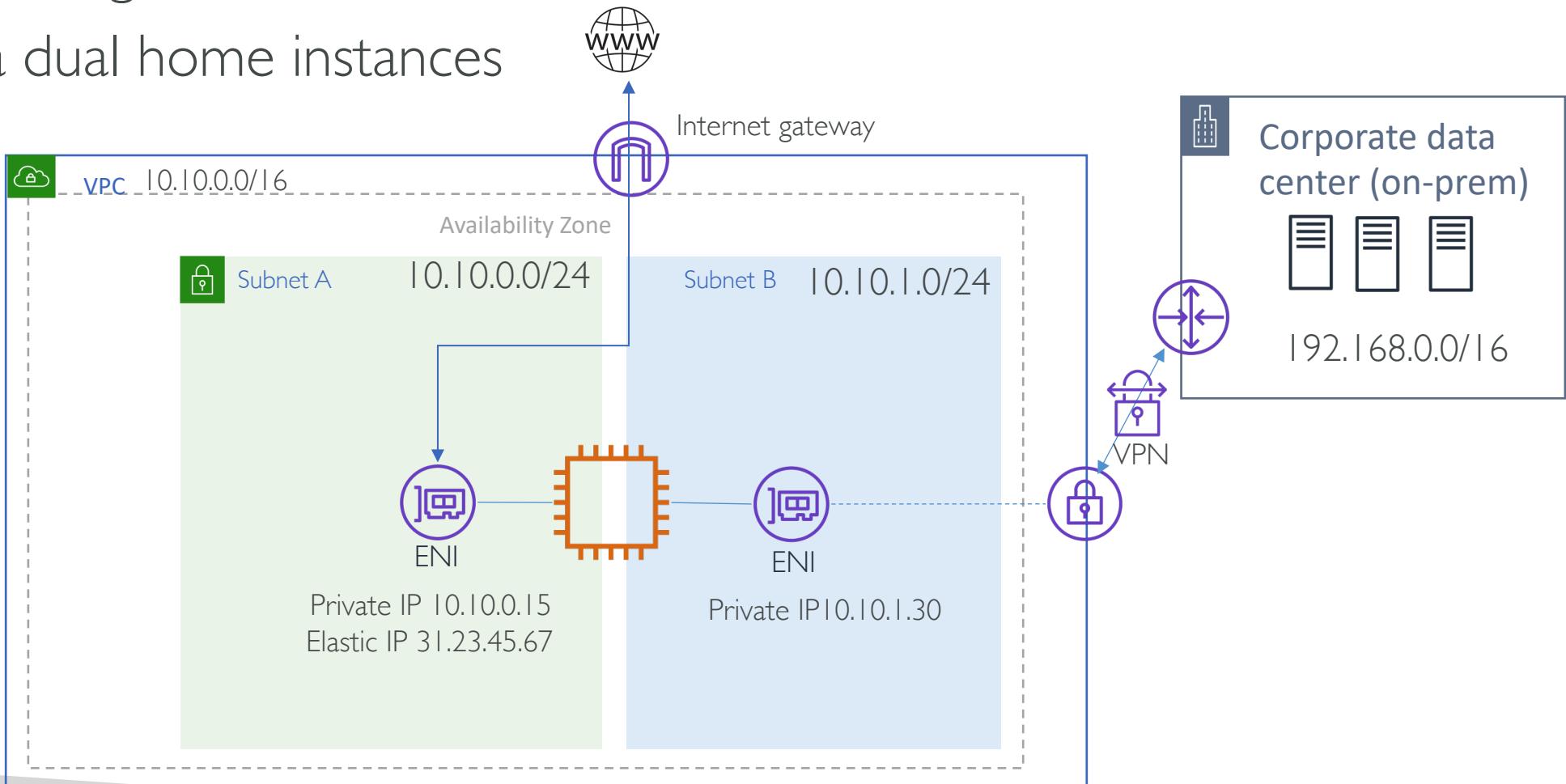
# Elastic Network Interfaces (ENI)

- Logical component in a VPC that represents a **virtual network card**
- The ENI can have the following attributes:
  - Primary private IPv4, one or more secondary IPv4
  - One Elastic IP (IPv4) per private IPv4
  - One Public IPv4
  - One or more security groups
  - A MAC address
- You can create ENI independently and attach them on the fly (move them) on EC2 instances for failover
- Bound to a specific availability zone (AZ)



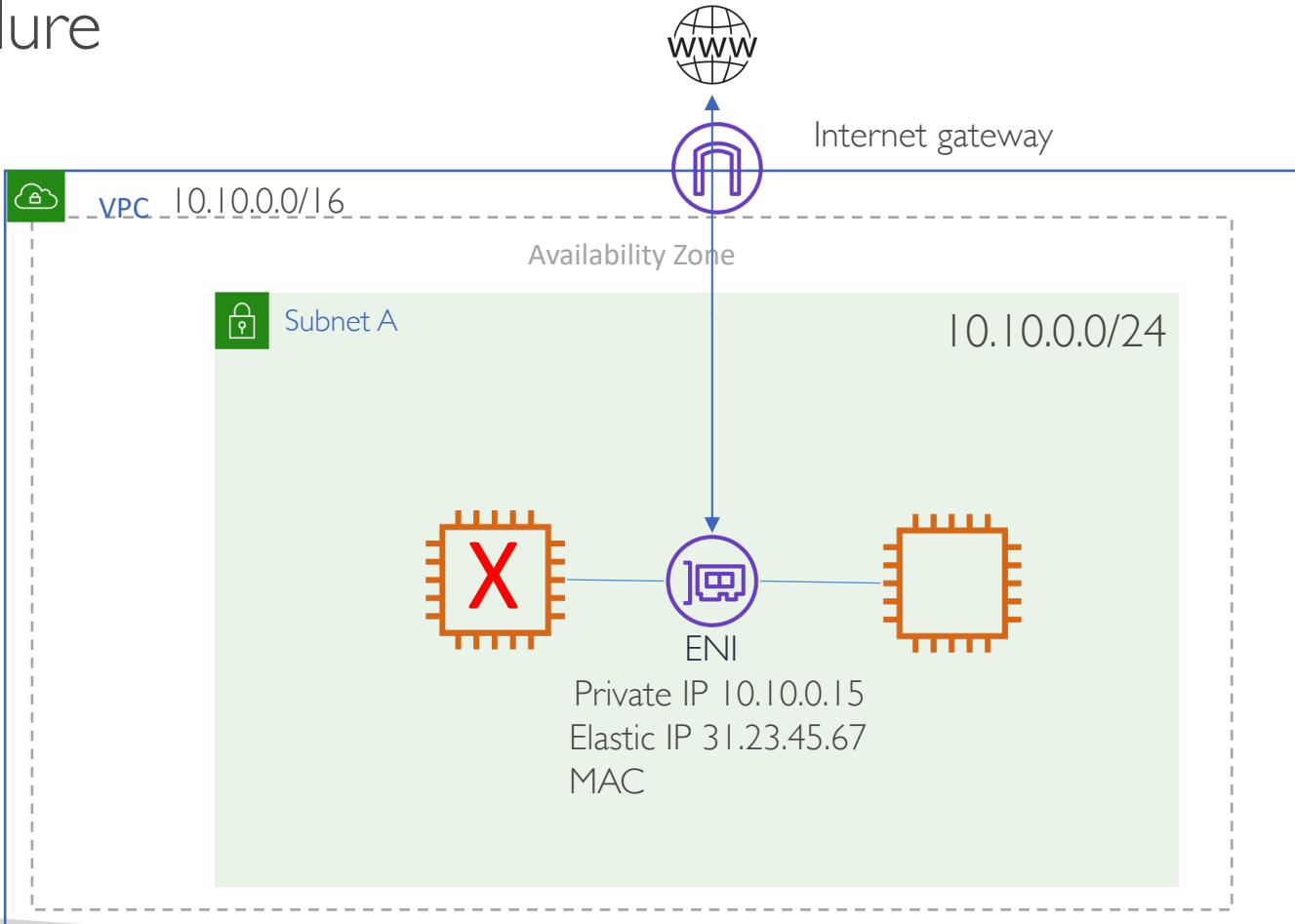
# ENI Use cases

- Creating Management Network
- Creating a dual home instances



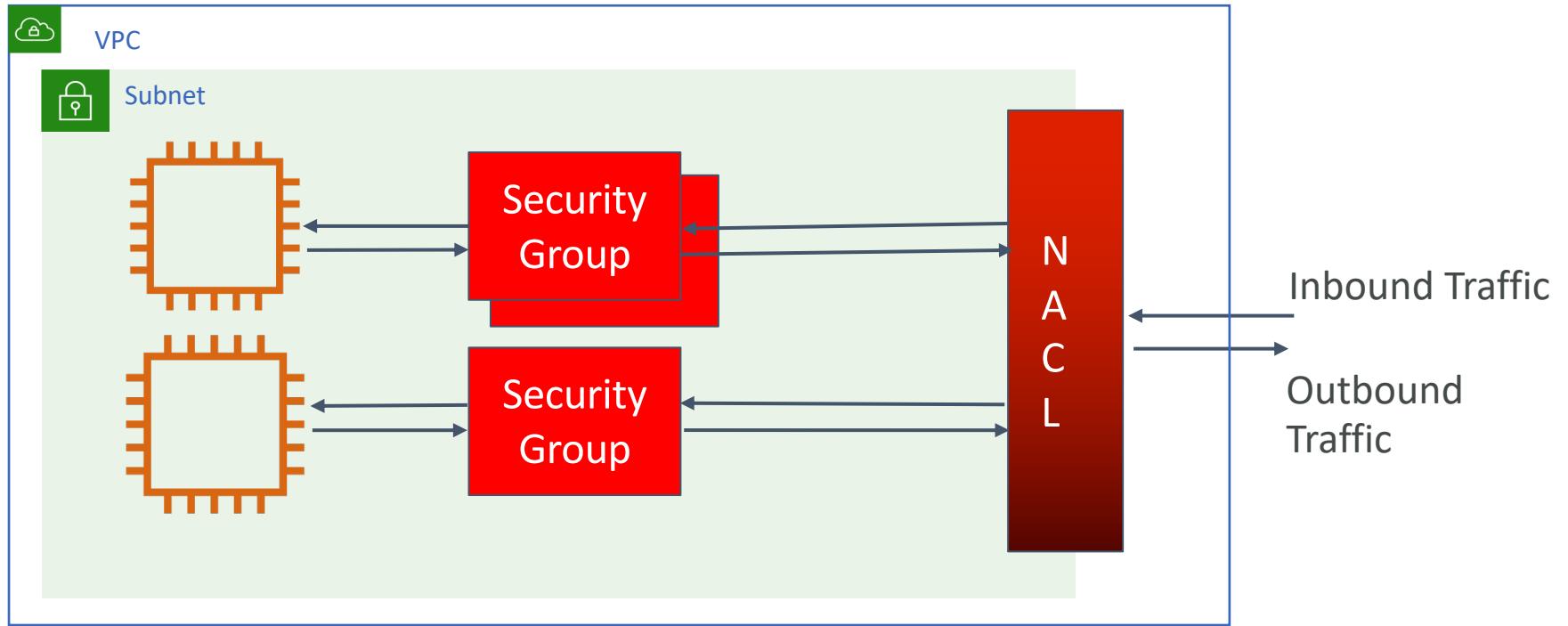
# ENI Use cases

- High Availability solution by attaching ENI to hot standby instance in case of failure



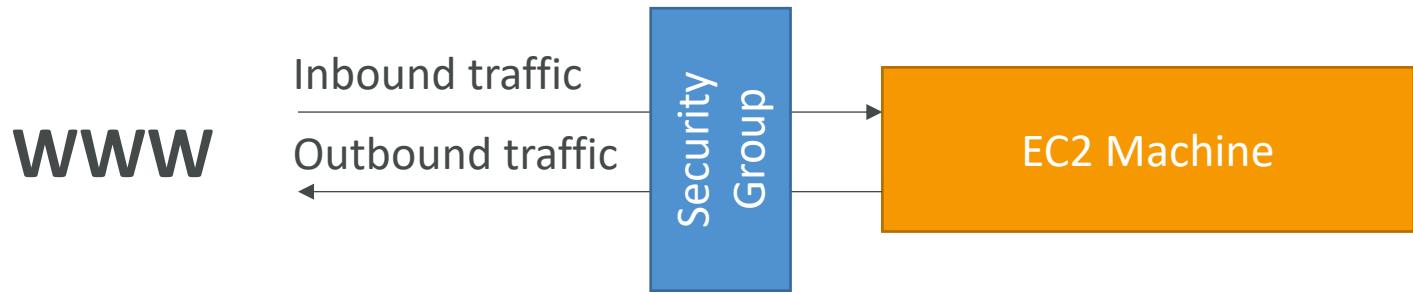
# Firewalls inside VPC

- Security Groups
- Network Access Control List (NACL)



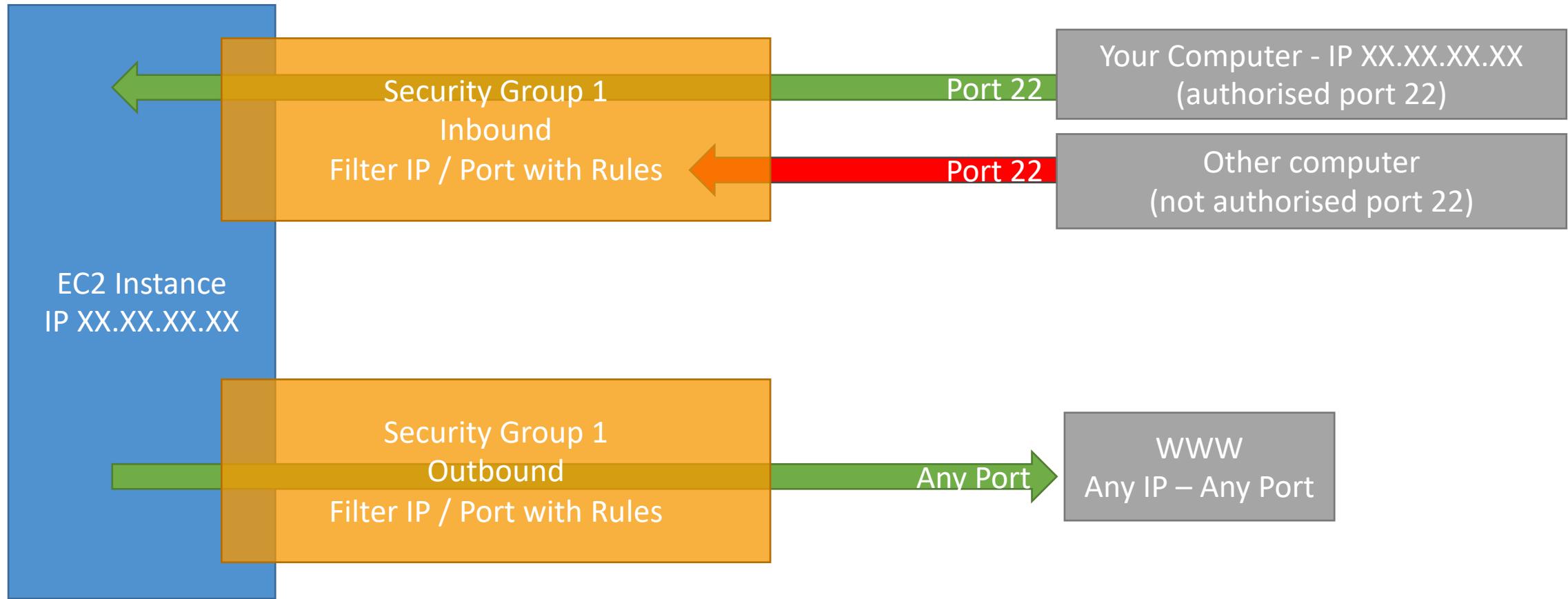
# Security Groups

- Security Groups are the fundamental of network security in AWS
- They control how traffic is allowed into or out of our EC2 Machines.



- It is the most fundamental skill to learn to troubleshoot networking issues
- In this lecture, we'll learn how to use them to **allow**, **inbound** and **outbound** ports

# Security Groups - Diagram



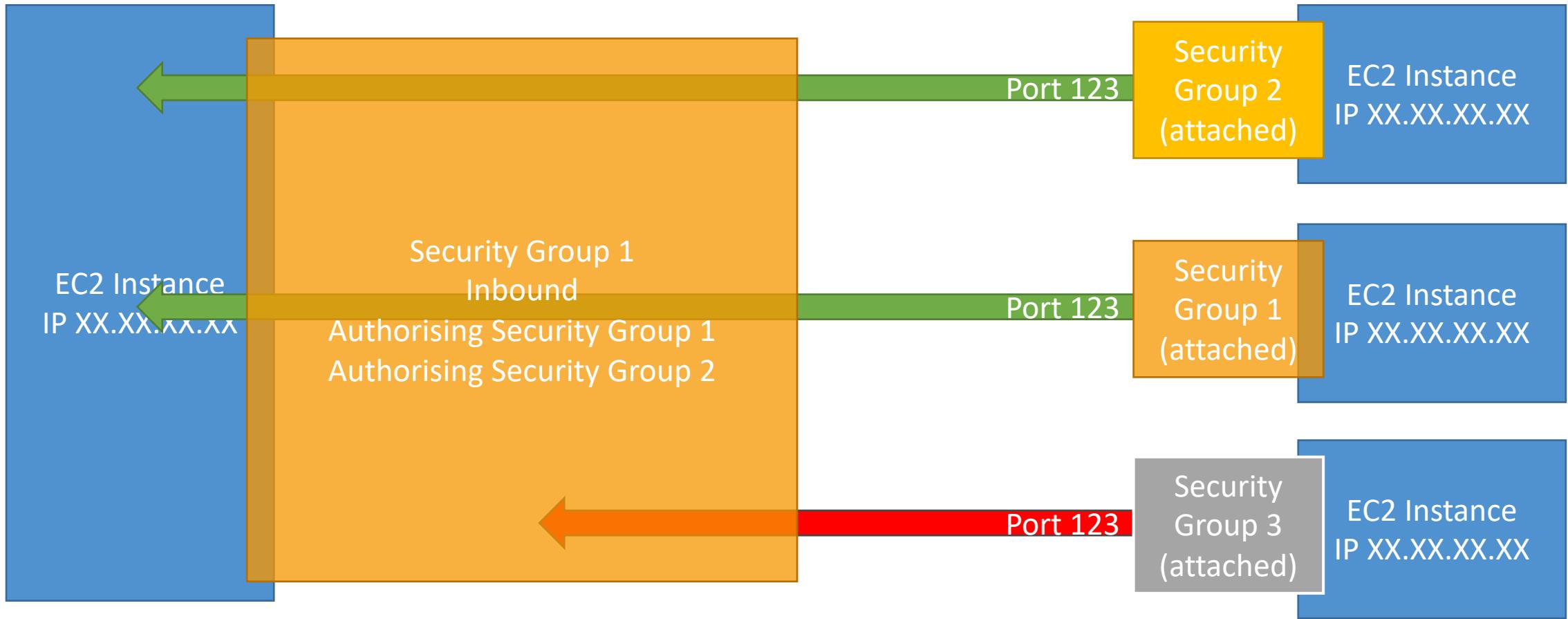
# Security Groups - Deeper Dive

- They regulate:
  - Access to Ports
  - Authorised IP ranges – IPv4 and IPv6
  - Control of inbound network (from other to the instance)
  - Control of outbound network (from the instance to other)
- Security groups are stateful
- You can reference another Security group as source

Inbound Rules - example

| Type            | Protocol | Port Range | Source            | Description    |
|-----------------|----------|------------|-------------------|----------------|
| HTTP            | TCP      | 80         | 0.0.0.0/0         | test http page |
| SSH             | TCP      | 22         | 122.149.196.85/32 |                |
| Custom TCP Rule | TCP      | 4567       | 0.0.0.0/0         | java app       |

# Referencing other security groups - Diagram



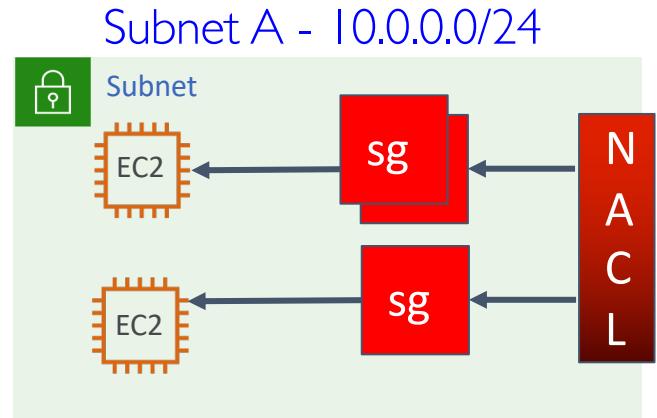
# Security Groups - Good to know

- Can be attached to multiple instances
- Locked down to a Region /VPC combination
- Does live “outside” the EC2 – if traffic is blocked the EC2 instance won’t see it
- If your application is not accessible (time out), then it’s a security group issue
- If your application gives a “connection refused“ error, then it’s an application error or it’s not yet in running state
- All inbound traffic is **blocked** by default
- All outbound traffic is **authorised** by default

| Security Group: sg-04cc11cb3874fa0c9 |          |            |             |             |
|--------------------------------------|----------|------------|-------------|-------------|
| <a href="#">Edit</a>                 |          |            |             |             |
| Type                                 | Protocol | Port Range | Destination | Description |
| All traffic                          | All      | All        | 0.0.0.0/0   |             |

# Network Access Control List (NACL)

- Works at Subnet level – Hence automatically applied to all instances
- Stateless – We need to explicitly open outbound traffic
- Contains both Allow and Deny rules
- Rules are evaluated in the order of rule number
- Default NACL allows all inbound and outbound traffic
- NACL are a great way of blocking a specific IP at the subnet level



| #Rule | Type        | Protocol | Port | Source            | Allow/Deny |
|-------|-------------|----------|------|-------------------|------------|
| 100   | ALL Traffic | ALL      | ALL  | 180.151.138.43/32 | DENY       |
| 101   | HTTPS       | TCP      | 443  | 0.0.0.0/0         | ALLOW      |
| *     | ALL Traffic | ALL      | ALL  | 0.0.0.0/0         | DENY       |

Network ACL inbound rules

# Network ACLs vs Security Groups

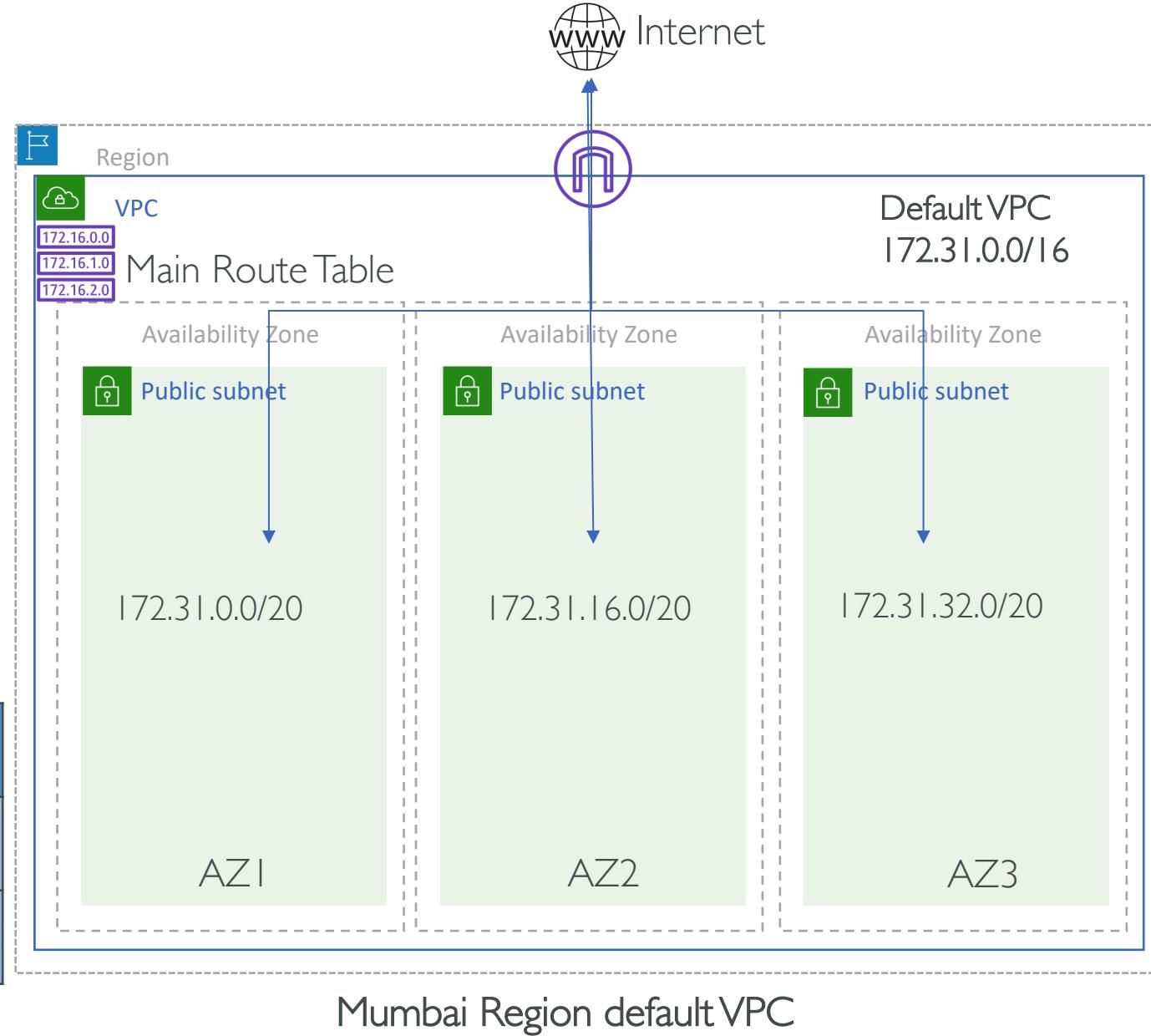
| Security Group   | Network ACL   |
|--|---|
| Operates at the instance level   | Operates at the subnet level  |
| Supports allow rules only  | Supports allow rules and deny rules   |
| Is stateful: Return traffic is automatically allowed, regardless of any rules  | Is stateless: Return traffic must be explicitly allowed by rules  |
| We evaluate all rules before deciding whether to allow traffic   | We process rules in number order when deciding whether to allow traffic   |
| Applies to an instance only if someone specifies the security group when launching the instance, or associates the security group with the instance later on | Automatically applies to all instances in the subnets it's associated with (therefore, you don't have to rely on users to specify the security group) |

[https://docs.aws.amazon.com/vpc/latest/userguide/VPC\\_Security.html#VPC\\_Security\\_Comparison](https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Security.html#VPC_Security_Comparison)

# Default VPC

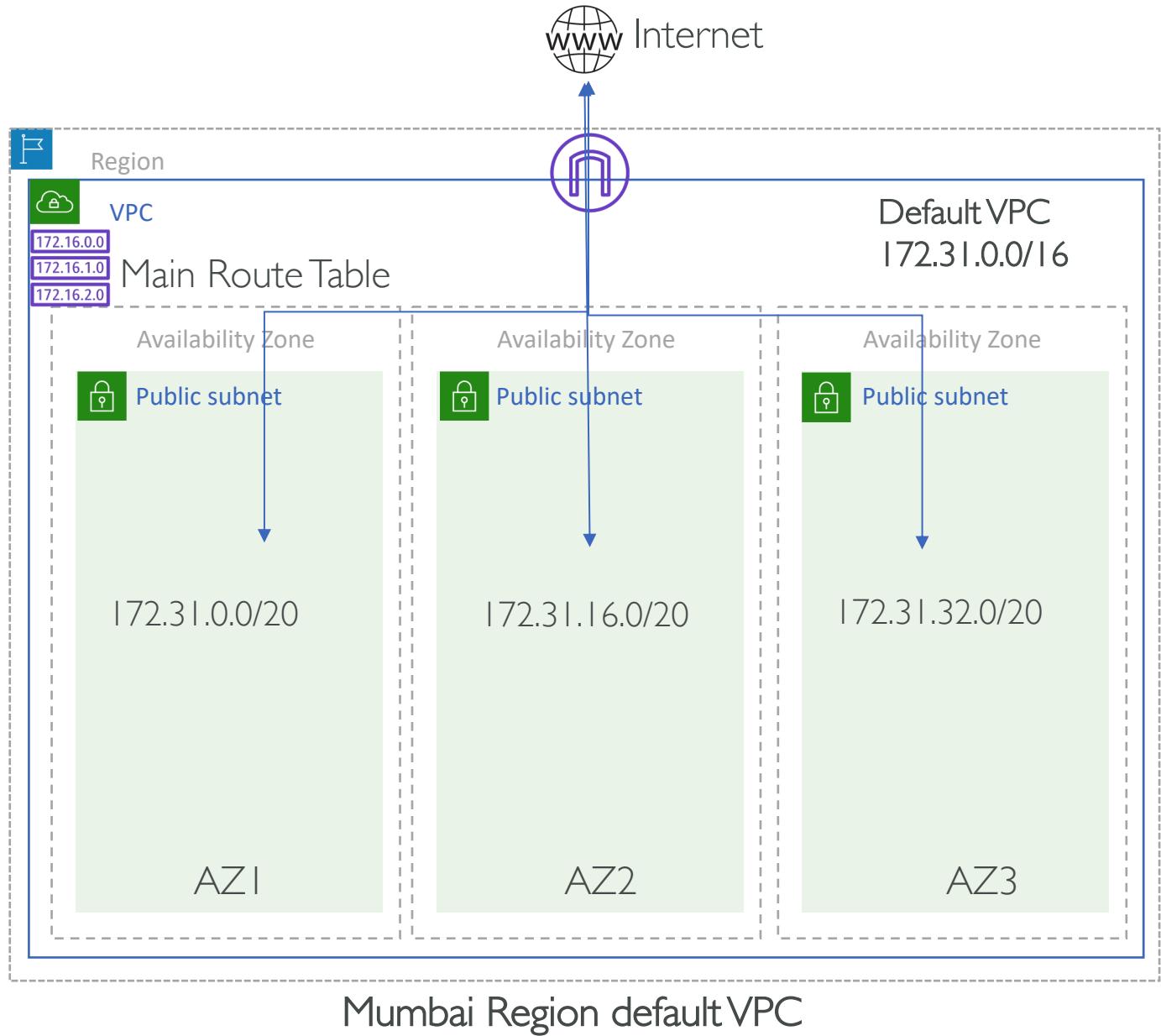
- AWS Creates Default VPC in each AWS region
- Creates VPC with CIDR - 172.31.0.0/16
- Creates Subnets in every AZ with CIDR /20
- Creates Internet Gateway
- Main route table with route to Internet which make all subnets public

| Destination   | Target     |
|---------------|------------|
| 172.31.0.0/16 | local      |
| 0.0.0.0/0     | igw-xxxxxx |

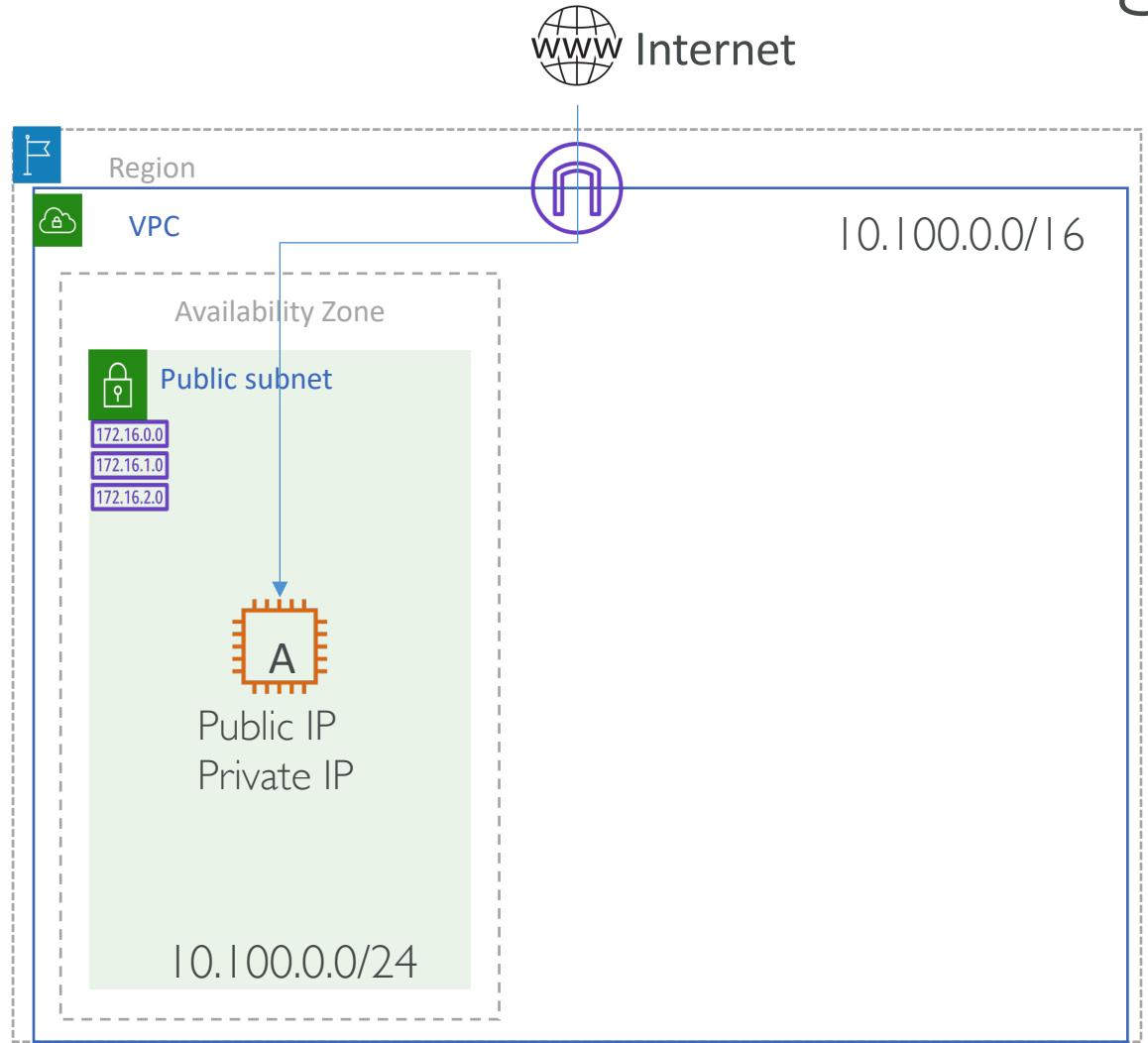


# Default VPC

- If deleted, you can recreate default VPC
- Console
  - > VPC Service
  - > Your VPCs
  - > Action
  - > Create Default VPC



# Demo - VPC with Single Public Subnet



Public Subnet Route Table

| Destination   | Target  |
|---------------|---------|
| 10.100.0.0/16 | local   |
| 0.0.0.0/0     | igw-xxx |

Note: For EC2 instance to be reachable from internet, it must be in Public Subnet and must have Public/Elastic IP

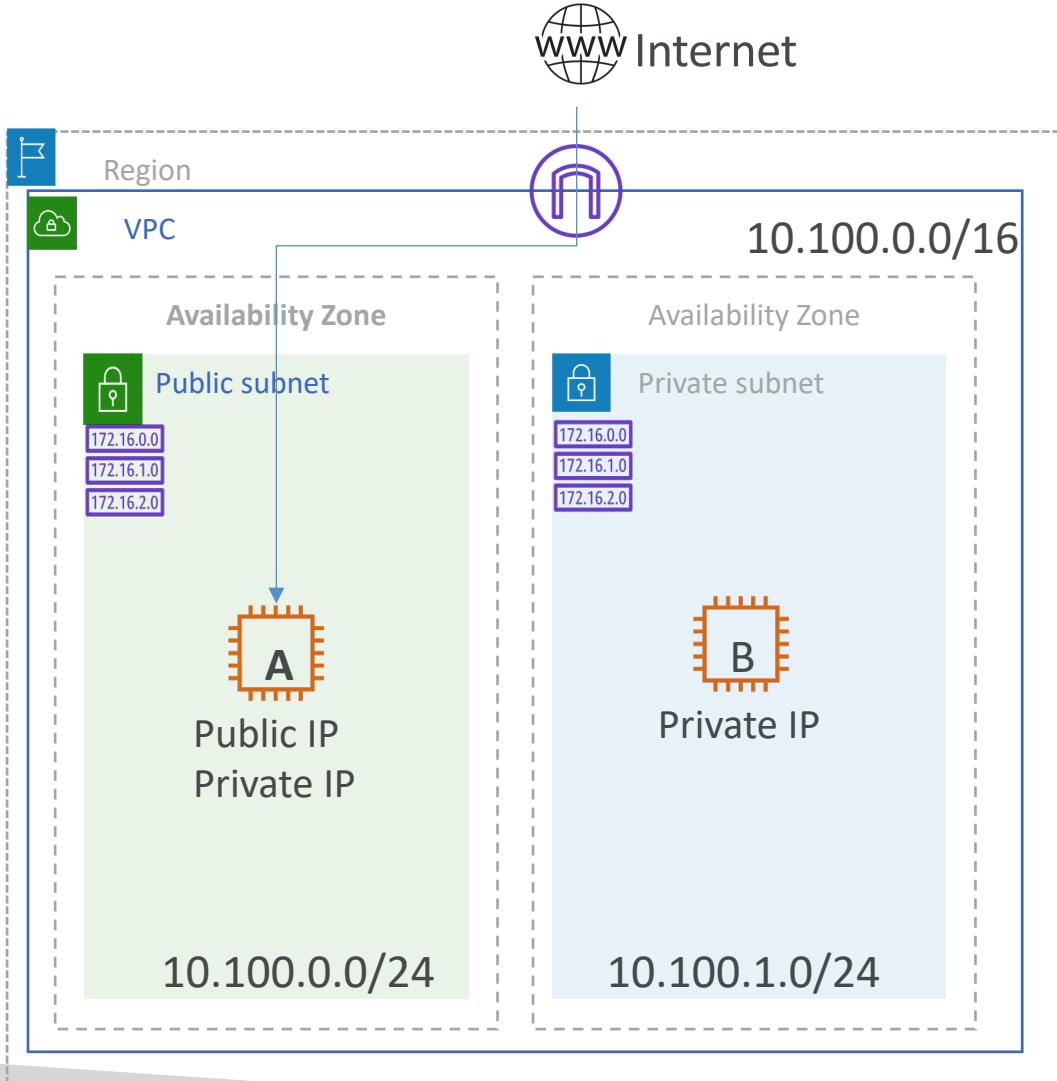
# Exercise I

1. Delete default VPC (We will create our own VPC)
2. Create VPC
  - a. Go to VPC service => Your VPCs => Create VPC (Name: MyVPC, CIDR: 10.100.0.0/16) => Create
3. Create Internet Gateway
  - a. Internet Gateways => Create internet gateway
4. Attach Internet Gateway to VPC
  - a. Select Internet gateway => Actions => Attach to VPC => Select your VPC
5. Create Subnet
  - a. Subnets => Create subnet (Name: MyVPC-Public, VPC: MyVPC, AZ: Select first AZ - ap-south-1a, CIDR: 10.100.0.0/24)
  - b. Select Subnet => Action => Modify Auto Assign Public IP => Enable => Save
6. Create Route table
  - a. Route Tables => Create Route Table (Name: MyVPC-Public, VPC: MyVPC)
  - b. Select Route table => Routes => Edit => Add another route (Destination: 0.0.0.0/0, Target: Internet gateway => igw-xxx) => Save

# Exercise I

6. Associate Route table with Subnet to make it Public subnet
  - a. Select Route table => Subnet Associations => Edit => Check the MyVPC-Public subnet => Save
7. Launch EC2 instance in newly created Public Subnet
  - a. Go to EC2 Service => Instances
  - b. Launch EC2 Instance => Select Amazon Linux 2 => Select t2.micro
  - c. Configure Instance Details:
    - i. Network: MyVPC
    - ii. Subnet: MyVPC-Public (rest all defaults)
  - d. Add storage (all defaults)
  - e. Add Tags
    - i. Key=Name, Value=EC2-A
  - f. Configure Security Group
    - i. Add rule for SSH port 22 for source as MyIP
  - g. Review and Launch
8. Connect to EC2 instance (Public IP) from your desktop/laptop using Putty or terminal (ec2-user)

# Demo - VPC with Public and Private Subnet



Private Subnet Route Table

| Destination   | Target |
|---------------|--------|
| 10.100.0.0/16 | local  |

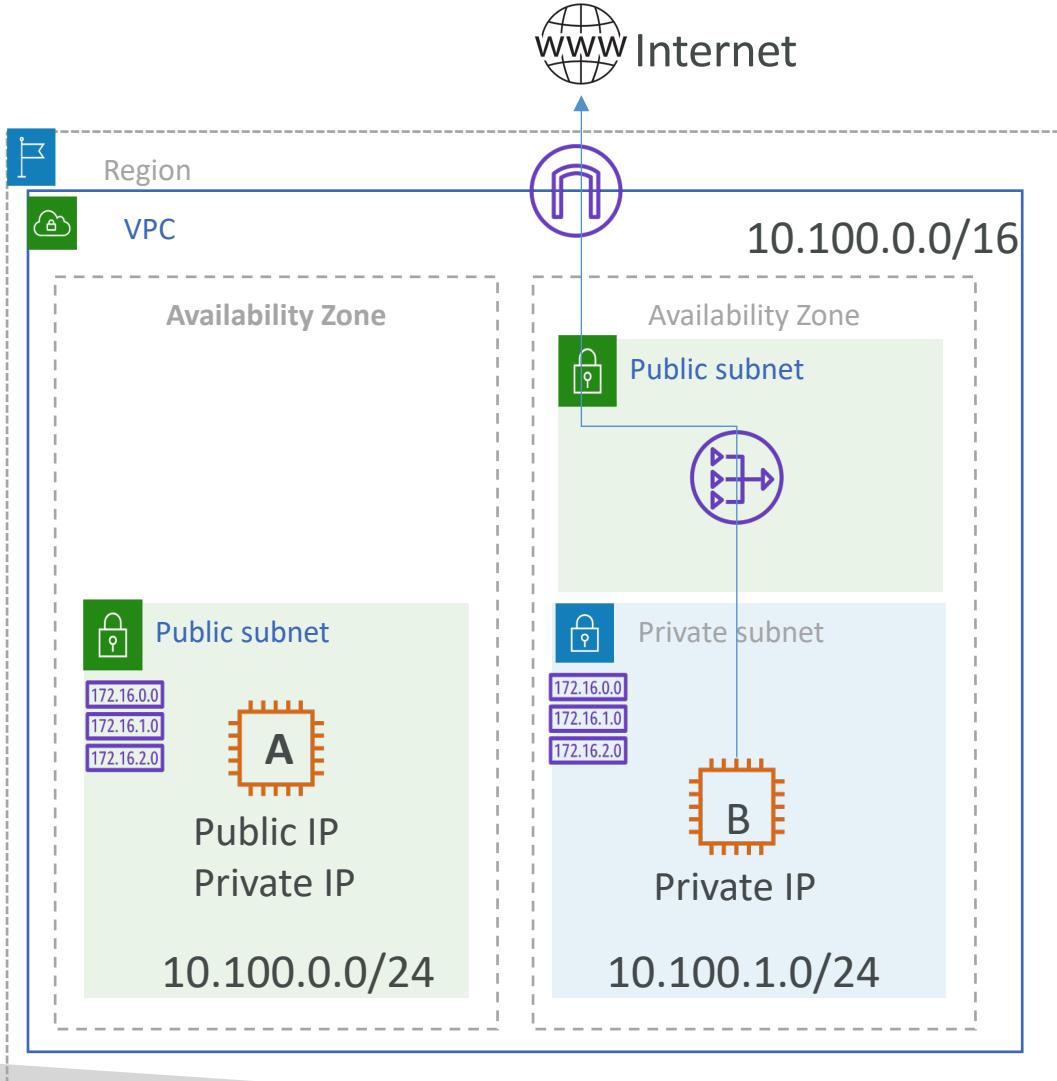
# Exercise 2 (Continuing with previous setup)

1. Create a Private subnet
  - a. Create subnet (Name: MyVPC-Private, VPC: MyVPC, AZ: Select different AZ (ap-south-1b), CIDR: 10.100.1.0/24)
2. Create Private route table
  - a. Route Tables => Create Route Table (Name: MyVPC-Private, VPC: MyVPC)
3. Associate Route table with Subnet to make it Private subnet
  - a. Select Route table => Subnet Associations => Edit => Check the MyVPC-Private subnet => Save
4. Launch another EC2 instance in same VPC but in newly created **Private subnet**.
  - a. Tag this instance with Name=EC2-B
  - b. **New security group**
    - Add rule SSH for CIDR of Public Subnet source CIDR
    - Add rule All-ICMP IPv4 for Public Subnet source CIDR
5. Note down EC2-B private IP address

# Exercise 2 (Continuing with previous setup)

6. Try to ping EC2-B Private IP from EC2-A instance => Should work
7. Try to connect to EC2-B instance from EC2-A (Permissions denied..Why?)
  - a. \$ssh ec2-user@10.100.1.x (Replace this ip with your EC2-B IP address)
6. Get your ssh .pem file on EC2-A instance
  - a. Open local .pem file with nodepad and copy the content (CTRLA => CTRL+C)
  - b. On EC2 A terminal => vi key.pem => enter => press i => paste using right click => esc => :wq => enter
  - c. chmod 600 key.pem
  - d. ssh -i key.pem ec2-user@10.100.1.x => should be able to connect
6. Try to ping google.com from EC2-B instance
  - a. ping google.com (You should not be able to ping.Why?)

# VPC with Public and Private Subnet

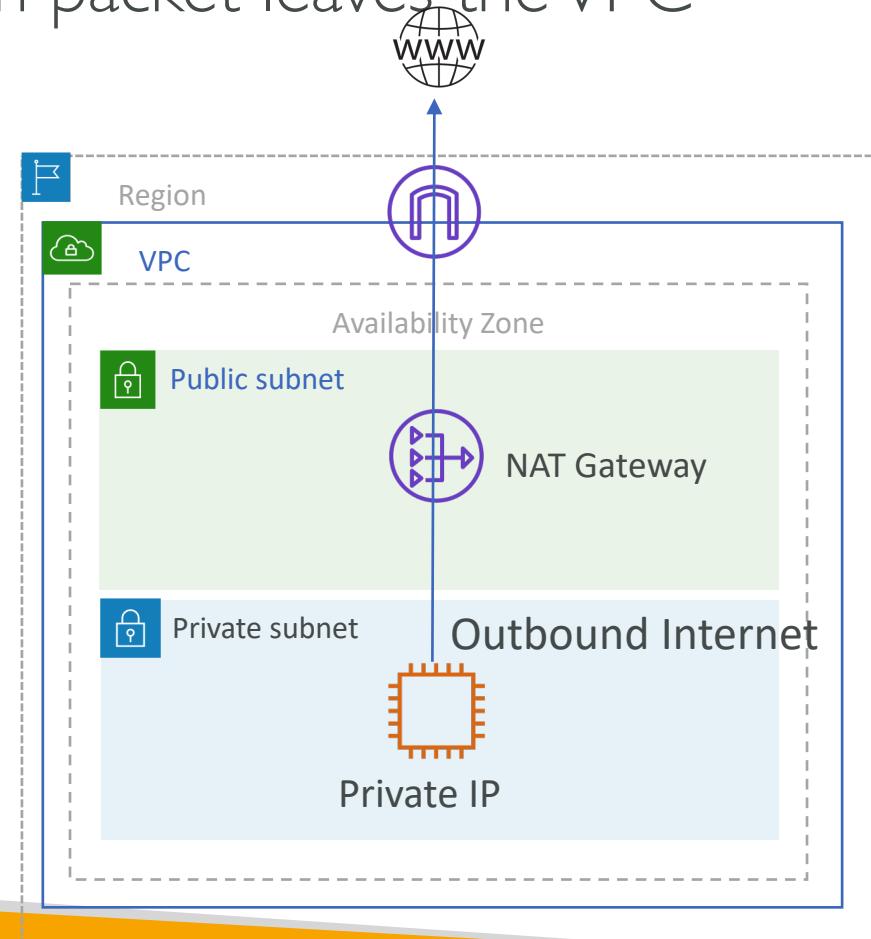


Private Subnet Route Table

| Destination   | Target |
|---------------|--------|
| 10.100.0.0/16 | local  |

# NAT Gateway and NAT Instance

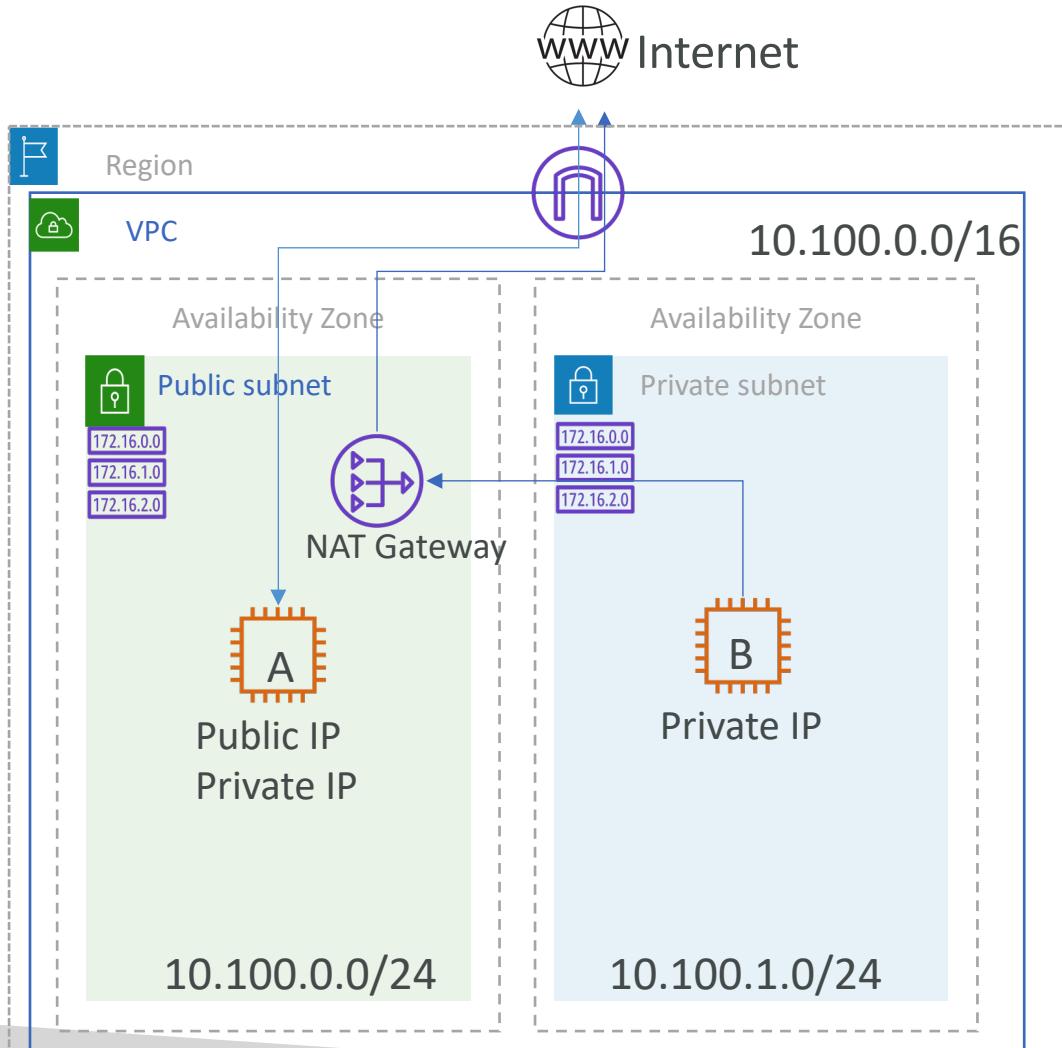
- To provide Internet Access to instances in private subnet without IGW
- Performs Network Address translation when packet leaves the VPC
- **Nat Gateway:**
  - Managed by AWS
  - Highly available within AZ
- **NAT Instance**
  - NAT EC2 can be launched using Amazon Linux Nat AMI
  - Disable Source/Destination check on instance
  - Allocate EIP



# NAT Gateway

- AWS managed NAT, higher bandwidth, better availability, no admin
- Pay by the hour for usage and bandwidth
- NAT is created in a specific AZ, uses an EIP
- 5 Gbps of bandwidth with automatic scaling up to 45 Gbps
- No security group to manage / required. NACL at subnet level applies to NAT Gateway.
- Supported protocols: TCP, UDP, and ICMP
- Uses ports 1024–65535 for outbound connection

# Demo – NAT Gateway



- NAT Gateway must be created in Public Subnet so that it can communicate with the Internet
- NAT Gateway should be allocated Elastic IP

Private Subnet Route Table

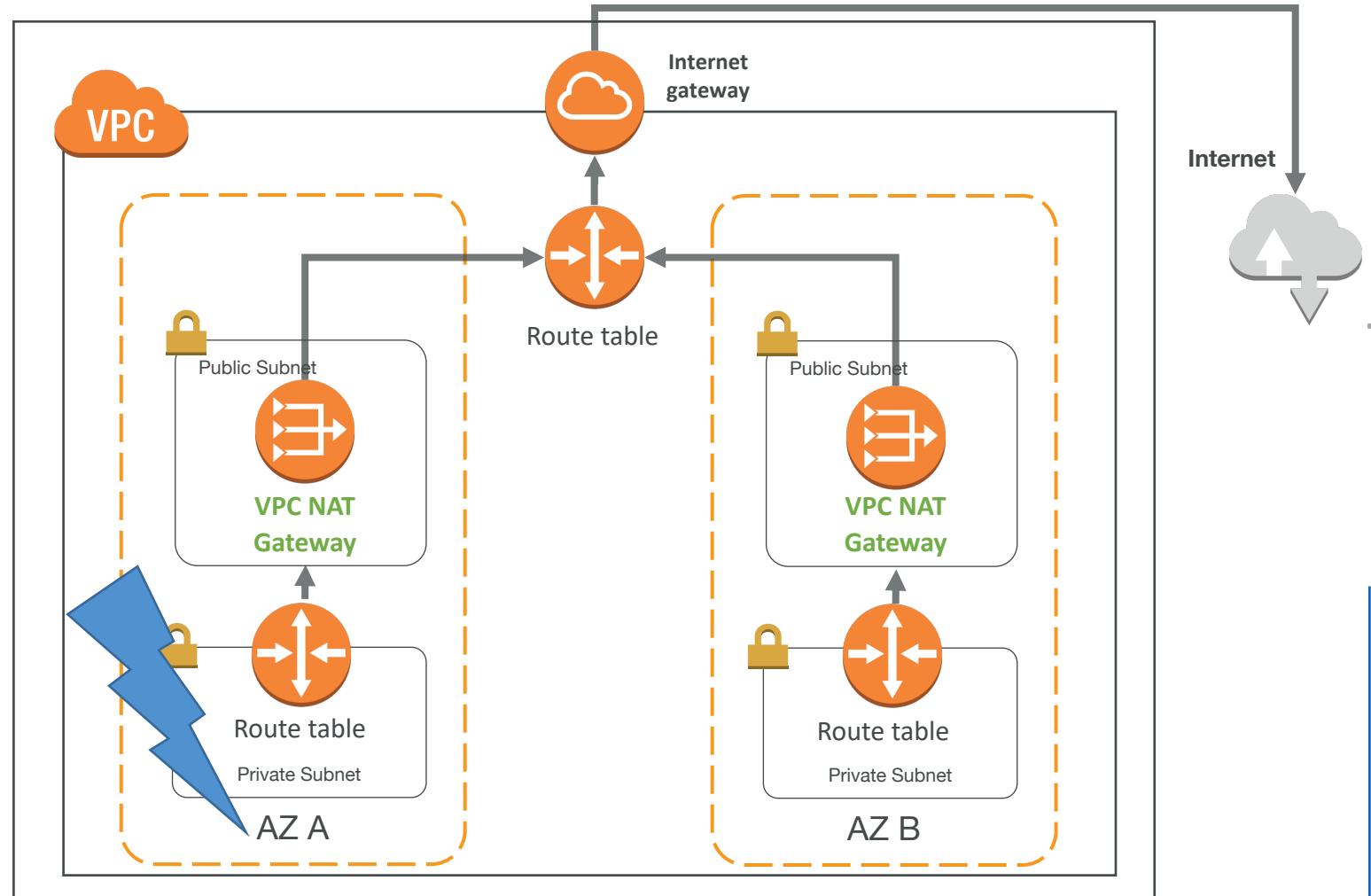
| Destination   | Target       |
|---------------|--------------|
| 10.100.0.0/16 | local        |
| 0.0.0.0/0     | nat-xxxxxxxx |

# Exercise 3 (Continuing with previous setup)

- Create a NAT Gateway in your VPC
  - VPC => NAT Gateways => Create NAT Gateway
    - Subnet: MyVPC-Public (Must select Public Subnet)
    - EIP: Create New EIP
    - Create NAT Gateway
    - It takes 5-10 minutes for NAT Gateway to be Active
- Add a route in Private subnet for internet traffic and route through NAT Gateway
  - Route Tables => Select MyVPC-Private route table
  - Routes => Edit => Add another route
    - Destination: 0.0.0.0/0
    - Target: nat-gateway
    - Save
- Now again try to ping google.com from EC2-B
  - ping google.com

# NAT Gateway with High Availability

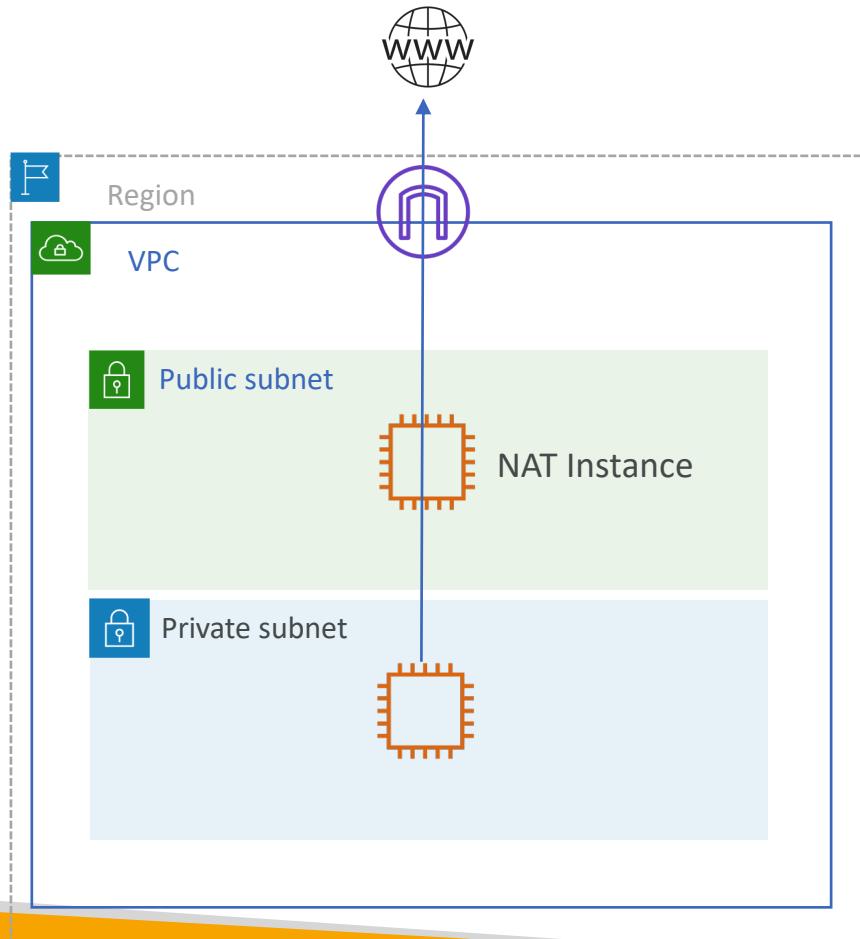
- NAT Gateway is resilient within a single-AZ
- Must create **multiple NAT Gateway** in multiple AZ for fault-tolerance
- There is no cross AZ failover needed because if an AZ goes down it doesn't need NAT



# Setup your own NAT on EC2 (NAT Instance)

- Must be in Public Subnet
- Must have Public or Elastic IP
- Should be launched using AWS provided NAT AMIs
- Disable Source/Destination Check
- Update Private subnet route tables.
- For internet traffic set target as NAT Instance ID

Reference: [Setting up NAT Instance](#)



# Nat Gateway vs NAT Instance

| Attribute           | NAT Gateway  | NAT Instance   |
|---------------------|--|--|
| Availability        | Highly available within AZ. Create a NAT Gateway in each Availability Zone to ensure zone-independent architecture.  | Use a script to manage failover between instances.   |
| Bandwidth           | Can scale up to 45 Gbps.   | Depends on the bandwidth of the instance type.   |
| Maintenance         | Managed by AWS. You do not need to perform any maintenance.  | Managed by you, for example, by installing software updates or operating system patches on the instance.             |
| Performance         | Software is optimized for handling NAT traffic.  | A generic Amazon Linux AMI that's configured to perform NAT.   |
| Cost                | Charged depending on the number of NAT Gateways you use, duration of usage, and amount of data that you send through the NAT Gateways.                         | Charged depending on the number of NAT Instances that you use, duration of usage, and instance type and size.        |
| Type and size       | Uniform offering; you don't need to decide on the type or size.  | Choose a suitable instance type and size, according to your predicted workload.                                      |
| Public IP addresses | Choose the Elastic IP address to associate with a NAT Gateway at creation.   | Use an Elastic IP address or a public IP address with a NAT Instance.  |
| Security groups     | Cannot be associated with a NAT Gateway. You can associate security groups with your resources behind the NAT Gateway to control inbound and outbound traffic. | Associate with your NAT Instance and the resources behind your NAT Instance to control inbound and outbound traffic. |
| Port forwarding     | Not supported.   | Manually customize the configuration to support port forwarding.   |
| Bastion servers     | Not supported.   | Use as a bastion server.   |

# NAT Gateway Troubleshooting / Rules / Limitations

- <https://docs.aws.amazon.com/vpc/latest/userguide/nat-gateway-troubleshooting.html>
- <https://docs.aws.amazon.com/vpc/latest/userguide/vpc-nat-gateway.html#nat-gateway-limits>
- <https://aws.amazon.com/premiumsupport/knowledge-center/vpc-resolve-port-allocation-errors/>
- <https://docs.aws.amazon.com/vpc/latest/userguide/nat-gateway-troubleshooting.html#nat-gateway-troubleshooting-tcp-issues>

# Recap / Important Concepts Covered

- We saw how Physical network transformed into virtual network in Cloud
- AWS Region, Availability zone and VPC relation
- VPC Addressing – CIDR
- VPC Components – Subnets, Route Tables, Internet Gateway
- IP Addresses – Private vs Public vs Elastic
- Elastic Network Interfaces
- Firewalls – Security groups and Network ACLs
- NAT – Nat Gateways and NAT Instance

# Exam Essentials

- Maximum size of VPC/Subnet is /16 which contains 65536 IP addresses
- Minimum size of VPC/Subnet is /28 which contains 16 IP addresses
- In every subnet 5 IPs are un-usuable – first 4 and last IP in the subnet
- Subnet is associated with AZ. One subnet can not span across AZs. However one AZ may contain any number of subnets.
- All subnets by default follow VPC main route table unless explicitly attached to custom route table
- All route tables have a default local route which you can't remove. This route enables interVPC communication between all subnets.

# Exam Essentials

- You can't block any IP with Security group. Use Network ACL.
- Security groups is stateful - No need to explicitly add outbound rule for incoming return traffic or inbound rules for outbound return traffic
- Network ACL is stateless – Rules must be created for both side of the traffic
- NAT Gateway must be created in each AZ for High Availability.
- NAT Gateways must be created in Public Subnet
- NAT Gateway does not have security group hence traffic should be controlled by using Network ACL
- You don't get access to NAT Gateway machine. It's managed by AWS.
- NAT Instance may be cost effective but may not be as performant and resilient as NAT Gateway.
- For NAT Instance, Source/Destination check must be disabled

# AWS VPC Advanced features

# Topics

- Extending VPC address space
- Elastic Network Interfaces (Revisit)
- VPC Traffic Monitoring
- VPC Traffic Mirroring

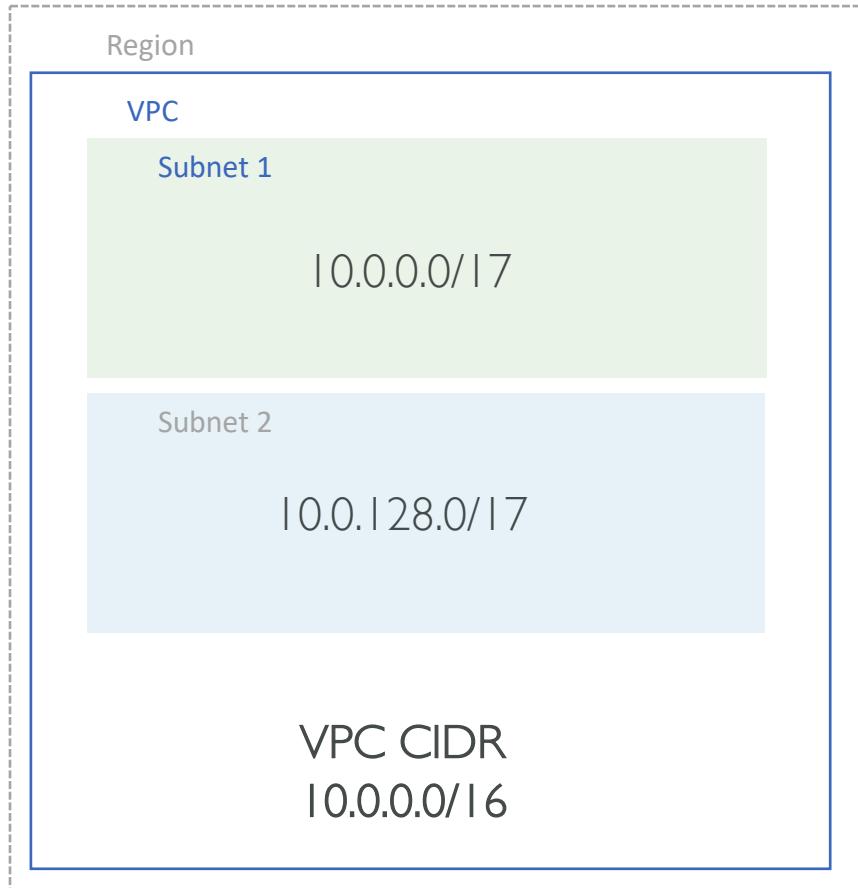
# Extending VPC address space

# VPC secondary CIDR blocks

1. You can add secondary VPC CIDRs to existing VPC
2. CIDR block must not overlap with existing CIDR or peered VPC CIDR
3. If Primary CIDR is from RFC1918 then you can not add secondary CIDR from other RFC1918 IP ranges (10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16)
4. CIDR block must not be same or larger than the CIDR range of routes in any of the VPC Route tables

For example, if VPC primary CIDR block is 10.0.0.0/16 and you want to associate a secondary CIDR block in the 10.2.0.0/16 range. You already have a route with a destination of 10.2.0.0/24 to a virtual private gateway, therefore you cannot associate a CIDR block of the same range or larger. However, you can associate a CIDR block of 10.2.0.0/25 or smaller.
5. You can have total 5 IPv4 and 1 IPv6 CIDR block for VPC

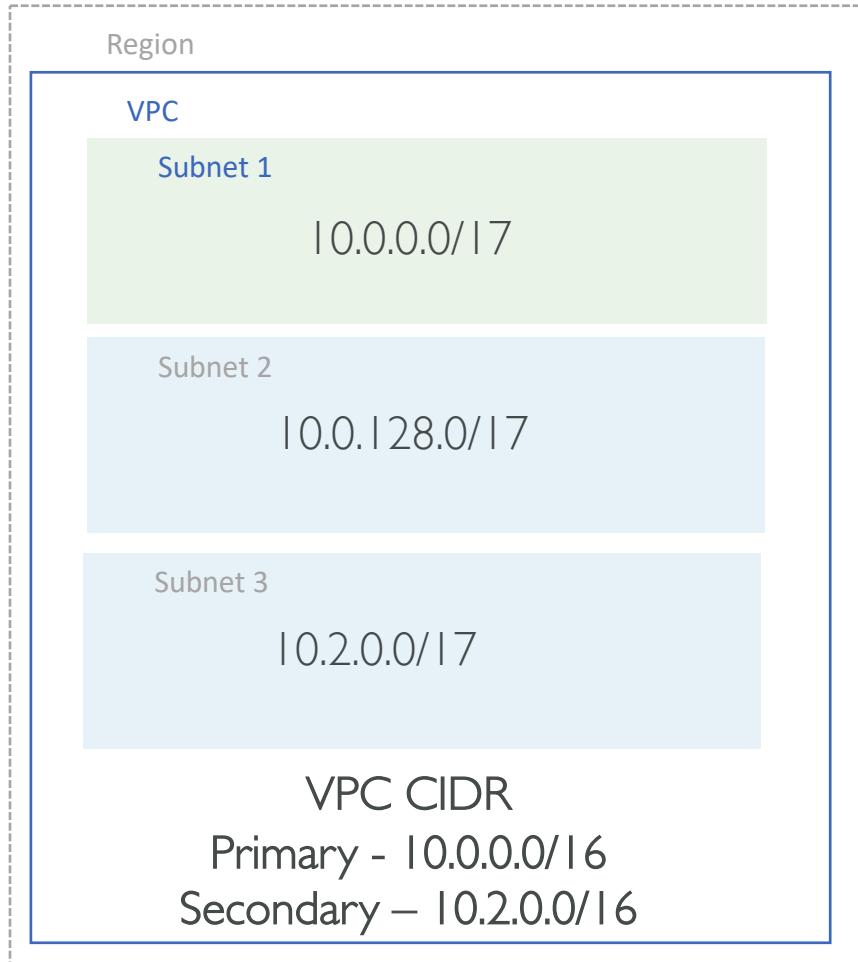
# VPC with Primary CIDR Block



Main Route Table

| Destination | Target |
|-------------|--------|
| 10.0.0.0/16 | local  |

# VPC with Primary and Secondary CIDR Block



Main Route Table

| Destination | Target |
|-------------|--------|
| 10.0.0.0/16 | local  |
| 10.2.0.0/16 | local  |

# Example |

- Create a VPC with CIDR 10.0.0.0/16
- Select VPC => Actions => Edit CIDR => Add new IPv4 CIDR from 10.1.0.0/16, 10.2.0.0/16 etc

| CIDR ⓘ      | Status     | Status reason |   |
|-------------|------------|---------------|---|
| 10.0.0.0/16 | associated | -             | ✖ |
| 10.1.0.0/16 | associated | -             | ✖ |
| 10.2.0.0/16 | associated | -             | ✖ |
| 10.3.0.0/16 | associated | -             | ✖ |
| 10.4.0.0/16 | associated | -             | ✖ |

**Add IPv4 CIDR**

# Example 2

- Create a VPC with CIDR 10.0.0.0/16
- Select VPC => Actions => Edit CIDR => Add other IPv4 CIDR from RFC1918 range 192.168.0.0/16 or 172.16.0.0/16

| CIDR  | Status     |
|---|------------|
| 10.0.0.0/16   | associated |
| <b>⚠ You cannot create a CIDR in this range. Choose a CIDR in a non restricted range <a href="#">Find out more</a> about restricted ranges.</b> |            |
| 192.168.0.0/16  |            |
| <b>Add IPv4 CIDR</b>  |            |

# Example 2

- Create a VPC with CIDR 10.0.0.0/16
- Select VPC => Actions => Edit CIDR => Add new IPv4 CIDR 192.168.0.0/16 or 172.16.0.0/16

| CIDR        | Status     |
|-------------|------------|
| 10.0.0.0/16 | associated |

**!** You cannot create a CIDR in this range. Choose a CIDR in a non restricted range [Find out more](#) about restricted ranges.

# Example 3

- Create a VPC with CIDR 10.0.0.0/16
- Select VPC => Actions => Edit CIDR => Add new IPv4 CIDR 100.64.0.0/16 (Non RFC1918 range)

| CIDR                 | Status     | Status reason |   |
|----------------------|------------|---------------|---|
| 10.0.0.0/16          | associated | -             | × |
| 100.64.0.0/16        | associated | -             | × |
| <b>Add IPv4 CIDR</b> |            |               |   |

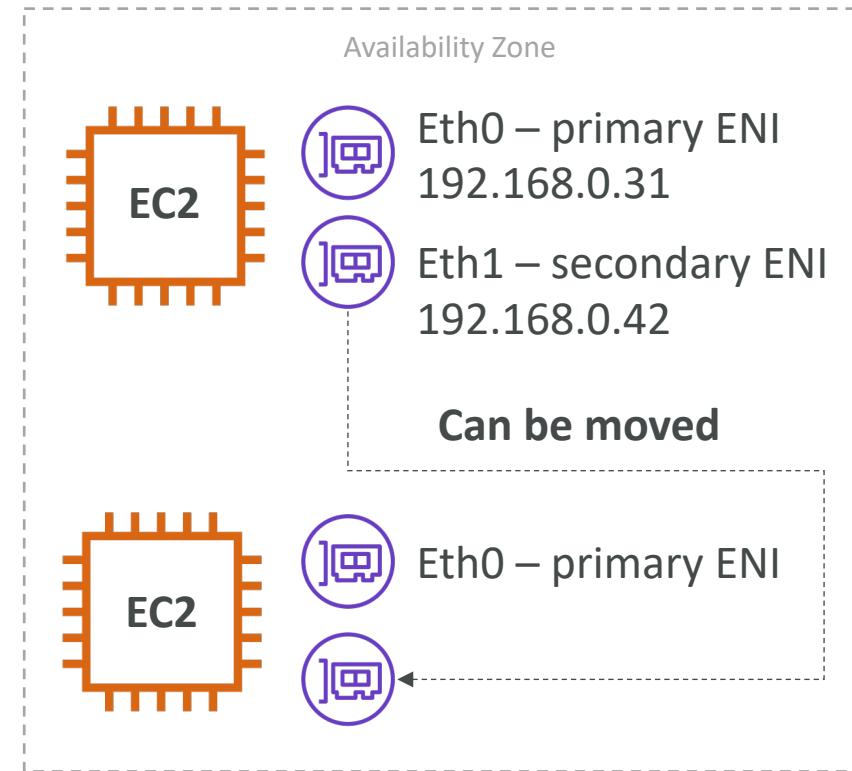
# Examples:

| Primary CIDR  | Allowed Secondary CIDR  | Not allowed Secondary CIDRs   |
|---------------|---|---|
| 10.0.0.0/16   | 10.1.0.0/16 likewise<br>100.64.0.0/16,<br>Public IPv4 Ranges    | 192.168.0.0/16,<br>172.31.0.0/16,<br>Any overlapping CIDR                 |
| 172.31.0.0/16 | 172.32.0.0/16 likewise,<br>100.64.0.0/16,<br>Public IPv4 Ranges | 192.168.0.0/16,<br>10.0.0.0/16,<br>Any overlapping CIDR                   |
| 100.64.0.0/16 | 100.65. 0.0/16 likewise,<br>Public IPv4 Ranges                  | 192.168.0.0/16,<br>10.0.0.0/16,<br>172.31.0.0/16,<br>Any overlapping CIDR |

# Elastic Network Interfaces

# Elastic Network Interfaces (ENI) - Revisit

- Logical component in a VPC that represents a **virtual network card**
- The ENI can have the following attributes:
  - Primary private IPv4, one or more secondary IPv4
  - One Elastic IP (IPv4) per private IPv4 address
  - One Public IPv4 address
  - One or more security groups
  - A MAC address
  - A source/destination check flag
- You can create ENI independently and attach them on the fly (move them) on EC2 instances for failover.
- When you move a network interface from one instance to another, network traffic is redirected to the new instance
- Bound to a specific availability zone (AZ)



# More about Elastic Network Interfaces (ENI)

1. You can not detach primary network interface from an instance
2. You associate security groups with network interfaces and not with individual IP addresses.
3. Second ENI allows instance to be multi-homed (subnets) **in same AZ**
4. ENIs can not be used for NIC teaming which means they can not be used together to increase instance network bandwidth
5. The number of ENIs that you can attach to instance and number of secondary IP addresses per ENI depends on EC2 instance type

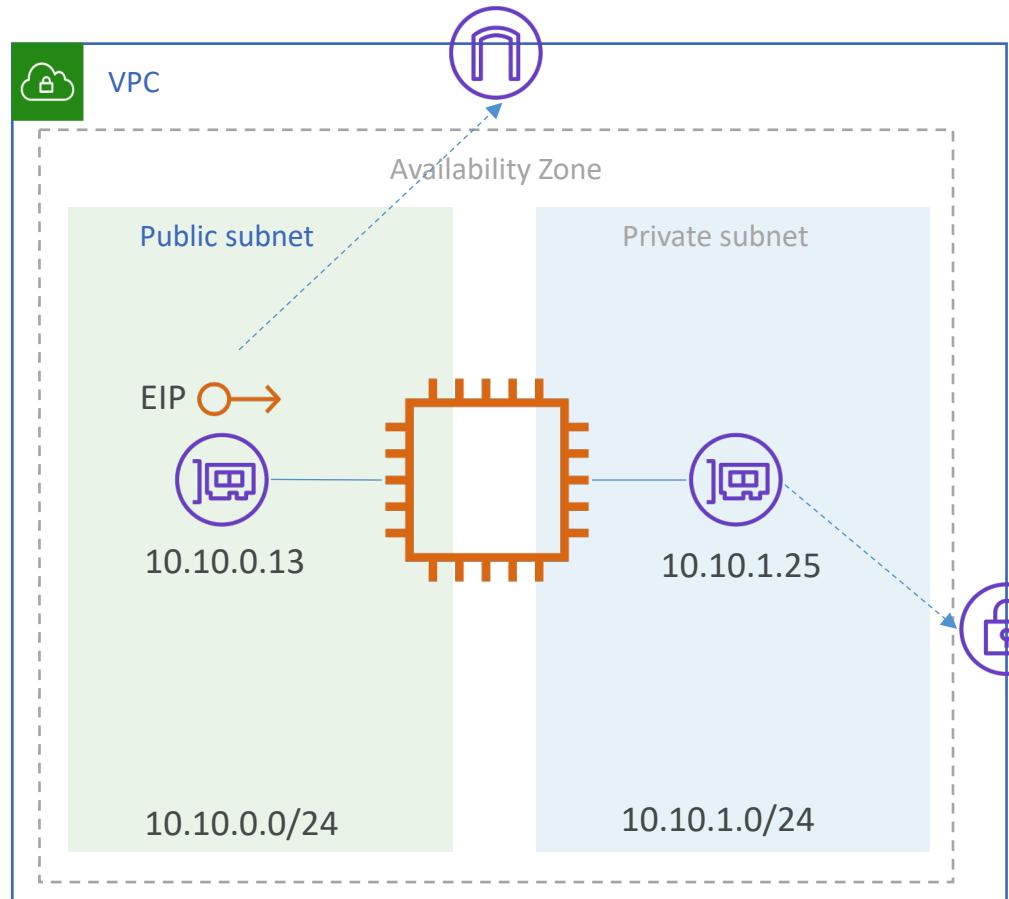
Example: c4.xlarge instance may be attached 4 ENIs with each ENI supporting 15 IPv4 private addresses and 15 IPv6 addresses

# IP Addresses per instance

| Instance type | Maximum network interfaces | Private IPv4 addresses per interface | IPv6 addresses per interface |
|---------------|----------------------------|--------------------------------------|------------------------------|
| a1.medium     | 2                          | 4                                    | 4                            |
| a1.large      | 3                          | 10                                   | 10                           |
| a1.xlarge     | 4                          | 15                                   | 15                           |
| a1.2xlarge    | 4                          | 15                                   | 15                           |
| a1.4xlarge    | 8                          | 30                                   | 30                           |
| a1.metal      | 8                          | 30                                   | 30                           |
| c1.medium     | 2                          | 6                                    | IPv6 not supported           |
| c1.xlarge     | 4                          | 15                                   | IPv6 not supported           |
| c3.large      | 3                          | 10                                   | 10                           |

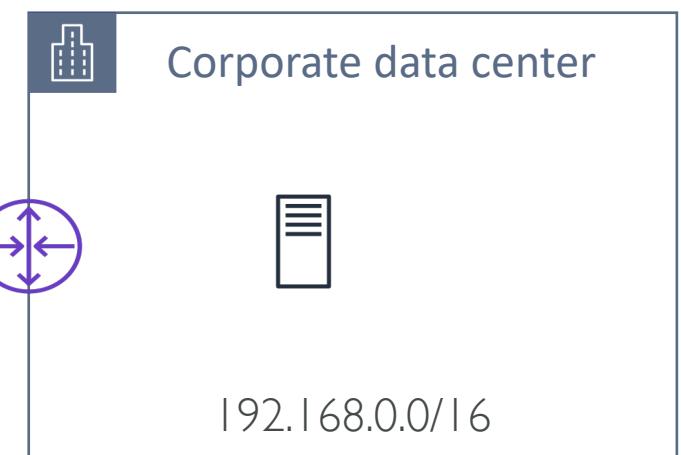
<https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html#AvailableIpPerENI>

# Elastic Network Interfaces – Dual Home



| Destination  | Target       |
|--------------|--------------|
| 10.10.0.0/16 | local        |
| 0.0.0.0/0    | lgw-xxxxxxxx |

| Destination    | Target       |
|----------------|--------------|
| 10.10.0.0/16   | local        |
| 192.168.0.0/16 | vgw-xxxxxxxx |



# More about Elastic Network Interfaces (ENI)

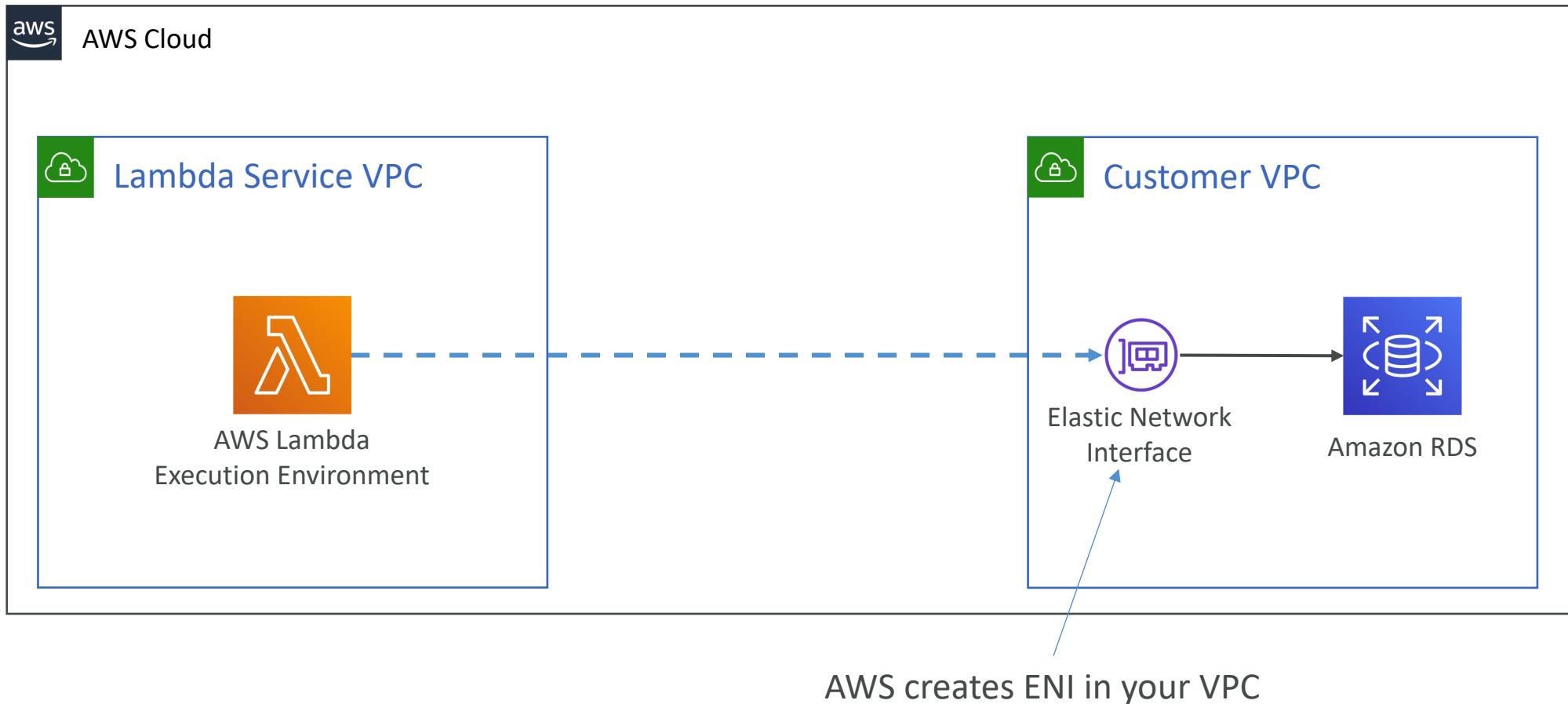
## 7. Cross-account Network Interface

- You can create the ENI into another account
- The cross-account network permission grants an AWS-authorized provider account permission to attach a customer network interface to an instance in the provider account.

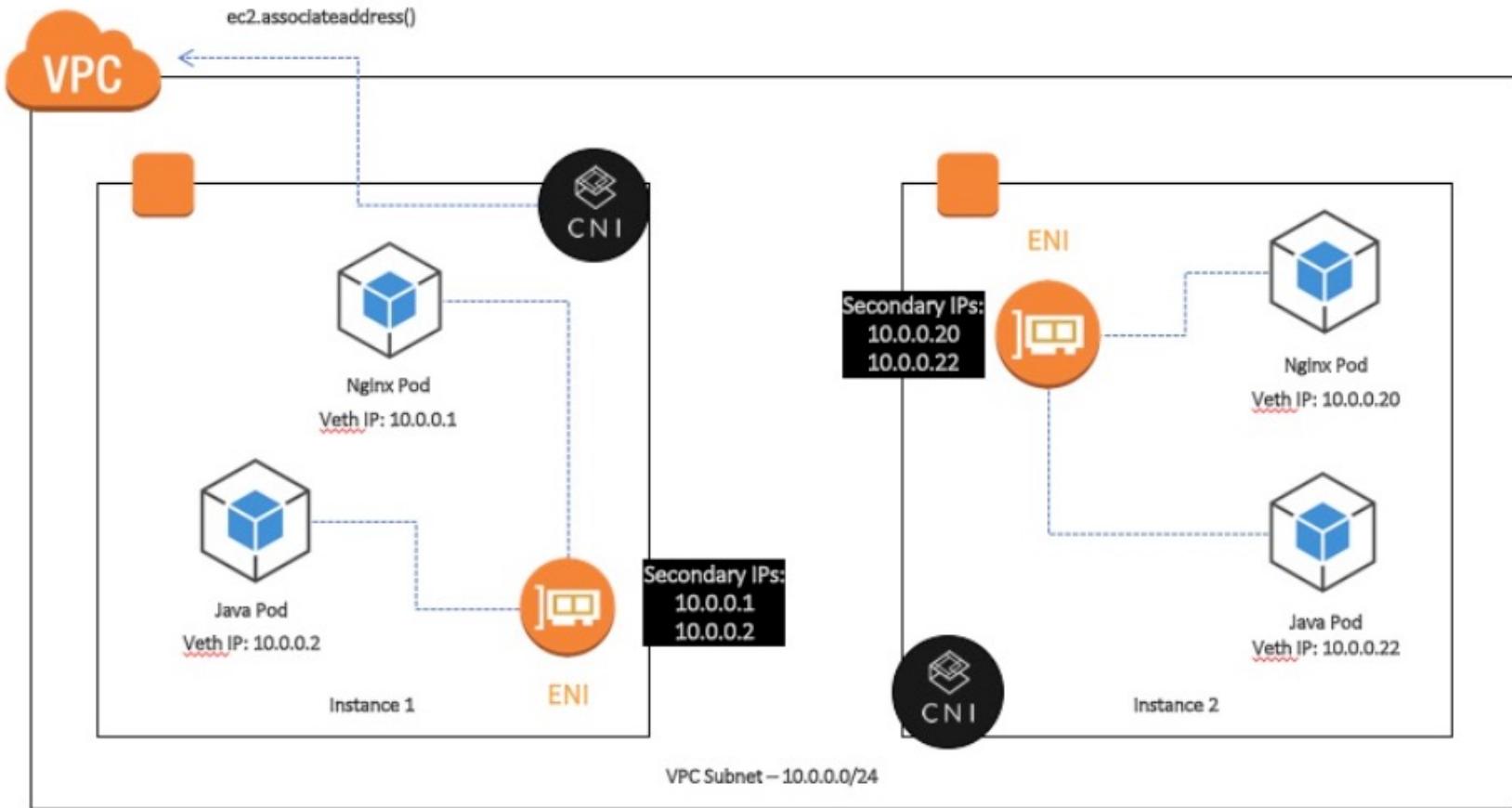
### Examples:

1. RDS instances reside in AWS managed VPC however the Network interface is created in customer VPC where customer can control the traffic using Security Groups. These are also called Requester managed network interfaces.
2. EKS (Control Plane) master nodes are launched in AWS managed VPC and it creates ENIs into your VPC so that it can communicate with EKS worker nodes
3. For AWS Workspaces/Appstream the underlying EC2 instances are launched inside AWS managed VPC and ENIs are created into your VPC so that those instances can communicate with applications inside your VPC

# Example: Lambda to access private resources inside your VPC

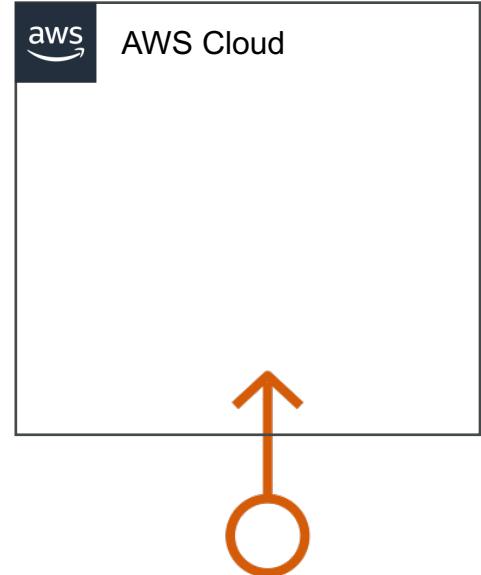


# Example: Secondary IPs for PODs in EKS



# Bring Your Own IP

- You can migrate your publicly routable IPv4 and IPv6 IP addresses to AWS
- But Why?
  - Keep your IP address reputation
  - Avoid changes to IP Address whitelisting
  - Move applications without change in the IP addresses
  - AWS as a hot standby



# Pre-requisites to BYOIP

- The address range must be registered with regional internet registry (RIR) – ARIN or RIPE or APNIC
- The addresses in the IP address range must have a clean history. AWS reserve the right to reject poor reputation IP address ranges.
- The most specific IPv4 address range that you can bring is /24
- The most specific IPv6 address range that you can bring is /48 for CIDRs that are publicly advertised
- The most specific IPv6 address range that you can bring is /56 for CIDRs that are not publicly advertised (can be advertised over Direct Connect if required)
- Create a Route Origin Authorization (ROA) to authorize Amazon ASNs 16509 and 14618 to advertise your address range

# Good to know about BYOIP

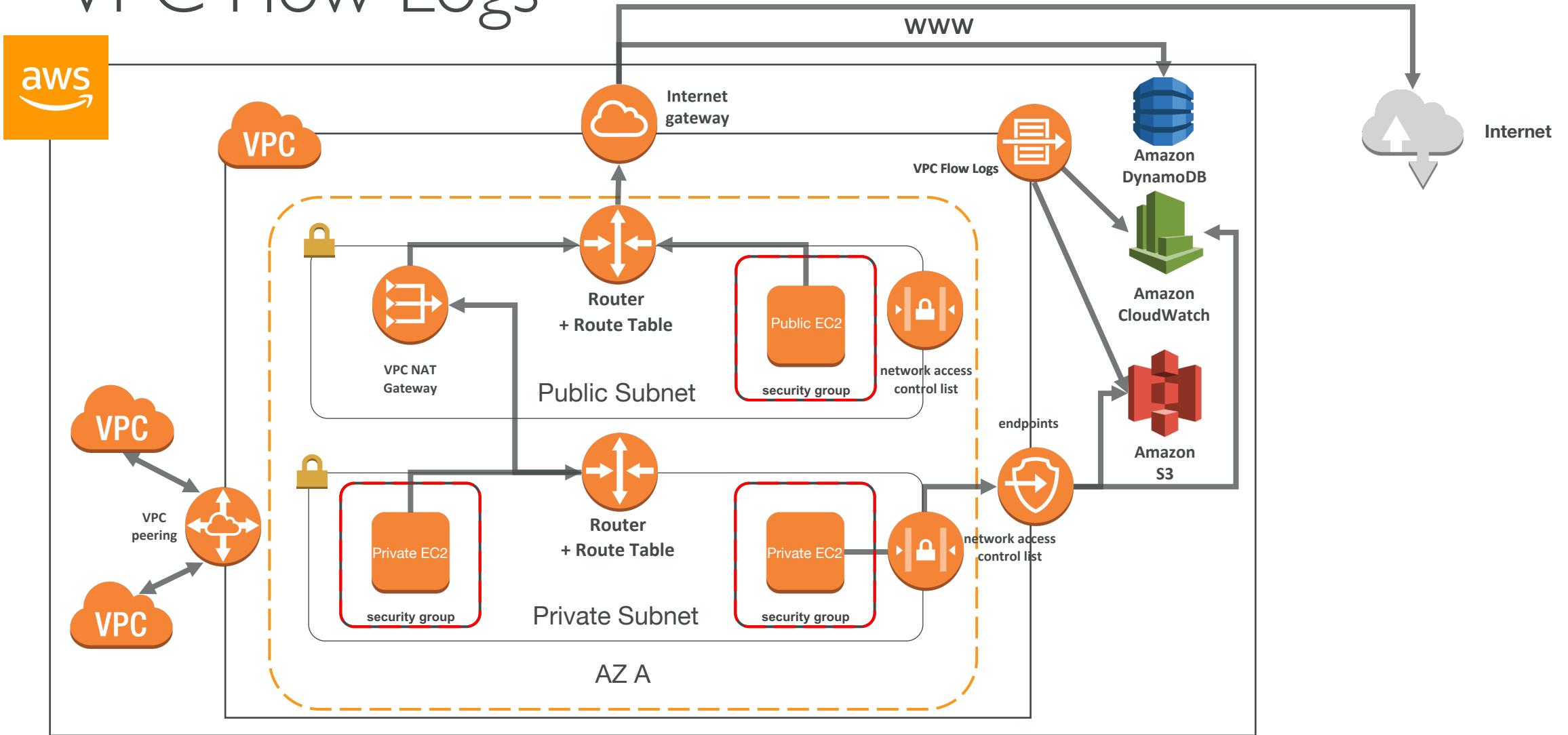
- You continue to own the address range, but AWS advertises it on the internet by default
- After you bring the address range to AWS, it appears in your AWS account as an address pool.
- You can associate these IP addresses to Amazon EC2 instances, Network Load Balancers, and NAT gateways
- You can bring a total of five IPv4 and IPv6 address ranges per Region to your AWS account.

# VPC Traffic Monitoring

# VPC Flow Logs

- Capture information about IP traffic going in/out of your interfaces:
  - VPC Flow Logs
  - Subnet Flow Logs
  - Elastic Network Interface Flow Logs
- Helps to monitor & troubleshoot connectivity issues
- Flow logs data can go to S3 / CloudWatch Logs
- Captures network information from AWS managed interfaces too: ELB, RDS, ElastiCache, Redshift, Amazon WorkSpaces

# VPC Flow Logs



# Flow Log Syntax

- <version> <account-id> <interface-id> <srcaddr> <dstaddr> <srcport> <dstport> <protocol> <packets> <bytes> <start> <end> <action> <log-status>
- Srcaddr, dstaddr help identify problematic IP
- Srcport, dstport help identity problematic ports
- Action : success or failure of the request due to Security Group / NACL
- Can be used for analytics on usage patterns, or malicious behavior
- Query VPC flow logs using Athena on S3 or CloudWatch Logs Insights

# Example:

|   | version               | account-id | interface-id | srcaddr               | dstaddr               | srcport       | dstport       | protocol | packets | bytes | start | end        | action     | log-status |        |    |
|---|-----------------------|------------|--------------|-----------------------|-----------------------|---------------|---------------|----------|---------|-------|-------|------------|------------|------------|--------|----|
| ▶ | 2020-12-26T18:53:0... | 2          | 523265502623 | eni-0e6a89b99a1bac86d | 27.57.246.36          | 10.100.0.224  | 28472         | 80       | 6       | 2     | 92    | 1608988983 | 1608989040 | ACCEPT     | OK     |    |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 10.100.0.224  | 27.57.246.36  | 80       | 28472   | 6     | 1     | 52         | 1608988983 | 1608989040 | ACCEPT | OK |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 27.57.246.36  | 10.100.0.224  | 28595    | 80      | 6     | 4     | 173        | 1608988983 | 1608989040 | ACCEPT | OK |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 10.100.0.224  | 27.57.246.36  | 80       | 28595   | 6     | 3     | 144        | 1608988983 | 1608989040 | ACCEPT | OK |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 45.129.33.8   | 10.100.0.224  | 58523    | 30018   | 6     | 1     | 40         | 1608988983 | 1608989040 | REJECT | OK |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 27.57.246.36  | 10.100.0.224  | 28478    | 443     | 6     | 1     | 52         | 1608988983 | 1608989040 | REJECT | OK |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 27.57.246.36  | 10.100.0.224  | 28090    | 443     | 6     | 2     | 104        | 1608988983 | 1608989040 | REJECT | OK |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 27.57.246.36  | 10.100.0.224  | 28103    | 443     | 6     | 1     | 52         | 1608988983 | 1608989040 | REJECT | OK |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 27.57.246.36  | 10.100.0.224  | 28472    | 443     | 6     | 1     | 52         | 1608988983 | 1608989040 | REJECT | OK |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 27.57.246.36  | 10.100.0.224  | 28605    | 22      | 6     | 6     | 528        | 1608988983 | 1608989040 | ACCEPT | OK |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 10.100.0.224  | 27.57.246.36  | 22       | 28605   | 6     | 6     | 240        | 1608988983 | 1608989040 | ACCEPT | OK |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 27.57.246.36  | 10.100.0.224  | 28108    | 443     | 6     | 2     | 104        | 1608988983 | 1608989040 | REJECT | OK |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 46.227.200.78 | 10.100.0.224  | 123      | 58455   | 17    | 1     | 76         | 1608988983 | 1608989040 | ACCEPT | OK |
| ▶ | 2020-12-26T18:53:0... |            | 2            | 523265502623          | eni-0e6a89b99a1bac86d | 10.100.0.224  | 46.227.200.78 | 58455    | 123     | 17    | 1     | 76         | 1608988983 | 1608989040 | ACCEPT | OK |

# Lab

- Analyze VPC Flows logs with CloudWatch Insights

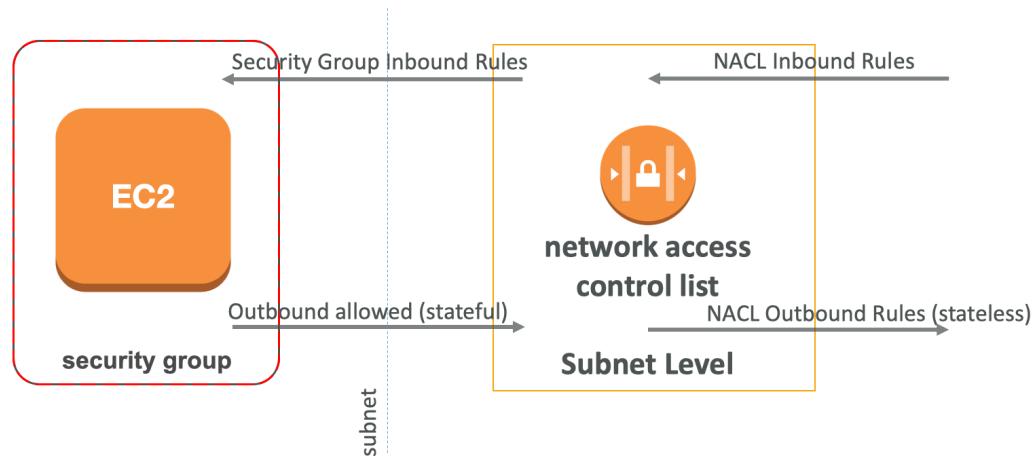
# Flow Logs: Looking at “ACTION” field

## How to troubleshoot SG vs NACL issue?

### For incoming requests

Inbound REJECT: NACL or SG

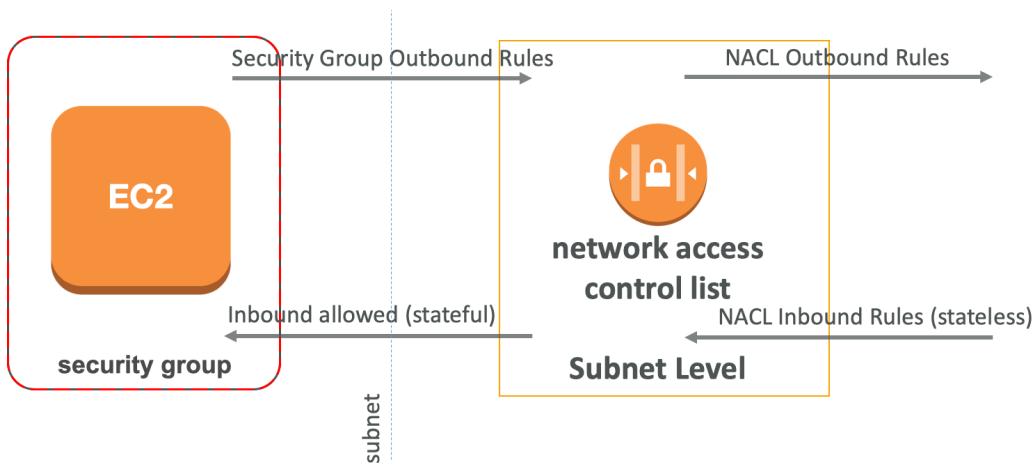
Inbound ACCEPT, outbound REJECT: NACL



### For outgoing requests

Outbound REJECT: NACL or SG

Outbound ACCEPT, inbound REJECT: NACL



# Flow Logs limitation

Amazon VPC Flow Logs do not record traffic

- To and from VPC-native DNS services
- Amazon EC2 metadata service
- Dynamic Host Configuration Protocol (DHCP) services
- Windows license activation server

# Other Traditional Tools for Network Monitoring

- Packet Capture (for deep packet inspection)
  - Wireshark (Windows/Linux) and tcpdump (Linux) which can be run on EC2 instance
- traceroute
- telnet
- nslookup - Used to resolve the hostnames into IP addresses
- ping
  - Ping records network round trips using Internet Control Message Protocol
  - ICMP traffic should be allowed through Security Groups, NACL

Security Group: sg-0dbfbf97c45a30961

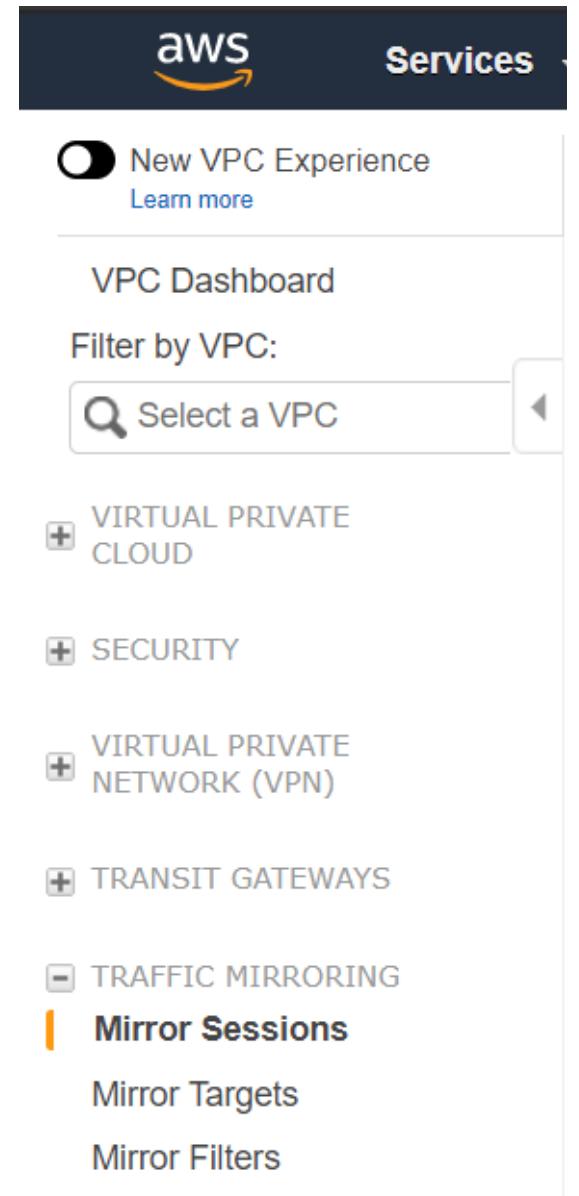
The screenshot shows the AWS Management Console interface for managing a security group. At the top, there are tabs for 'Description', 'Inbound' (which is selected), 'Outbound', and 'Tags'. Below the tabs, there is an 'Edit' button. The main area displays two inbound rules in a table format:

| Type            | Protocol | Port Range | Source           | Description           |
|-----------------|----------|------------|------------------|-----------------------|
| SSH             | TCP      | 22         | 182.70.87.195/32 | SSH Traffic           |
| All ICMP - IPv4 | All      | N/A        | 0.0.0.0/0        | PING - ICMP Traffi... |

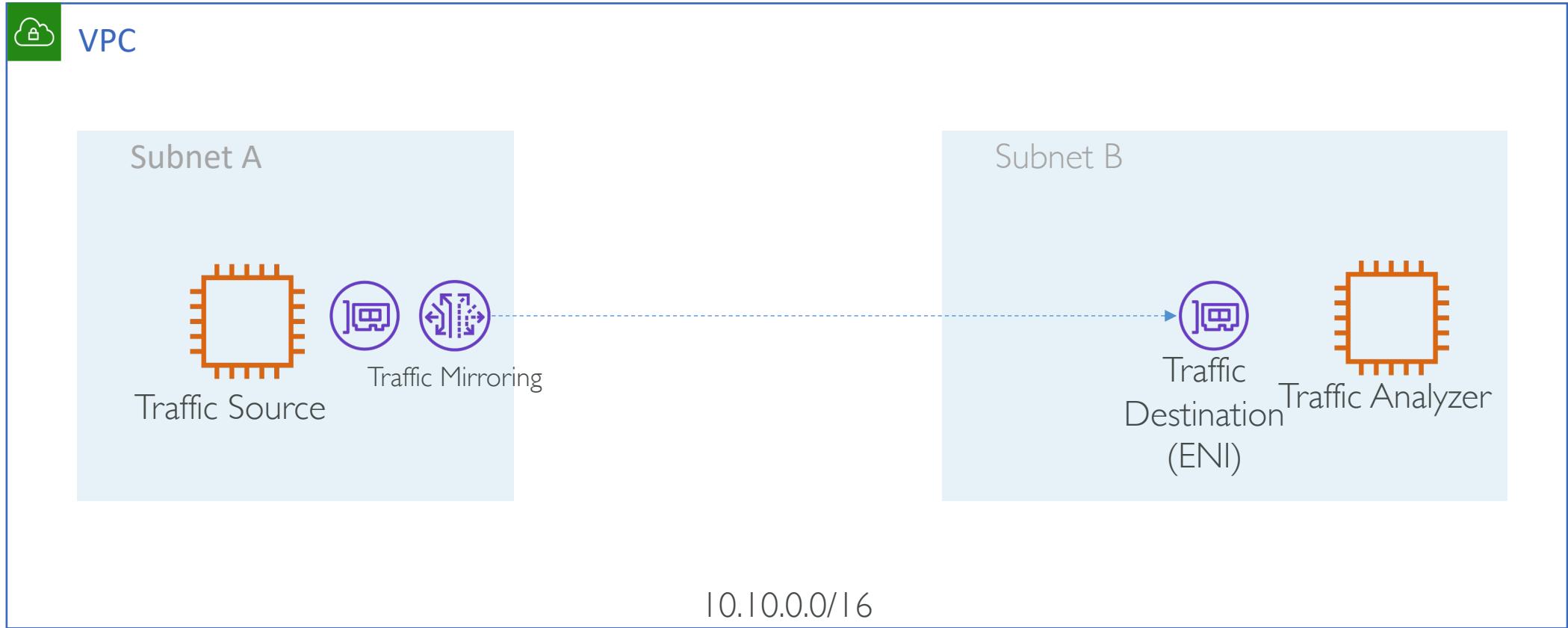
# VPC Traffic Mirroring

# VPC Traffic Mirroring (new)

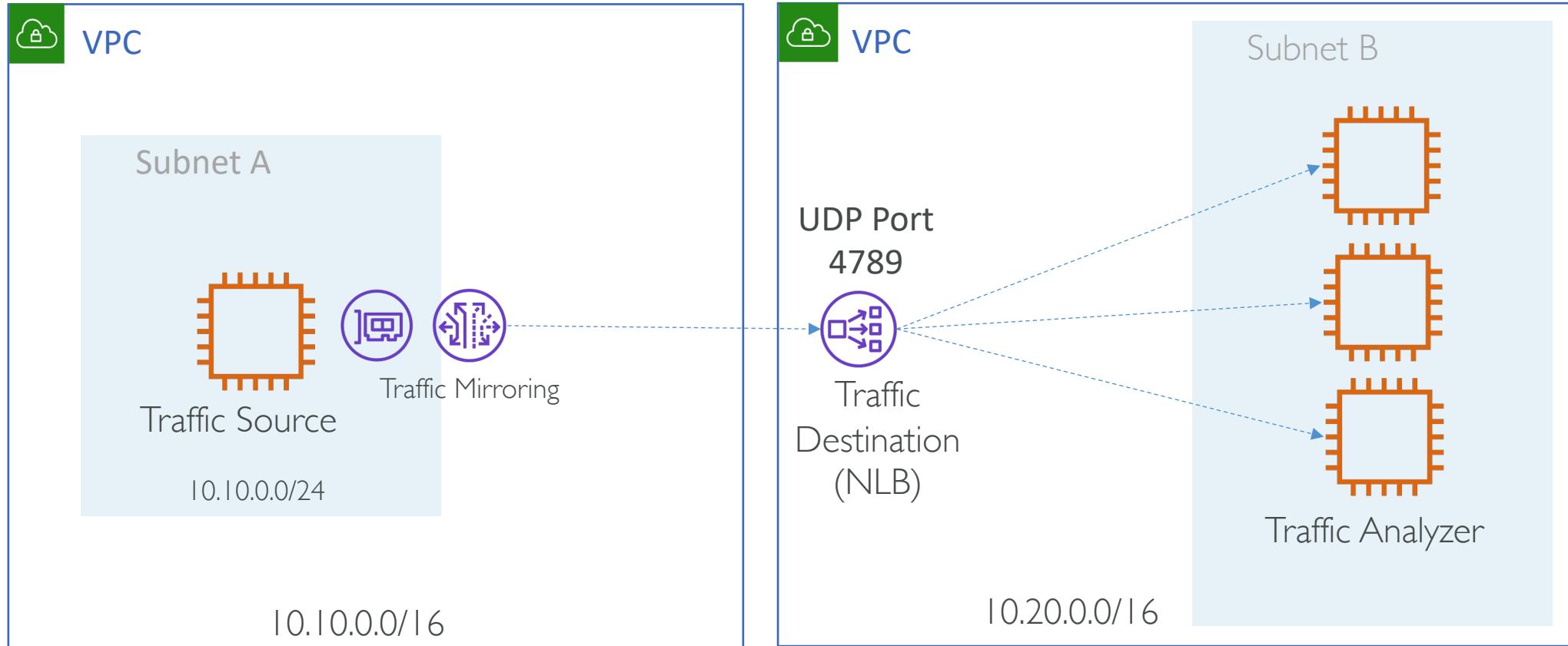
- Copy network traffic from an elastic network interface of Amazon EC2 instances
- Send the traffic to out-of-band security and monitoring appliances for Content inspection or threat monitoring or for troubleshooting
- How to set up Traffic Mirroring (via AWS VPC console)
  - Create the Mirror Target
  - Define the Traffic Filter
  - Create Mirror Session



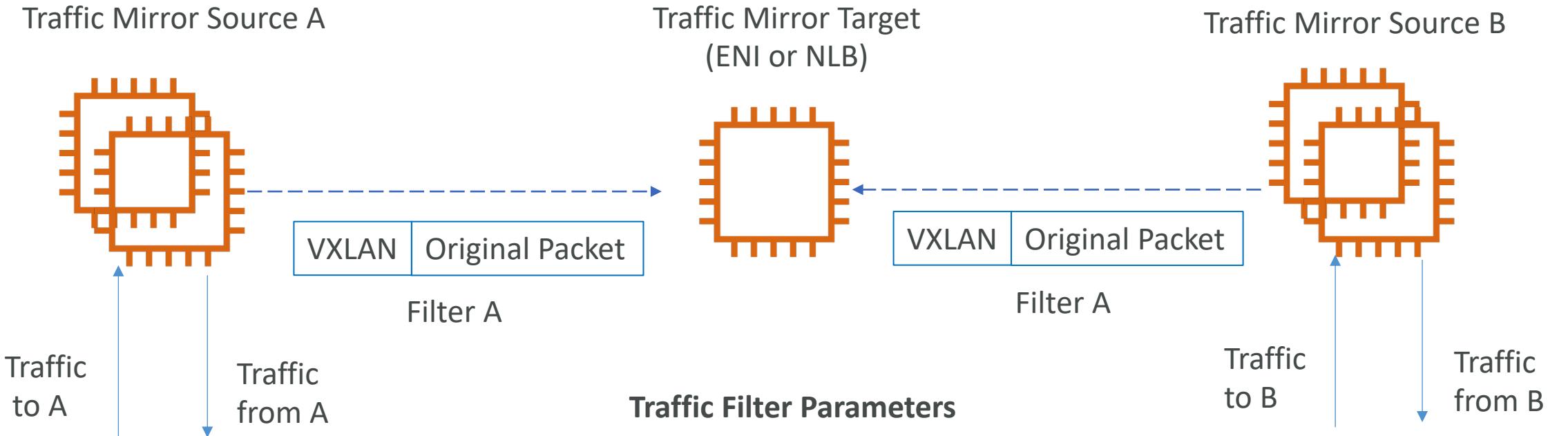
# VPC Traffic Mirroring – ENI as Target



# VPC Traffic Mirroring – NLB as Target



# VPC Traffic Mirroring Filters



# VPC Traffic Mirroring – Good to know

- Mirrors the traffic flowing in/out of VPC and routes it to network analysis tools to detect the potential network & security anomalies
- Mirror source is ENI
- Mirror target could be another ENI or Network Load Balancer (UDP - 4789)
- Mirror Filter – To capture only the traffic you are interesting in
  - Filter the traffic by specifying Protocols, Source/Destination port ranges, and CIDR Blocks
  - Define rules (numbered) to send the traffic to respective destination
- The traffic mirror source and the traffic mirror target (monitoring appliance) can be in the same VPC or they can be in a different VPC connected via intra-Region VPC peering or a transit gateway
- Source and Destination can be in different AWS accounts

# VPC DNS and DHCP

# DNS and DHCP in VPC

- When you launch EC2 instance in default VPC it receives

|                       |   |                   |   |
|-----------------------|---|-------------------|---|
| Instance:             | i-045ee5e966bef85d1 (EC2)   | Public DNS:       | ec2-13-232-197-112.ap-south-1.compute.amazonaws.com   |
| Description           | Status Checks   | Monitoring        | Tags  |
| Instance ID           | i-045ee5e966bef85d1   | Public DNS (IPv4) | ec2-13-232-197-112.ap-south-1.compute.amazonaws.com   |
| Instance state        | running   | IPv4 Public IP    | 13.232.197.112  |
| Instance type         | t2.micro  | IPv6 IPs          | -   |
| Finding               | Opt-in to AWS Compute Optimizer for recommendations. <a href="#">Learn more</a> | Elastic IPs       |   |
| Private DNS           | ip-10-10-0-211.ap-south-1.compute.internal                                      | Availability zone | ap-south-1a   |
| Private IPs           | 10.10.0.211   | Security groups   | <a href="#">SG-1</a> , <a href="#">view inbound rules</a> , <a href="#">view outgoing rules</a> |
| Secondary private IPs |   | Scheduled events  | No scheduled events   |
| VPC ID                | vpc-07a621b582810cd2d (demo-vpc)  | AMI ID            | amzn2-ami-hvm-2.0.20200617.0<br><a href="#">0732b62d210b000071</a>                              |

# DNS and DHCP in VPC

- Hostname

```
ec2-user@ip-10-10-0-211:~  
           _.-| | _ / )     Amazon Linux 2 AMI  
          _\ \_ |__| |  
  
https://aws.amazon.com/amazon-linux-2/  
13 package(s) needed for security, out of 32 available  
Run "sudo yum update" to apply all updates.  
[ec2-user@ip-10-10-0-211 ~]$ hostname  
ip-10-10-0-211.ap-south-1.compute.internal
```

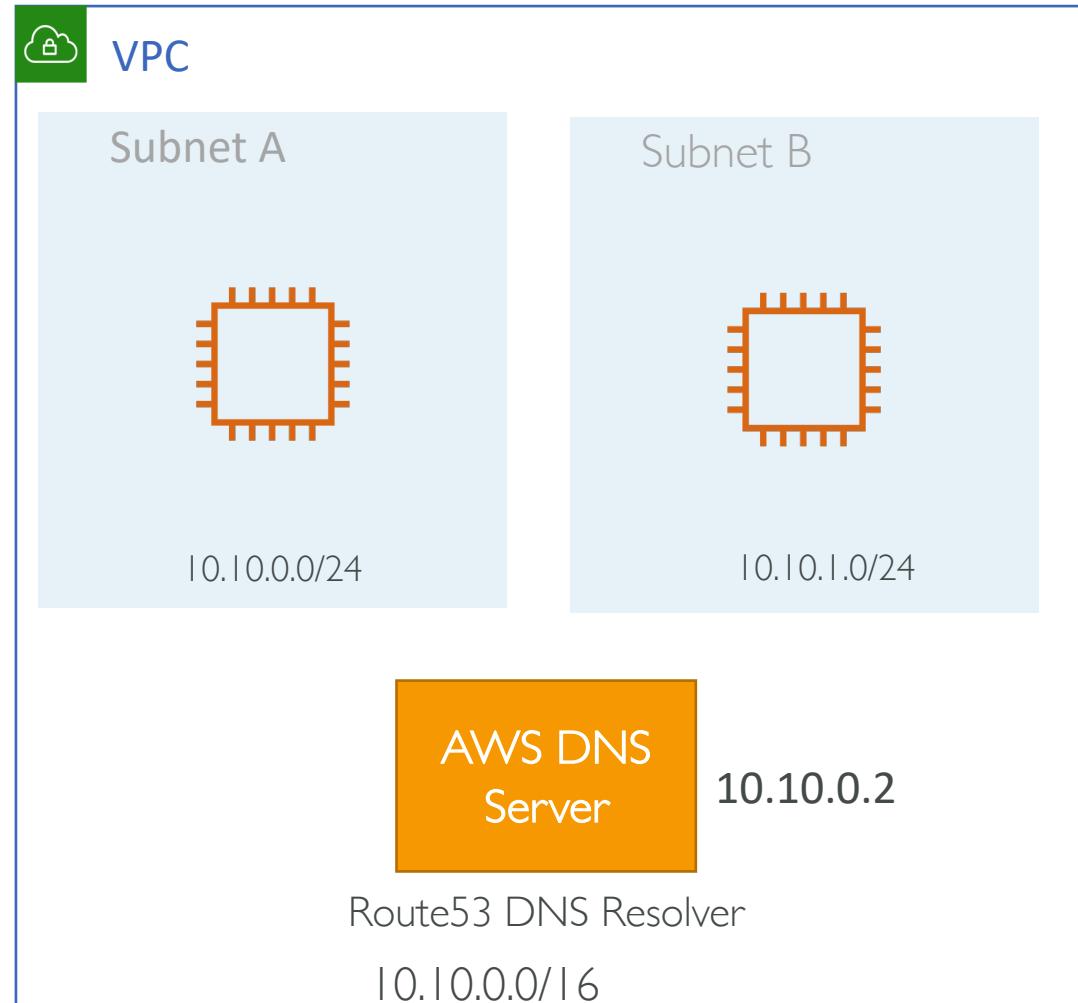
- /etc/resolv.conf

Amazon DNS Server runs at  
VPC CIDR + 2 IP address

```
[ec2-user@ip-10-10-0-211 ~]$ cat /etc/resolv.conf  
; generated by /usr/sbin/dhclient-script  
search ap-south-1.compute.internal  
options timeout:2 attempts:5  
nameserver 10.10.0.2
```

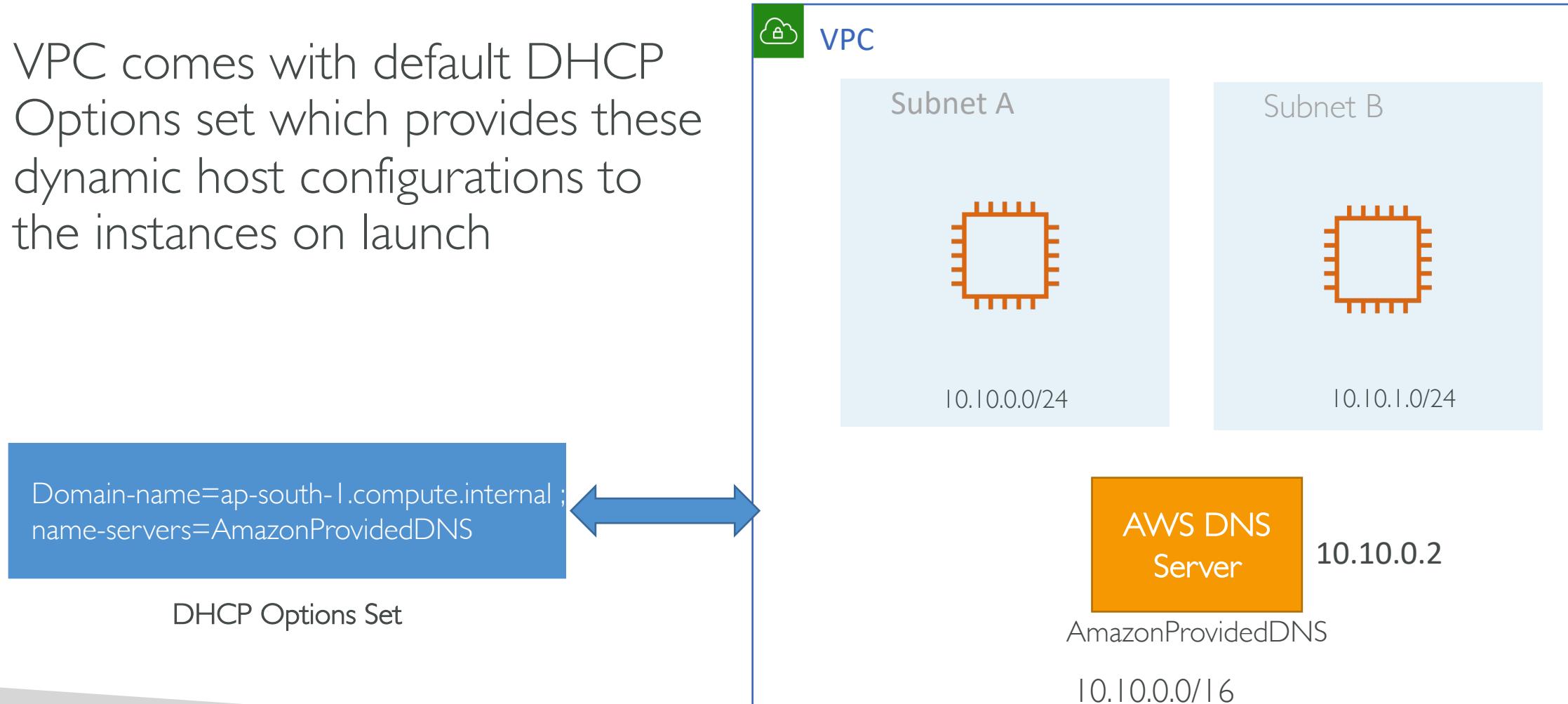
# Amazon DNS Server - Amazon Route 53 Resolver

- VPC comes with default DNS server also called as Route53 DNS Resolver
- Runs at VPC Base + 2 Address
- .2 is also reserved in every subnet
- Resolves DNS requests from Private and Public Route53 Hosted Zones & forwards other requests to Public DNS
- Only accessible from within VPC



# How VPC knows about default DNS Server - DHCP Options Set

- VPC comes with default DHCP Options set which provides these dynamic host configurations to the instances on launch



# DHCP Options Sets

The screenshot shows the AWS VPC DHCP Options Sets page. The left sidebar is titled "VIRTUAL PRIVATE CLOUD" and includes links for Your VPCs, Subnets, Route Tables, Internet Gateways, Egress Only Internet Gateways, and DHCP Options Sets (which is currently selected). The main content area has a header with "Create DHCP options set" and "Actions". A search bar says "Filter by tags and attributes or search by keyword". Below it is a table with columns: Name, DHCP options set ID, Options, and Owner. One row is shown: dopt-d35ea6b8, domain-name = ap-south-1.compute.internal; domain-name-servers = AmazonProvidedDNS, 523265502623. A red box highlights the "Options" column value. The bottom section shows the details for dopt-d35ea6b8, including the DHCP options set ID, owner (523265502623), and a "Tags" tab.

New VPC Experience [Learn more](#)

VPC Dashboard

Filter by VPC: [Select a VPC](#)

**VIRTUAL PRIVATE CLOUD**

Your VPCs

Subnets

Route Tables

Internet Gateways

Egress Only Internet Gateways

**DHCP Options Sets**

Elastic IPs

Managed Prefix Lists

Endpoints

Endpoint Services

NAT Gateways

Create DHCP options set Actions

Filter by tags and attributes or search by keyword

1 to 1 of 1

| Name | DHCP options set ID | Options   | Owner        |
|------|---------------------|---|--------------|
|      | dopt-d35ea6b8       | domain-name = ap-south-1.compute.internal; domain-name-servers = AmazonProvidedDNS; | 523265502623 |

DHCP Options Set: dopt-d35ea6b8

Details Tags

DHCP options set ID: dopt-d35ea6b8

Owner: 523265502623

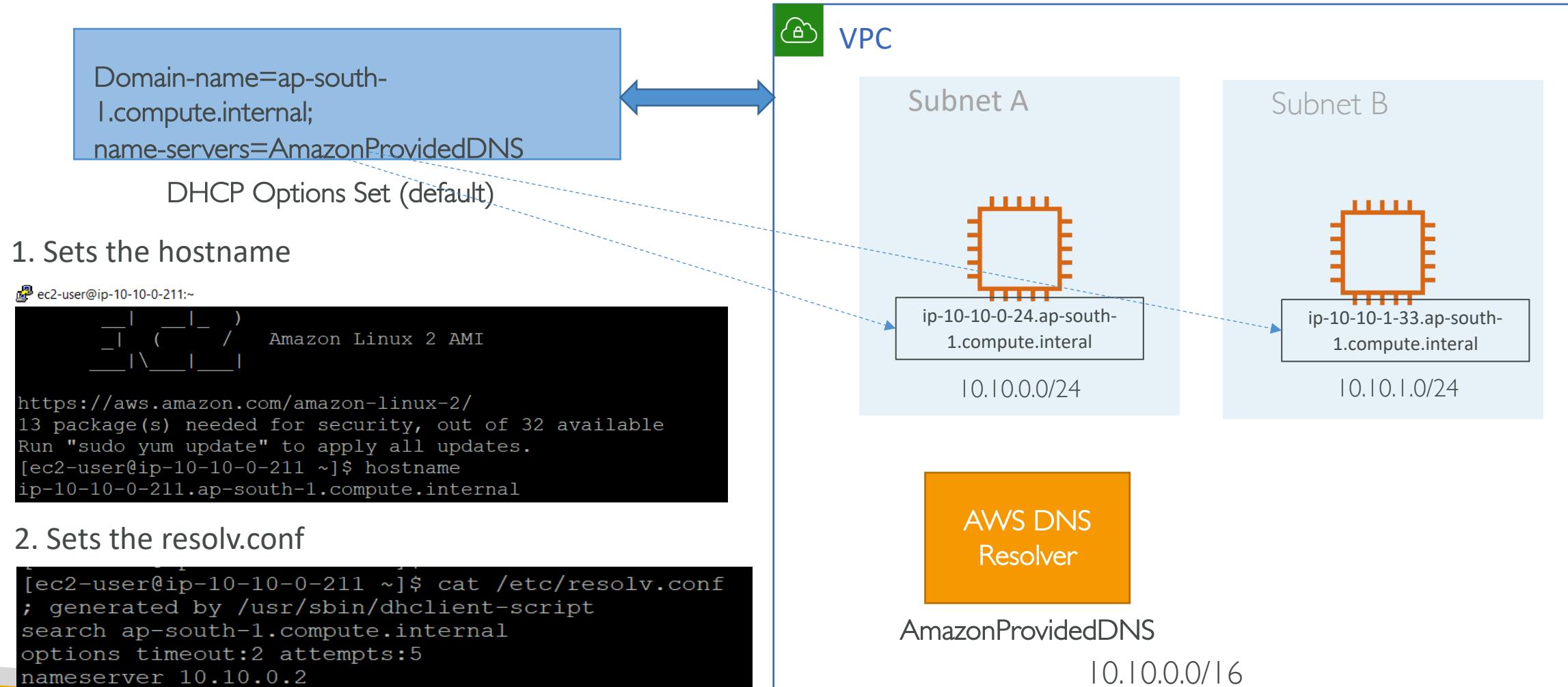
Options: domain-name = ap-south-1.compute.internal; domain-name-servers = AmazonProvidedDNS;

Feedback English (US) © 2008 - 2020, Amazon Internet Services Private Ltd. or its affiliates. All rights reserved. Privacy Policy Terms of Use

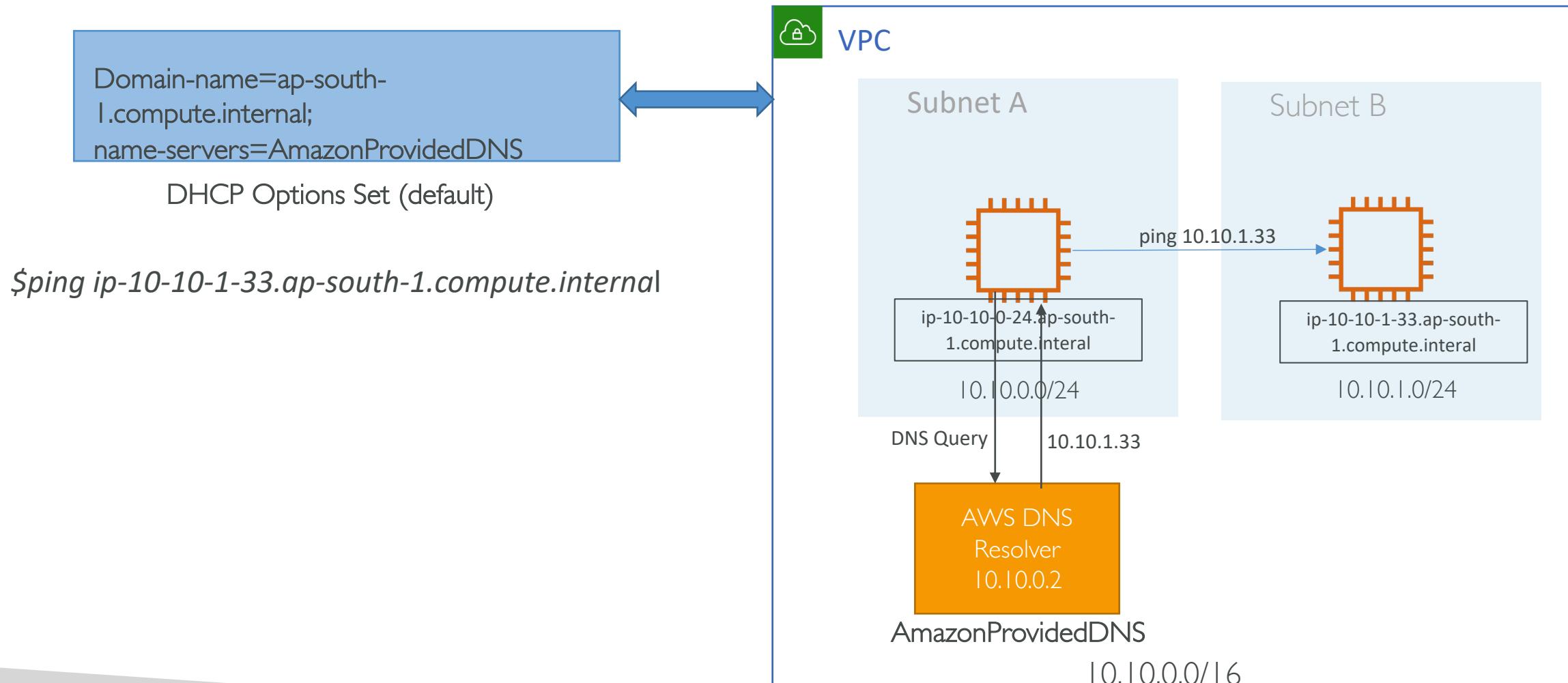
# DHCP Option Sets

- The options field of a Dynamic Host Configuration Protocol message contains the configuration parameters like domain name, domain name server, and the NetBIOS node type
- AWS automatically creates and associates a DHCP option set for your VPC upon creation and sets following parameters:
  - **domain-name-servers**: This defaults to **AmazonProvidedDNS**
  - **domain-name**: This defaults to the internal Amazon domain name for your region (e.g <ip>.ap-south-1.compute.internal)

# DHCP Options set – How it works



# DHCP Options set – How it works



# EC2 Private and Public DNS names

# AWS assigned domain names for EC2

- Internal DNS
  - ip-private-ipv4-address.ec2.internal (for the US-East-1 region)
  - ip-private-ipv4-address.region.compute.internal (for all other regions)
- External DNS (If instance has Public IP)
  - ec2-public-ipv4-address.compute-1.amazonaws.com (for the US-East-1 region)
  - ec2-public-ipv4-address.region.amazonaws.com (for other regions)

# DHCP Option sets - Attributes

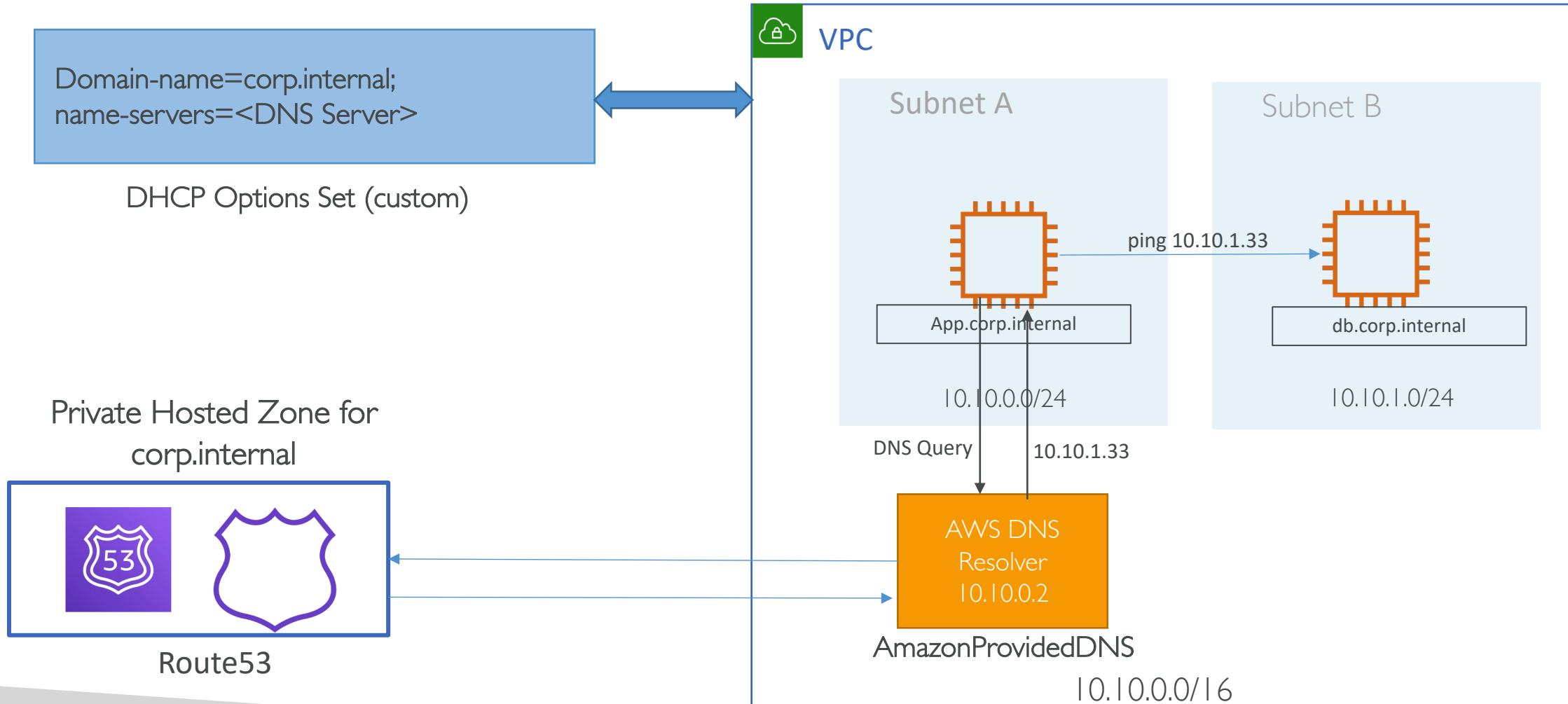
- **enableDnsSupport:** (= DNS Resolution setting)
  - Default True
  - Helps decide if DNS resolution is supported for the VPC
  - If True, queries the AWS DNS server at 169.254.169.253
- **enableDnsHostname:** (= DNS Hostname setting)
  - False by default for newly created VPC, True by default for Default VPC
  - Won't do anything unless enableDnsSupport=true
  - If True, Assign public hostname to EC2 instance if it has a public IP
- If you use custom DNS domain names in a private zone in Route 53, you must set both these attributes to true

# Demo

- When enableDnsHostname is not set
- When enableDnsHostname is set

# Custom Domain Name

# DHCP Options set – Custom domain name with Route53 Private hosted zone



# Demo – Custom DHCP options set

- Create custom DHCP Options set
- Create new VPC and attach
- Configure Route53 Private Hosted Zone

## Create DHCP options set

Dynamic Host Configuration Protocol (DHCP) provides a standard for passing configuration information to hosts on a TCP/IP network. parameters.

Name  i

### DHCP options (configuration parameters)

Specify at least one of the following configuration parameters:

Domain name  i

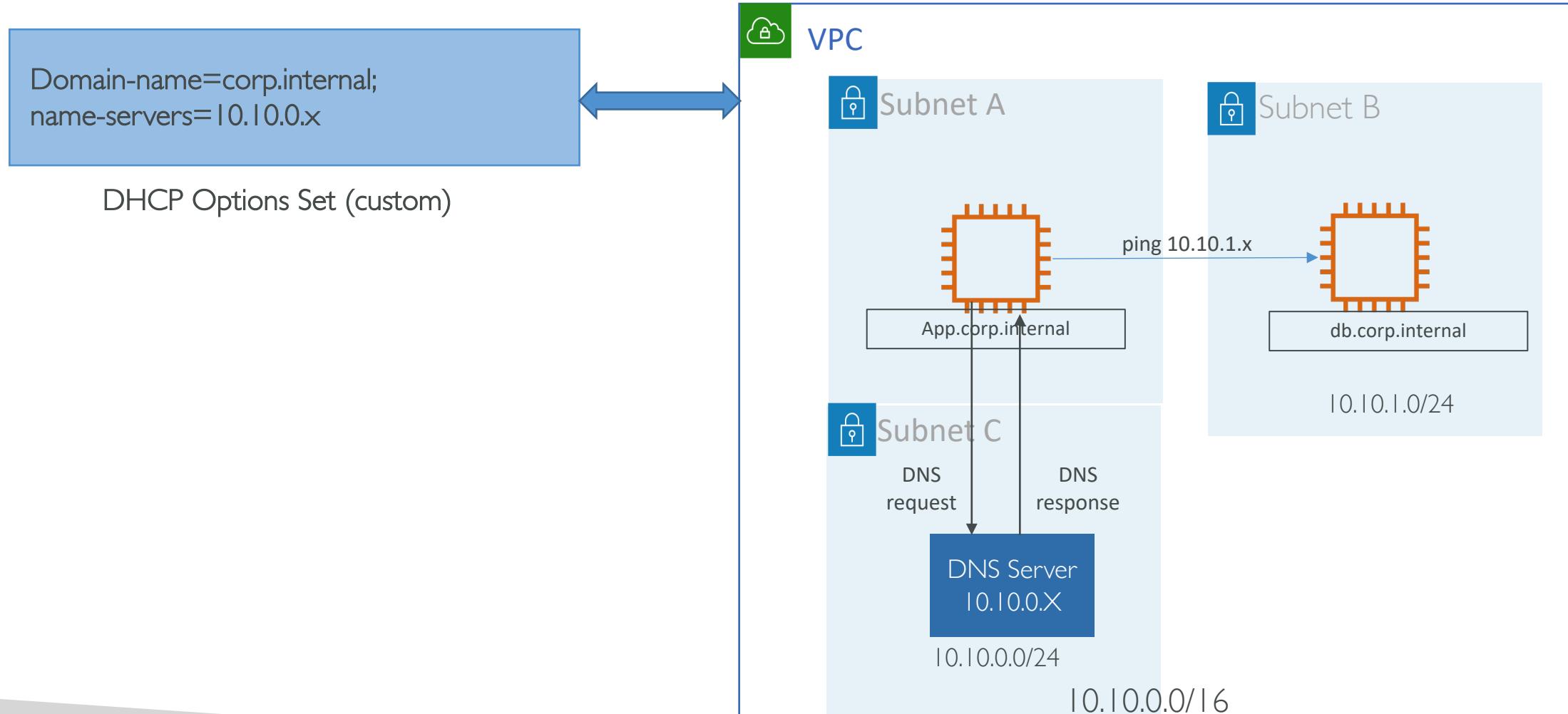
Domain name servers  i

NTP servers  i

NetBIOS name servers  i

NetBIOS node type  i

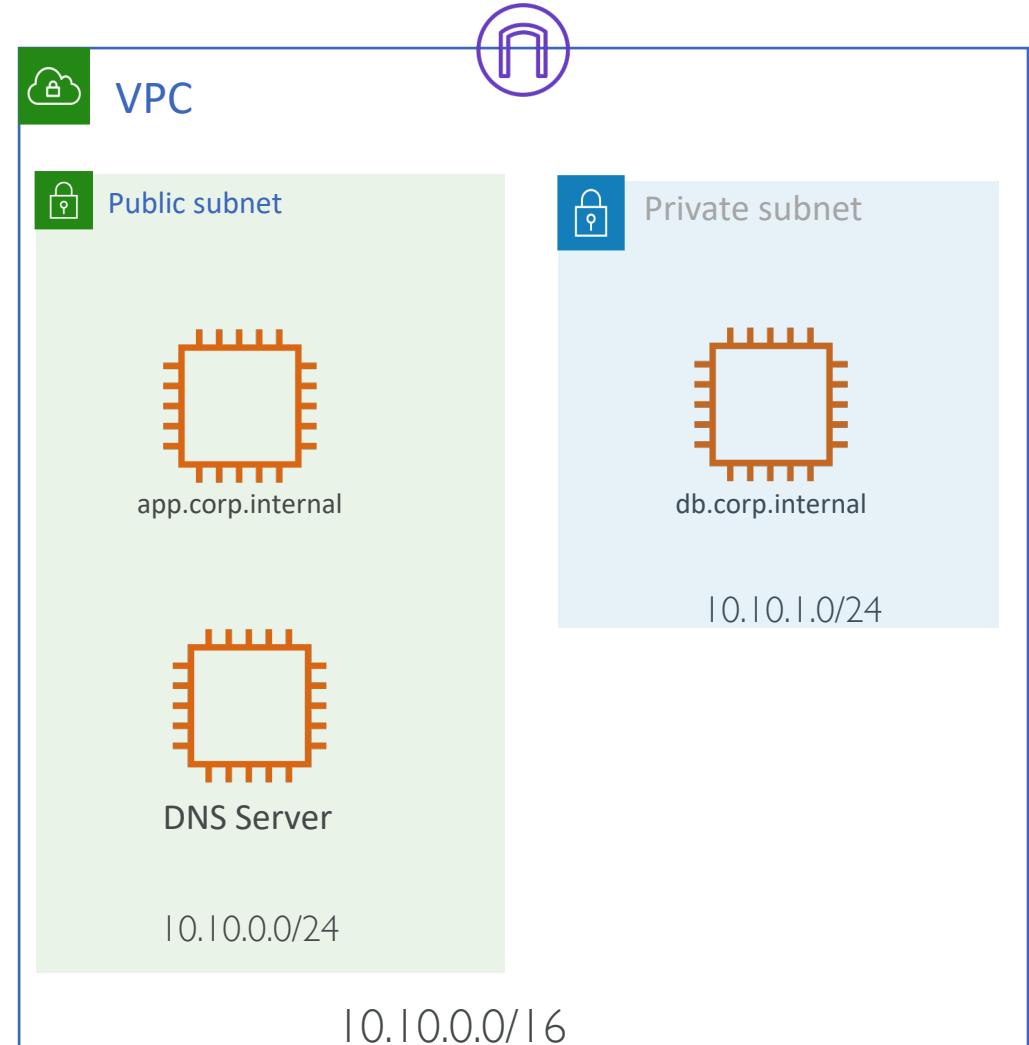
# Hands on: DHCP Options set – Custom domain name with custom DNS server



# Step 1 – Setup a VPC and launch instances

- Create a VPC with public and private subnets
- Launch DNS server ec2 instance:
  - Security group to allow UDP 53 from VPC CIDR, SSH (22)
- Launch an app server & db server ec2 instances:
  - Security group to allow SSH (22), ICMP IPv4 All (ping)

Already created to save time



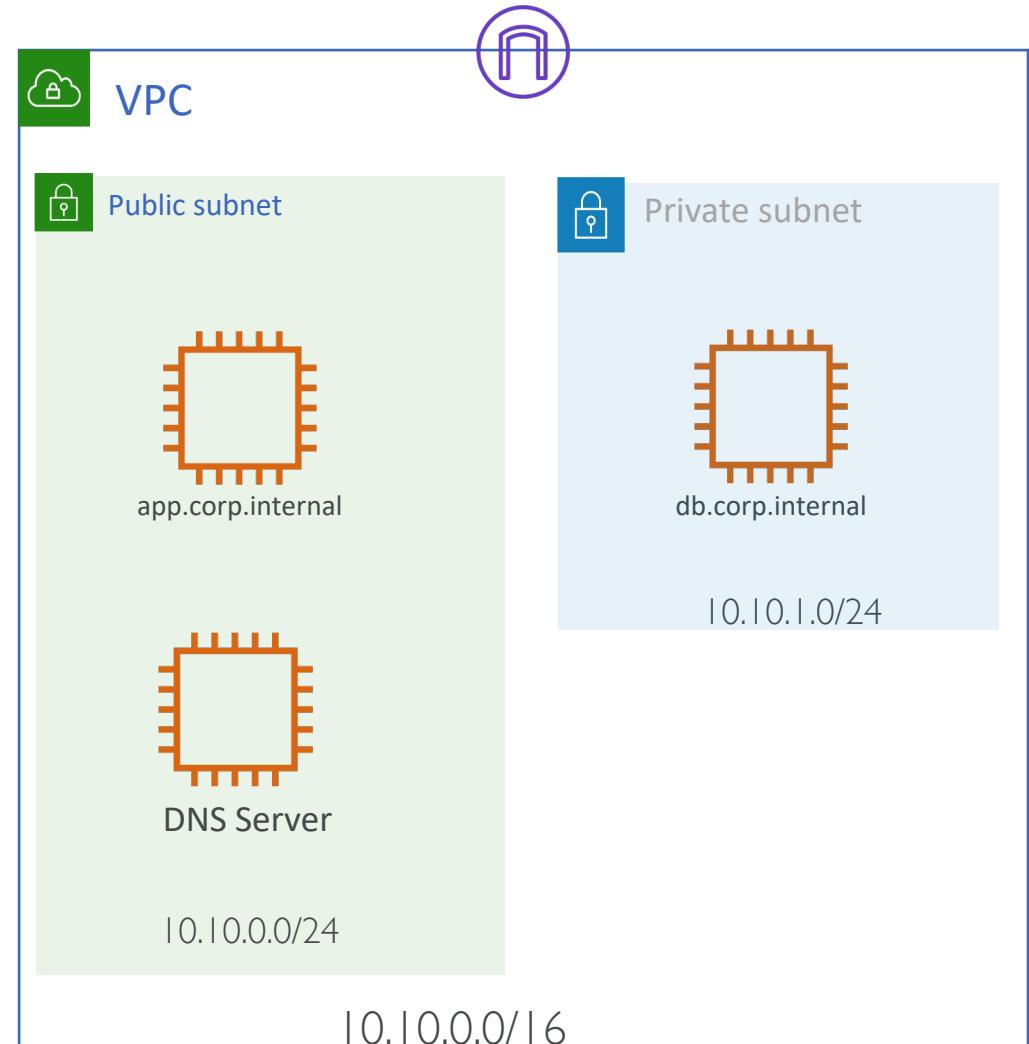
# Step 2a – Setup DNS server

1. Login to DNS server
2. Install DNS server packages

```
$sudo su
```

```
$yum update -y
```

```
$yum install bind bind-utils -y
```

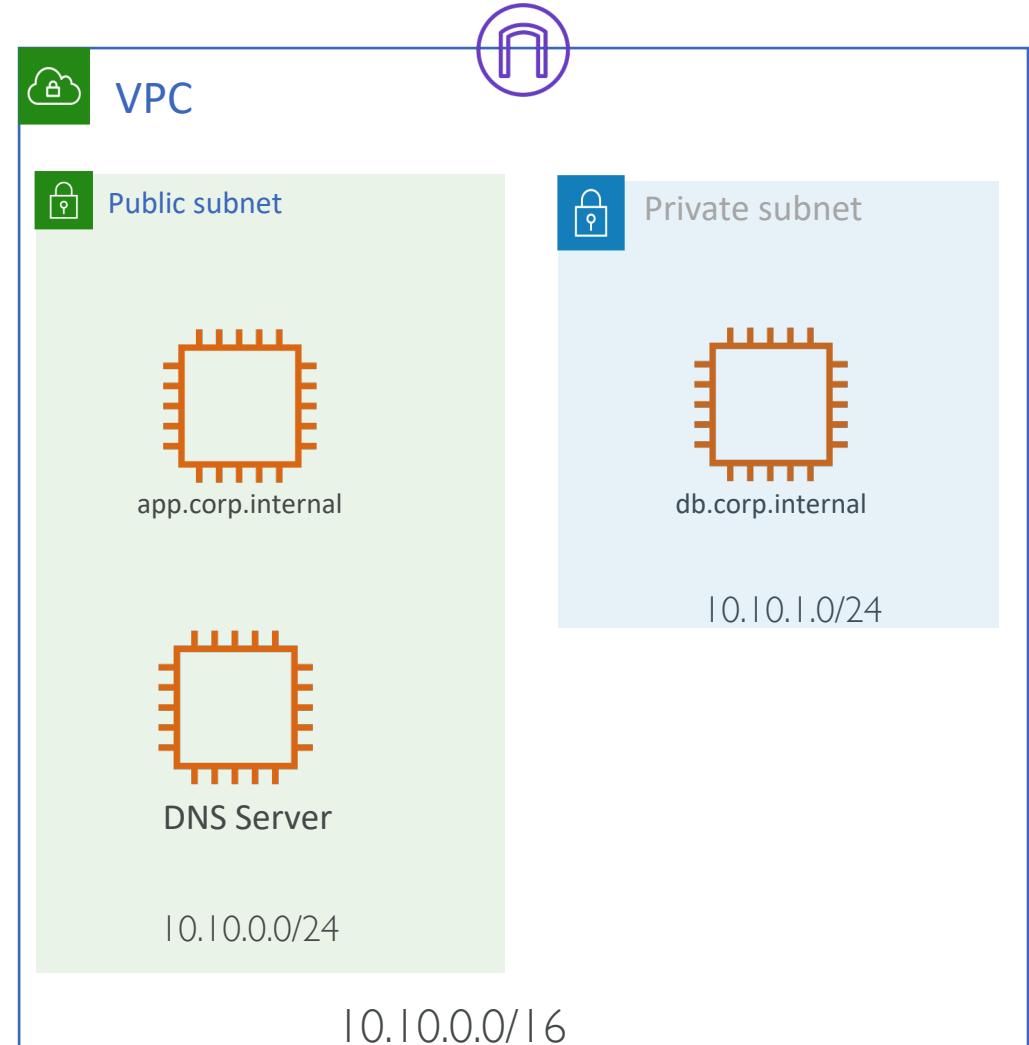


# Step 2b – Configure DNS server

3. Create file /var/named/corp.internal.zone [Replace X]

```
$TTL 86400
@ IN SOA ns1.corp.internal. root.corp.internal. (
    2013042201 ;Serial
    3600 ;Refresh
    1800 ;Retry
    604800 ;Expire
    86400 ;Minimum TTL
)
; Specify our two nameservers
IN NS dnsA.corp.internal.
IN NS dnsB.corp.internal.
; Resolve nameserver hostnames to IP, replace with your two droplet IP addresses.
dnsA IN A 1.1.1.1
dnsB IN A 8.8.8.8

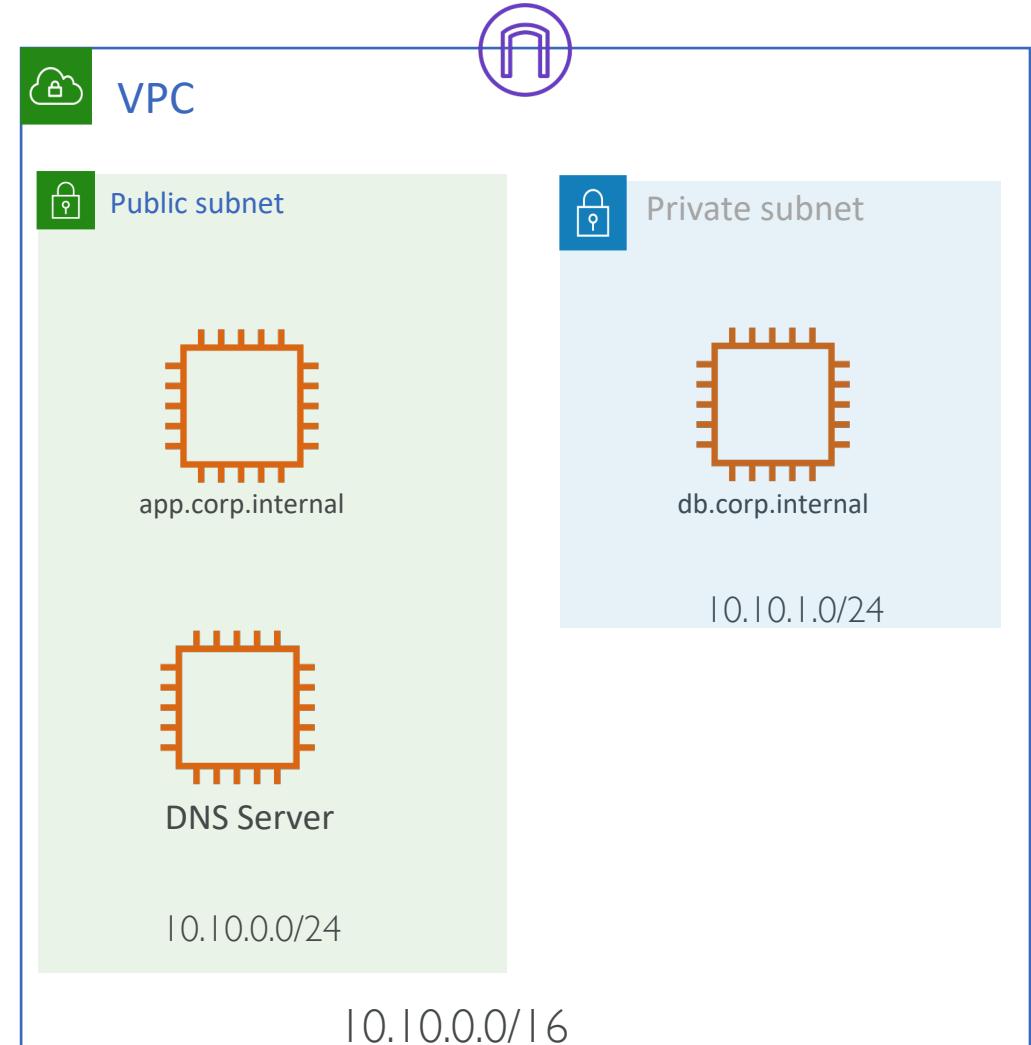
; Define hostname -> IP pairs which you wish to resolve
@ IN A 10.10.0.x
app IN A 10.10.0.x
db IN A 10.10.1.x
```



# Step 2c – Configure DNS server

4. Create file /etc/named.conf [Replace X with your DNS server IP]

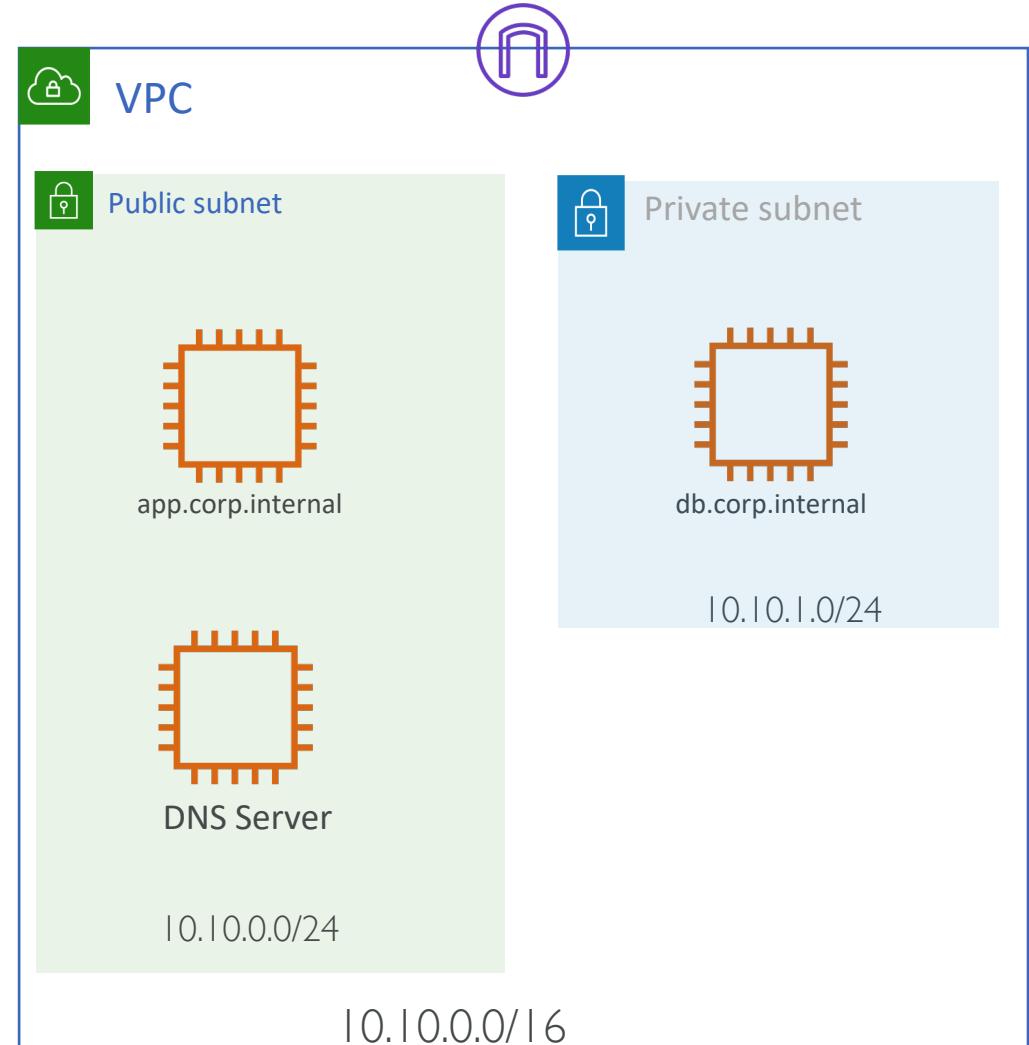
```
options {  
    directory "/var/named";  
    dump-file "/var/named/data/cache_dump.db";  
    statistics-file "/var/named/data/named_stats.txt";  
    memstatistics-file "/var/named/data/named_mem_stats.txt";  
    allow-query { any; };  
    allow-transfer { localhost; 10.10.0.X; };  
    recursion yes;  
    forward first;  
    forwarders {  
        10.10.0.2;  
    };  
    dnssec-enable yes;  
    dnssec-validation yes;  
    dnssec-lookaside auto;  
    /* Path to ISC DLV key */  
    bindkeys-file "/etc/named.iscdlv.key";  
    managed-keys-directory "/var/named/dynamic";  
};  
zone "corp.internal" IN {  
    type master;  
    file "corp.internal.zone";  
    allow-update { none; };  
};
```



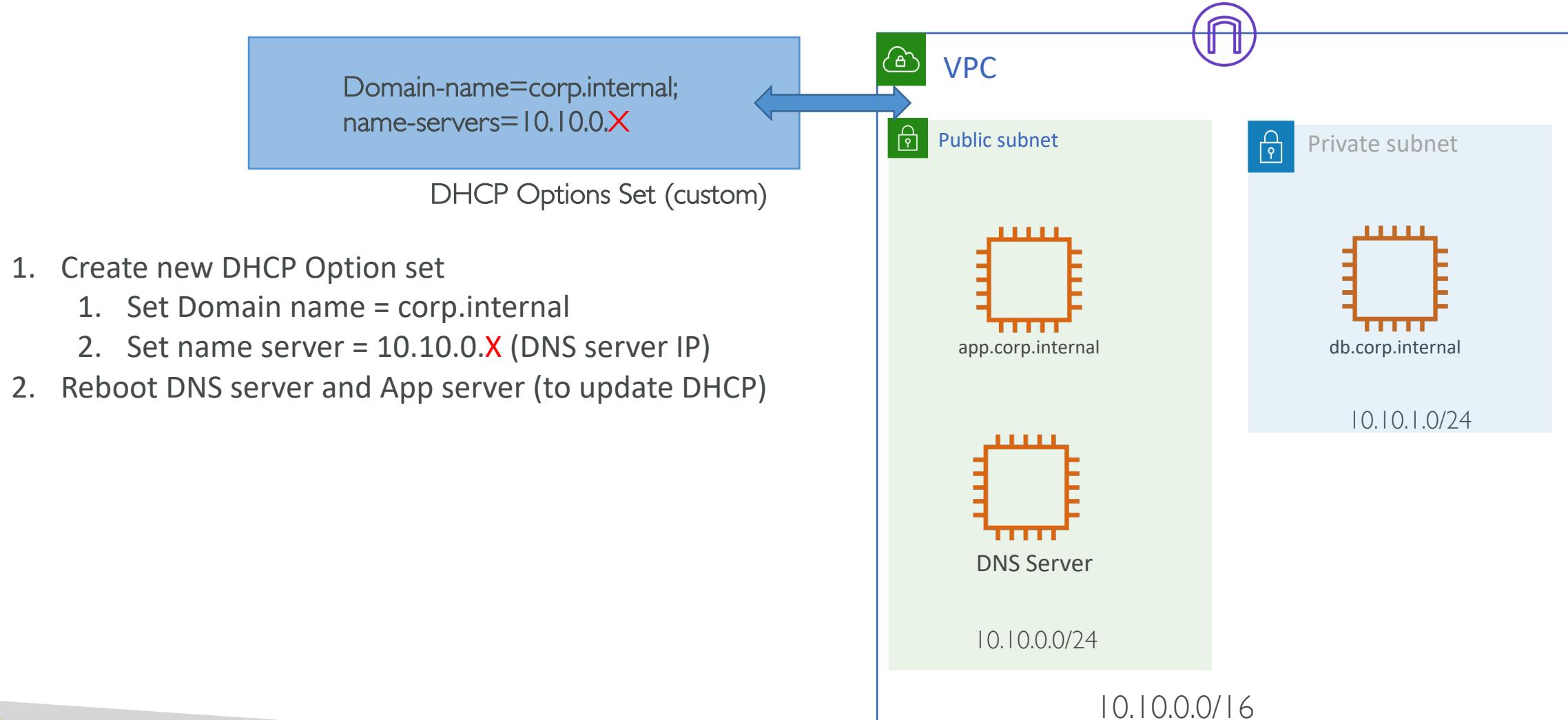
# Step 2d – Configure DNS server

## 5. Restart **named** service

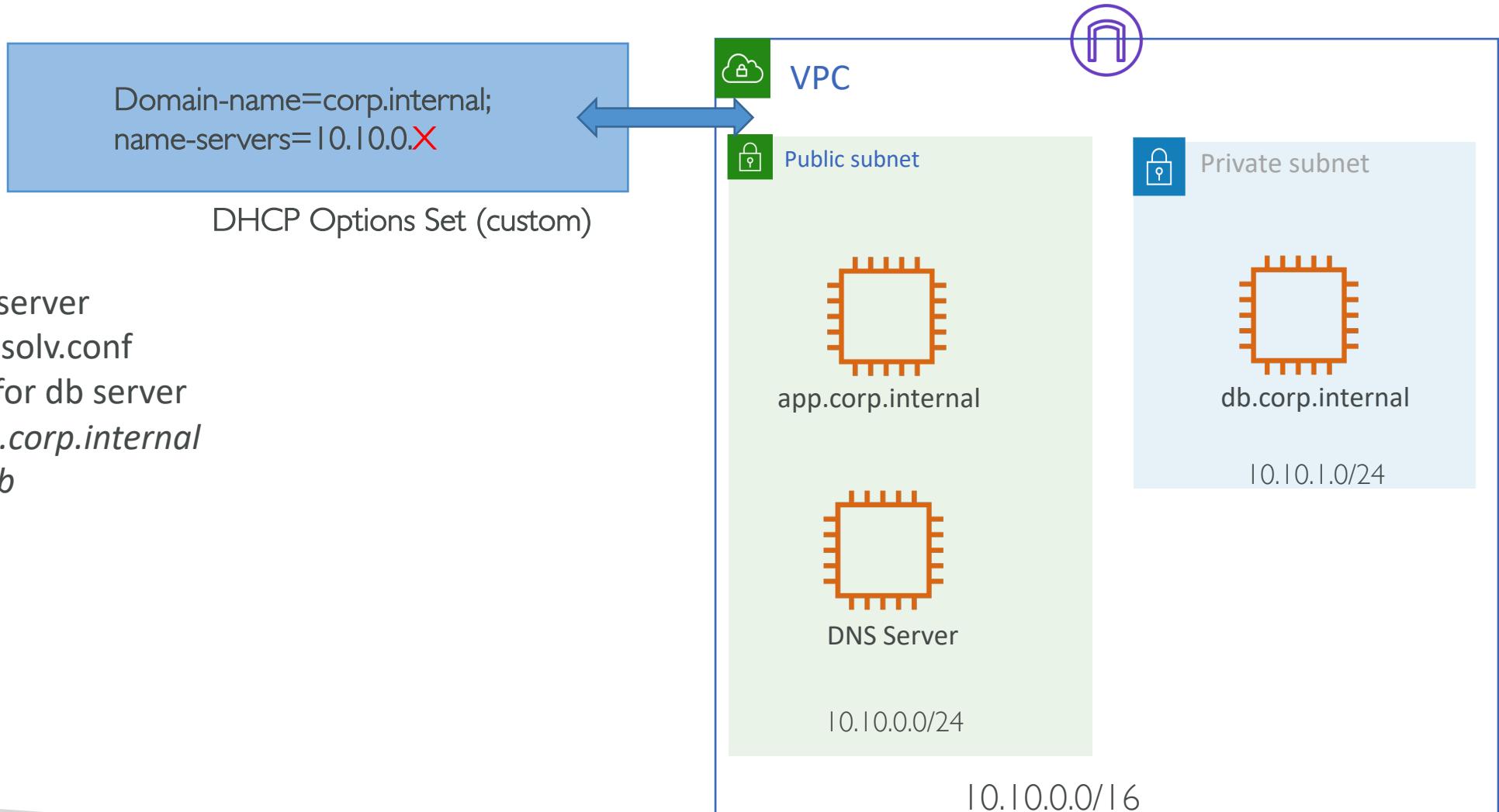
```
$service named restart  
$chkconfig named on
```



# Step 3 – Create DHCP Option set



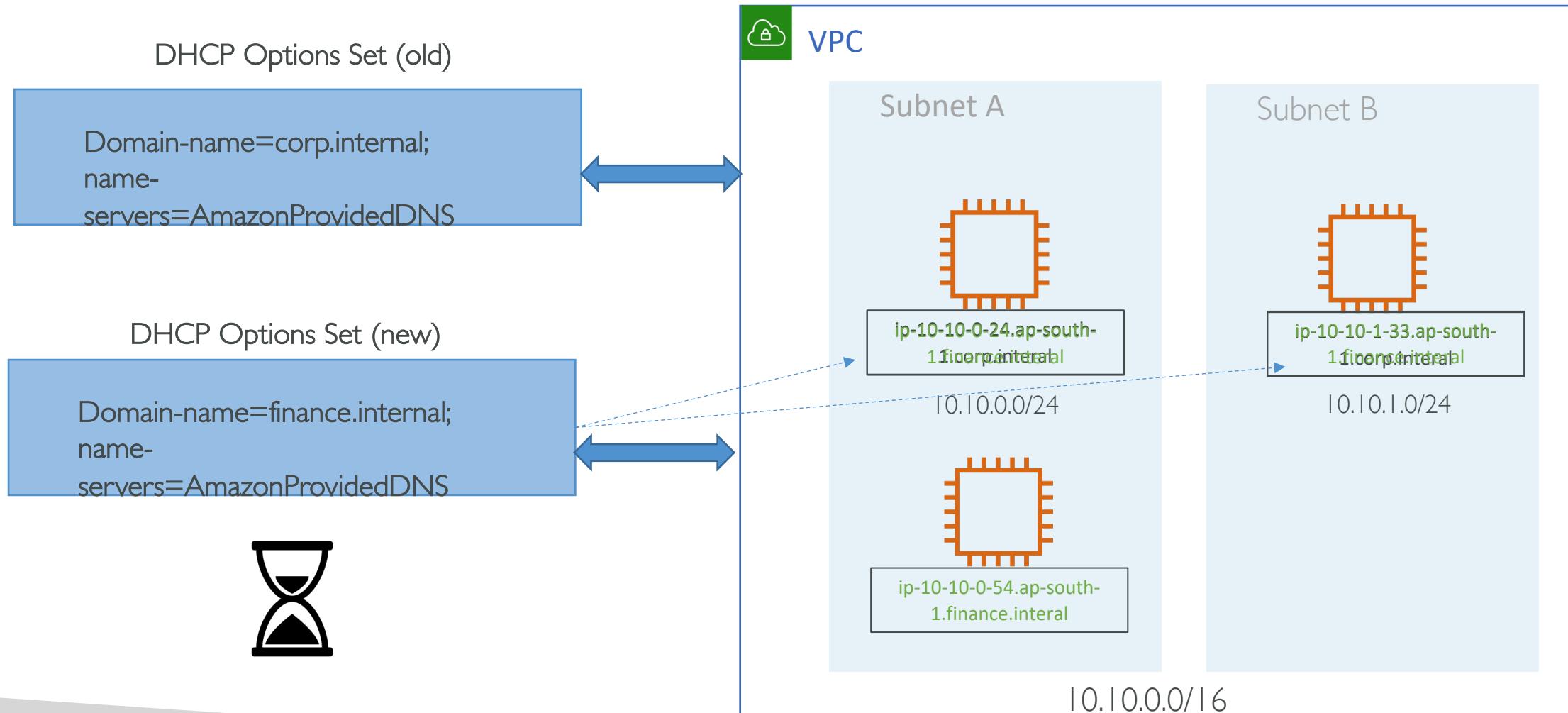
# Step 4 – Verify custom domain names



1. Login to App server
  2. Check /etc/resolv.conf
  3. Resolve DNS for db server
- \$nslookup db.corp.internal  
\$nslookup db  
\$ping db

# More about DHCP Options set

# Attaching VPC a new DHCP Options set



# DHCP Options Sets – good to know

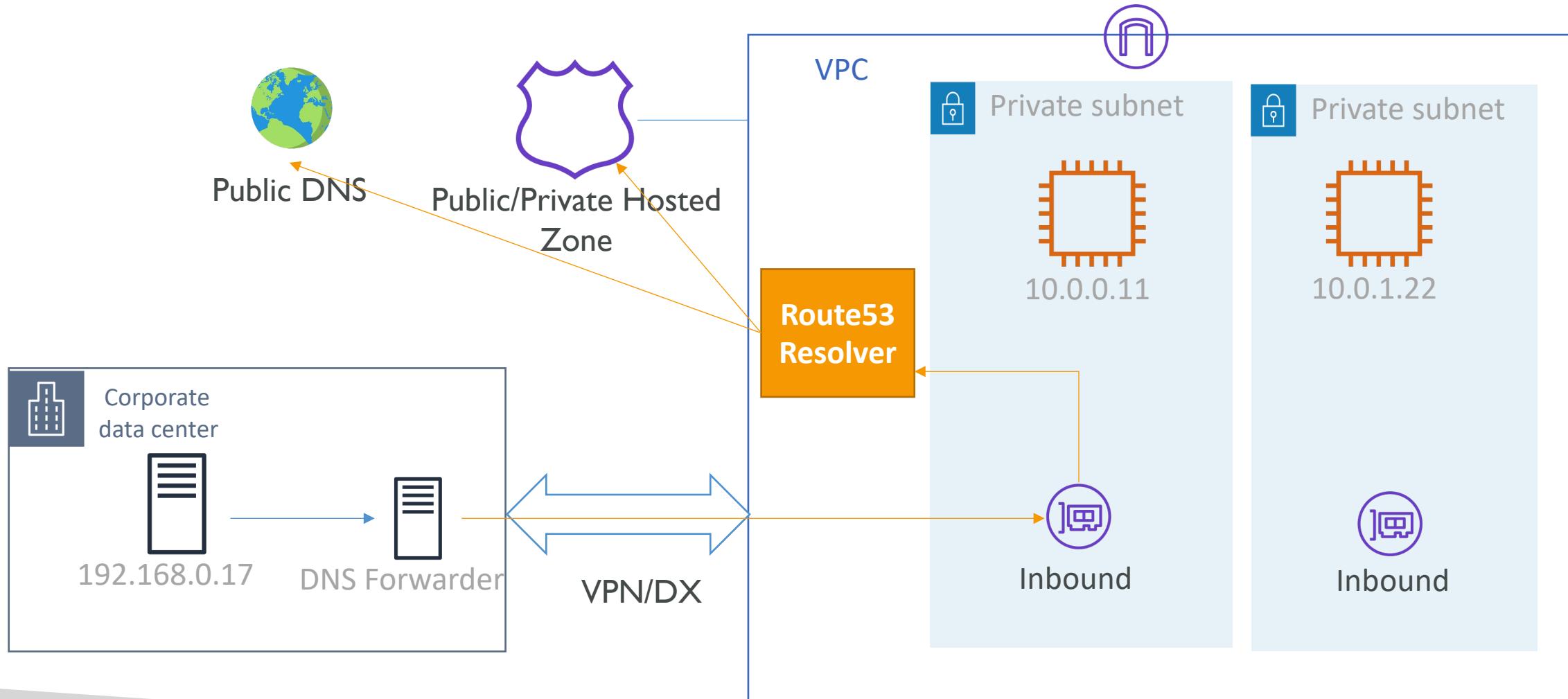
- Once created, you **can not** modify DHCP Options set however you can create new DHCP Options set and associate it with VPC
- You can only associate a single DHCP Options set with a VPC
- VPC can also be setup with No DHCP Options set
- After DHCP Option set is associated with VPC, the instances automatically use new option set, but this may take few hours
- You can also refresh the DHCP option parameters using an operating system command:
  - Example Linux command: `$sudo dhclient -r eth0`

# Hybrid DNS (introduction)

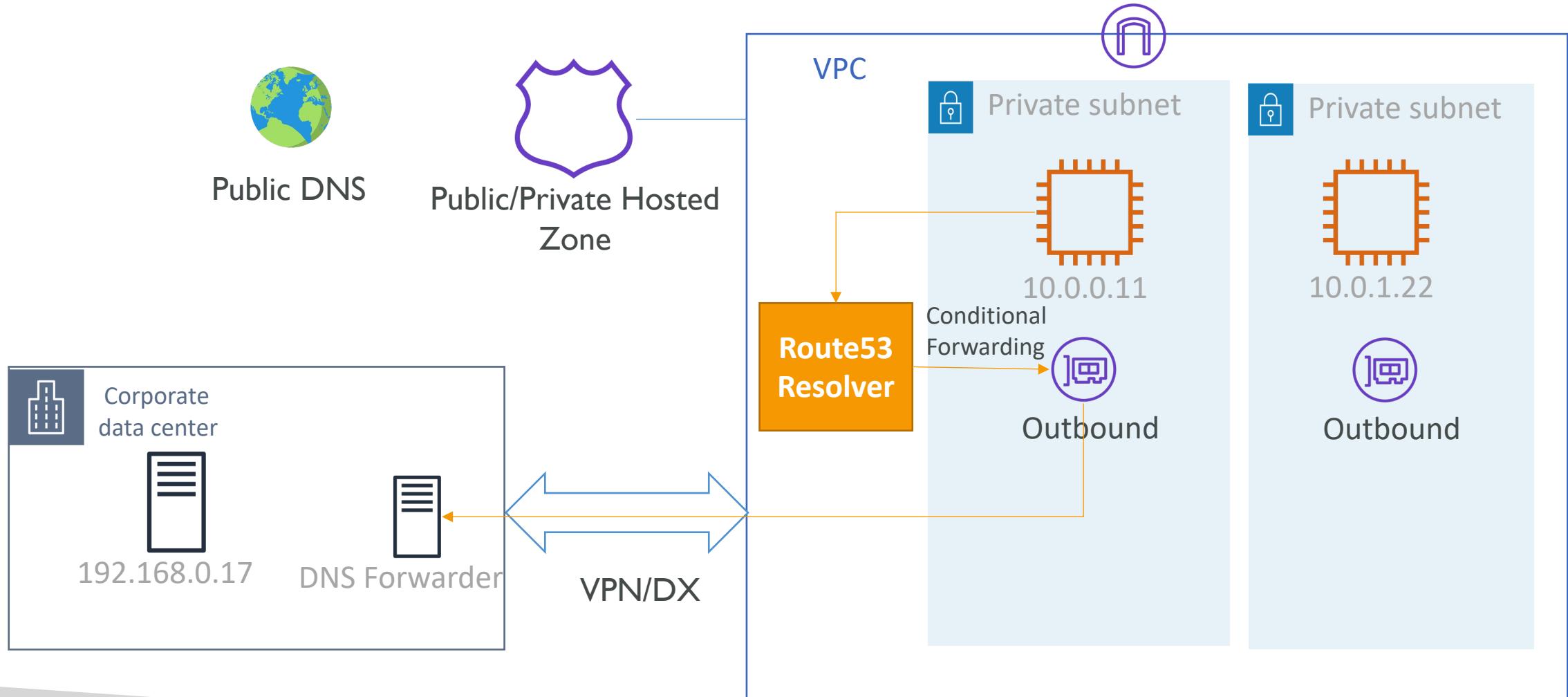
# Route53 Resolver Endpoint

- Officially named the .2 DNS resolver to Route 53 Resolver
- Provides Inbound & Outbound Route53 resolver endpoint
- Provisions ENI in the VPC which are accessible over VPN or DX
- Inbound -> On-premise forwards DNS request to R53 Resolver
- Outbound -> Conditional forwarders for AWS to On-premise

# Route53 Resolver Endpoint - Inbound



# Route53 Resolver Endpoint - Outbound



# Summary

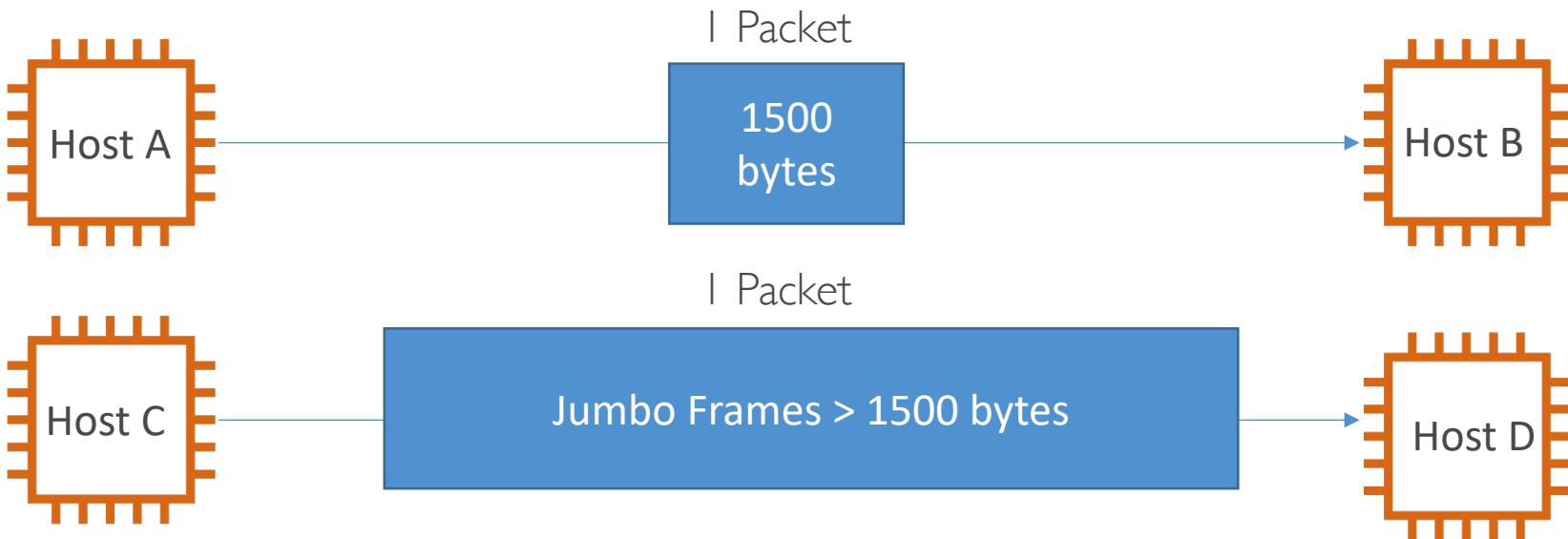
- VPC has a default DNS server AWSProvidedDNS
- VPC DNS settings can be changed using DHCP Options set
- AWSProvidedDNS can resolve the DNS from Route53 Private Hosted Zone
- For hybrid DNS resolution between VPC and on-premises network, use Route53 Resolver endpoints.
- DHCP Option set can not be edited. Create new one and associate it with VPC and you can have only one DHCP option set associated at a time.
- For hostname resolution, we should enable both `enableDnsSupport` and `enableDnsHostname`
- AWS Provided DNS server runs at VPC base + 2 IP address. You can also query DNS server at this IP or 169.254.169.253 virtual IP within VPC

# VPC Network Performance and Optimization

# Network Performance - Basics

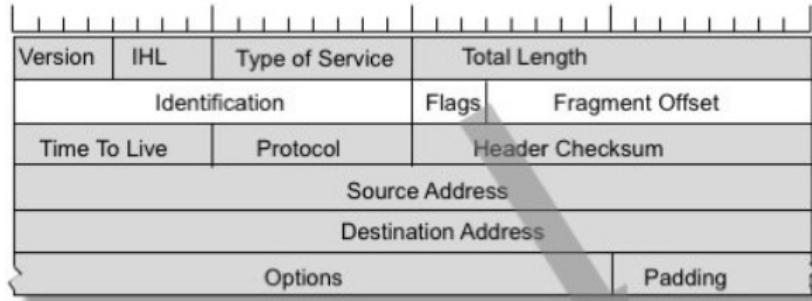
- Bandwidth – Maximum rate of transfer over the network
- Latency – Delay between two points in a network
  - Delays include propagation delays for signals to travel across medium
  - Also includes the processing delays by network devices
- Jitter – Variation in inter-packet delays.
- Throughput – Rate of successful data transfer (measured in bits per sec)
  - Bandwidth, Latency and Packet loss directly affects the throughput
- Packet Per Second (PPS) – How many packets processed per seconds
- Maximum Transmission Unit (MTU) – Largest packet that can be sent over the network

# MTU



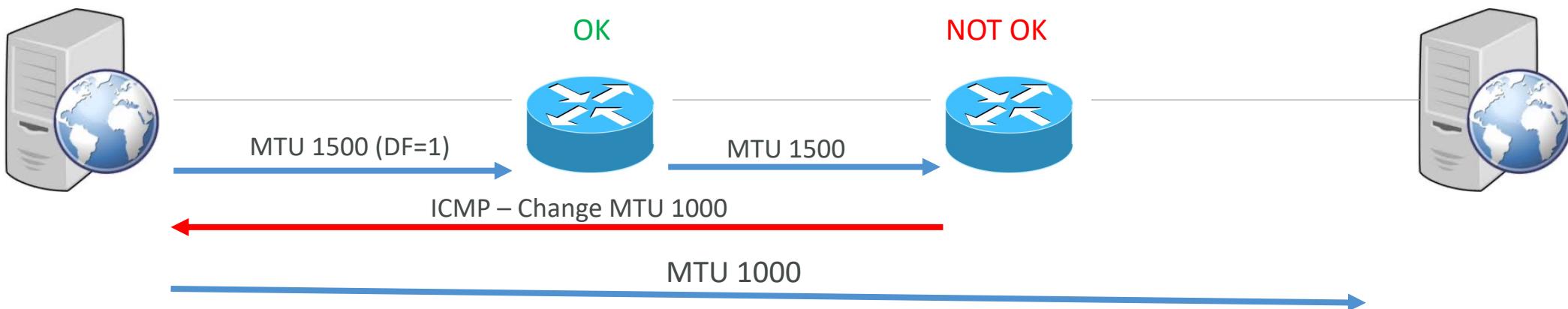
- Most of the networks support MTU of 1500 bytes
- Jumbo Frames packets are larger than 1500 bytes up to 9001 bytes
- Benefits of using Jumbo frames
  - Less packets
  - More throughput
  - Increasing the MTU increases the throughput when you can't increase the Packet Per Second (PPS)

# Path MTU Discovery example



Flags: bit 0 – Reserved  
bit 1 - Don't Fragment  
bit 2 – More Fragments

For MTU Path Discovery  
ICMP must be allowed

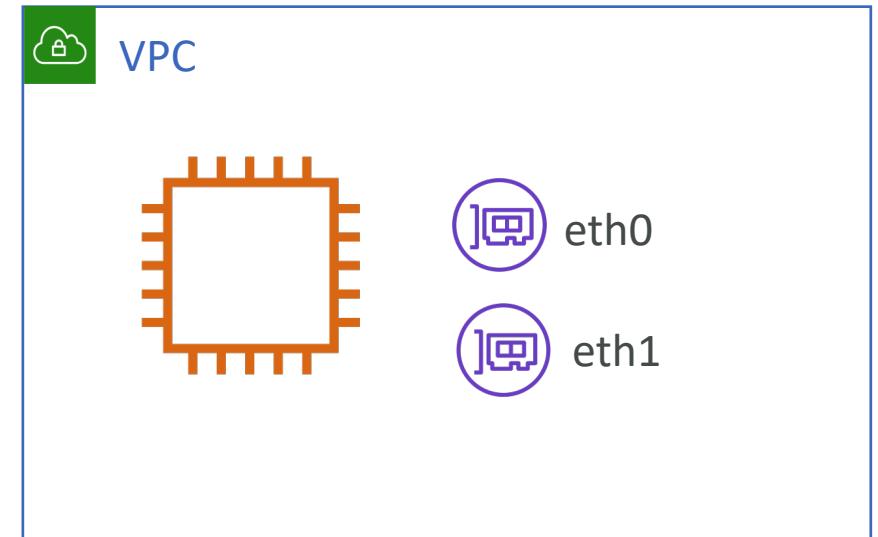


# Jumbo Frames

- 9001 MTU
- Jumbo frames are enabled in a VPC by default
- AWS supports Jumbo frame within VPC however traffic leaving VPC over IGW or inter region VPC peering does not support Jumbo frames (Limited to 1500 byte)
- Jumbo frames are also supported between VPC and on-premises network using AWS Direct Connect.
- Using Jumbo frames inside EC2 cluster placement groups provides maximum network throughput
- Jumbo frames should be used with caution for traffic leaving the VPC. If packets are over 1500 bytes, they are fragmented, or they are dropped if the Don't Fragment flag is set in the IP header.

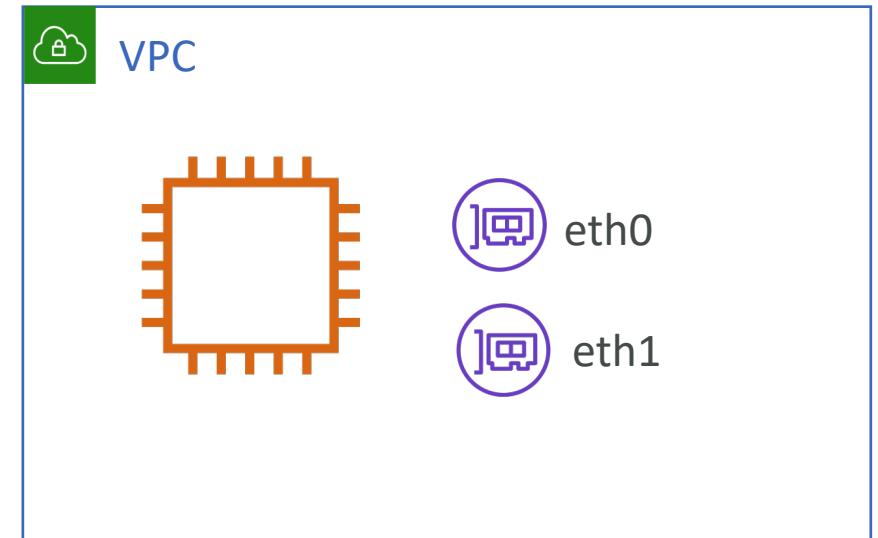
# Defining MTU on EC2 instances

- MTU also depends on Instance Type
- Defined at the ENI level
- You can check the path MTU between your device and target endpoint using tracepath command
  - *tracepath amazon.com*
- To check the MTU on your interface
  - *ip link show eth0*
- To set MTU value on Linux
  - *sudo ip link set dev eth0 mtu 900*



# Demo – MTU for EC2

1. Launch 2 EC2 instances in Public Subnet and connect to any one
2. Check MTU using EC2 Public IP: \$tracepath <Public IP>
3. Check MTU using EC2 Private IP: \$tracepath <Private IP>
4. Check MTU on EC2 interface  
`$ip link show eth0`



# MTU

- Within AWS:
  - Within VPC : Supports Jumbo frames (9001 bytes)
  - Over the VPC Endpoint : MTU 8500 bytes
  - Over the Internet Gateway : MTU 1500 bytes
  - Intra region VPC Peering: MTU 9001 bytes
  - Inter region VPC Peering : MTU 1500 bytes
- On-premise network:
  - Over the VPN using VGW : MTU 1500 bytes
  - Over the VPN via Transit Gateway : MTU 1500 for traffic for Site to Site VPN
  - Over the DirectConnect (DX) : Supports Jumbo frames (9001 bytes)
  - Over the DX via Transit Gateway : MTU 8500 for VPC attachments connected over the Direct Connect

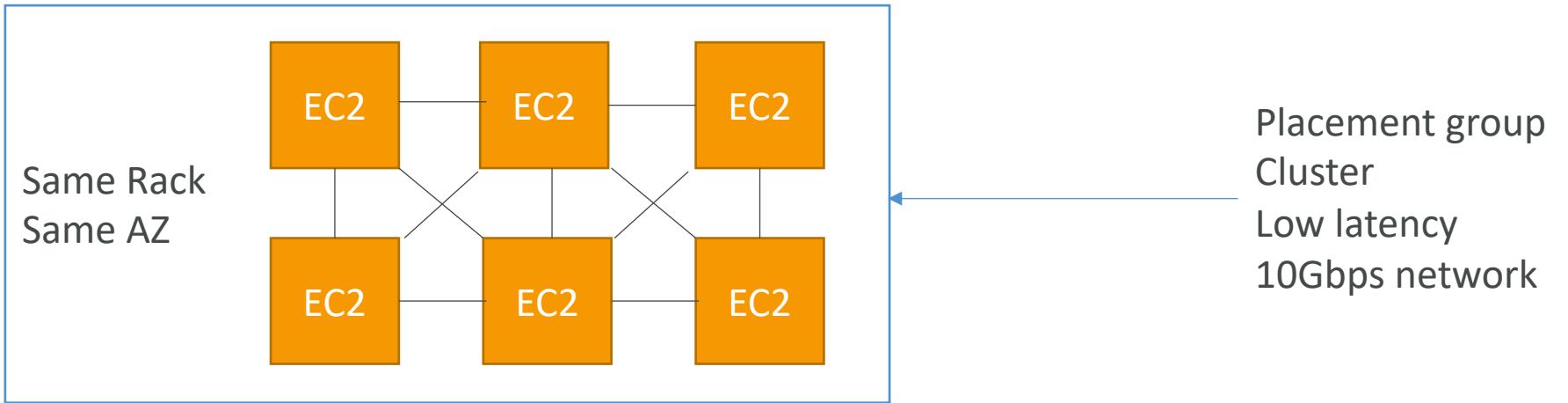
# Optimizing Network Performance

# EC2 Network optimization

- Cluster Placement Groups
- EBS Optimized Instances
- Enhanced Networking

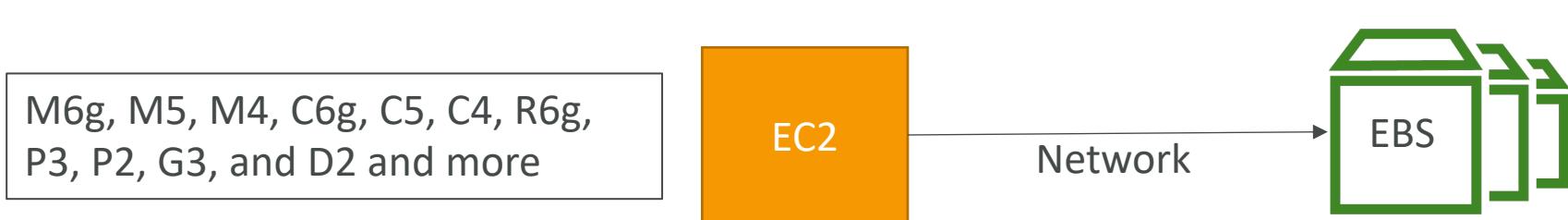
# Placement Groups – Cluster

- A logical grouping of instances within single Availability zone
- Ideal for distributed applications that require low latency like HPC



# EBS Optimized Instances

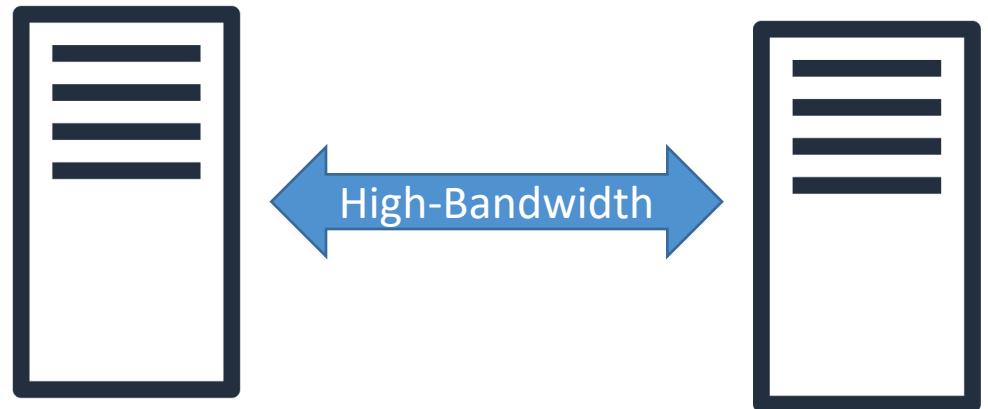
- EBS is a network drive (i.e. not a physical drive)
  - It uses the network to communicate the instance, which means there might be a bit of latency
- Hence, EBS input and output affect network performance
- **Amazon EBS-optimized instances** deliver dedicated throughput between Amazon EC2 and Amazon EBS
- This minimizes contention between Amazon EBS Input/Output (I/O) and other traffic from Amazon EC2 instance



# Enhanced Networking

# Enhanced Networking – What is it?

- Over 1M PPS performance
- Reduces instance-to-instance latencies
- SR-IOV with PCI passthrough, to get the hypervisor out of the way and for consistent performance
- Enabled using intel ixgbevf driver or Elastic Network Adapter (ENA)



# SR-IOV and PCI for Enhanced Networking

- SR-IOV and PCI passthrough are methods of device virtualization that provide higher I/O performance and lower CPU utilization
- SR-IOV allows a single physical NIC to present itself as multiple vNICs
- PCI passthrough enables PCI devices such as ENI to appear as if they are physically attached to the guest operating system bypassing hypervisor
- Ultimately in combination this allows low latency, high rate data transfer (>1 M PPS)

# Enhanced Networking pre-requisites

- Depending on Instance Type, Enhanced Networking can be enabled using one of the following Network drivers
  - Option 1: Intel 82599 VF up to 10 Gbps (VF uses ixgbevf driver)
  - Option 2: Elastic Network Adapter (ENA) up to 100 Gbps
- The eligible EC2 instance families support **either of** the above two drivers

## Remember:

For Enhanced Networking, it requires support from both EC2 operating system (AMI) and Instance Type that is flagged for Enhanced Networking

# Supported Instance types

- Instances supporting Elastic Network Adapter (ENA) for speed upto 100 Gbps
  - AI, C5, C5a, C5d, C5n, C6g, F1, G3, G4, H1, I3, I3en etc
- Instances supporting Intel 82599 Virtual Function (VF) interface for speed upto 10 Gbps
  - C3, C4, D2, I2, M4 (excluding m4.16xlarge), and R3 etc

# Verifying enhanced networking

Intel VF ixgbevf

```
$ ethtool -i eth0  
driver: ixgbevf
```

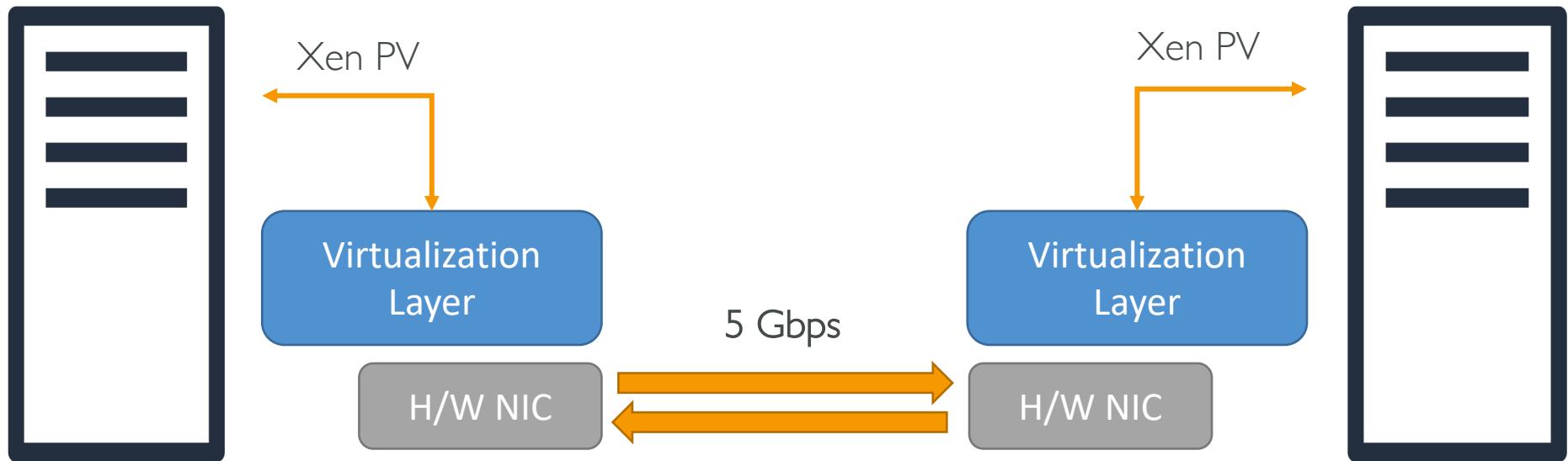
C3, C4, D2, I2, R3, M4 (not  
m4.16xlarge)

ENA

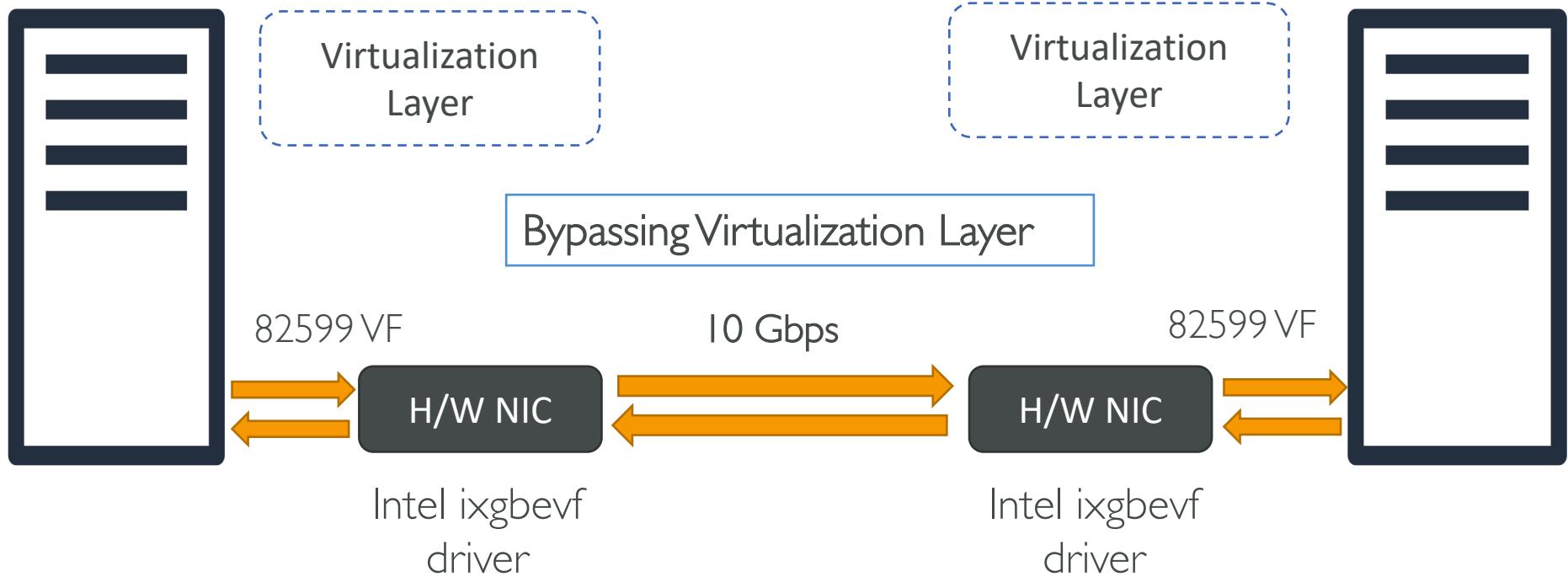
```
$ ethtool -i eth0  
driver: ena
```

C5, F1, G3, P2, P3, R4, I3, X1,  
X1e, m4.16xlarge

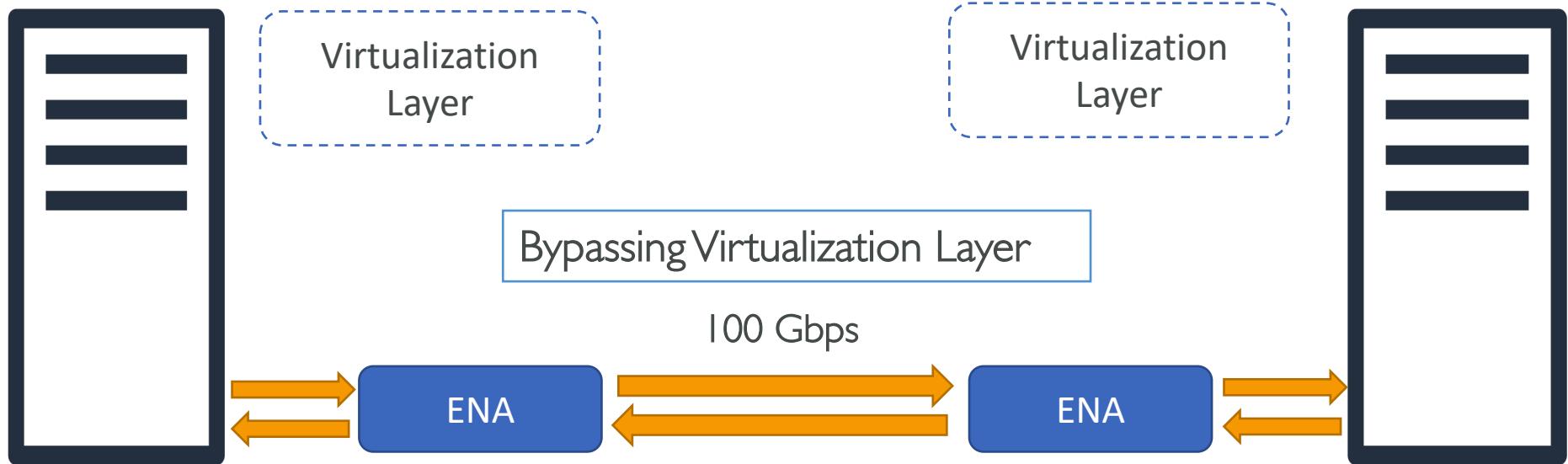
# EC2 Networking - default



# EC2 Enhanced Networking – with Intel VF



# EC2 Enhanced Networking – with ENA



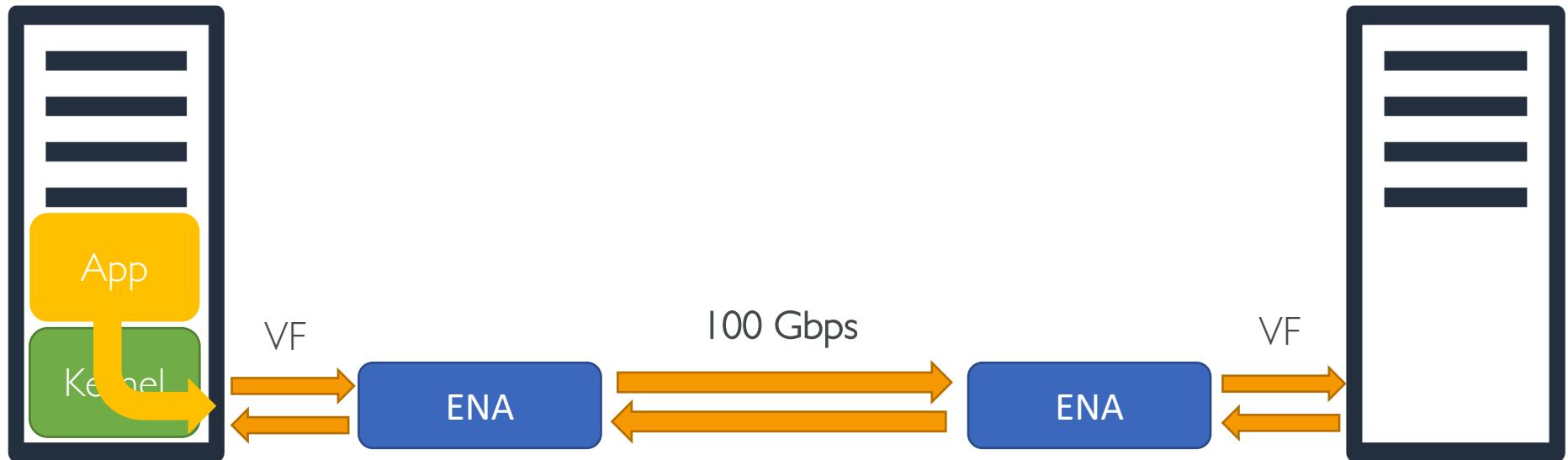
# Additional Network optimization techniques

- DPDK & EFA

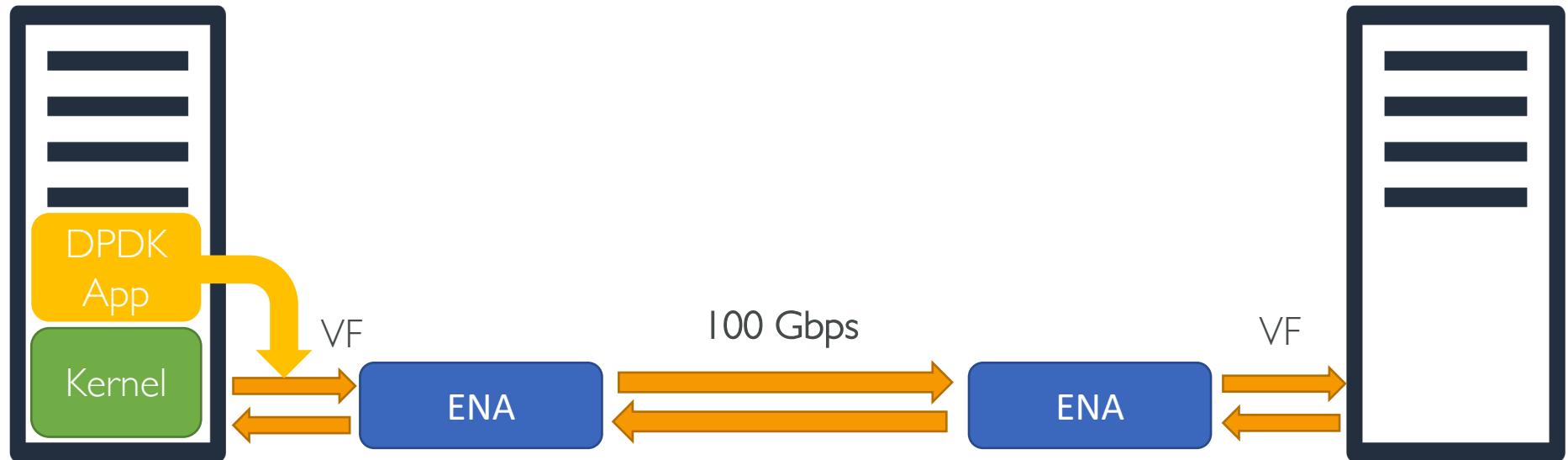
# Additional Tuning & Optimization - DPDK

- Intel Data Plane Development Kit (DPDK) is a set of libraries and drivers for fast packet processing.
- While Enhanced Networking and SR-IOV reduce overhead of packet processing between Instance and Hypervisor, DPDK reduces overhead of packet processing inside the **Operating System**
- DPDK provides
  - Lower latency due to Kernel bypass
  - More control of packet processing
  - Lower CPU overhead

# Packet processing without DPDK



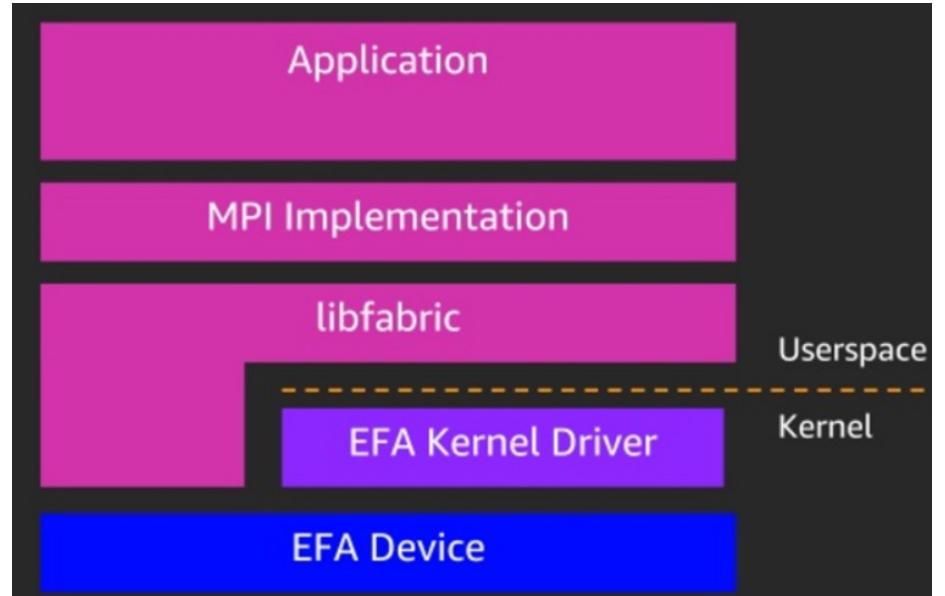
# Packet processing with DPDK



Application is developed to use DPDK libraries

# EFA – Elastic Fabric Adapter

- EFA is an ENA with added capabilities
- Provides lower latency and higher throughput
- Provides OS bypass functionality (Linux)
- For windows instance, it acts just as ENA
- With an EFA, HPC applications use MPI to interface with the Libfabric API which bypasses OS kernel and communicates directly with the EFA device to put packets on the network

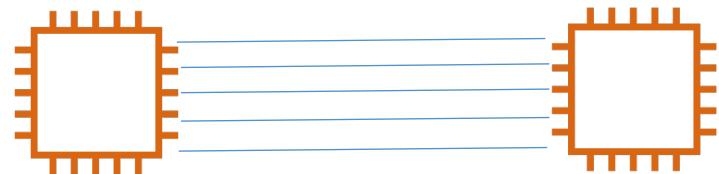


c5n.18xlarge, p3dn.24xlarge

# Network Bandwidth Limits

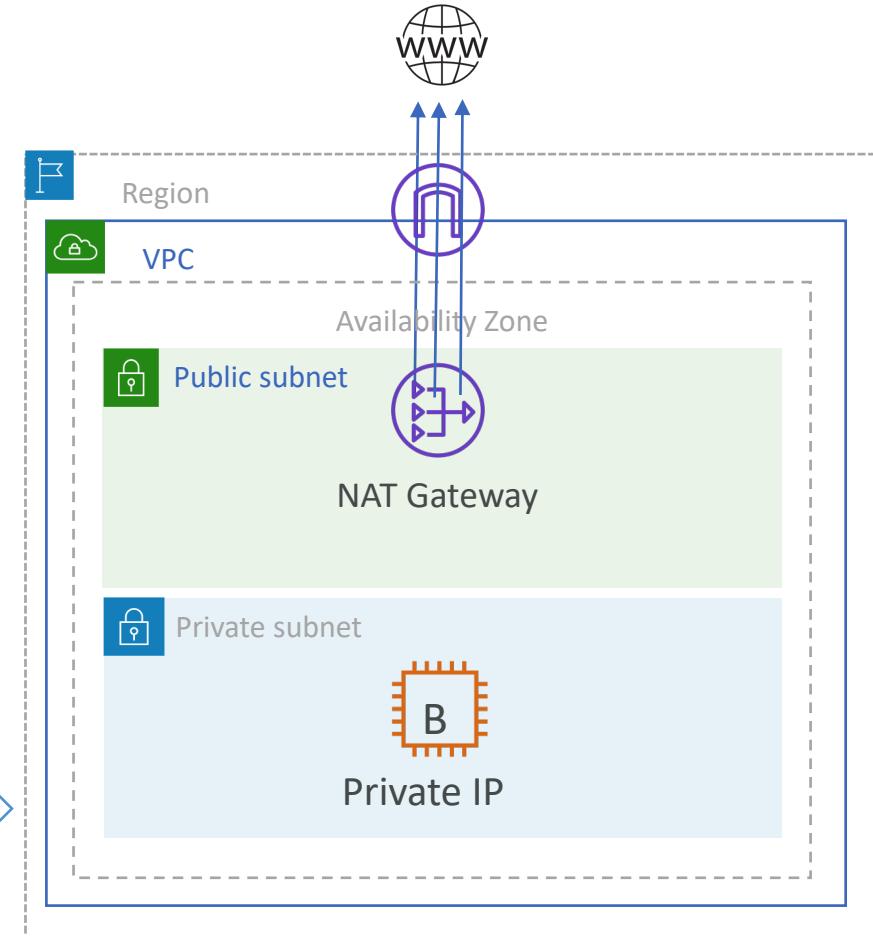
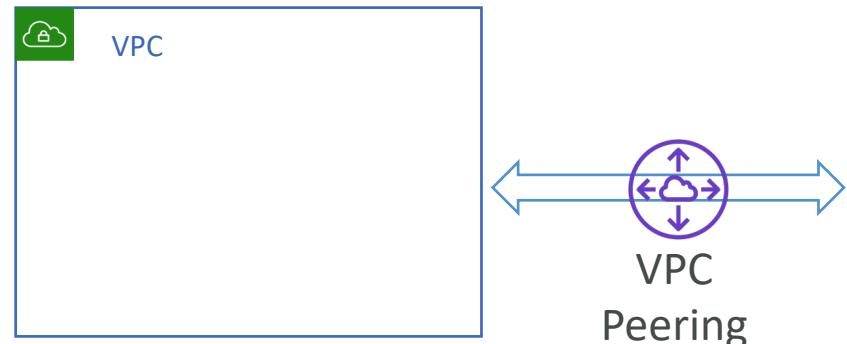
# Let's discuss..

- VPC Bandwidth Limits – Internet Gateway, NAT Gateway, VPC Peering
- EC2 Bandwidth Limits
- Bandwidth over a VPN Connection & AWS Direct Connect & Transit Gateway
- Network Flow
  - Network flow is a 5 tuple point to point connection (Protocol Src IP, Src Port, Dest IP, Dest Port)
  - Multiple flows allow to scale the network performance



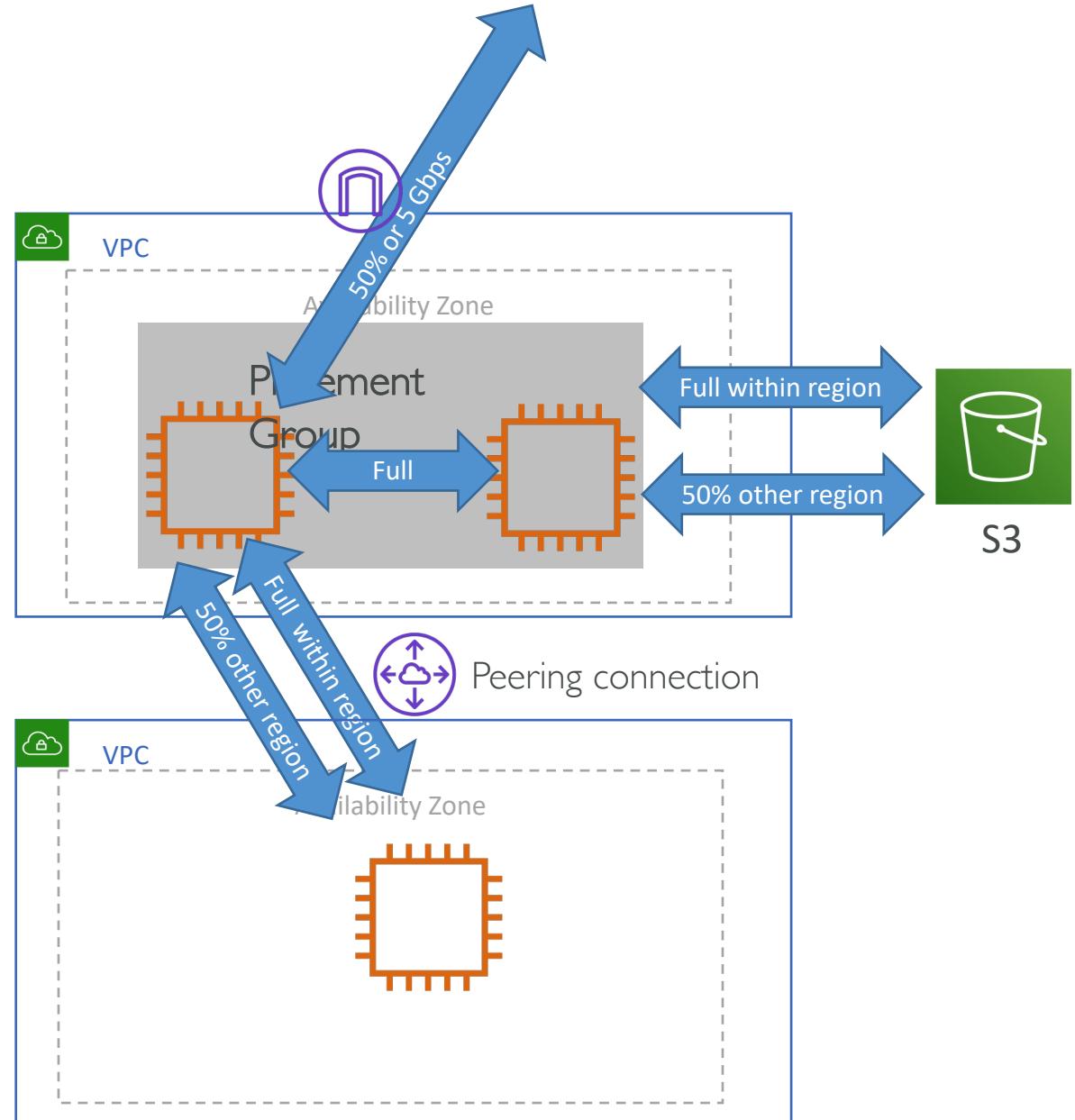
# VPC Bandwidth limits

- No VPC specific limits
- No limit for any Internet Gateway
- No limit for VPC peering
- Each NAT gateway can provide up to 45 Gbps. Use multiple NAT gateways to scale beyond 45 Gbps.



# EC2 Bandwidth limits

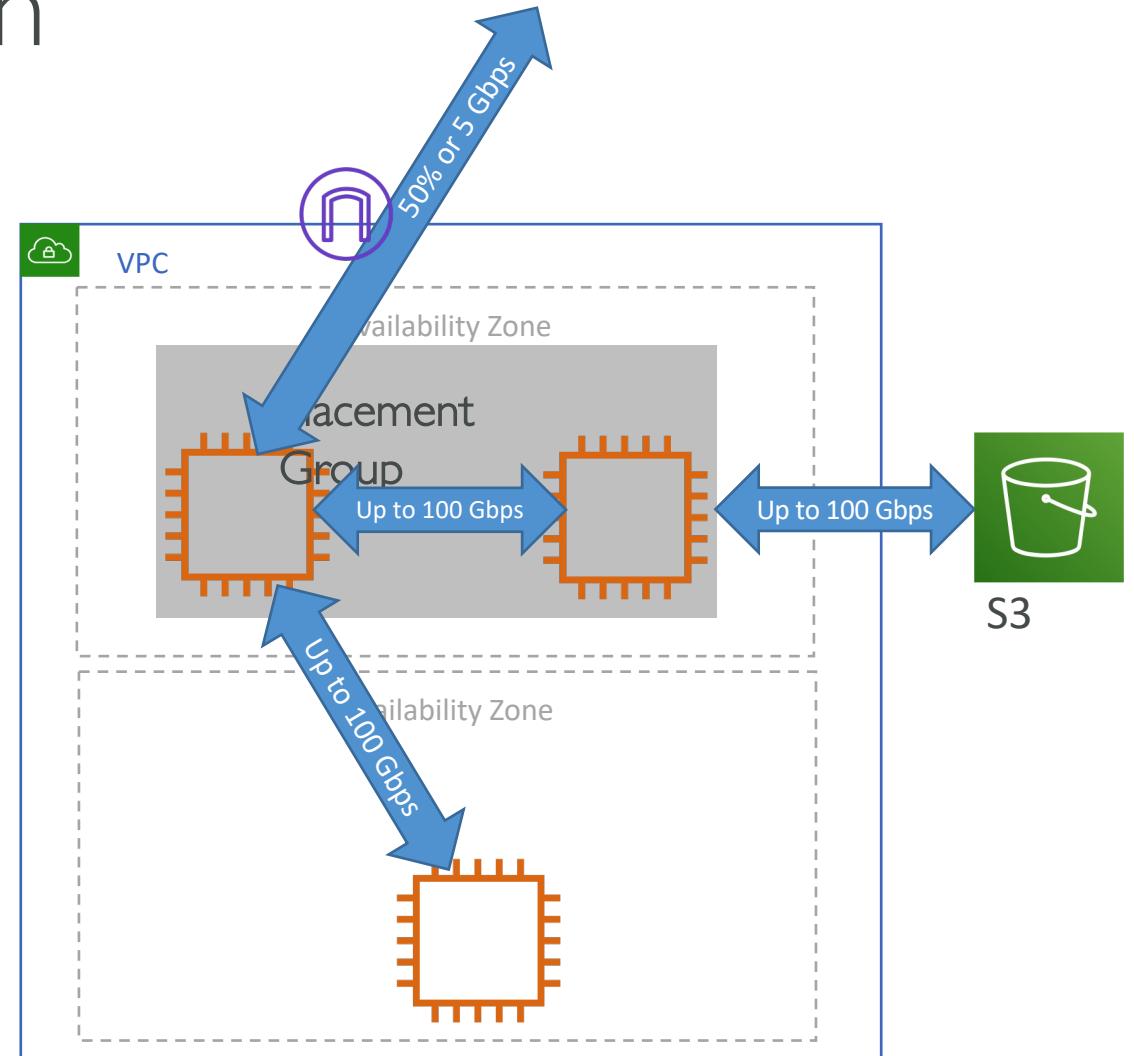
- Depends on factors like Instance family, vCPU, traffic destination etc.
- Within the Region
  - Can utilize the full network bandwidth available to the instance.
- To other Regions, an internet gateway, or Direct Connect
  - Can utilize up to 50% of the network bandwidth (current generation instance with a minimum of 32 vCPUs)
  - Otherwise limited to 5 Gbps.



# EC2 maximum bandwidth

- With Intel 82599 VF interface
  - 10 Gbps aggregate and 5 Gbps flow-based bandwidth limit
- With AWS ENA driver
  - 10 Gbps flow limit inside a placement group
  - 5 Gbps flow limit outside of a placement group
  - Aggregate bandwidth of 100 Gbps with multiple flows within a VPC or a peered VPC or to S3 (using VPC endpoint) in the same region

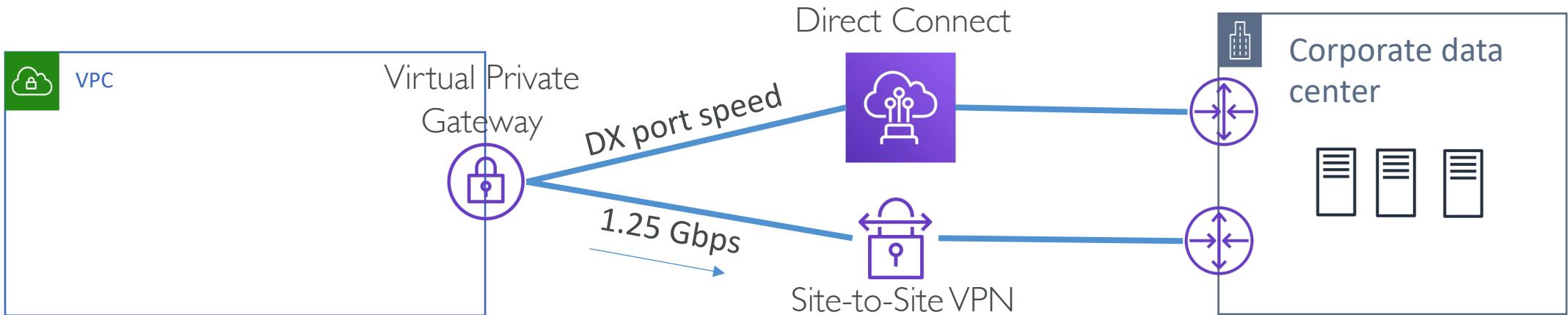
AWS P4d instances deployed in UltraClusters supercomputer provides 400 Gbps networking



EC2 max bandwidth with ENA

# VPN and DX Bandwidth

- 1.25 Gbps aggregate bandwidth per Virtual Private Gateway for traffic from AWS to on-premises
- Multiple VPN connections to the same Virtual Private Gateway are bound by an aggregate throughput limit
- AWS Direct connect bandwidth is defined by Port Speed opted
- For AWS Direct Connect connection on a Virtual Private Gateway, the throughput is bound by the Direct Connect physical port itself.
- Transit Gateway supports 1.25Gbps per VPN tunnel and 50 Gbps total VPN bandwidth



# Network I/O Credits

# Network Credits

- Instance families such as R4 and C5 use a network I/O credit mechanism
- Most application do not consistently need a high network performance
- These instances perform well above baseline n/w performance during peak requirement
- Make sure that you consider the accumulated n/w credits before doing performance benchmark for instances supporting network I/O credits mechanism

# Use cases for high performance networks

- High Performance Computing (HPC) workloads
  - Using cluster placement groups with HPC enables access to a low-latency, high-bandwidth network for tightly coupled, IO-intensive, and storage-intensive workloads.
  - For faster Amazon EBS IO, it is recommended using Amazon EBS-optimized instances and Provisioned IOPS volumes for high performance.
- Real Time Media
  - For applications like VoIP, Media streaming using RTP and RTMP
  - Enhanced networking provides smoother packet delivery with less packet loss and jitter
- More use cases: Data Processing, Backup, On-prem Data Transfer, etc.

# Summary

# Summary

- For high network bandwidth and throughput, consider using Jumbo Frames, EC2 Enhanced Networking, Placement groups, EBS optimized instances, DPDK
- Instance level network optimization with
  - Enhanced Networking (SR-IOV, ENA, Intel VF 82599)
  - Placement Groups
  - EBS-Optimized Instances
- Operating system level network optimization with DPDK
- EFA is ENA with additional OS-bypass capability which provides further improved network performance for HPC workloads

# Exam Essentials

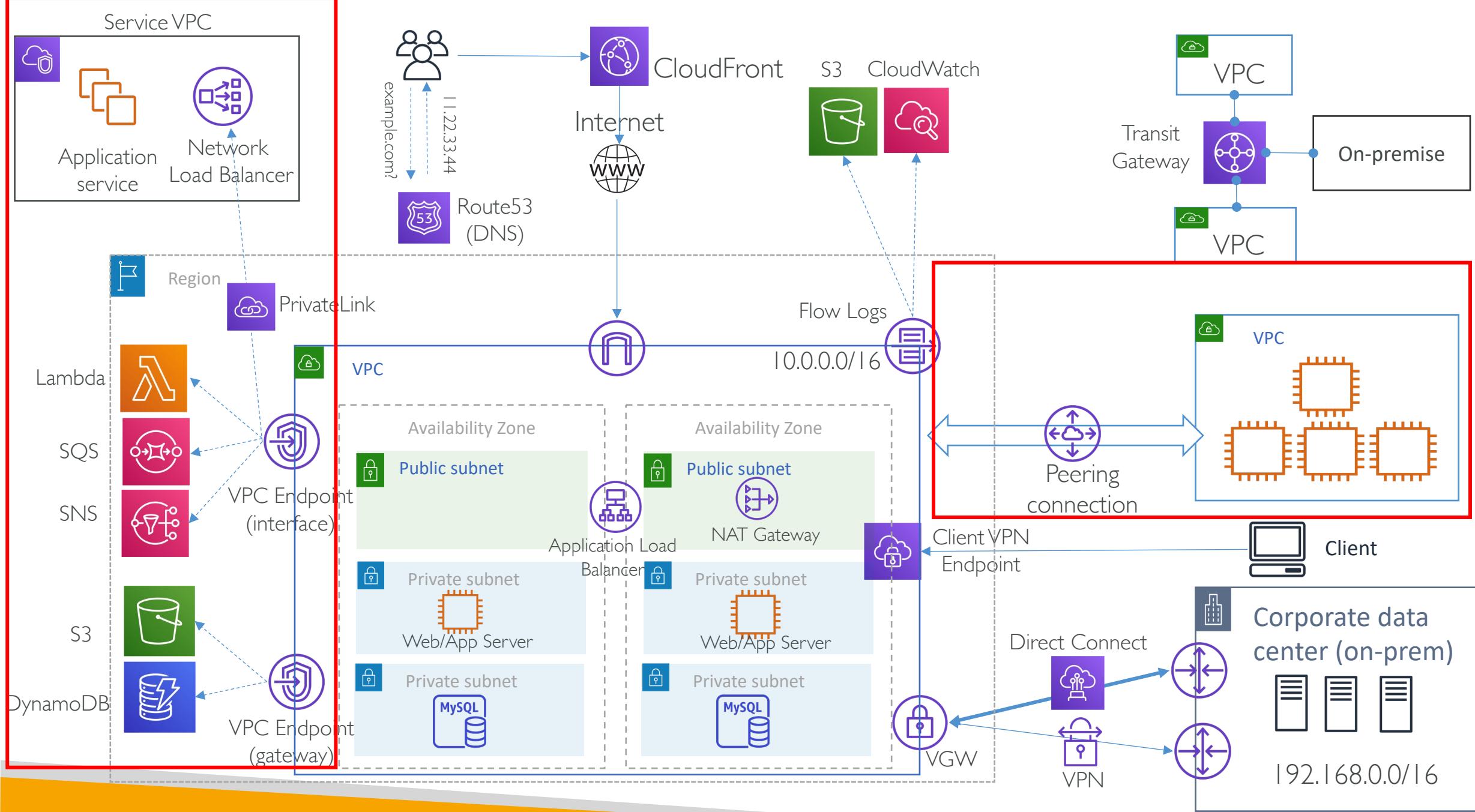
# Exam Essentials

- Within VPC the MTU size can be 9001 bytes with Jumbo Frames
- For the following cases the maximum MTU is 1500 bytes:
  - Traffic over an internet gateway
  - Traffic over an inter-region VPC peering connection
  - Traffic over VPN connections
- When PPS is bottleneck, increased MTU provides more throughput
- While Enhanced Networking and placement groups lowers the latency between EC2 and hypervisor, DPDK improves the packet processing at OS level

# Exam Essentials

- EC2 network bandwidth depends on many factors including Instance family, size, enhanced networking support etc.
- Bandwidth for aggregate multi-flow traffic available to an instance depends on the destination of the traffic
  - Within the Region
    - Full bandwidth of the [EC2 instance](#)
    - Within the AWS region you can get up to 100 Gbps bandwidth between EC2 instances or between EC2 and S3 by using multiple-flows
  - Outside of the Region, Internet or Direct Connect
    - 50% of the available network bandwidth for current generation instances with minimum 32 vcpu
    - 5 Gbps for the instances with less than 32 vcpu
- Bandwidth for single-flow (5-tuple) traffic is limited to 5 Gbps and can get up to 10 Gbps if the instances are within the Cluster placement group

# Private Connectivity options in VPC



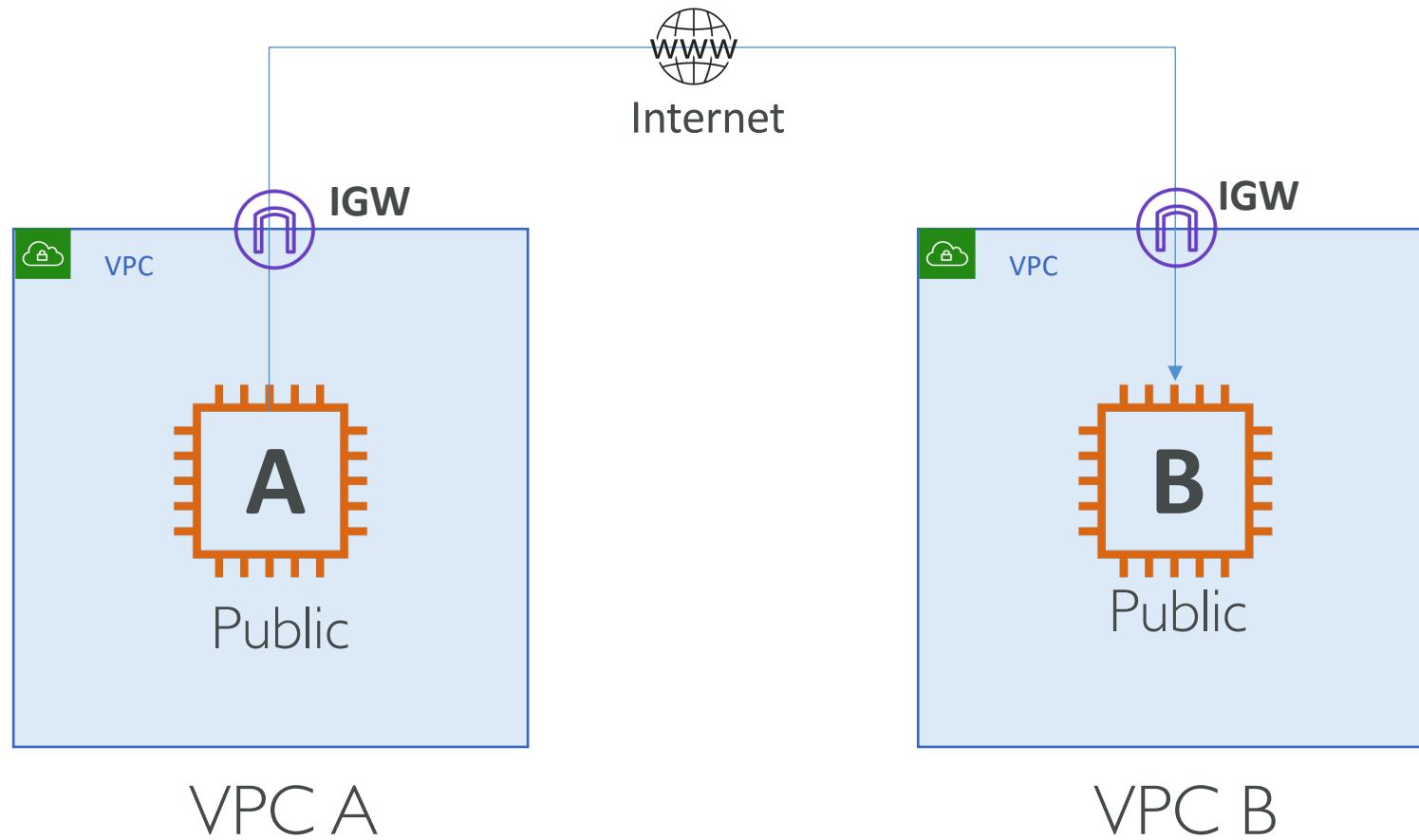
# Topics

- VPC Peering
- VPC Endpoints – Gateway and Interface
- VPC Private Link
- Transit VPC
- Transit Gateway

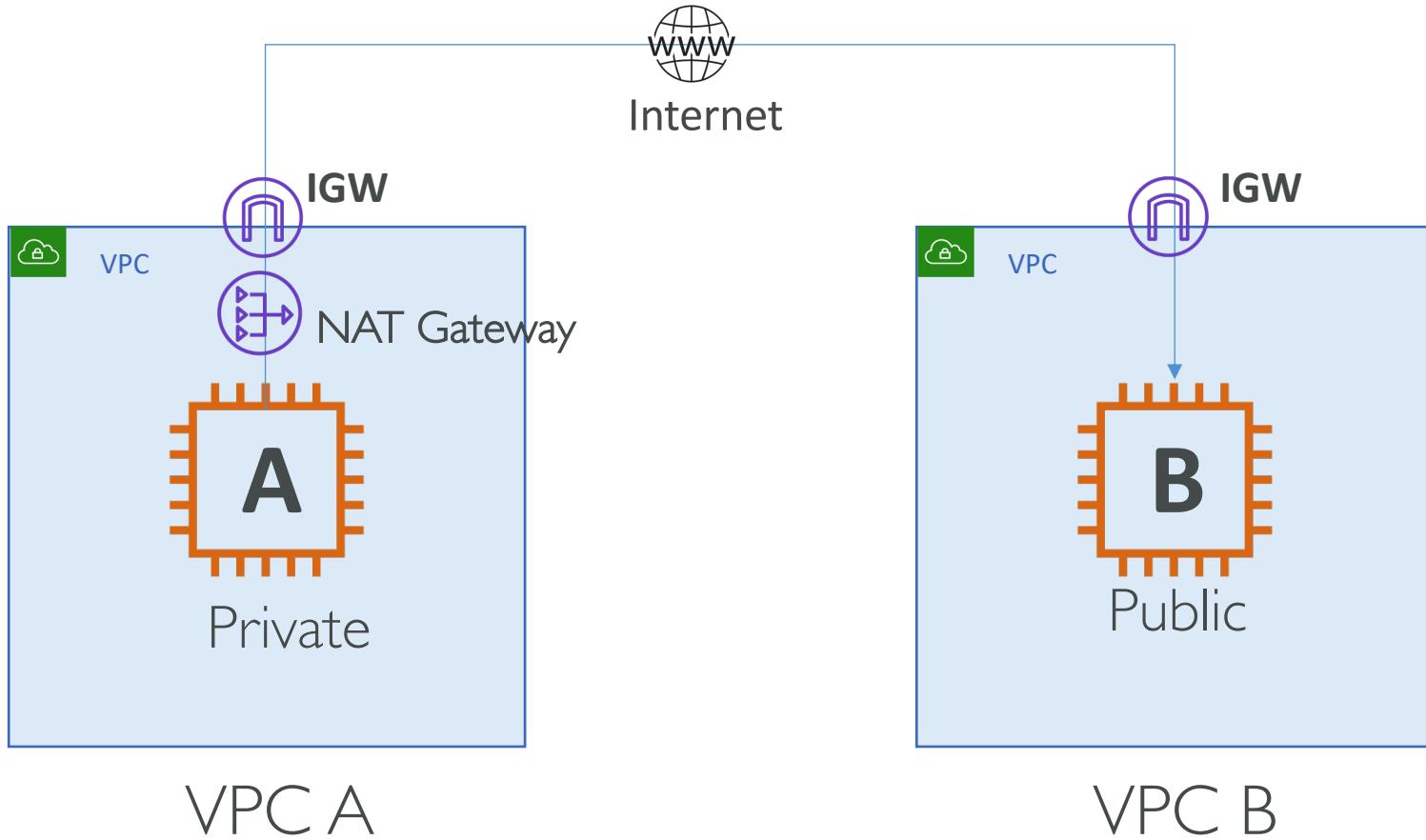
# Why do we need Private Connectivity?

# Options for EC2 communication across VPCs

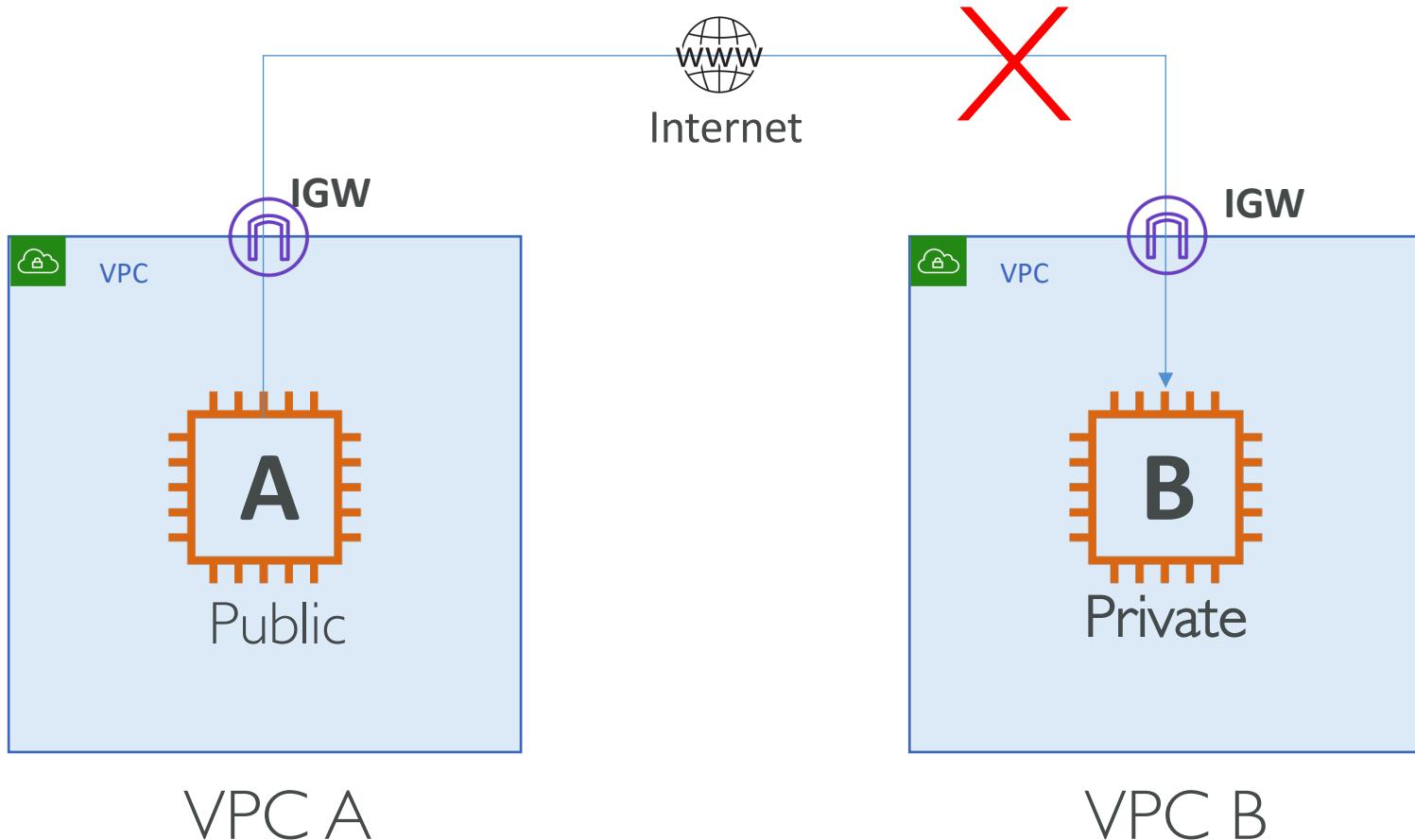
## – Public to Public



# Options for instance communication across VPCs – Private to Public

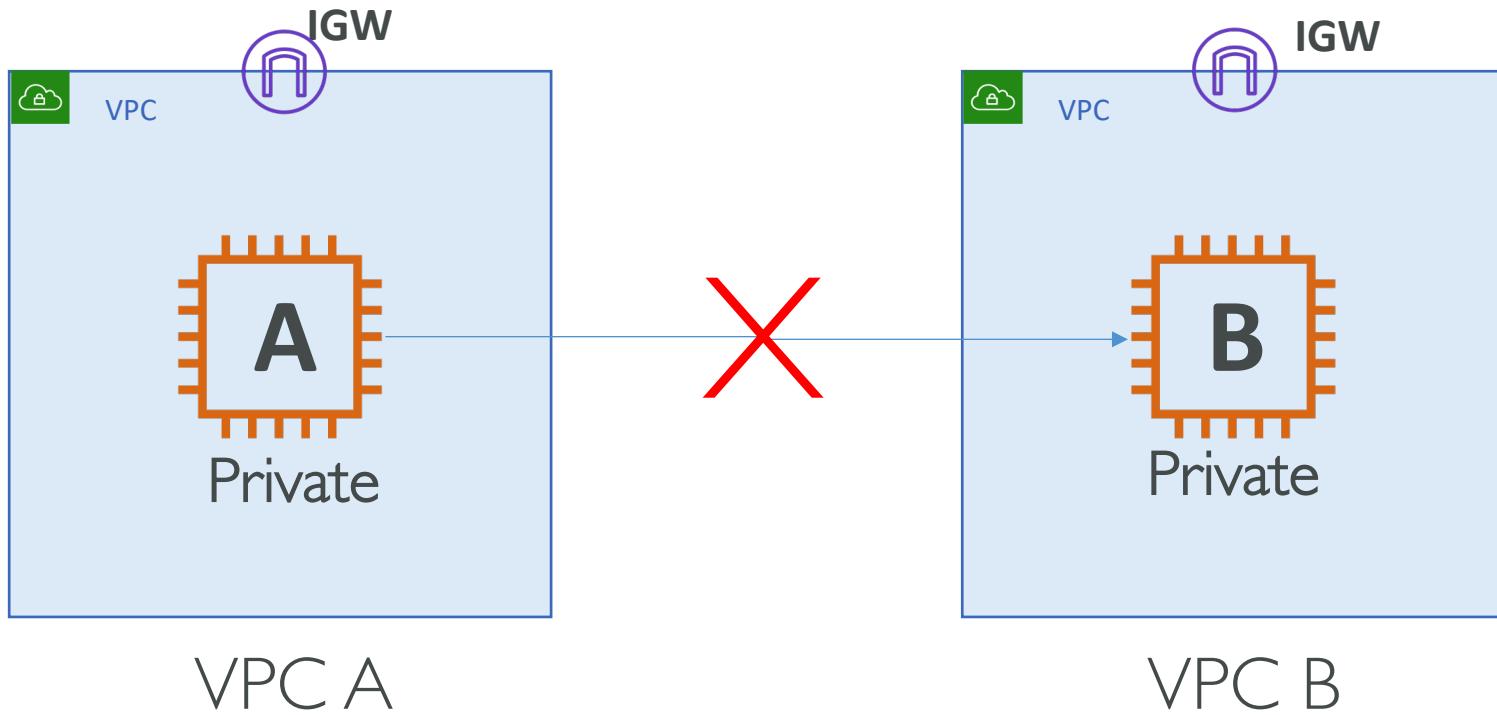


# Options for instance communication across VPCs – Public to Private



Not possible without SNAT  
or Port forwarding

# Options for instance communication across VPCs – Private to Private



Not possible without some  
Private connectivity

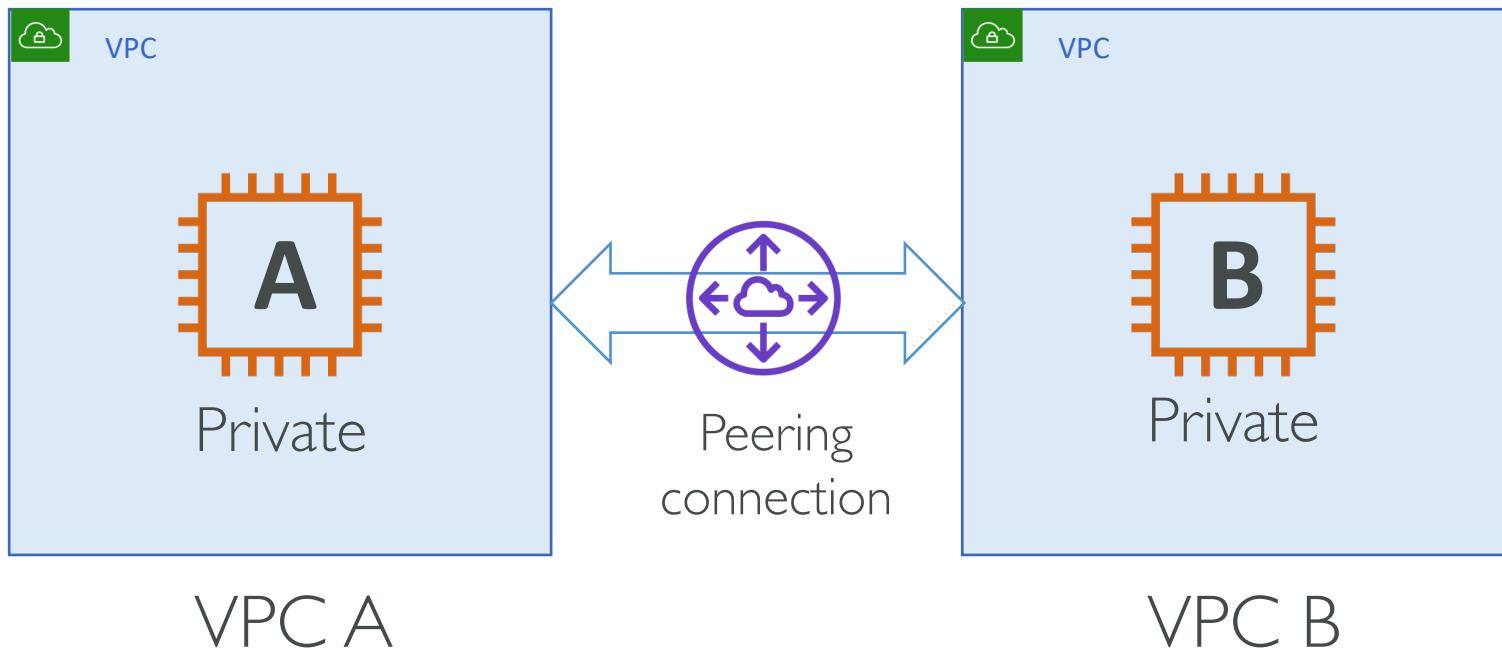
# Disadvantages of having Internet based traffic

- Traffic flows over the internet (can be considered less secured)
- Internet connectivity may not provide consistent bandwidth and latency
- NAT devices comes with considerable per hour running cost and data processing charges
- You might have to un-necessarily expose your servers publicly even if they don't need to be public (e.g Databases, Application servers, Intranet application servers)

Lets look at how VPC Peering, VPC endpoints and VPC PrivateLink help solve this problem !

# VPC Peering

# VPC Peering



# VPC Peering

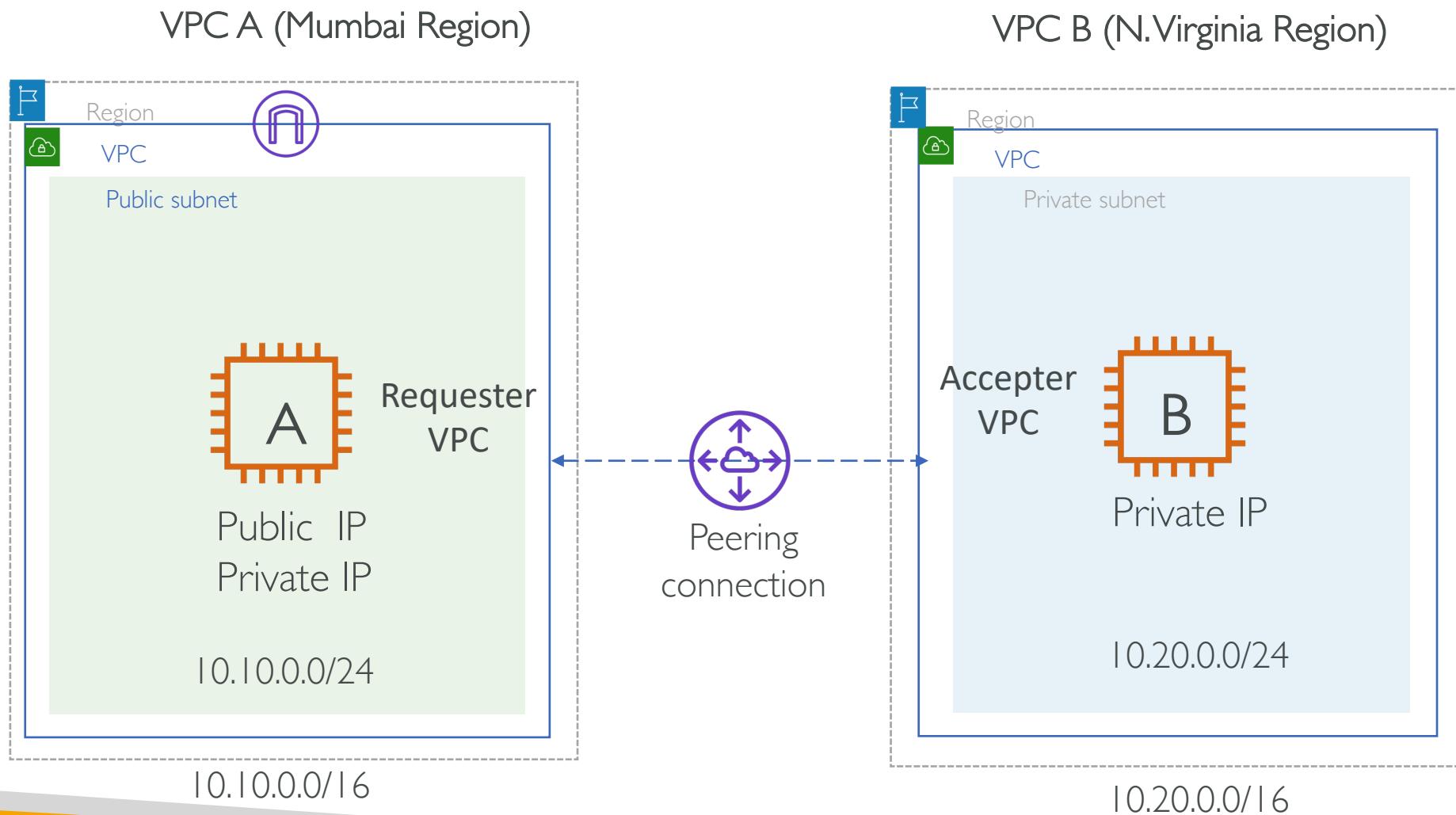
- Connect two VPC, privately using AWS' network
- Make them behave as if they were in the same network
- Peered VPCs can be in same AWS region or across AWS regions
- You can do VPC peering with another AWS account

## Caveats:

- VPC CIDRs should be non-overlapping
- You must update route tables in each VPC's subnets to ensure instances can communicate across VPC

# VPC Peering Demo

# VPC Peering - Demo



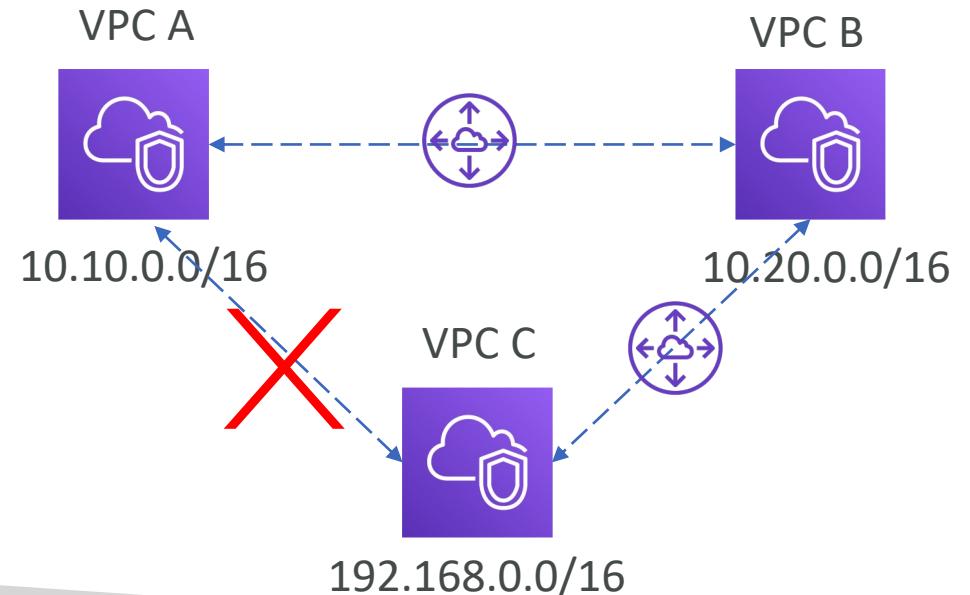
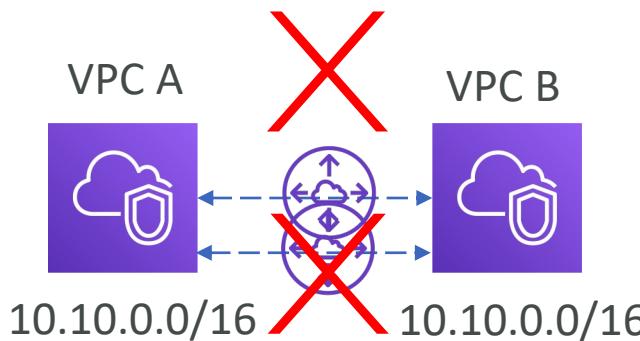
# Steps

- Create VPC-A in Mumbai (ap-south-1) region (CIDR: 10.10.0.0/16)
- Create Internet Gateway and Attach to VPC-A
- Create Public Subnet in VPC-A (10.10.0.0/24)
- Launch EC2 instance and assign Public IP. Open inbound port 22 for your IP.
- Create VPC-B in North Virginia (us-east-1) region (CIDR: 10.20.0.0/16)
- Create Private Subnet in VPC-B (10.20.0.0/24)
- Launch EC2 instance in private subnet. Open inbound port 22 and ICMP for VPC-A CIDR range
- Create VPC peering connection request from VPC-A to VPC-B
- Accept the connection request in VPC-B
- Modify route tables on both sides for traffic on other VPC
- Login To EC2-A instance and try to connect to EC2-B (ping or SSH)

# VPC Peering Limitations

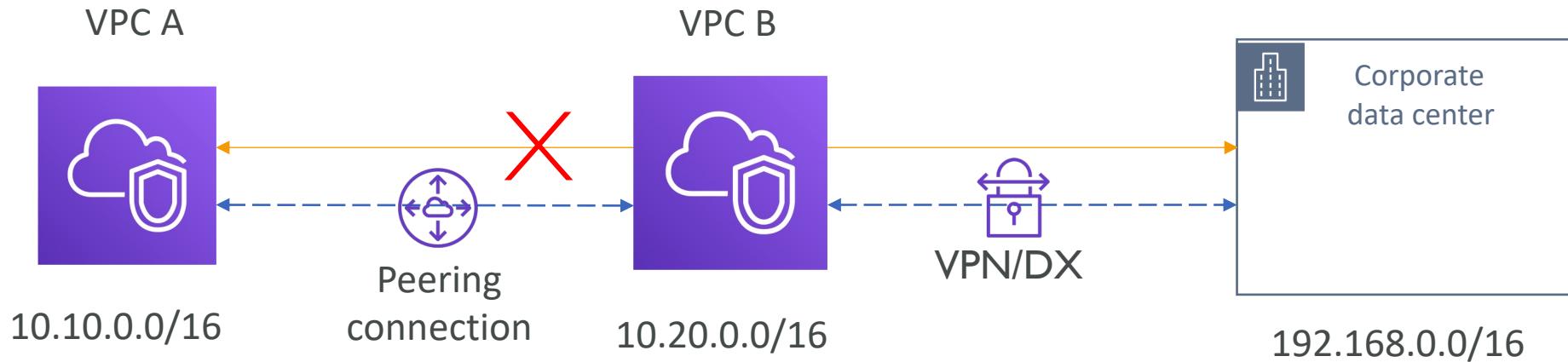
# VPC Peering Limitations

- Must not have overlapping CIDR
- VPC Peering connection is **not transitive** (must be established for each VPC that need to communicate with one another)
- You can setup only 1 VPC peering connection between 2 VPCs
- Maximum 125 VPC peering connections per VPC



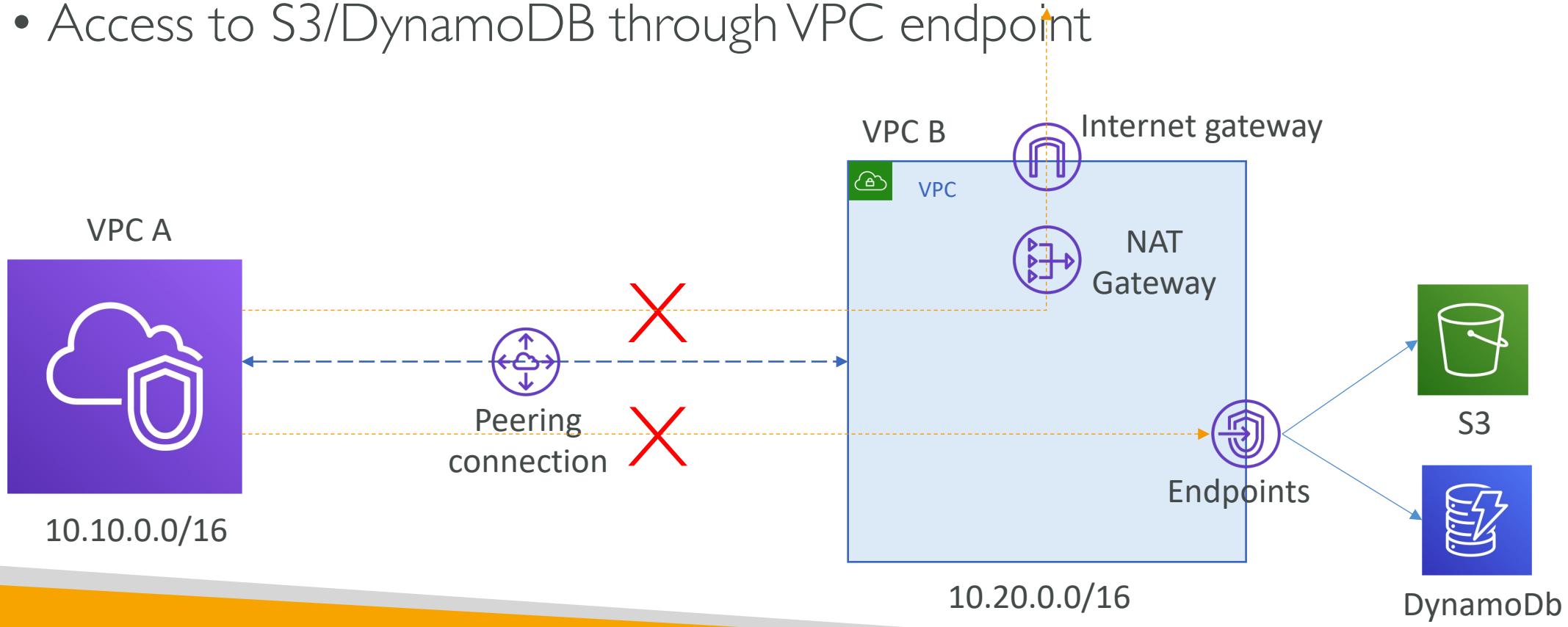
# VPC Peering invalid scenarios – VPN or DX

- VPN or Direct Connect connection to on-premises network



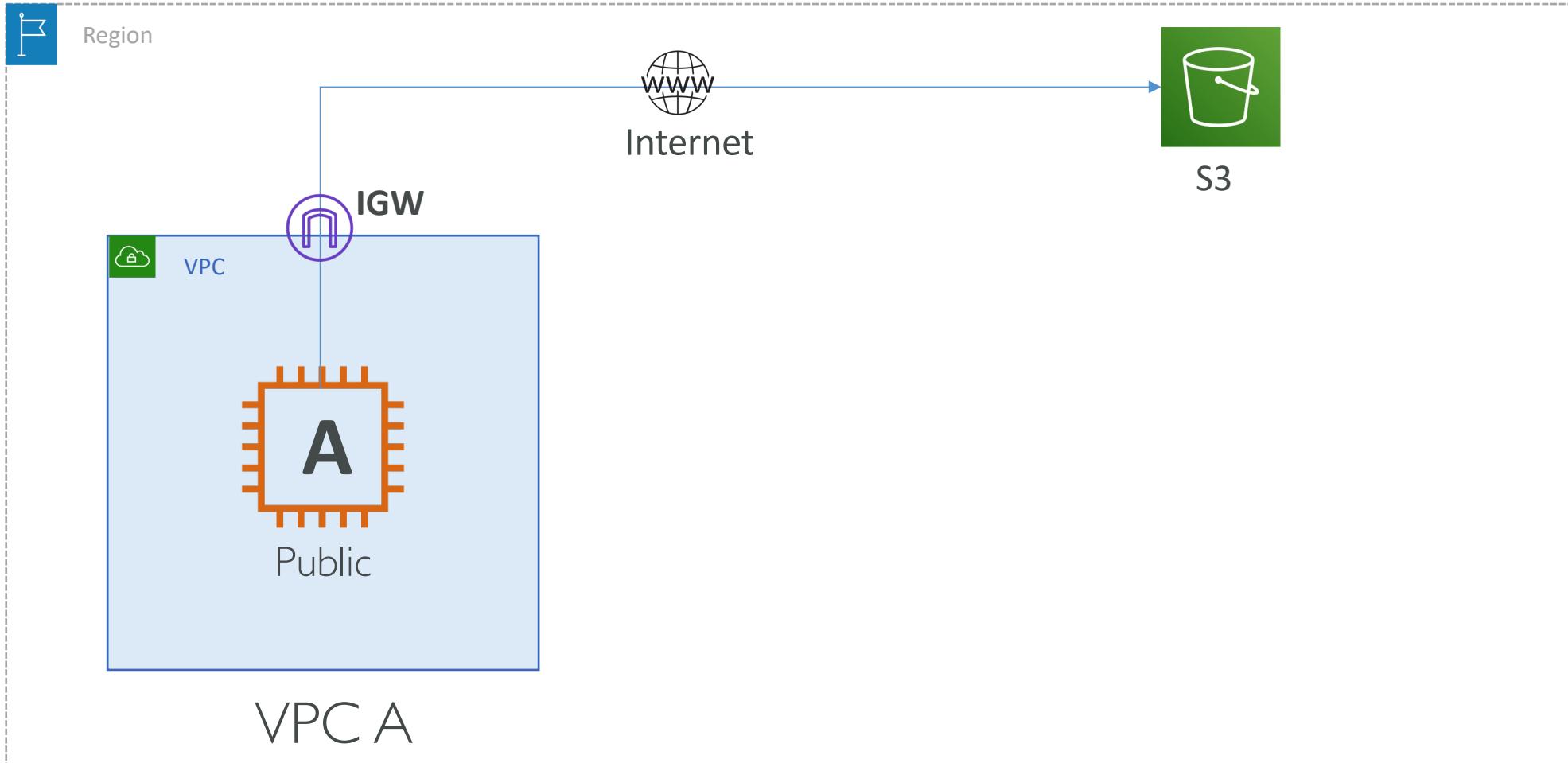
# VPC Peering invalid scenarios – IGW

- Internet access through peered VPC internet gateway
- Internet access through peered VPC NAT Gateway
- Access to S3/DynamoDB through VPC endpoint

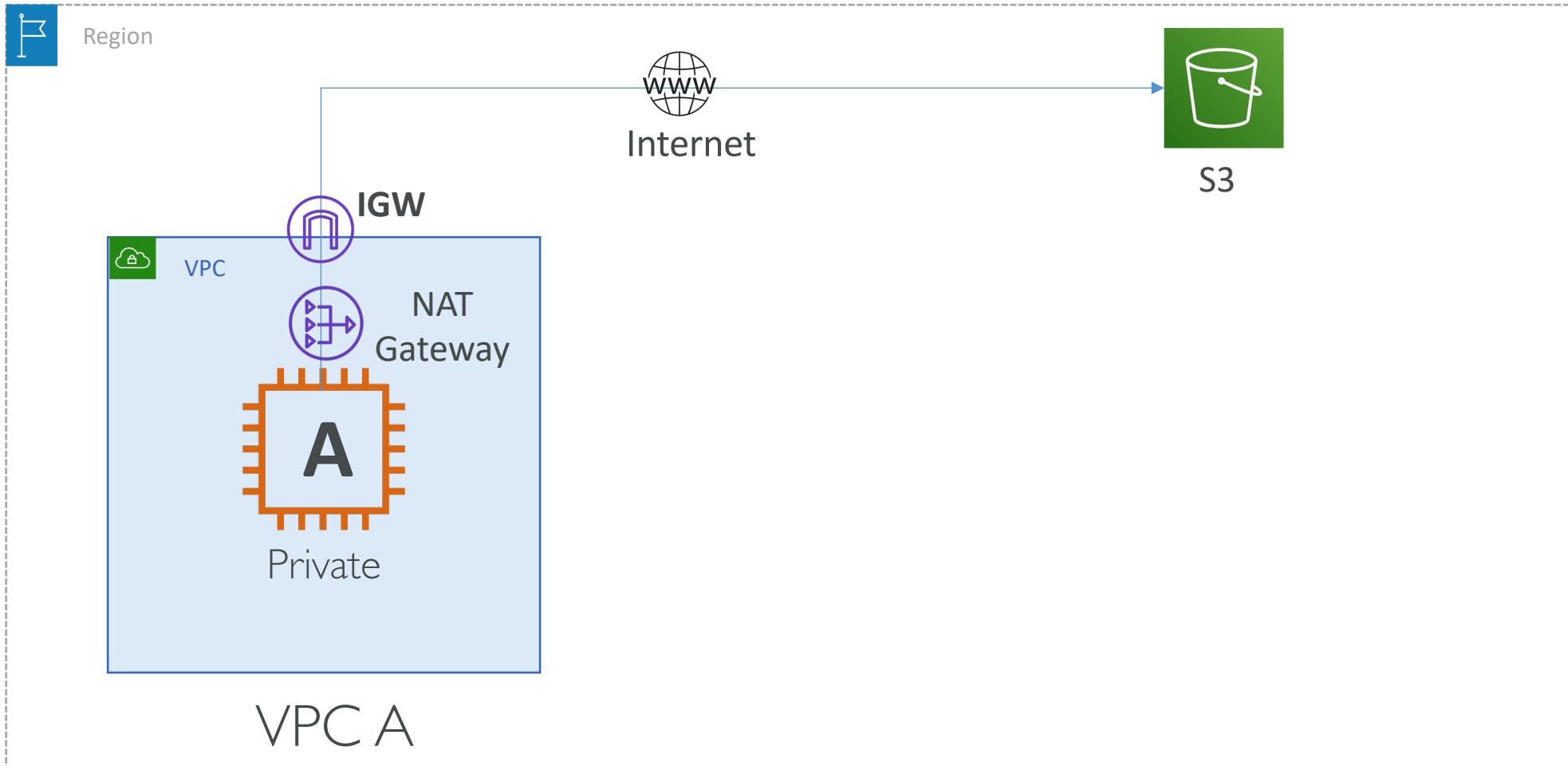


# VPC Endpoints

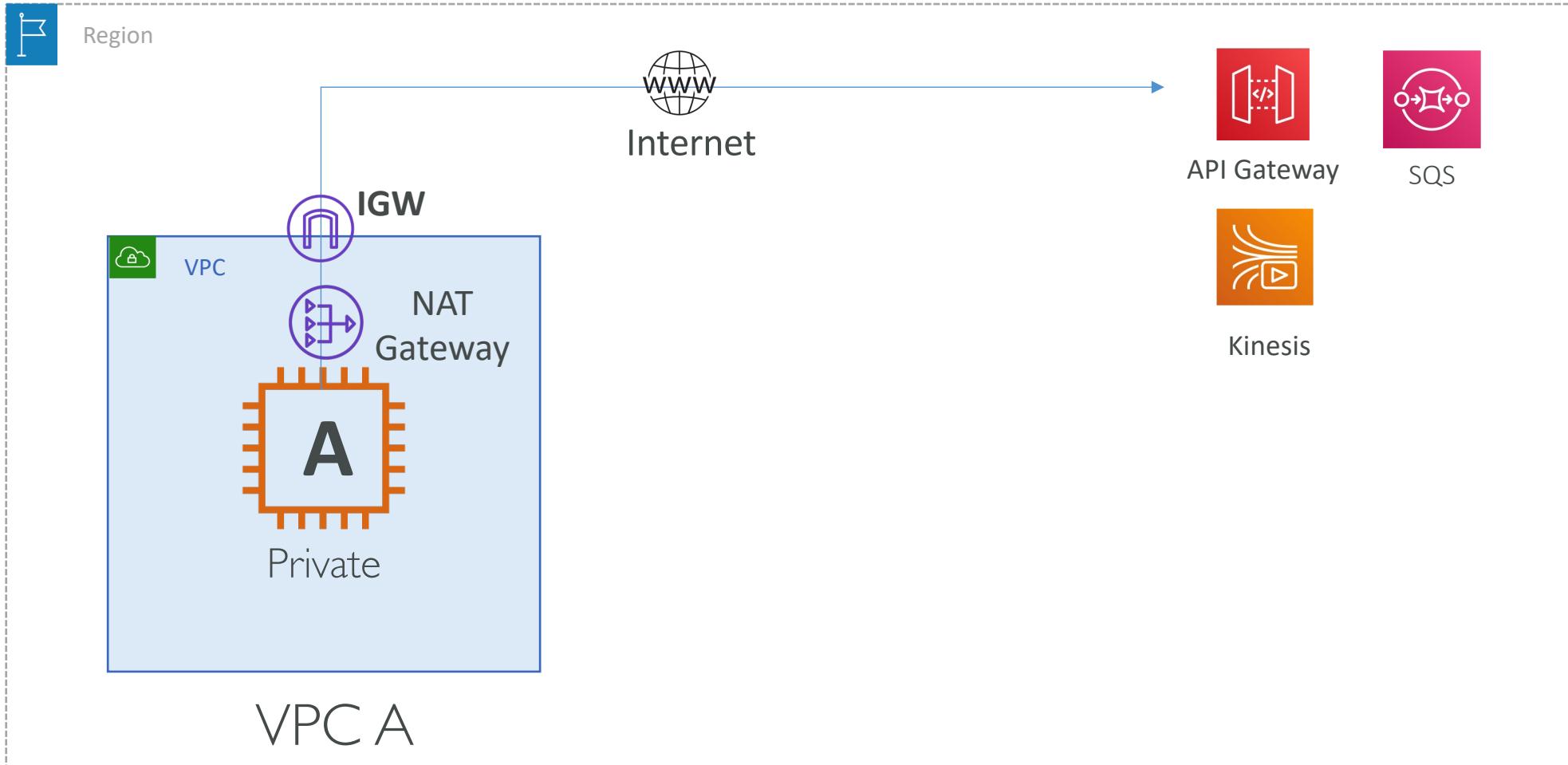
# Accessing AWS services from VPC



# Accessing AWS services from VPC – via NAT



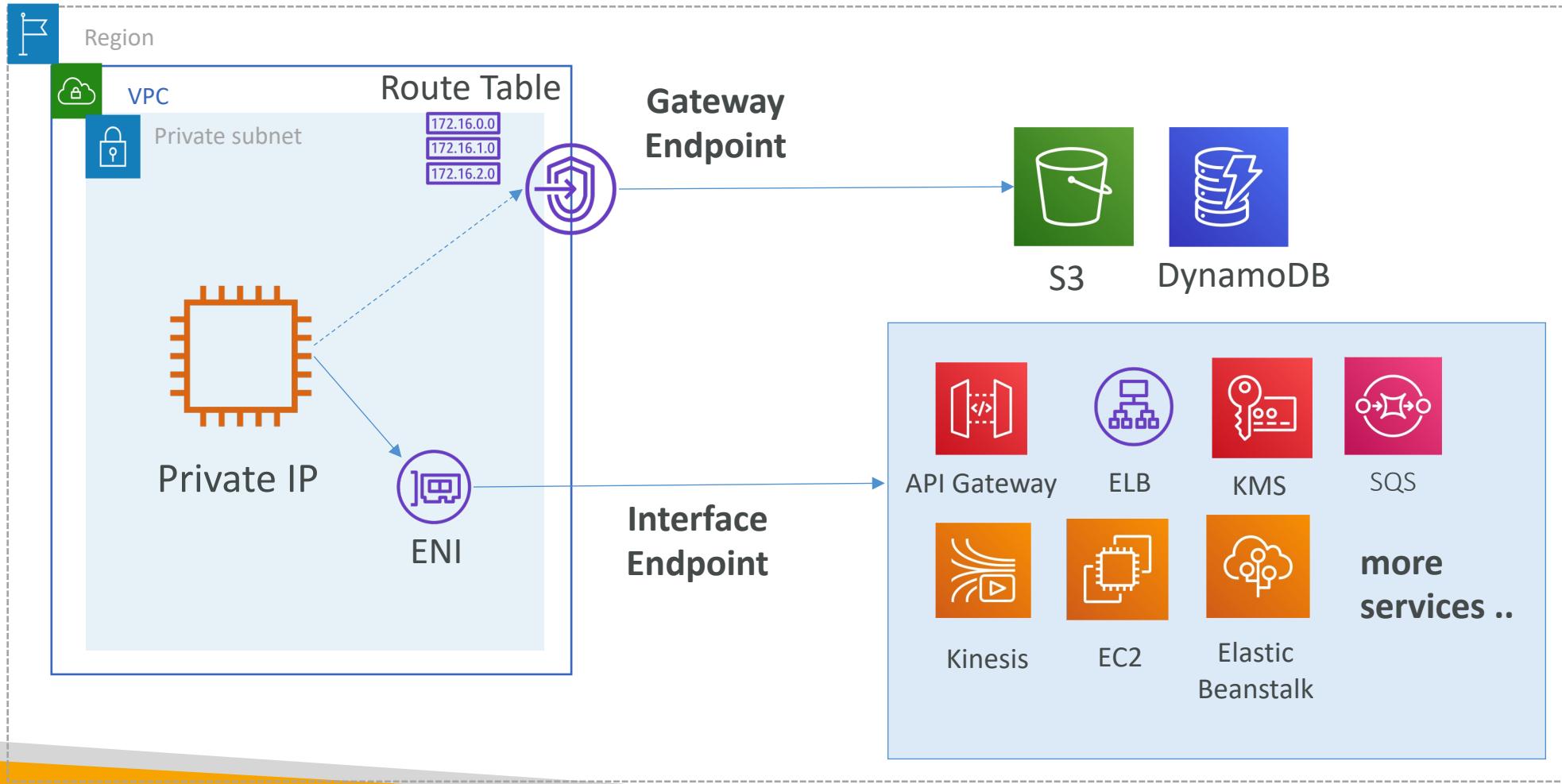
# Accessing AWS services APIs from VPC



# VPC Endpoints

- Endpoints allow you to connect to AWS Services using a private network instead of the public network
- They remove the need of IGW, NAT GW to access AWS Services
- Endpoint devices are horizontally scaled, redundant and highly available without any bandwidth constraint on your network traffic
- Gateway Endpoint: provisions a target and must be used in a route table – S3 and DynamoDB
- Interface Endpoint: provisions an ENI (private IP) as an entry point – most other AWS services

# VPC Endpoints: Gateway and Interface



# VPC Gateway Endpoint

# VPC Gateway Endpoint

- Enables private connection between VPC and S3/DynamoDB
- Need to modify the route tables and add an entry to route the traffic to S3 or DynamoDB through the gateway VPC endpoint
- When we create an Amazon S3 endpoint, a *prefix list* is created in VPC
- The prefix list is the collection of IP addresses that Amazon S3 uses.
- The Prefix list is formatted as pl-xxxxxxxx and becomes an available option in both subnet routing tables and security groups

| Destination   | Target                 | Status |
|---|------------------------|--------|
| 10.0.0.0/16   | local                  | active |
| pl-78a54011 (com.amazonaws.ap-south-1.s3, 52.219.62.0/23, 3.5.212.0/23, 3.5.208.0/22, 52.219.64.0/22) | vpce-0e89398e630ab0bcf | active |
| 0.0.0.0/0   | nat-085190f27ccf2ca3e  | active |

# VPC Gateway Endpoint

- Prefix list should be added in Security group Outbound rule (if Security group outbound rules do not have default “Allow All” rule)

| Security Group: sg-17f47973 |                           |                             |                              |
|-----------------------------|---------------------------|-----------------------------|------------------------------|
| Description                 | Inbound                   | Outbound                    | Tags                         |
| <a href="#">Edit</a>        |                           |                             |                              |
| Type <small>i</small>       | Protocol <small>i</small> | Port Range <small>i</small> | Destination <small>i</small> |
| HTTPS                       | TCP                       | 443                         | pl-68a54001                  |
| HTTP                        | TCP                       | 80                          | pl-68a54001                  |

# VPC Endpoint Security

- Endpoints allows more granular access to VPC resources as compared to broad access through VPC peering connection
- Access to S3 through VPC endpoint can be secured using bucket policies and endpoint policies
- VPC Endpoint policy
  - An IAM policy which is attached to VPC endpoint
  - Default policy allows full control to the AWS service

# VPC Endpoint Policy

- VPC Endpoint Policy to restrict access to a specific S3 bucket or a DynamoDB table

```
{  
  "Statement": [  
    {  
      "Sid": "Access-to-specific-bucket-only",  
      "Principal": "*",  
      "Action": [  
        "s3:GetObject",  
        "s3:PutObject"  
      ],  
      "Effect": "Allow",  
      "Resource": ["arn:aws:s3:::my_secure_bucket",  
                  "arn:aws:s3:::my_secure_bucket/*"]  
    }  
  ]  
}
```

Restrict access to  
S3 Bucket

```
{  
  "Statement": [  
    {  
      "Sid": "AccessToSpecificTable",  
      "Principal": "*",  
      "Action": [  
        "dynamodb:Batch*",  
        "dynamodb>Delete*",  
        "dynamodb:DescribeTable",  
        "dynamodb:.GetItem",  
        "dynamodb:PutItem",  
        "dynamodb:Update*"  
      ],  
      "Effect": "Allow",  
      "Resource": "arn:aws:dynamodb:us-east-1:123456789012:table/StockTable"  
    }  
  ]  
}
```

Restrict access to  
DynamoDB table

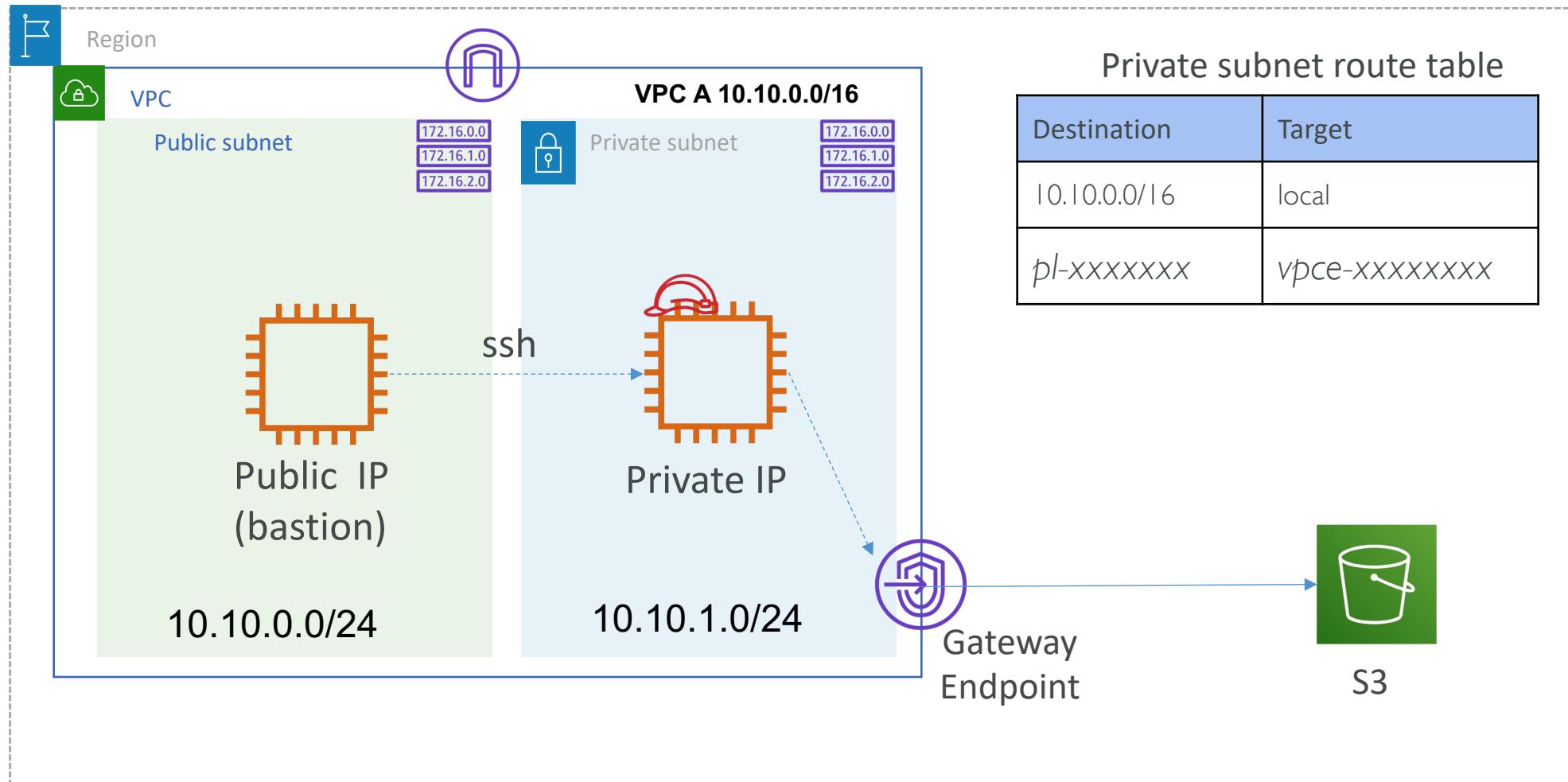
# Resource based policy - S3 bucket policy

- S3 Bucket policy to restrict access to specific VPC endpoint

```
{  
    "Version": "2012-10-17",  
    "Id": "Policy1415115909152",  
    "Statement": [  
        {  
            "Sid": "Access-to-specific-VPCE-only",  
            "Principal": "*",  
            "Action": "s3:*",  
            "Effect": "Deny",  
            "Resource": ["arn:aws:s3:::my_secure_bucket",  
                        "arn:aws:s3:::my_secure_bucket/*"],  
            "Condition": {  
                "StringNotEquals": {  
                    "aws:sourceVpce": "vpce-1a2b3c4d"  
                }  
            }  
        }  
    ]  
}
```

# VPC Gateway Endpoint - Demo

# VPC Endpoint for S3 - Demo



# Steps

1. Create VPC with Public and Private Subnet
2. Launch EC2 instances in both the subnets
3. Add IAM role to EC2 instance in Private Subnet to allow access to S3
4. Create VPC Gateway endpoint for S3
5. Modify Private subnet route table to route traffic to S3 via VPC gateway endpoint
6. Login to Public EC2 instance and from there SSH to Private EC2 instance
7. Test the connectivity to S3 by uploading/downloading some files

# VPC Gateway endpoint and S3 policies

# VPC Endpoint Policy & S3 bucket policy

- S3 bucket policy may have
  - Condition: "aws:sourceVpce": "vpce-1a2b3c4d" to Deny any traffic that doesn't come from a specific VPC endpoint (more secure)
  - Condition: "aws:sourceVpc": "vpc-111bbb22" for a specific VPC
- The **aws:sourceVpc** condition only works for VPC Endpoints, in case you have multiple endpoints and want to manage access to your S3 buckets for all your endpoints
- The S3 bucket policies can restrict access only from a specific public IP address or an elastic IP address. You can't restrict based on private IP
- Therefore aws:SourceIp condition doesn't apply for VPC endpoints

# Example S3 bucket policies

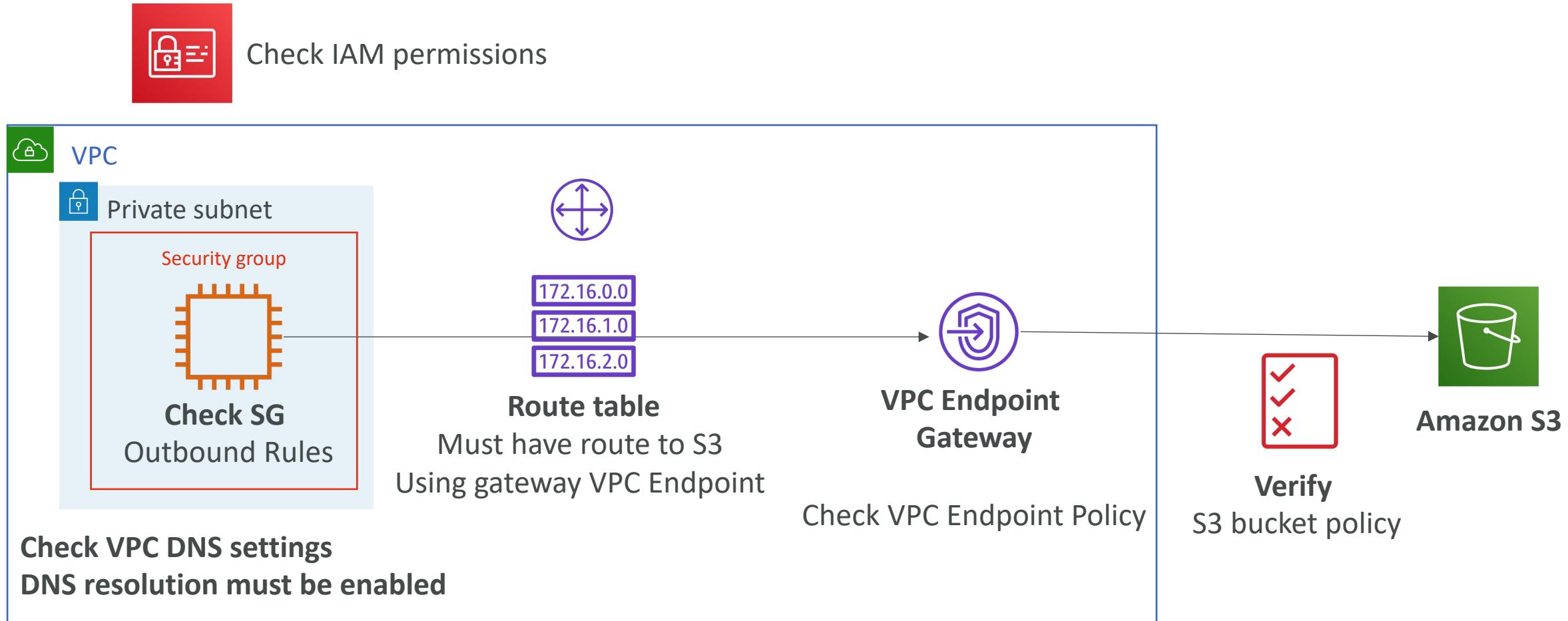
- S3 bucket policy to restrict to one specific VPC Endpoint

```
{  
  "Version": "2012-10-17",  
  "Id": "Policy1415115909152",  
  "Statement": [  
    {  
      "Sid": "Access-to-specific-VPCE-only",  
      "Principal": "*",  
      "Action": "s3:*",  
      "Effect": "Deny",  
      "Resource": ["arn:aws:s3:::my_secure_bucket",  
                  "arn:aws:s3:::my_secure_bucket/*"],  
      "Condition": {  
        "StringNotEquals": {  
          "aws:sourceVpce": "vpce-1a2b3c4d"  
        }  
      }  
    }  
  ]  
}
```

- S3 bucket policy to restrict to an entire VPC (multiple VPC Endpoints)

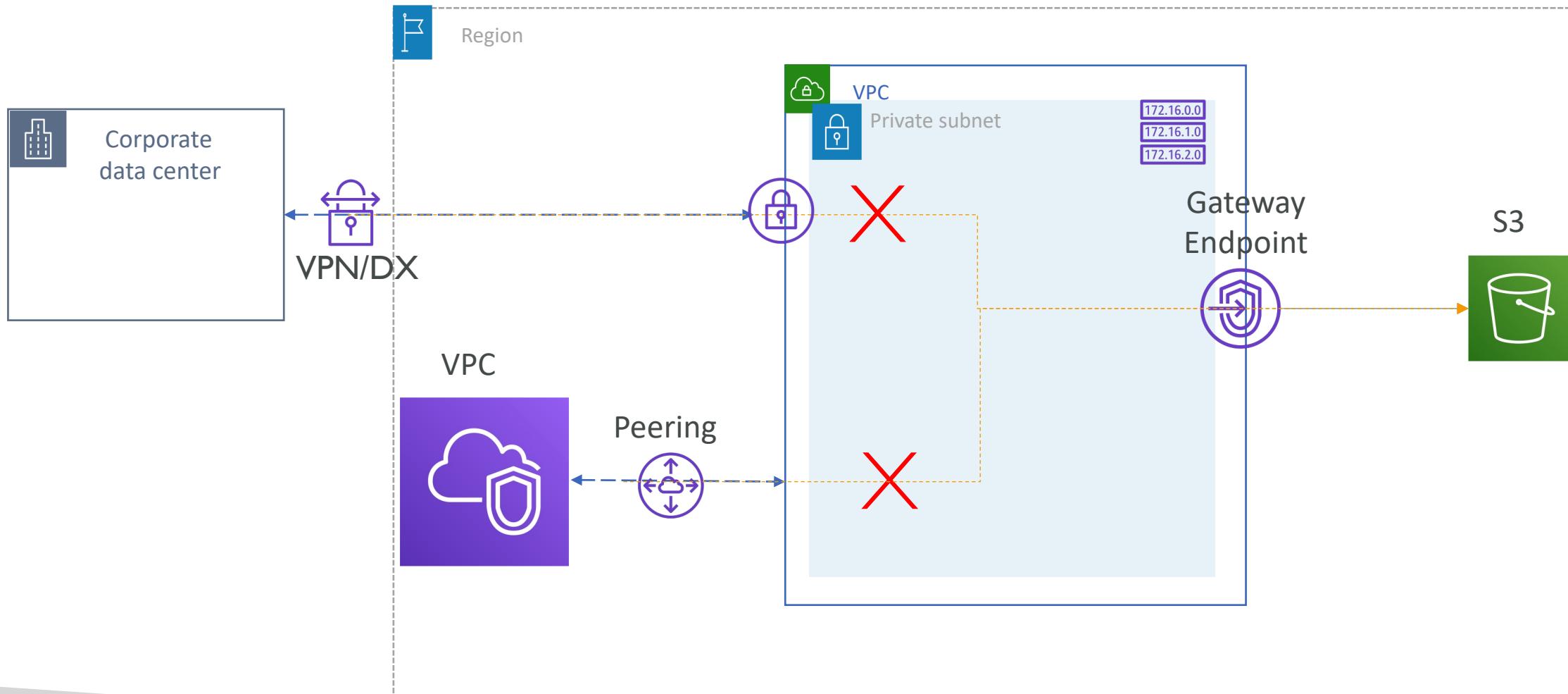
```
{  
  "Version": "2012-10-17",  
  "Id": "Policy1415115909152",  
  "Statement": [  
    {  
      "Sid": "Access-to-specific-VPCE-only",  
      "Principal": "*",  
      "Action": "s3:*",  
      "Effect": "Deny",  
      "Resource": ["arn:aws:s3:::my_secure_bucket",  
                  "arn:aws:s3:::my_secure_bucket/*"],  
      "Condition": {  
        "StringNotEquals": {  
          "aws:sourceVpc": "vpc-111bbb22"  
        }  
      }  
    }  
  ]  
}
```

# VPC Endpoint Policies for S3 Troubleshooting



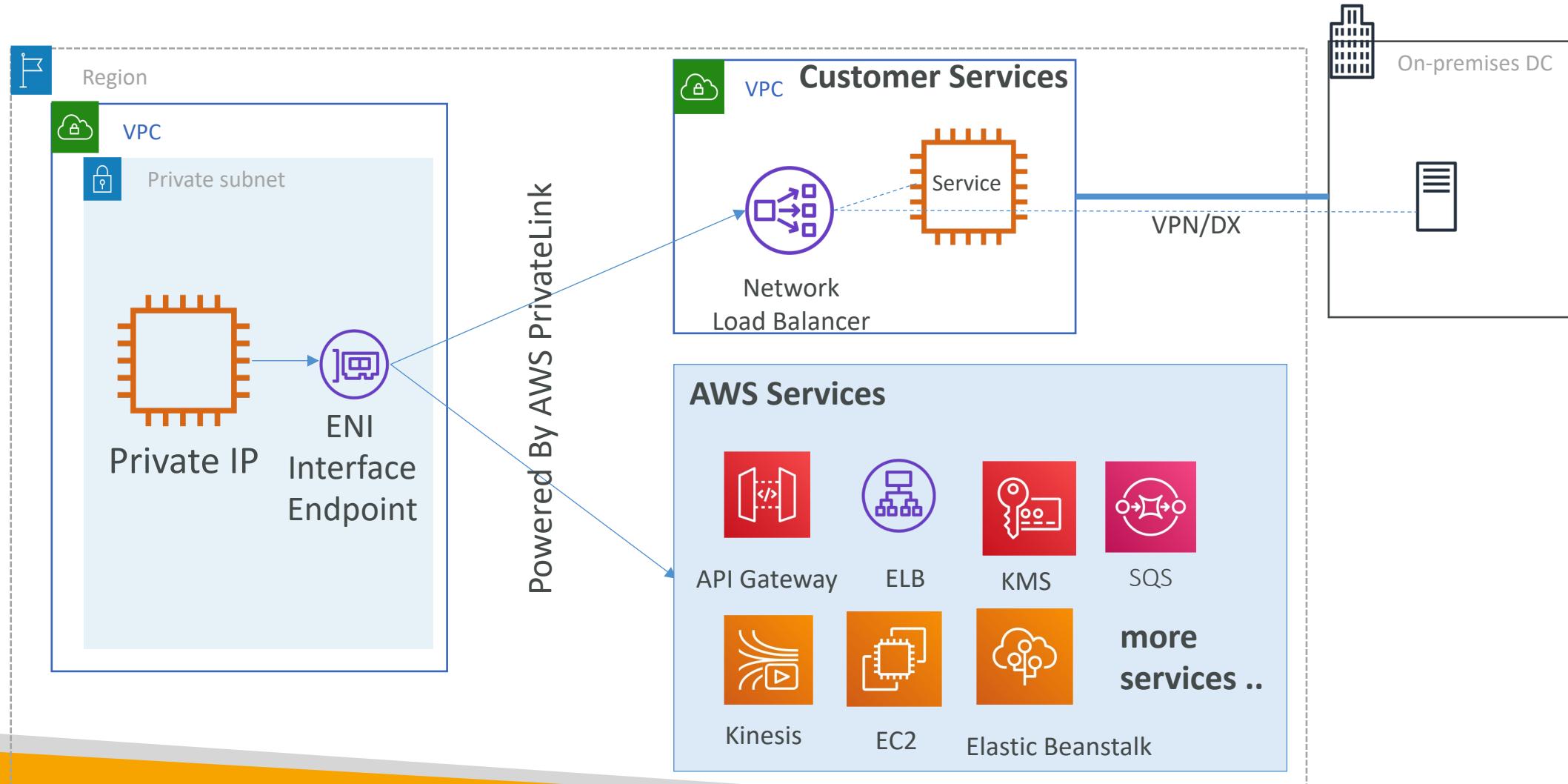
# VPC Gateway endpoint access from remote networks

# Gateway endpoint access from Remote Networks



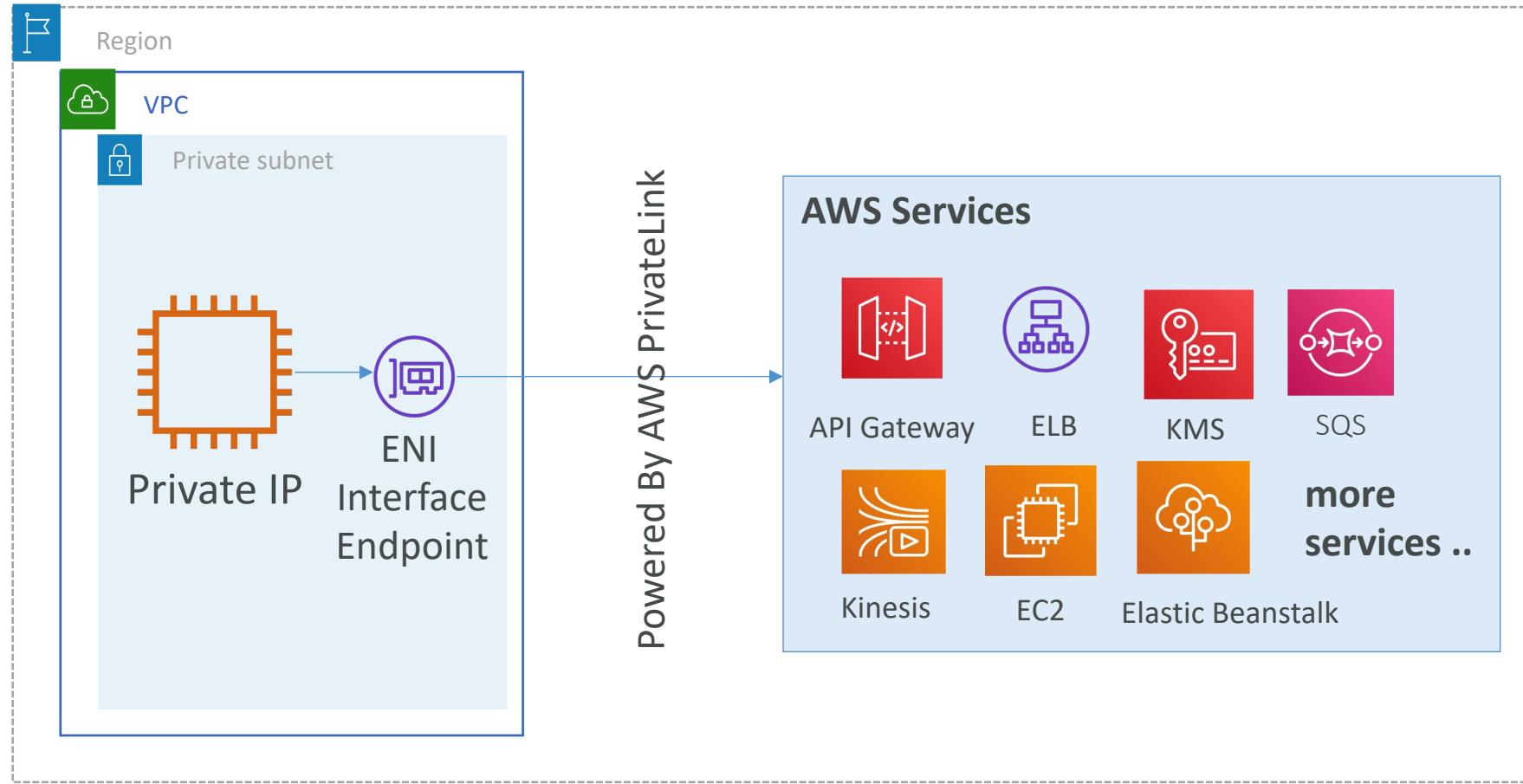
# VPC Interface Endpoint (PrivateLink)

# VPC Interface Endpoint – Accessing AWS or Customer Services



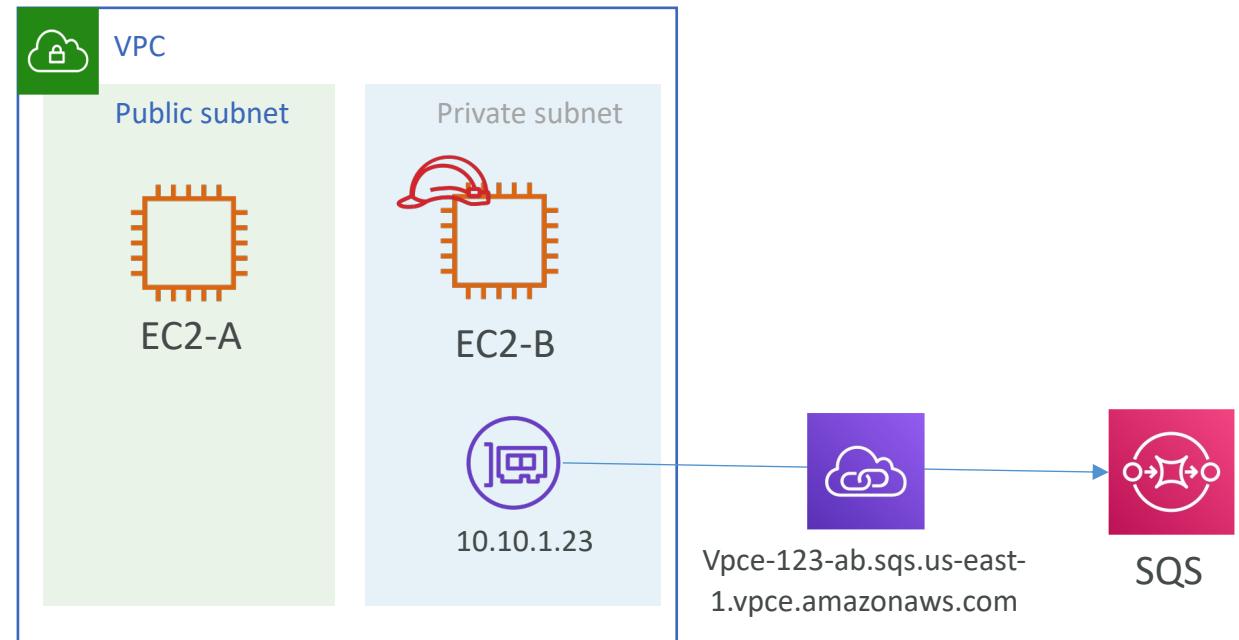
# VPC Interface Endpoint – Accessing AWS Services

# VPC Interface Endpoint – Accessing AWS Services



# Let's first create VPC interface endpoint

- Create a VPC and Public and Private Subnets
- Launch EC2 instance in both the subnets
- Create Interface endpoint for SQS service in Private Subnet
- Login to Private EC2 and Access SQS (PutMessage) using Interface endpoint



# Interface VPC Endpoint

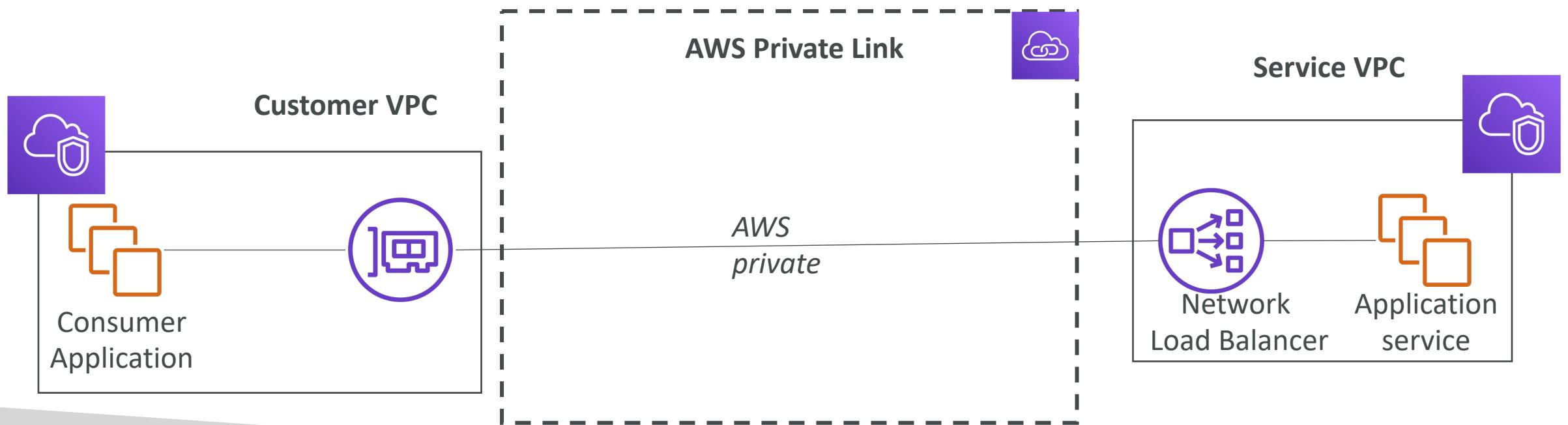
- Interface endpoints create local IP addresses (using ENI) in your VPC
- You create one interface endpoint per Availability zone for high availability
- Leverage Security Groups for security
- For interface endpoints, AWS creates Regional and zonal DNS entries that resolves to private IP address of interface
- Interface endpoint supports only IPv4 traffic

# VPC Interface endpoint – Accessing Customer service

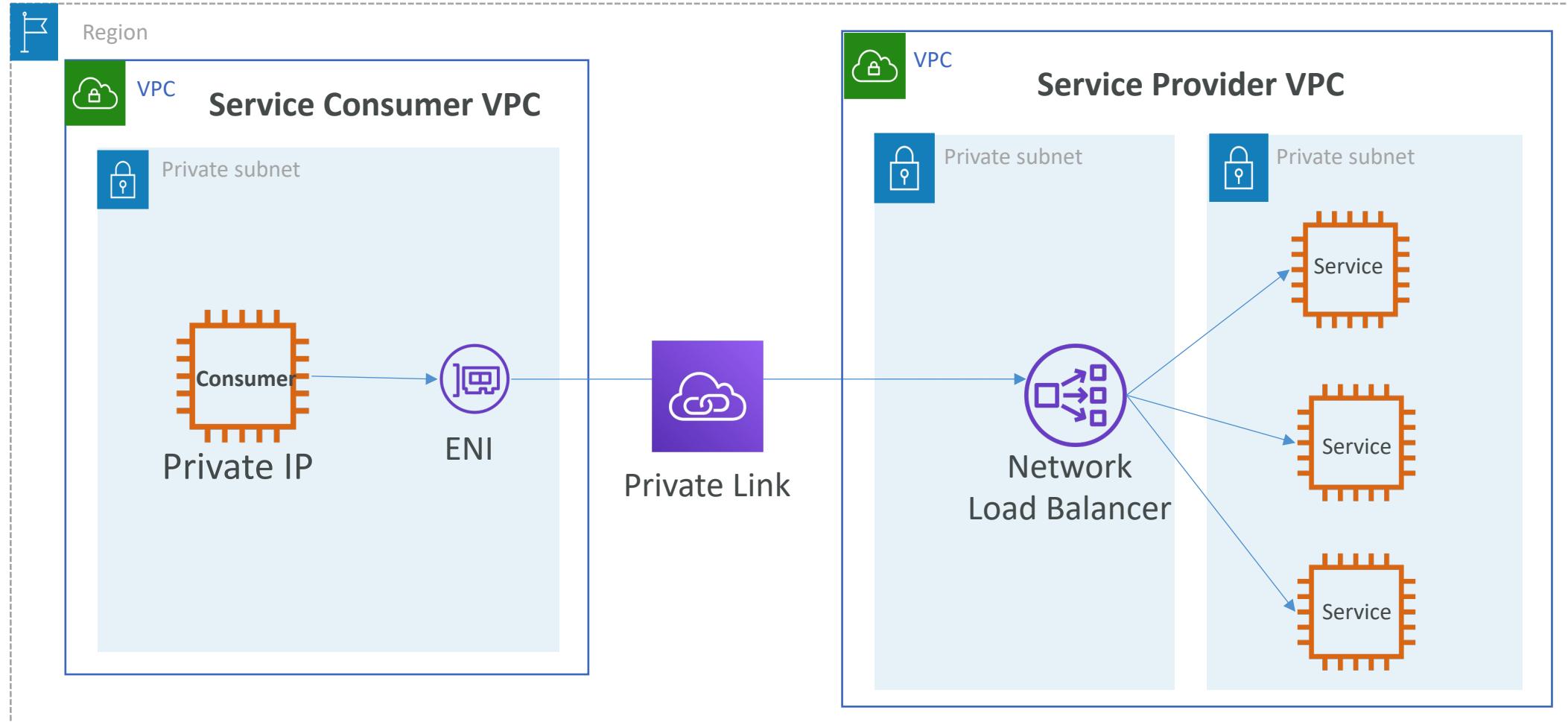
# AWS PrivateLink (VPC Endpoint Services)



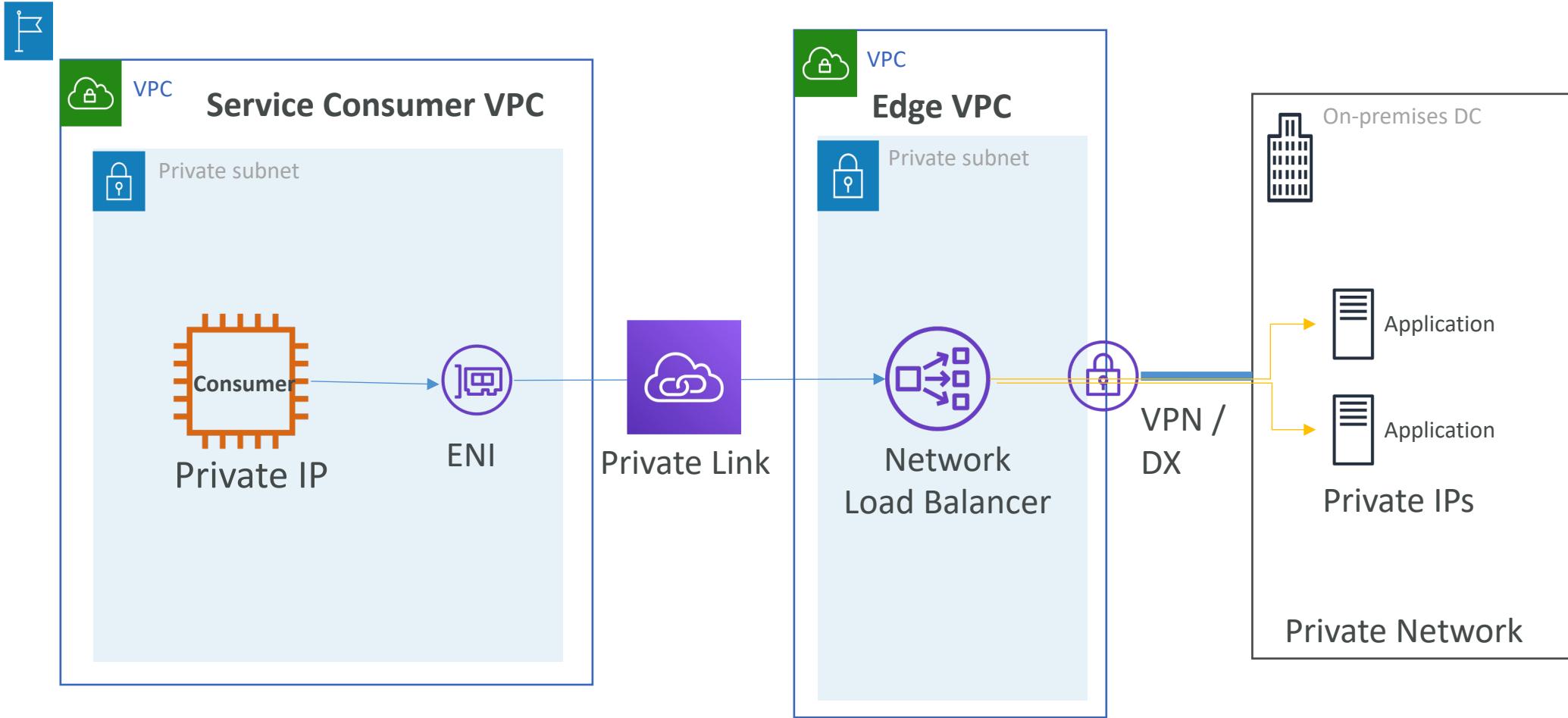
- Most secure & scalable way to expose a service to 1000s of VPC (own or other accounts)
- Does not require VPC peering, internet gateway, NAT, route tables...
- Requires a network load balancer (Service VPC) and ENI (Customer VPC)
- If the NLB is in multiple AZ, and the ENI in multiple AZ, the solution is fault tolerant!



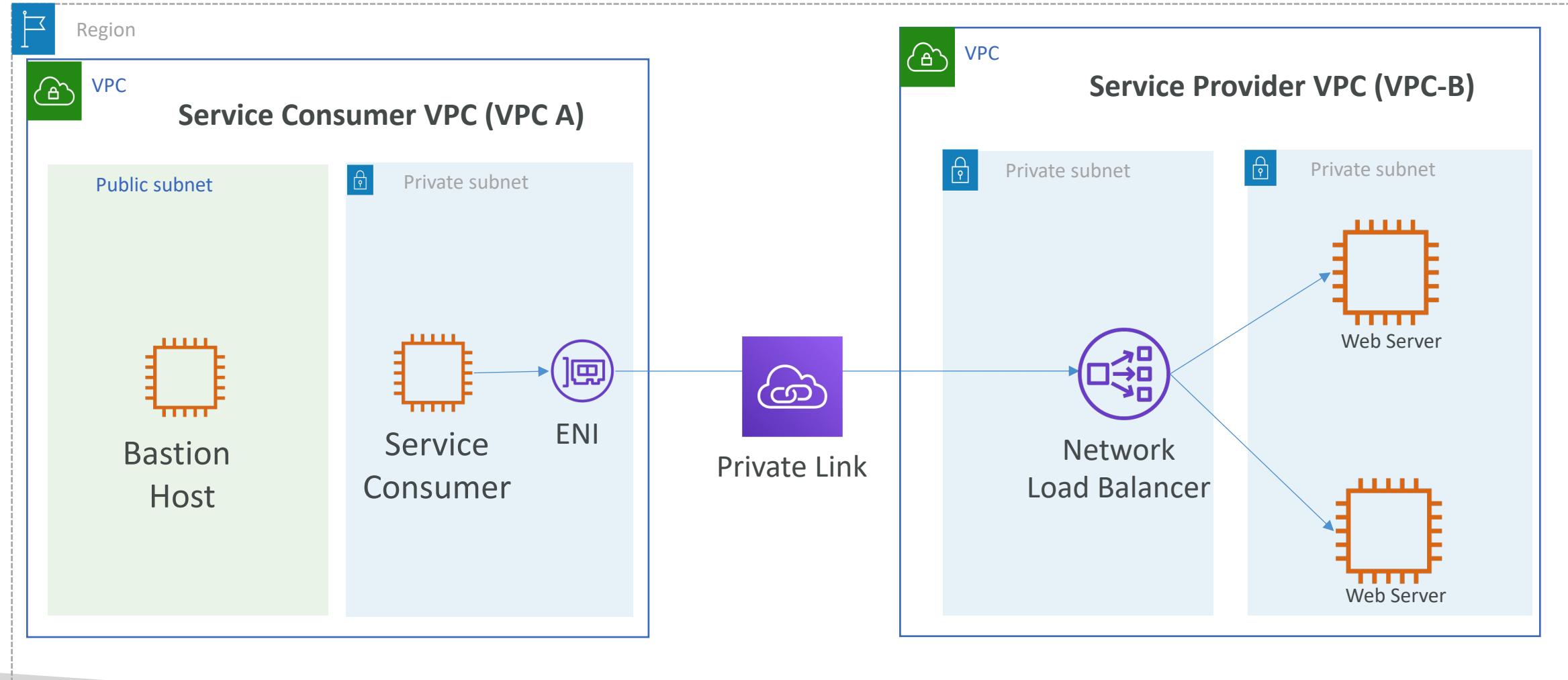
# Interface VPC Endpoint – Accessing Customer VPC services



# Interface VPC Endpoint – Accessing Customer On-premises services



# Accessing Customer Service - Demo



# Setup setups

- Pre-requisites – Create EC2 AMI having httpd webserver. We need this to launch Private EC2 instance in VPC-B to host the dummy service.
- Create VPC-B with 2 Private Subnets
- Launch EC2 instance in a Private Subnet using AMI created earlier
- Create NLB in another Private Subnet and Register EC2 instance behind NLB
- Create VPC Endpoint Service in VPC-B and associate NLB
- Whitelist the VPC-A AWS account (if both VPCs are in different AWS accounts)
- Create Service Consumer VPC (VPC-A) with Public and Private subnets
- Create VPC endpoint in VPC-A and search for endpoint service created above
- Login to Private EC2 instance in Consumer VPC and access VPC endpoint DNS

# How to create AMI

- Use below userdata to launch an EC2 instance in a default VPC
- Once launched, verify that you can access web page using Public IP
- Create AMI from this instance

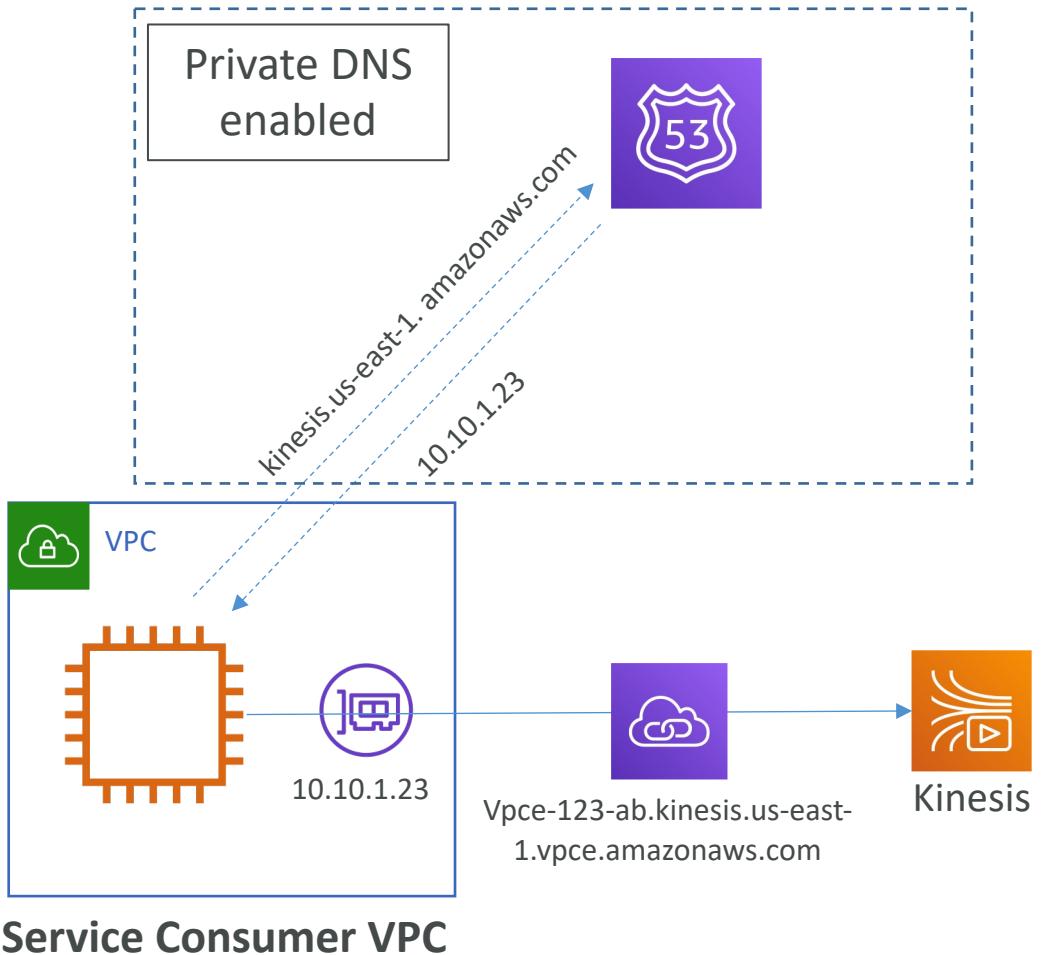
```
#!/bin/bash  
yum install -y httpd  
systemctl start httpd  
systemctl enable httpd  
echo THIS IS A SERVICE HOSTED BEHIND AWS PRIVATELINK >  
/var/www/html/index.html
```

# VPC Interface Endpoint – DNS

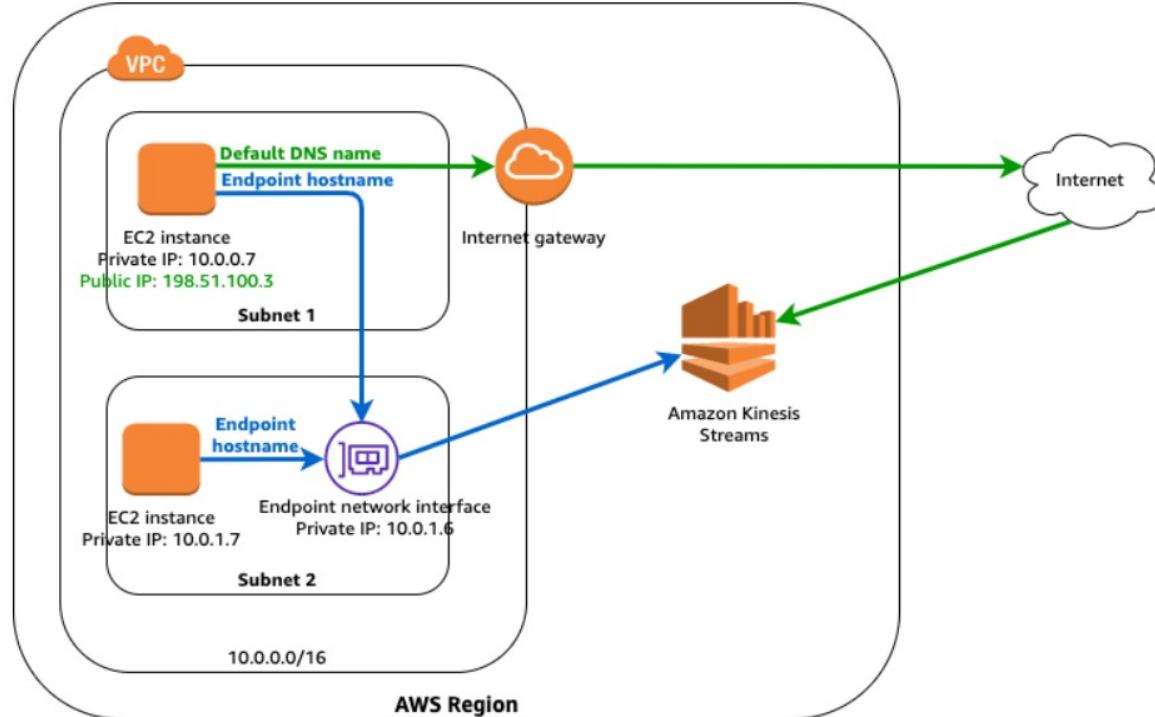
# VPC Interface Endpoint

- Provisions an ENI that will have a private endpoint interface hostname
- Private DNS settings for Interface endpoint
  - The public hostname of a service will resolve to the private Endpoint Interface hostname
  - VPC Setting: "Enable DNS hostnames" and "Enable DNS Support" must be 'true'
  - Example for Athena:
    - Regional: vpce-0b7d2995e9dfe5418-mwrths3x.athena.us-east-1.vpce.amazonaws.com
    - Zonal: vpce-0b7d2995e9dfe5418-mwrths3x-us-east-1a.athena.us-east-1.vpce.amazonaws.com
    - Zonal: vpce-0b7d2995e9dfe5418-mwrths3x-us-east-1b.athena.us-east-1.vpce.amazonaws.com
    - Service DNS: athena.us-east-1.amazonaws.com (private DNS name)
- With Private DNS enabled, the consumer VPC can access the endpoint services using Service's default DNS e.g **ec2.us-east-1.amazonaws.com** instead of using endpoint specific DNS e.g vpce-12345-ab.ec2.us-east-1.vpce.amazonaws.com

# VPC Interface Endpoint DNS

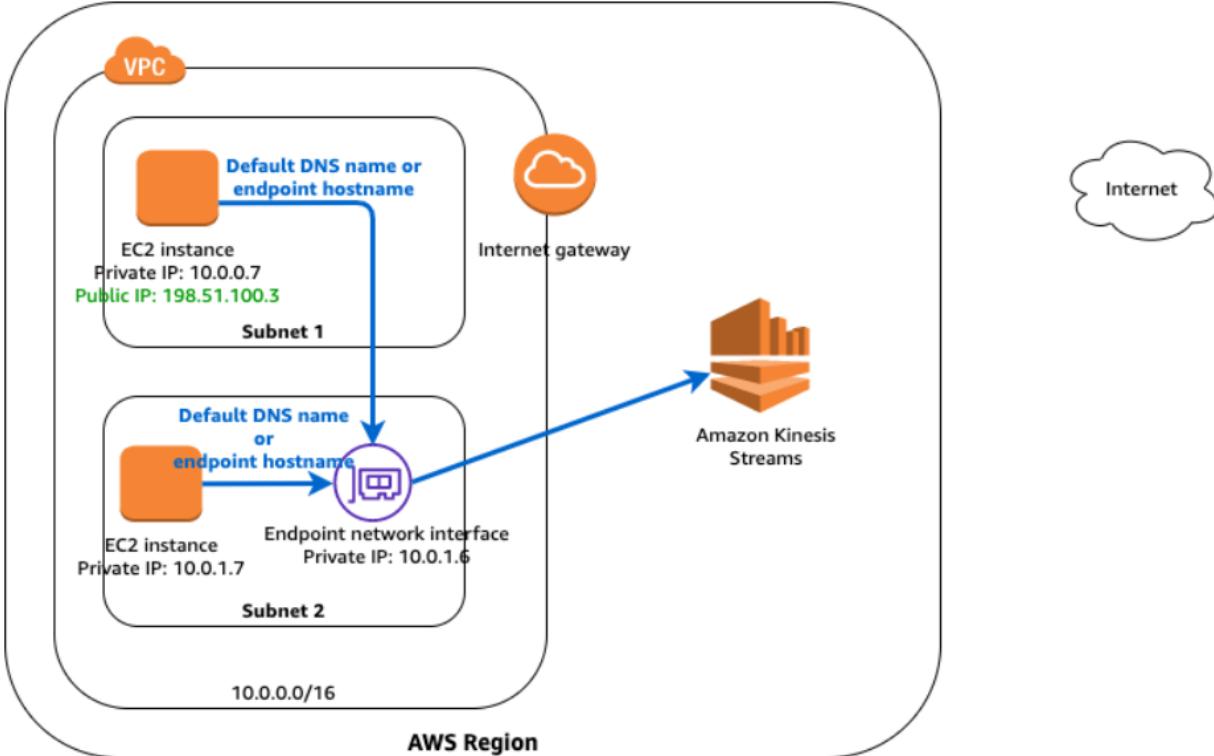


# VPC Interface Endpoint – Private DNS disabled



- Subnet 1 -> kinesis.us-east-1.amazonaws.com -> Via internet
- Subnet 1 -> vpce-123-ab.kinesis.us-east-1.vpce.amazonaws.com -> Via Interface endpoint
- Subnet 2 (No IGW) -> Only option is via interface endpoint using interface-endpoint DNS

# VPC Interface Endpoint – Private DNS enabled



- Subnet 1/2 -> kinesis.us-east-1.amazonaws.com -> Via Interface endpoint
- Subnet 1/2 -> vpce-123-ab.kinesis.us-east-1.vpce.amazonaws.com -> Via Interface endpoint

# VPC endpoints DNS Summary

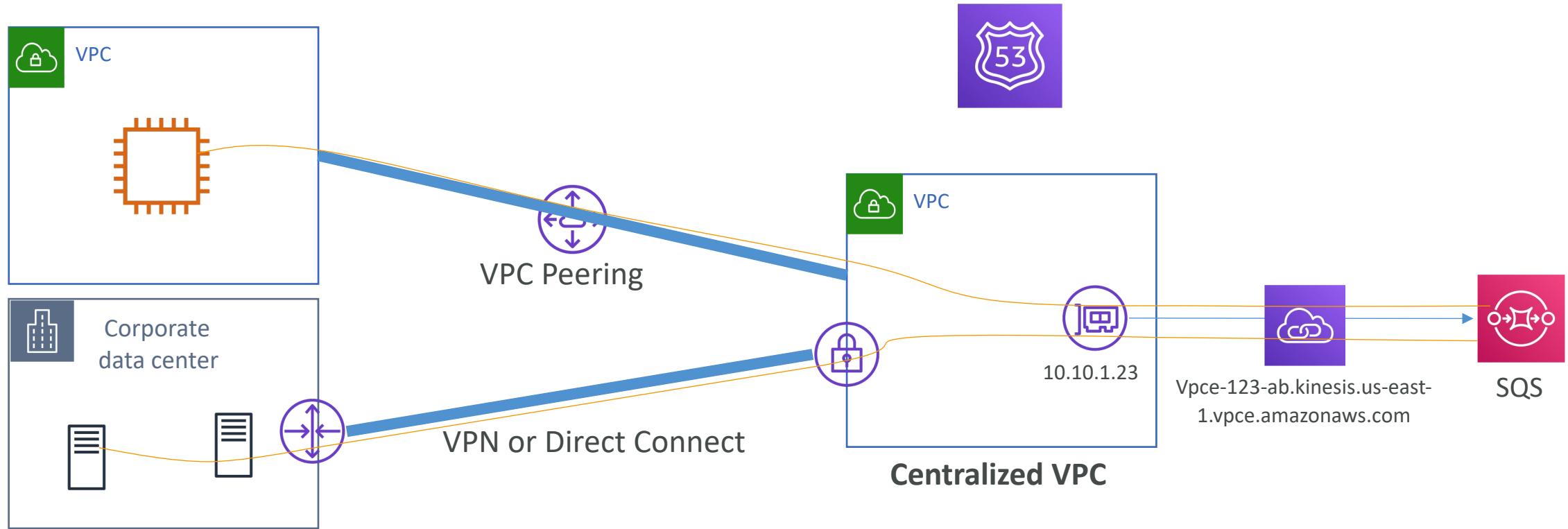
- With Private DNS enabled
  - The public hostname of a service will resolve to the private Endpoint Interface hostname
  - For AWS services (e.g SQS, Kinesis etc) AWS creates Private Hosted Zone and associate it with your VPC
  - Example: With Private DNS enabled, the consumer VPC can access the endpoint services using Service's default DNS e.g `ec2.us-east-1.amazonaws.com` instead of using endpoint specific DNS e.g `vpce-12345-ab.ec2.us-east-1.vpce.amazonaws.com`
- VPC Setting: “Enable DNS hostnames” and “Enable DNS Support” must be ‘true’

# VPC Interface Endpoint – Remote Access

# VPC Interface endpoint

- An Interface endpoint can be accessed through
  - Direct Connect
  - AWS Managed VPN
  - VPC peering connection

# Centralized VPC Interface endpoint



- Peered VPCs can resolve the private DNS of Interface endpoint if attached to custom Route53 Private Hosted Zone
- For On-premise DNS resolution the DNS queries should be forwarded to custom Route53 Resolver
- Reference: <https://d1.awsstatic.com/whitepapers/building-a-scalable-and-secure-multi-vpc-aws-network-infrastructure.pdf>

# VPC PrivateLink vs VPC Peering

# AWS PrivateLink vs VPC Peering

- VPC peering is useful when there are many resources that should communicate between peered VPCs
- PrivateLink should be used when you want to allow access to only single application hosted in your VPC to other VPCs without peering the VPCs
- When there is overlapping CIDRs, VPC peering connection can not be created. However private link does support overlapping CIDR
- We can create a maximum of 125 peering connections. There is no limit on private link connections.
- VPC peering enables bidirectional traffic origin. PrivateLink allows only consumer to originate the traffic.

# Summary

# Summary

- VPC peering enables the communication between 2 VPCs in same region or across regions
- A VPC endpoint enables you to privately connect your VPC to supported AWS services and Services powered by PrivateLink without requiring an internet gateway, NAT device, VPN connection, or AWS Direct Connect connection.
- Instances in your VPC do not require public IP addresses to communicate with resources in the service.
- Traffic between your VPC and the other service does not leave the Amazon network.
- Gateway VPC endpoints allow you to connect to services but do not exist as a networking component in your VPC
- Interface VPC endpoints are elastic network interfaces, private IP addresses, and DNS names that allow you to access AWS Cloud services privately from within your VPC
- AWS PrivateLink is an extension of interface VPC endpoints, and it allows you to create your own endpoints or consume endpoints that others have created

# Exam Essentials

# Exam essentials

- VPC peering does not support transitive routing. Hence you can not access IGW, NAT of other VPC through peering connection
- VPC peering allows using Security group as source for inbound rules from other VPC
- You can create maximum 125 VPC peering connections
- VPC gateway endpoint enables private connectivity to S3 or DynamoDB in same AWS region without requiring IGW or NAT
- To route traffic through gateway endpoint, you should modify the route table of the required subnet
- Gateway endpoint is not accessible over Direct Connect/VPN or VPC Peering

# Exam essentials

- VPC interface endpoint create an ENI into your subnets
- Interface endpoint receives the Regional and zonal DNS name.
- You can also use Route53 private hosted zone to use your custom DNS with Alias record to interface DNS
- Interface endpoint can be accessed over Direct Connect connection AWS managed VPN and VPC peering connection
- Traffic originates from the resources in the VPC and endpoint service can only respond to the request. Endpoint service cannot initiate the request.

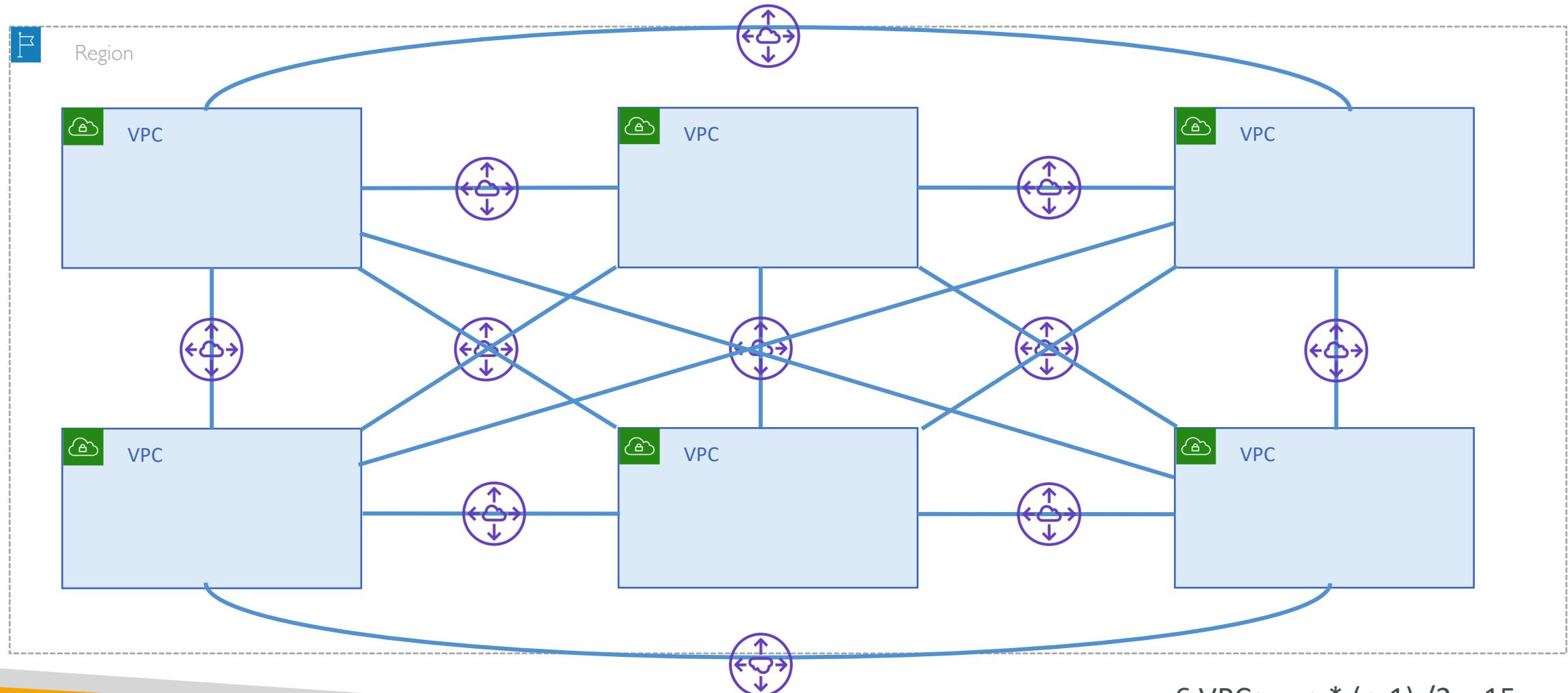
# Transit Gateway (TGW)



# What is Transit Gateway?

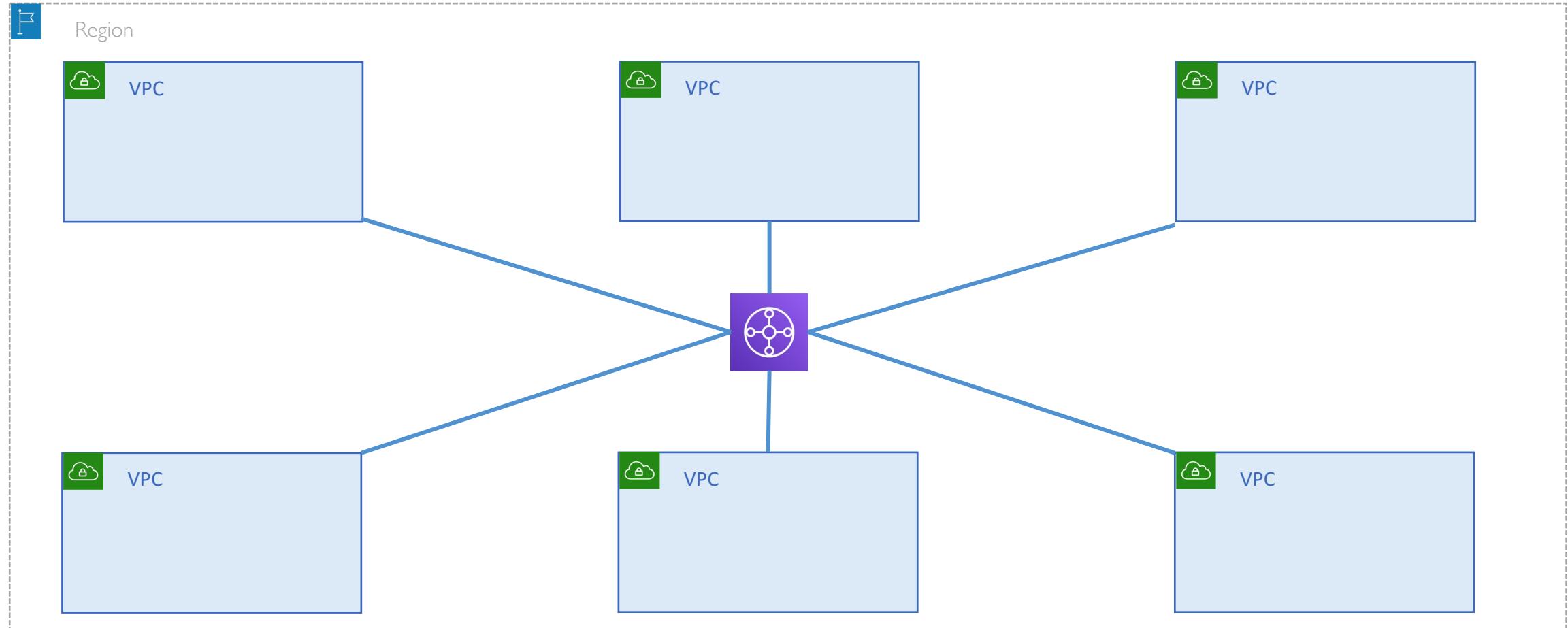
- Allows customers to interconnect thousands of Virtual Private Clouds (VPCs) and on-premises networks.
- Transit Gateway supports attachments to
  - One or more VPCs
  - VPN
  - Direct Connect Gateway
  - Peering connection with another Transit Gateway
  - A Connect SD-WAN/third-party network appliance

# Multiple VPCs without Transit Gateway

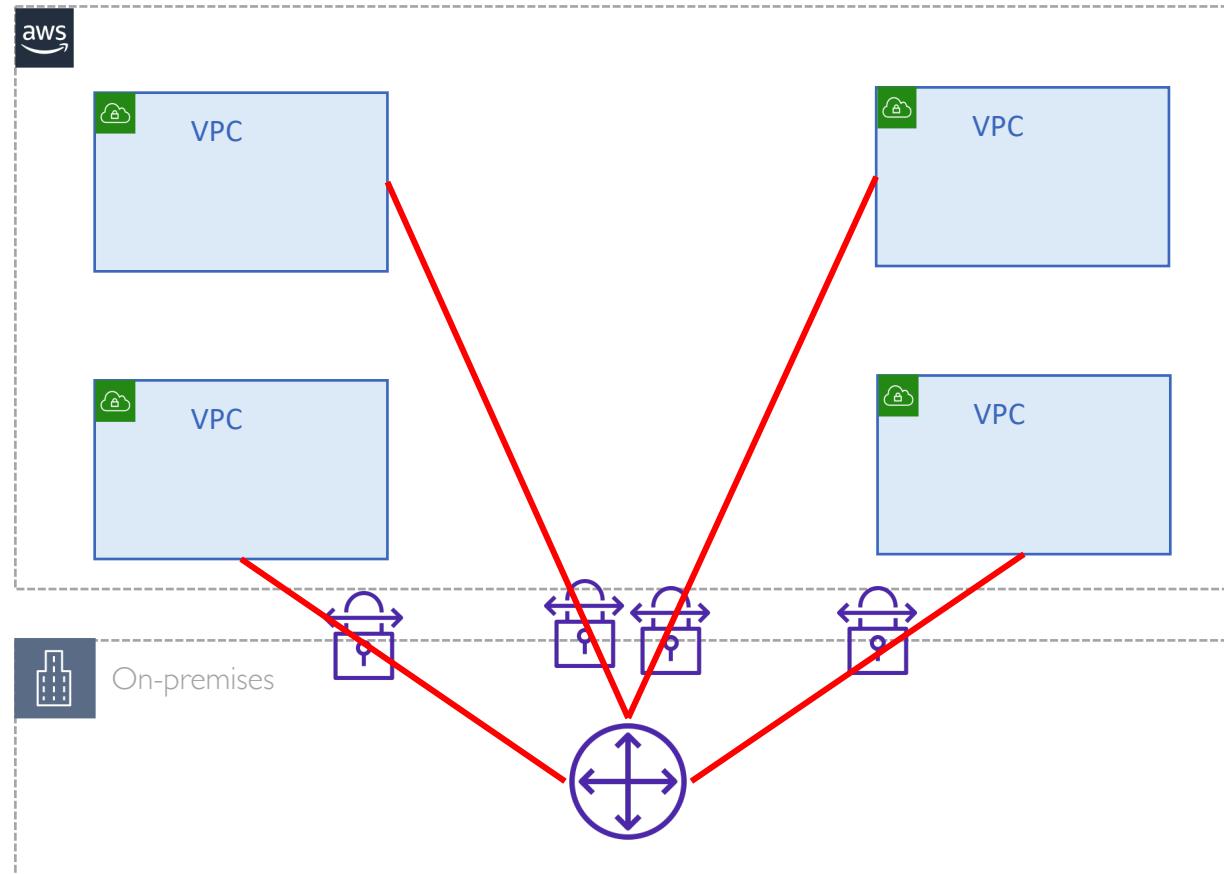


$$6 \text{ VPCs} = n * (n-1) / 2 = 15$$

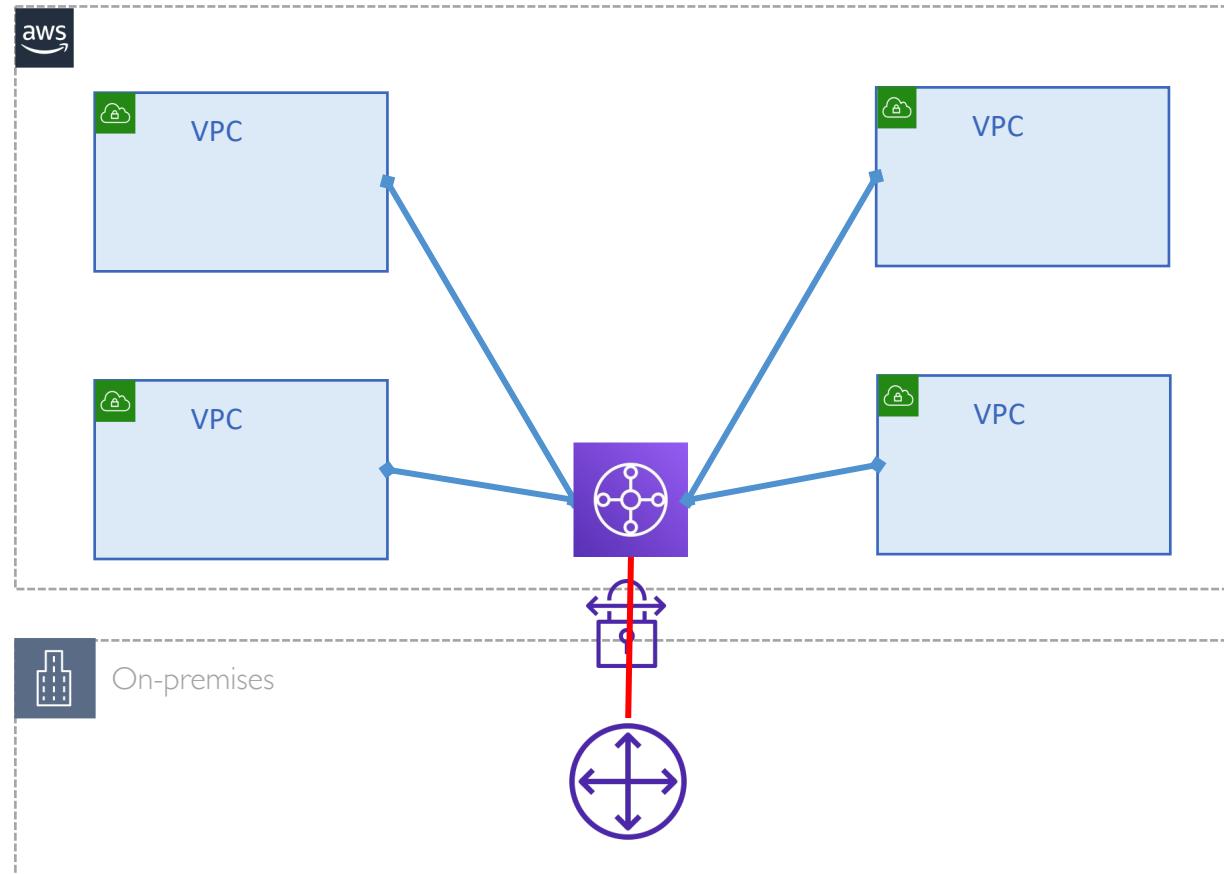
# Multiple VPCs with Transit Gateway



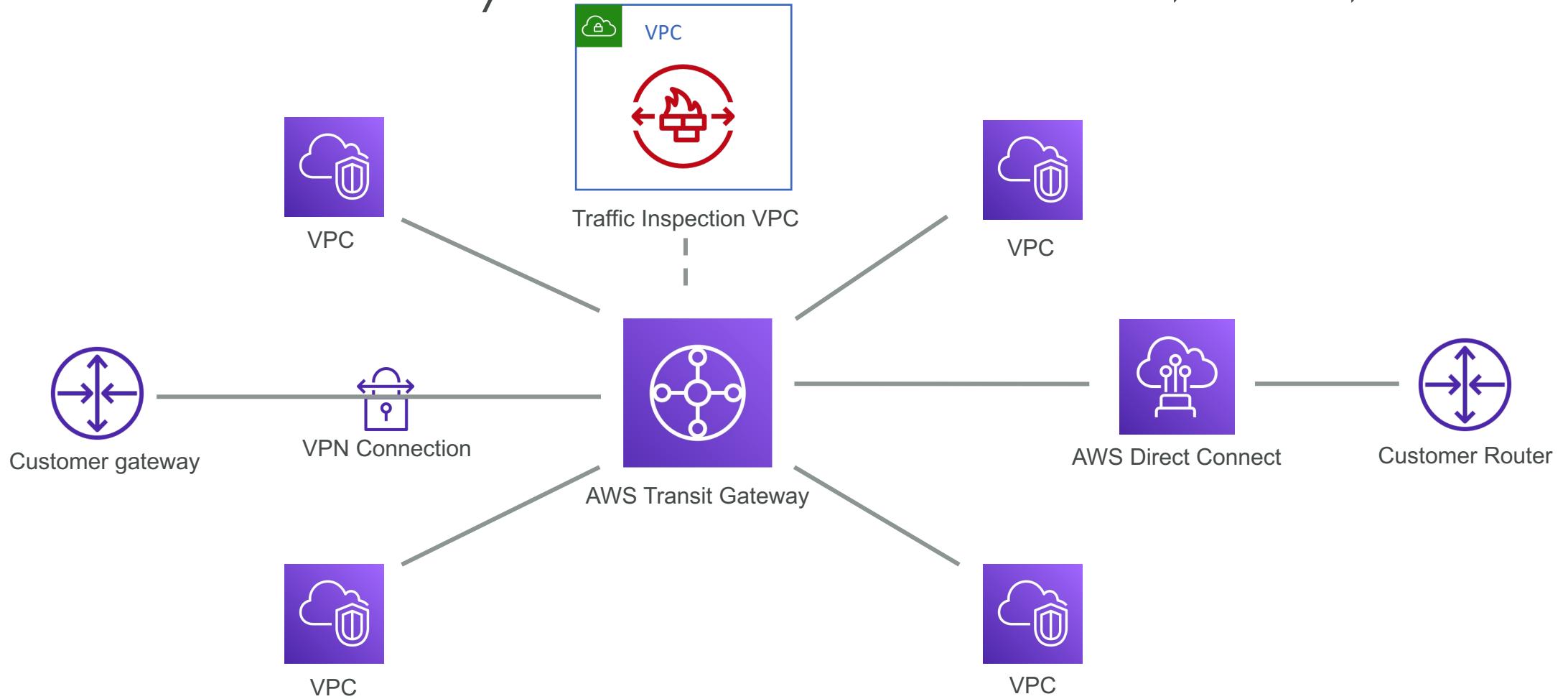
# Multiple VPCs and VPN



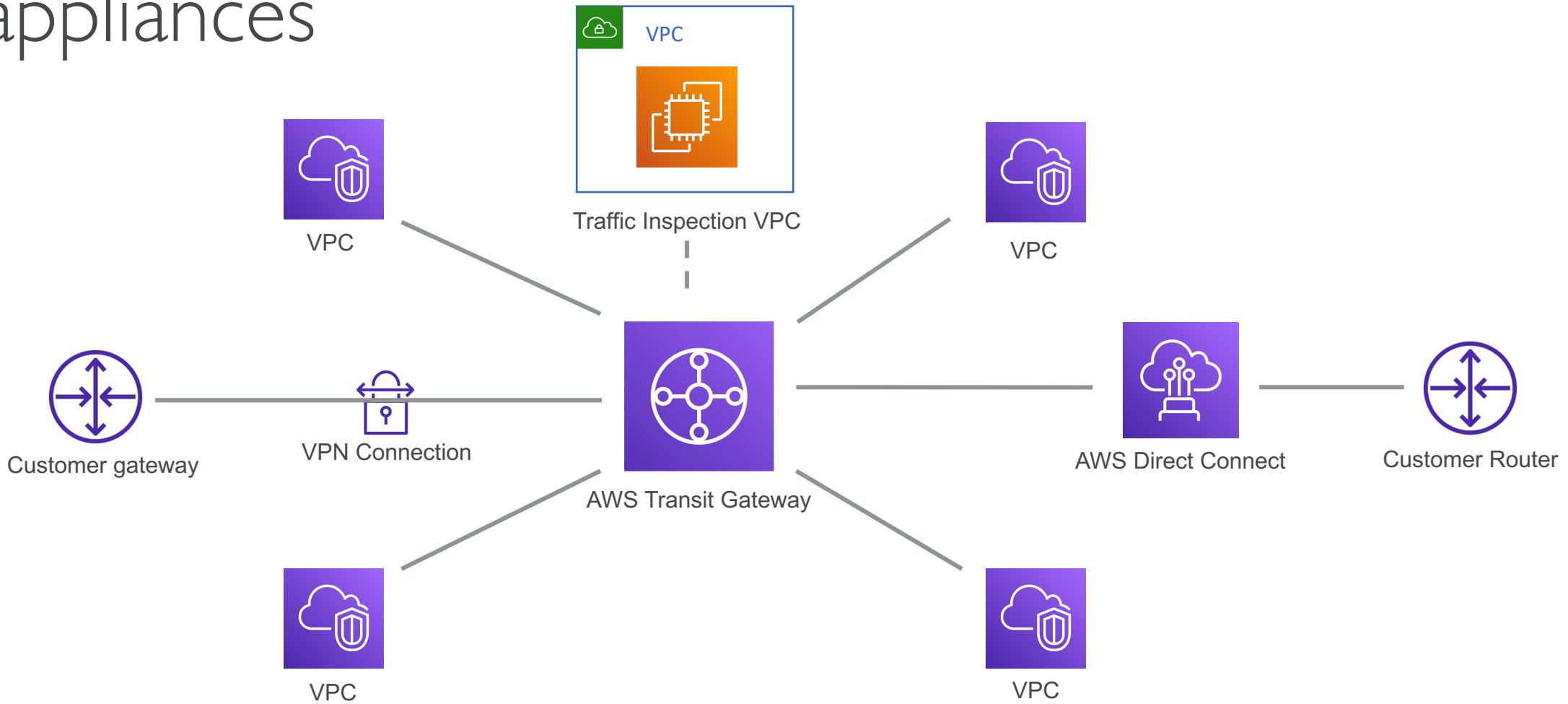
# Multiple VPCs and VPN with Transit Gateway



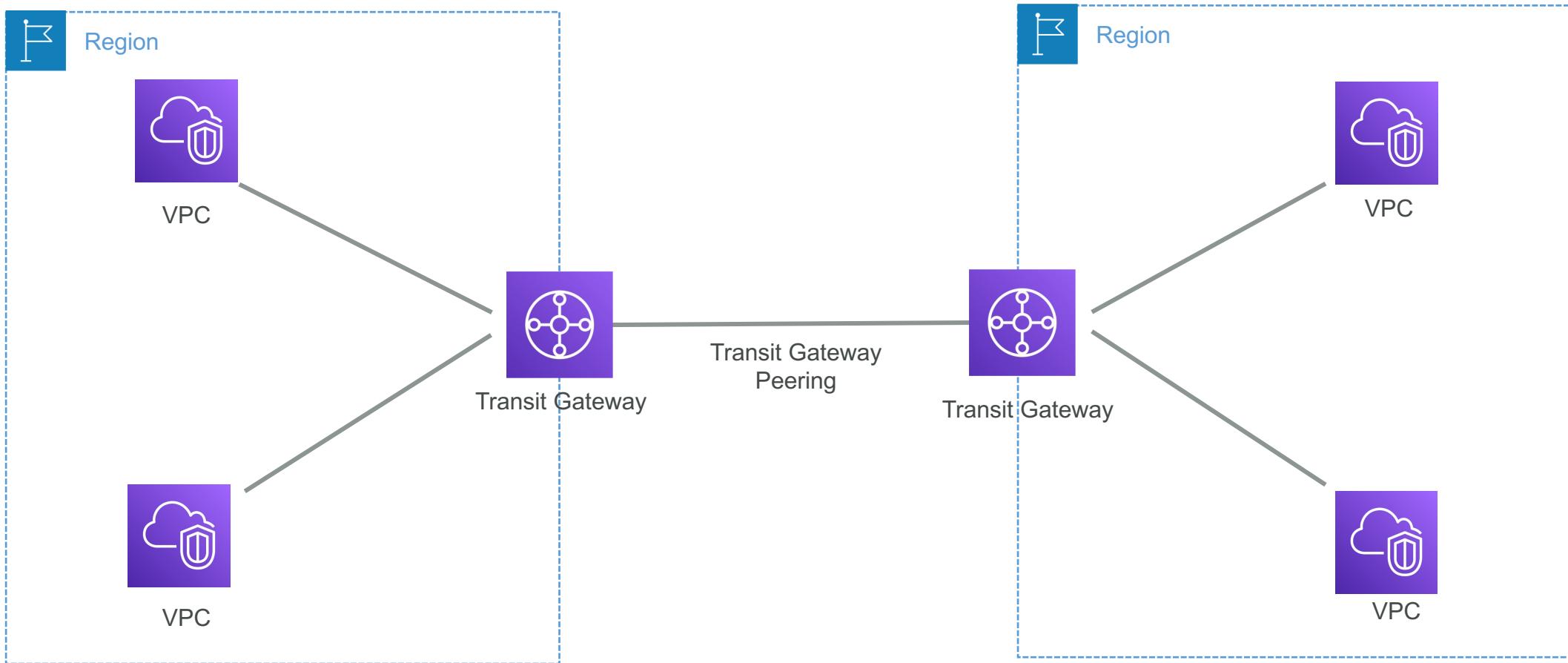
# Transit Gateway attachments – VPC,VPN, DX



# Transit Gateway attachments – 3<sup>rd</sup> party appliances

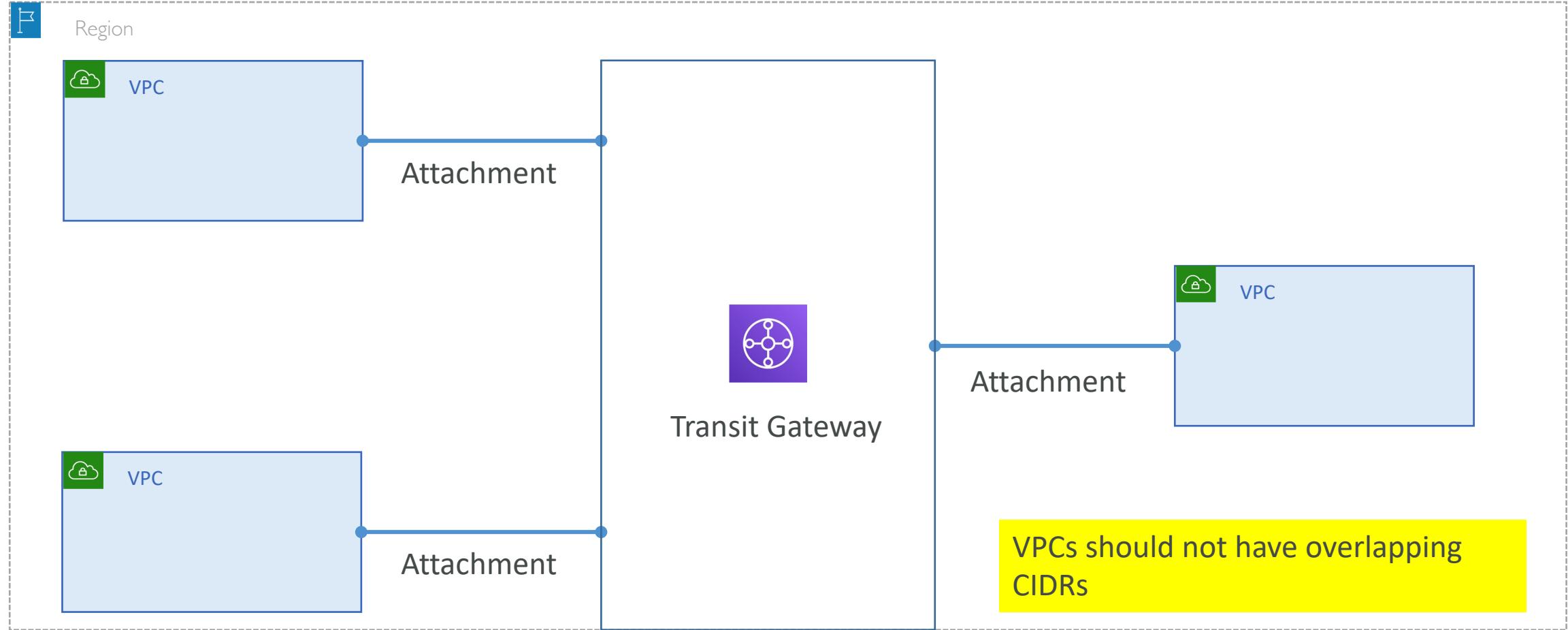


# Transit Gateway attachments - Peering

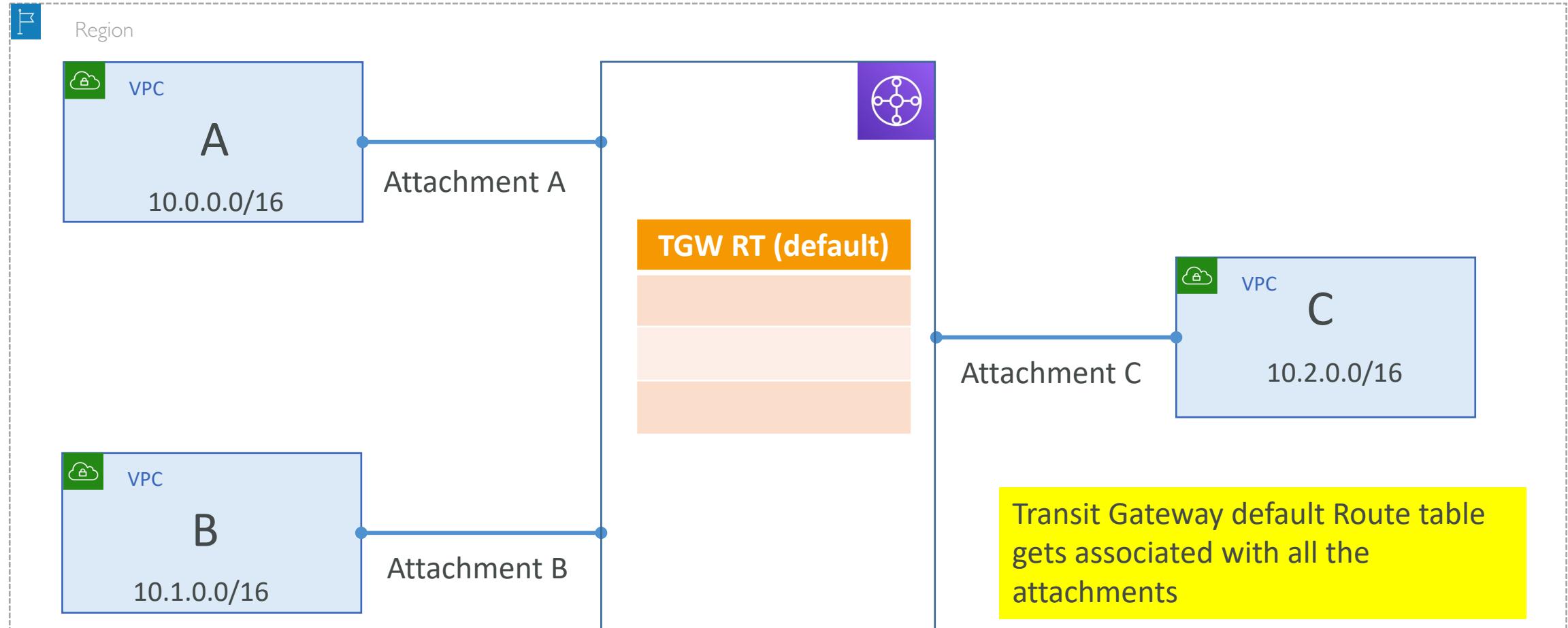


# Transit Gateway VPC attachments

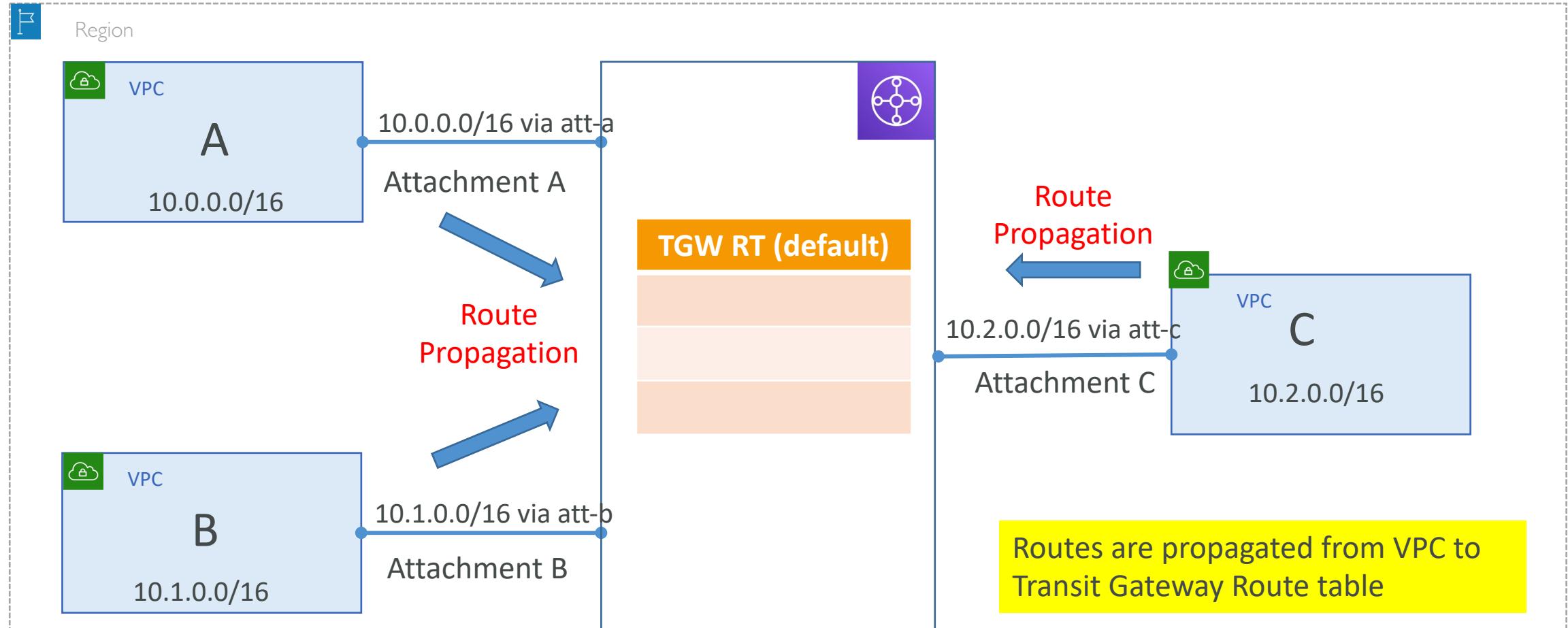
# Transit Gateway attachments



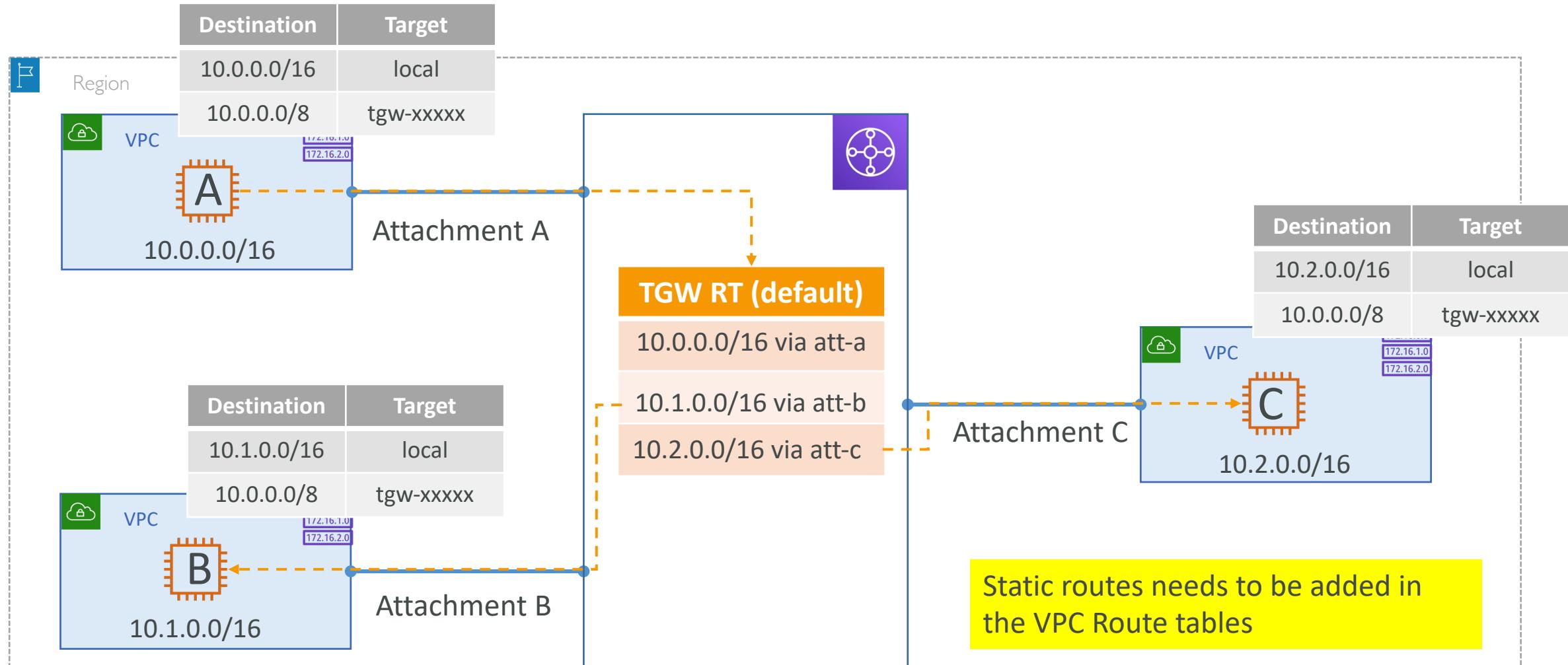
# Transit Gateway Route tables



# Transit Gateway Route tables

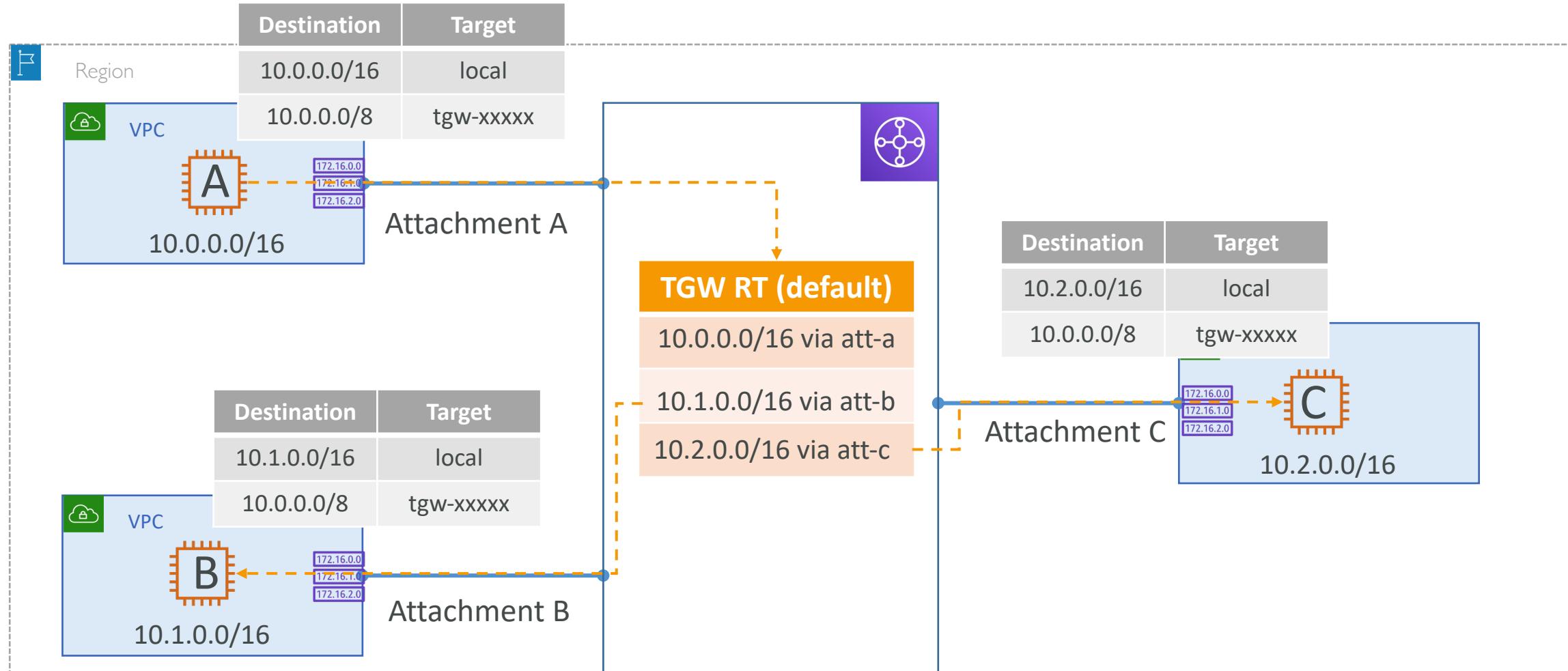


# VPC Route tables

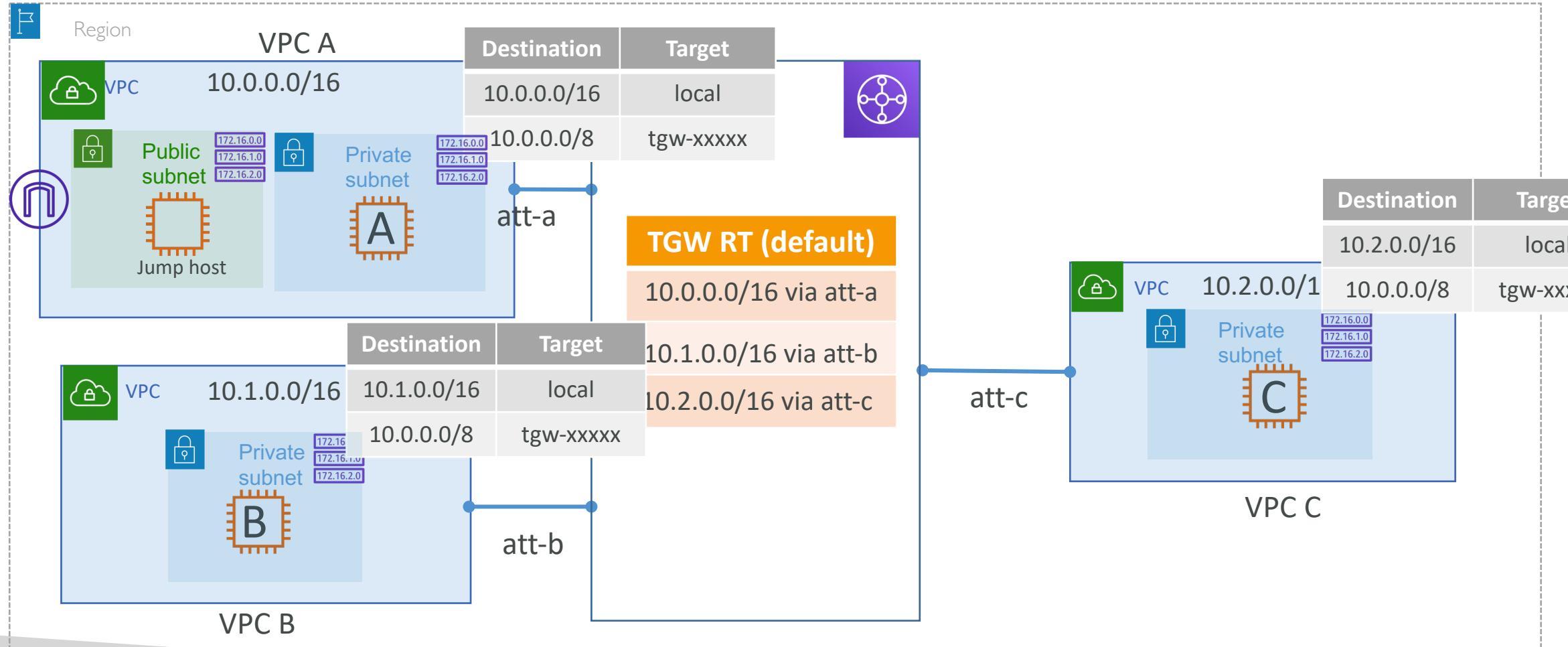


# Transit Gateway Lab – Three VPCs with full connectivity

# Lab – Let's set this up and test connectivity



# Lab setup

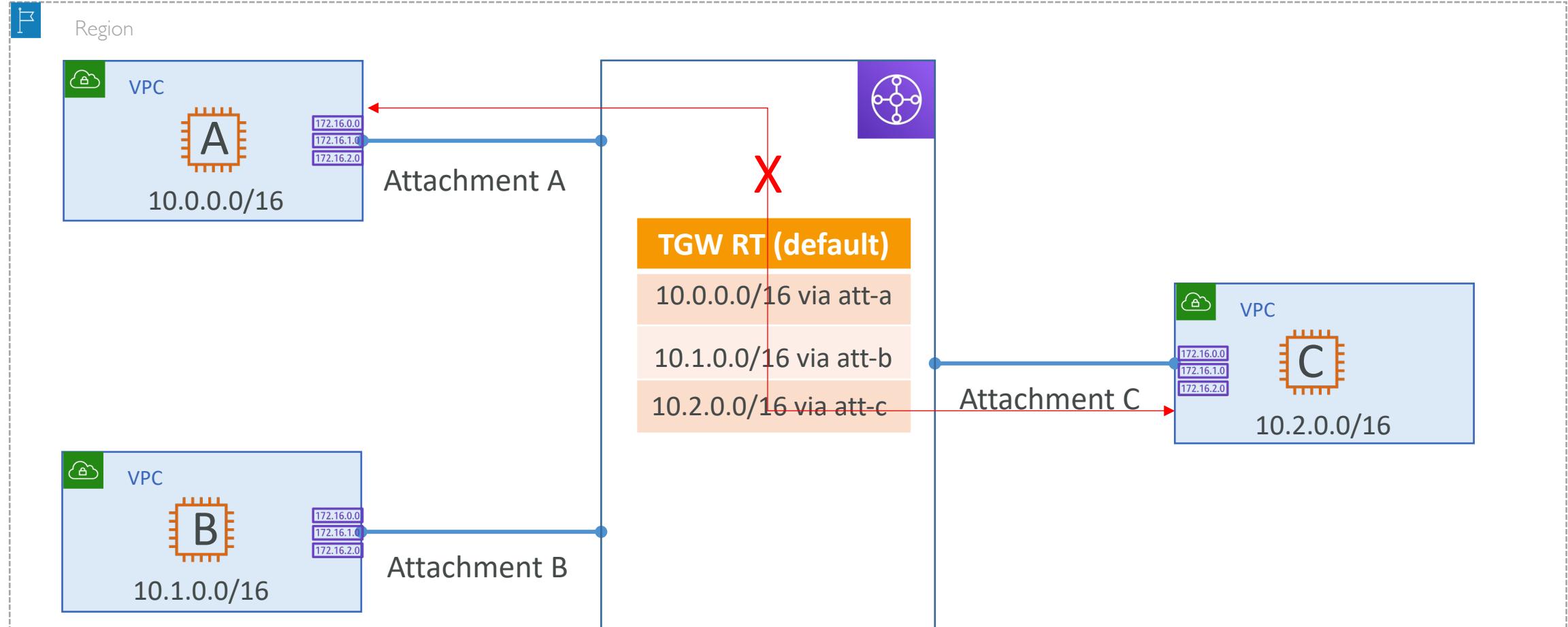


# Lab Steps

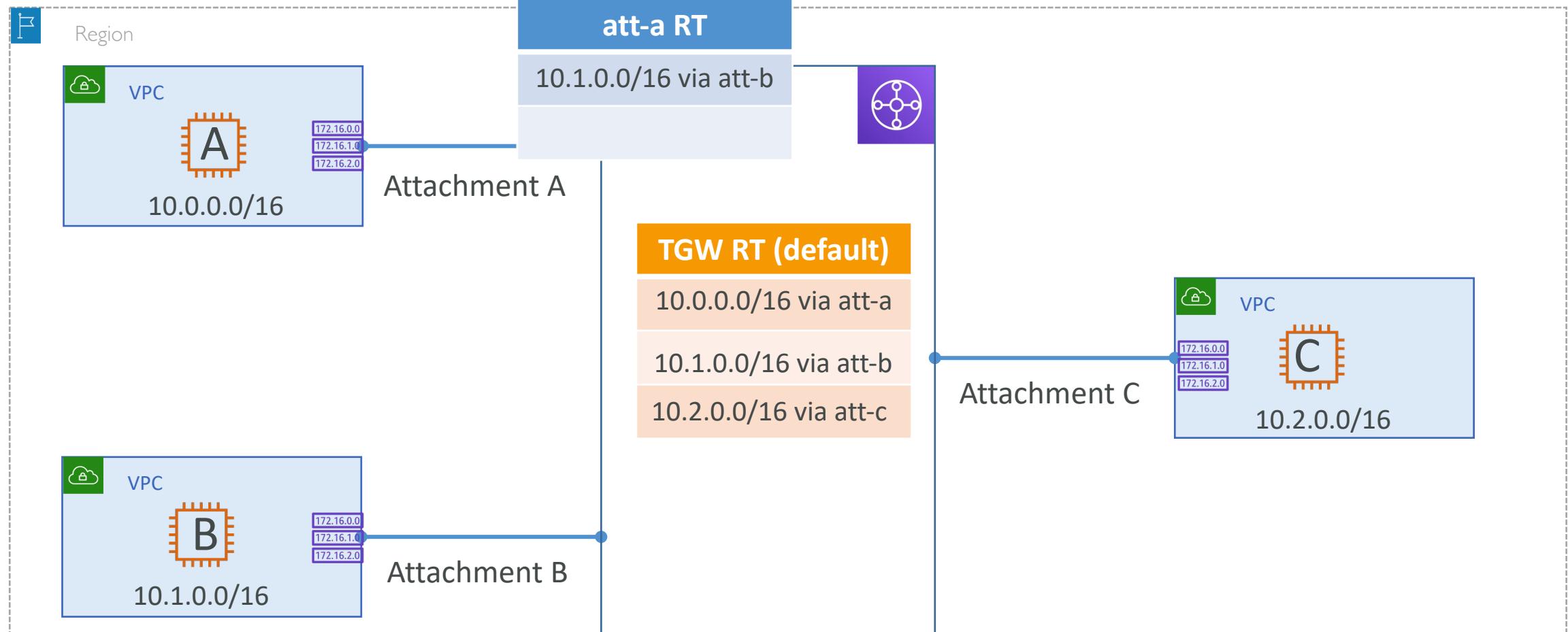
1. Create 3 VPCs and corresponding private subnets. For VPC A – additionally create a Public Subnet (we need that for jump host)
2. Create Transit Gateway
3. Create 3 VPC attachments for the Transit Gateway
4. Modify all Private subnet route table and add route for 10.0.0.0/8 via the Transit gateway attachment
5. SSH to VPC A Jump host -> SSH to EC2-A -> Ping to EC2-B or EC2-C using a Private IP

# Transit Gateway Attachment specific routing

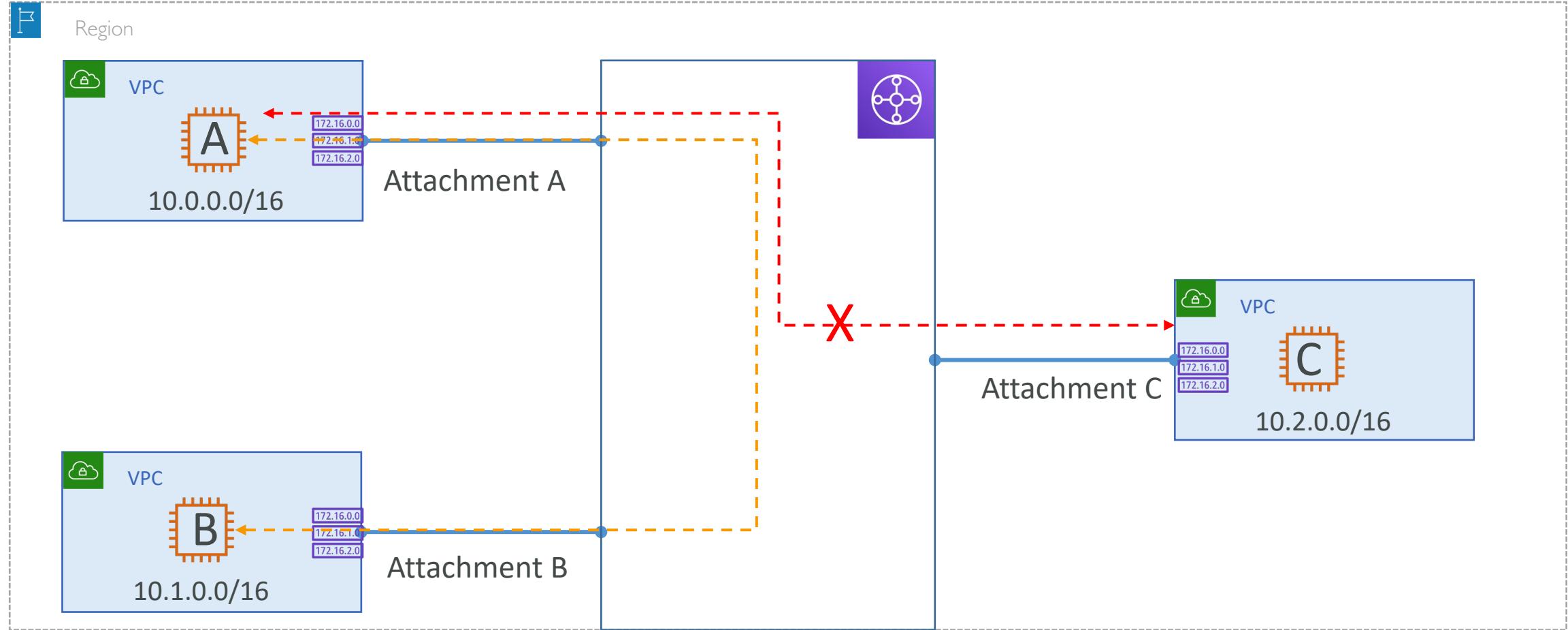
# Routing control with attachment Route Tables



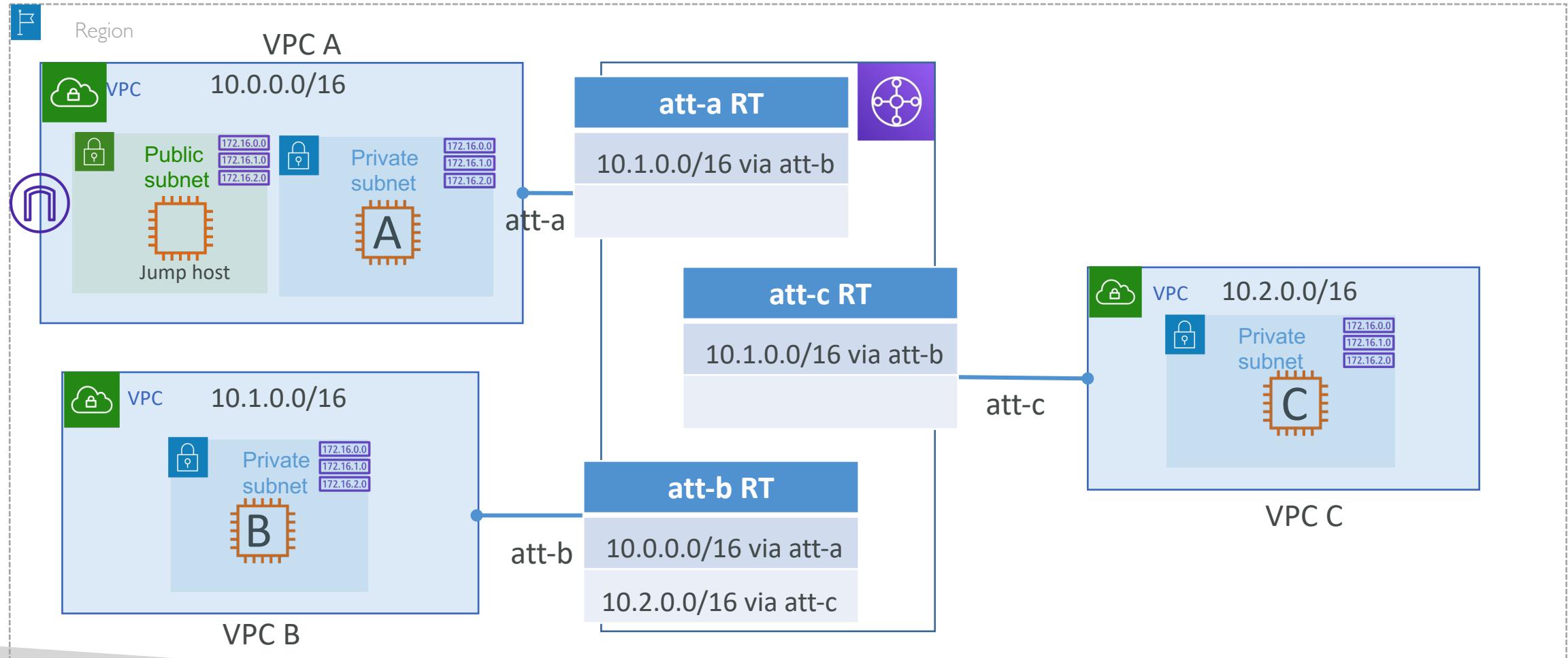
# Create attachment specific route table



# Lab – Let's set this up and test connectivity



# Lab setup

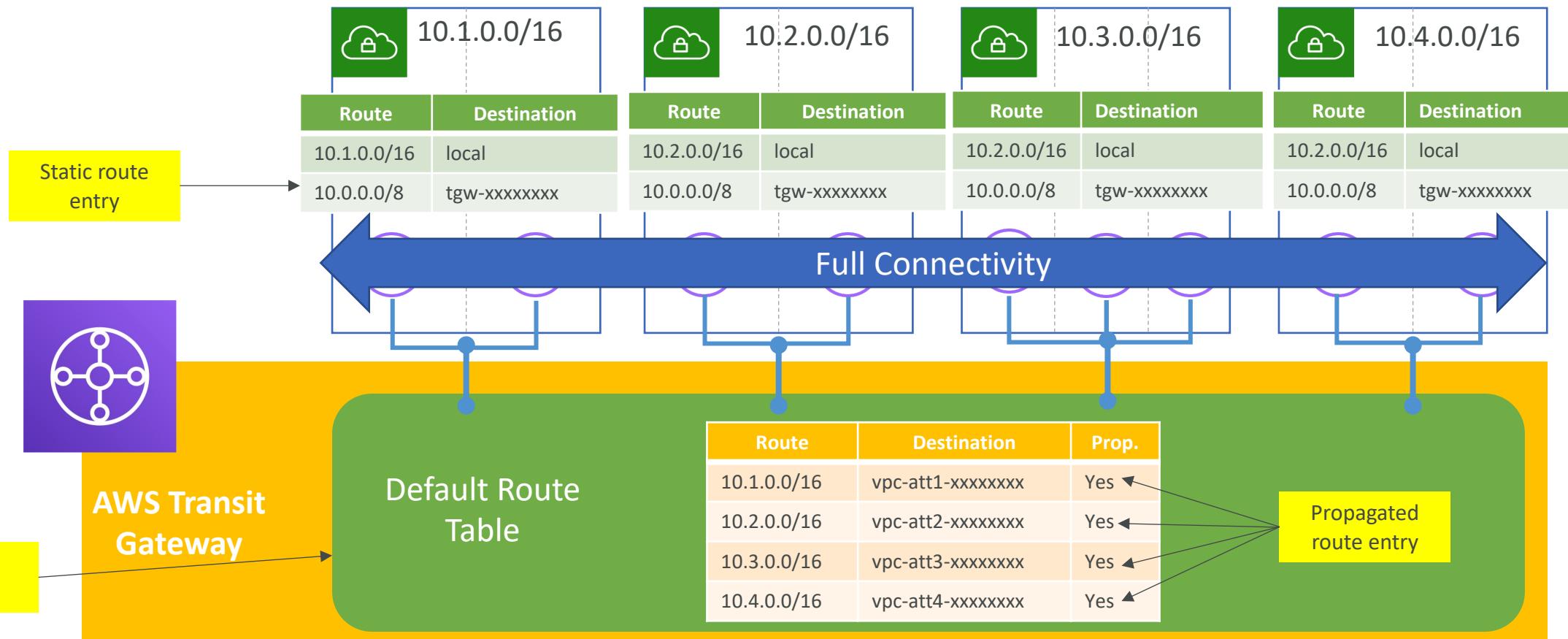


# Lab Steps

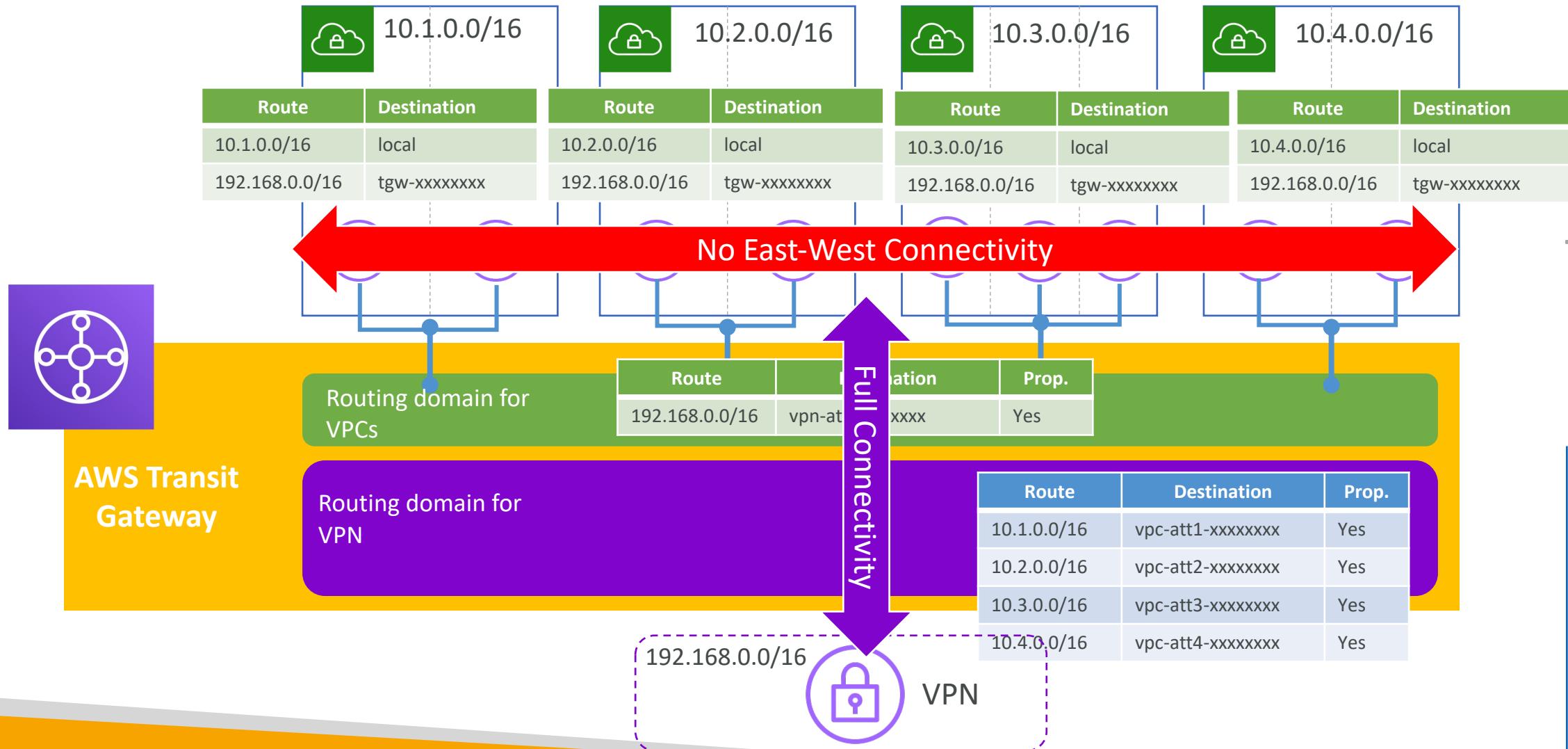
1. Create 3 VPCs and corresponding private subnets. For VPC A –additionally create a Public Subnet (we need that for jump host)
2. Create Transit Gateway (Do not enable default route table association and default route table propagation)
3. Create 3 Transit gateway VPC attachments for each VPC
4. Create 3 transit gateway route tables and associate each with corresponding VPC attachments.
5. For att-a route table add the route propagation for att-b
6. For att-b route table add the route propagation for att-a and att-c
7. For att-c route table add the route propagation for att-b
8. SSH to VPC A Jump host -> SSH to EC2-A -> Ping to EC2-B or EC2-C using a Private IP. Connectivity to EC2-B should work however EC2-C should be denied.
9. Similarly from EC2-B try to SSH/Ping to EC2-A and EC2-C. Connectivity to both the instances should be successful

# Transit Gateway – VPC Network patterns

# Flat network

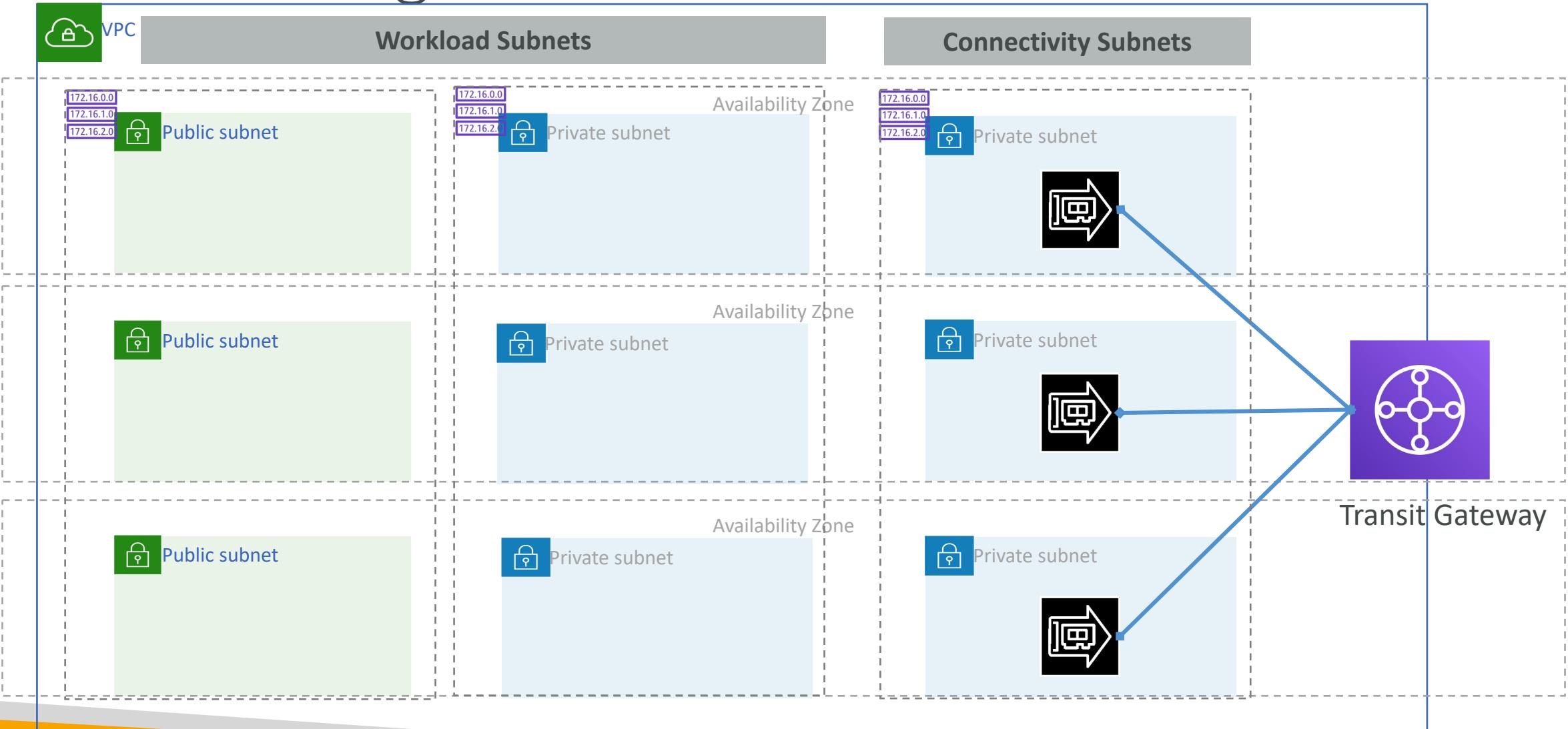


# Segmented Network



# VPC & Subnets design for Transit Gateway

# VPC design for TGW

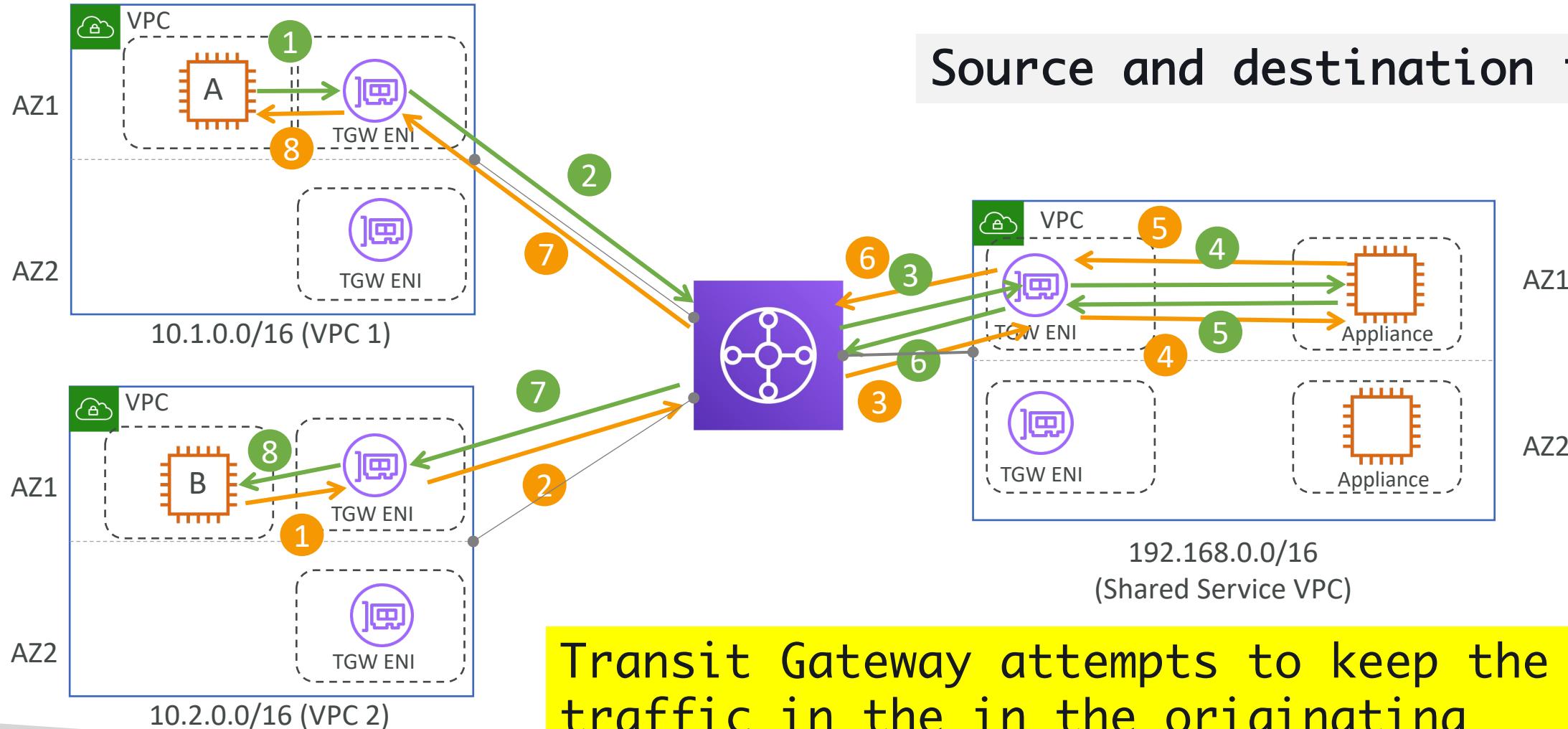


# Availability zone considerations

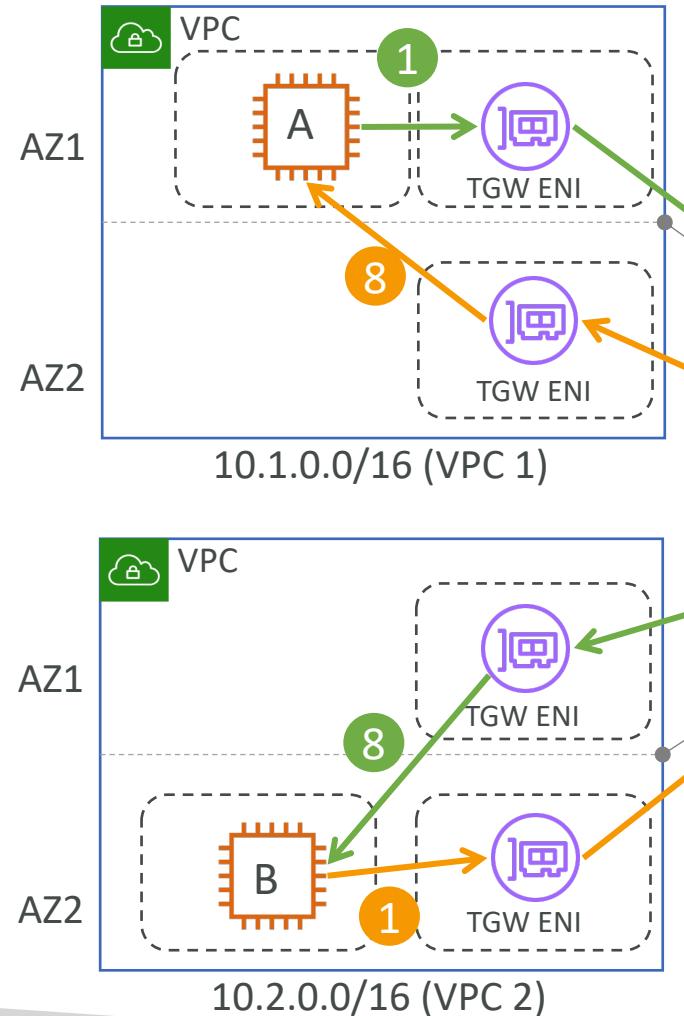
- When you attach a VPC to a transit gateway, you must enable one or more Availability Zones to be used by the transit gateway to route traffic to resources in the VPC subnets
- To enable each Availability Zone, you specify exactly one subnet (typically /28 range to save IPs for workload subnets)
- The transit gateway places a network interface in that subnet using one IP address from the subnet
- After you enable an Availability Zone, traffic can be routed to all subnets in that zone, not just the specified subnet
- Resources that reside in Availability Zones where there is no transit gateway attachment cannot reach the transit gateway

# Transit Gateway AZ affinity & Appliance mode

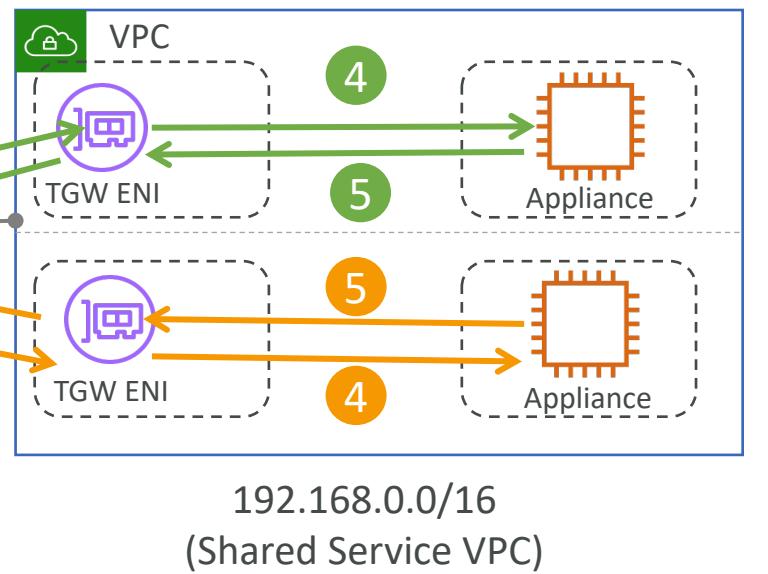
# Transit Gateway – AZ Affinity



# Transit Gateway – AZ Affinity

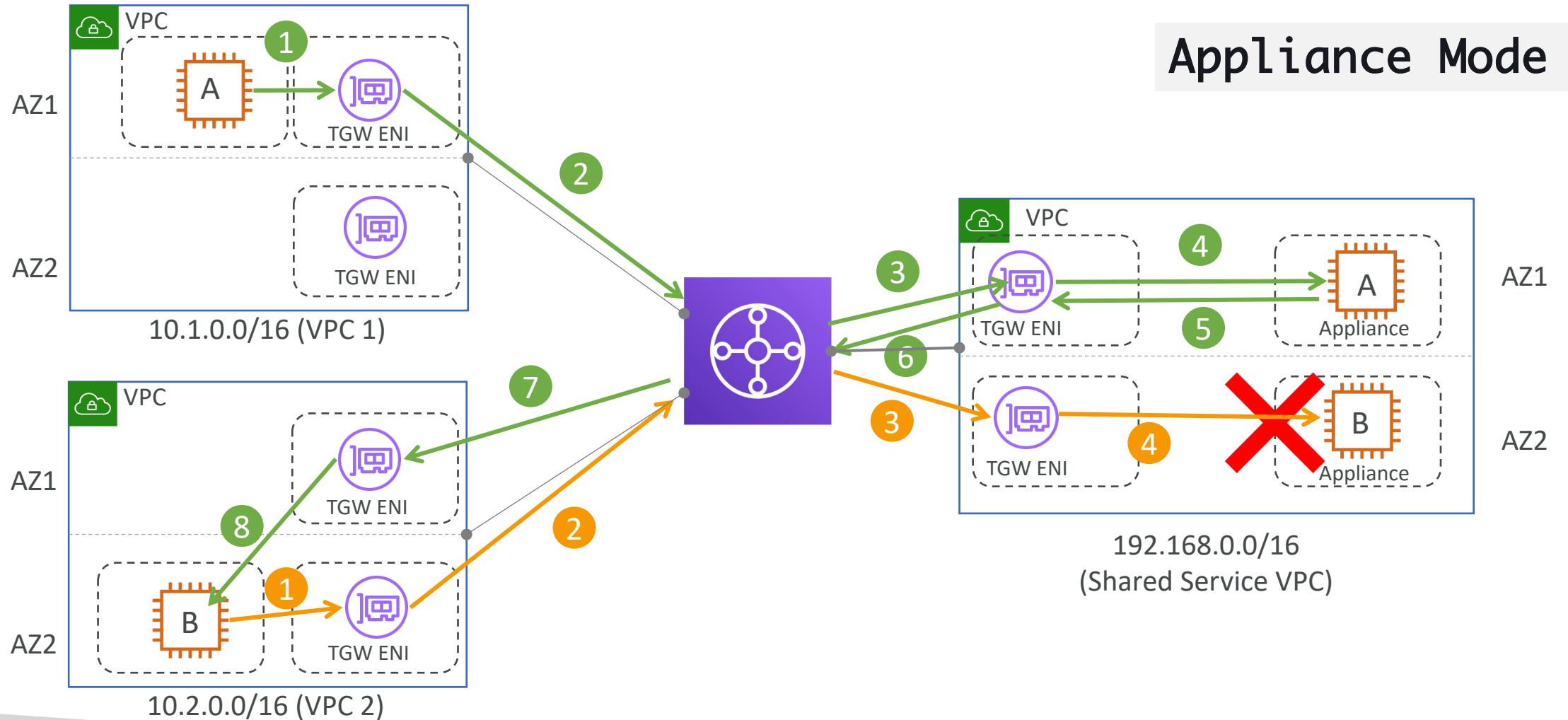


Source and destination in different AZs

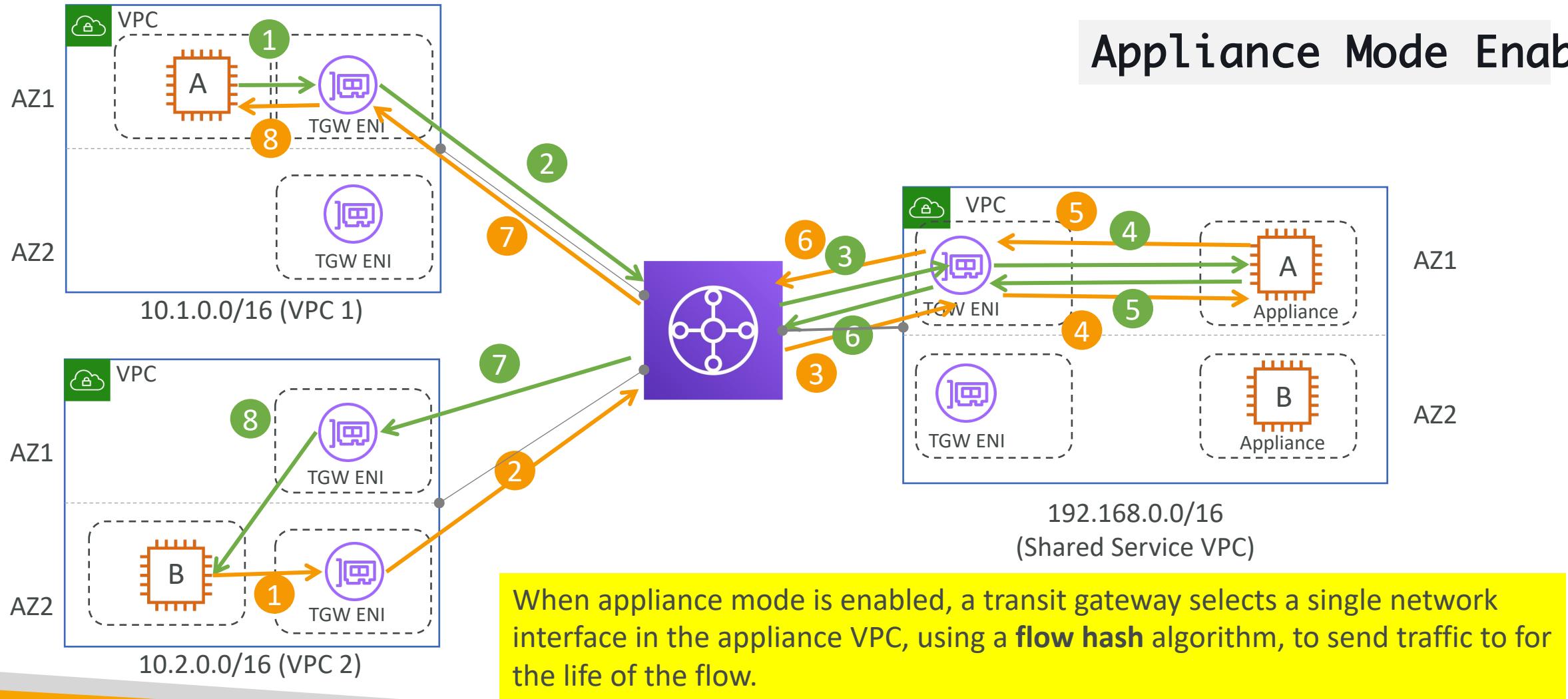


This causes Asymmetric Routing

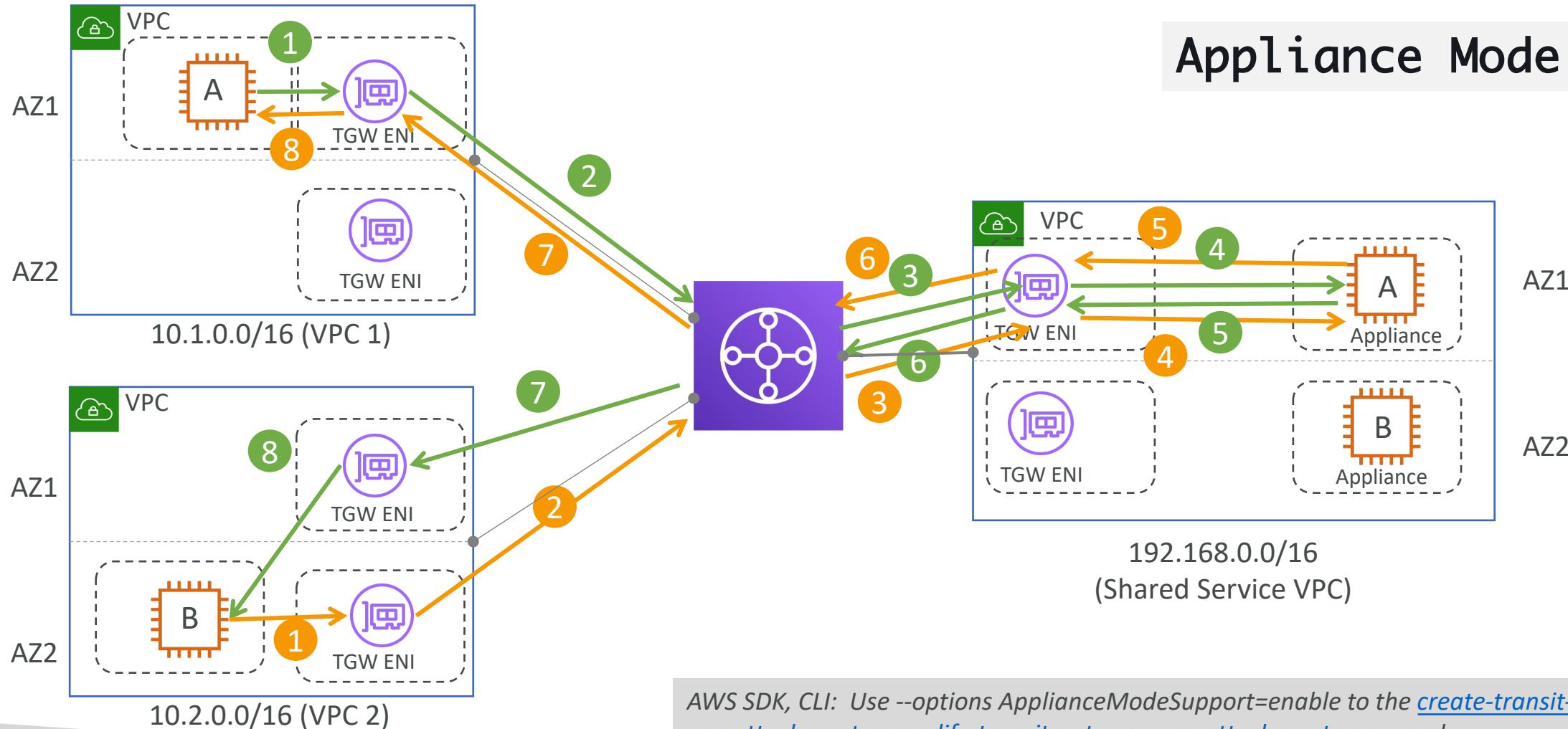
# Transit Gateway – Stateful Appliance



# Transit Gateway – Stateful Appliance



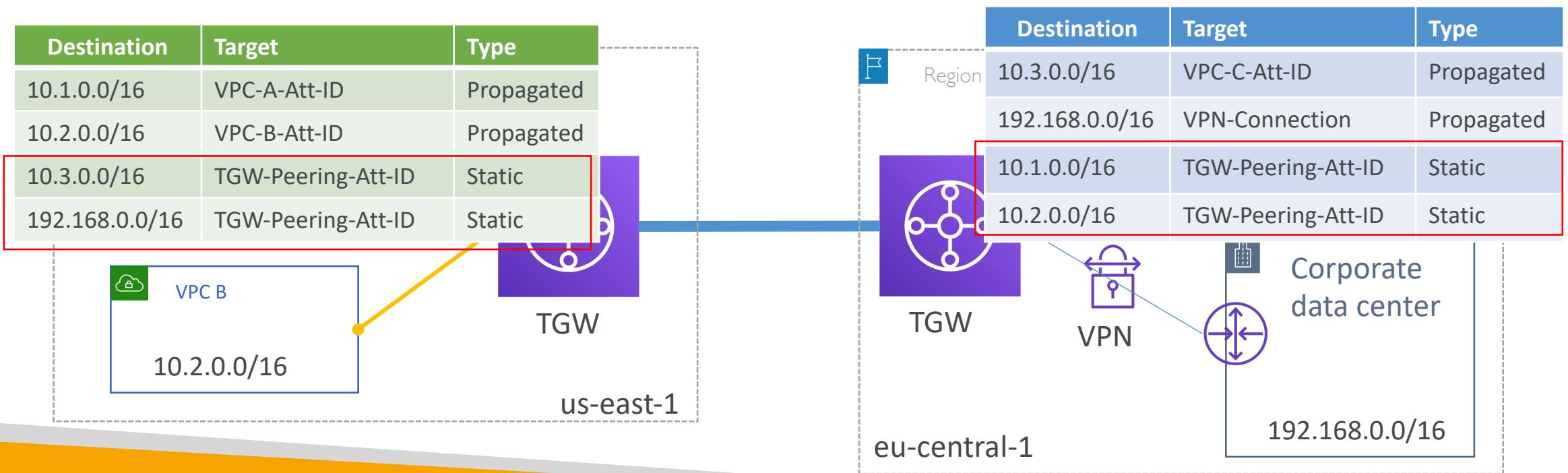
# Transit Gateway – Stateful Appliance



# Transit Gateway Peering

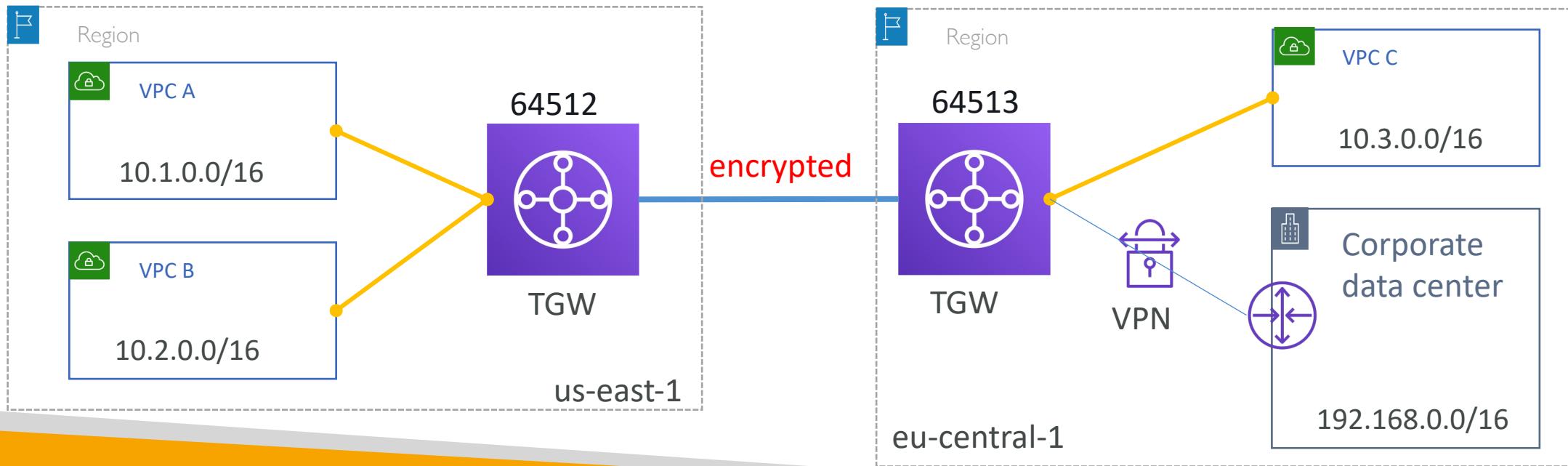
# Transit Gateway Peering

- Transit gateways are regional router which means you can connect VPCs from the same region
- For inter region network connectivity you can peer the transit gateways across the regions
- Static routes needs to be added for peering connection



# Transit Gateway Peering

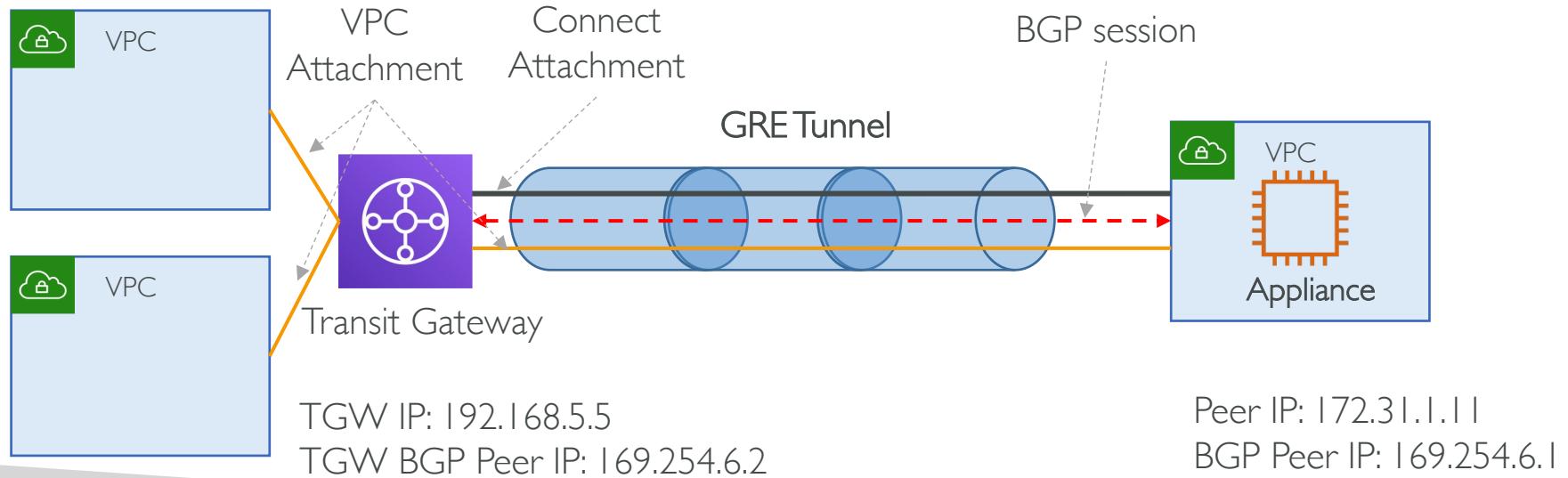
- Transit gateways are regional router which means you can connect VPCs from the same region
- For inter region network connectivity you can peer the transit gateways across the regions
- Static routes needs to be added for peering connection (No BGP)
- The inter-Region traffic is encrypted, traverses the AWS global network, and is not exposed to the public internet. Supports bandwidth up to 50 Gbps.
- Use unique ASNs for the peered transit gateways (as much possible)



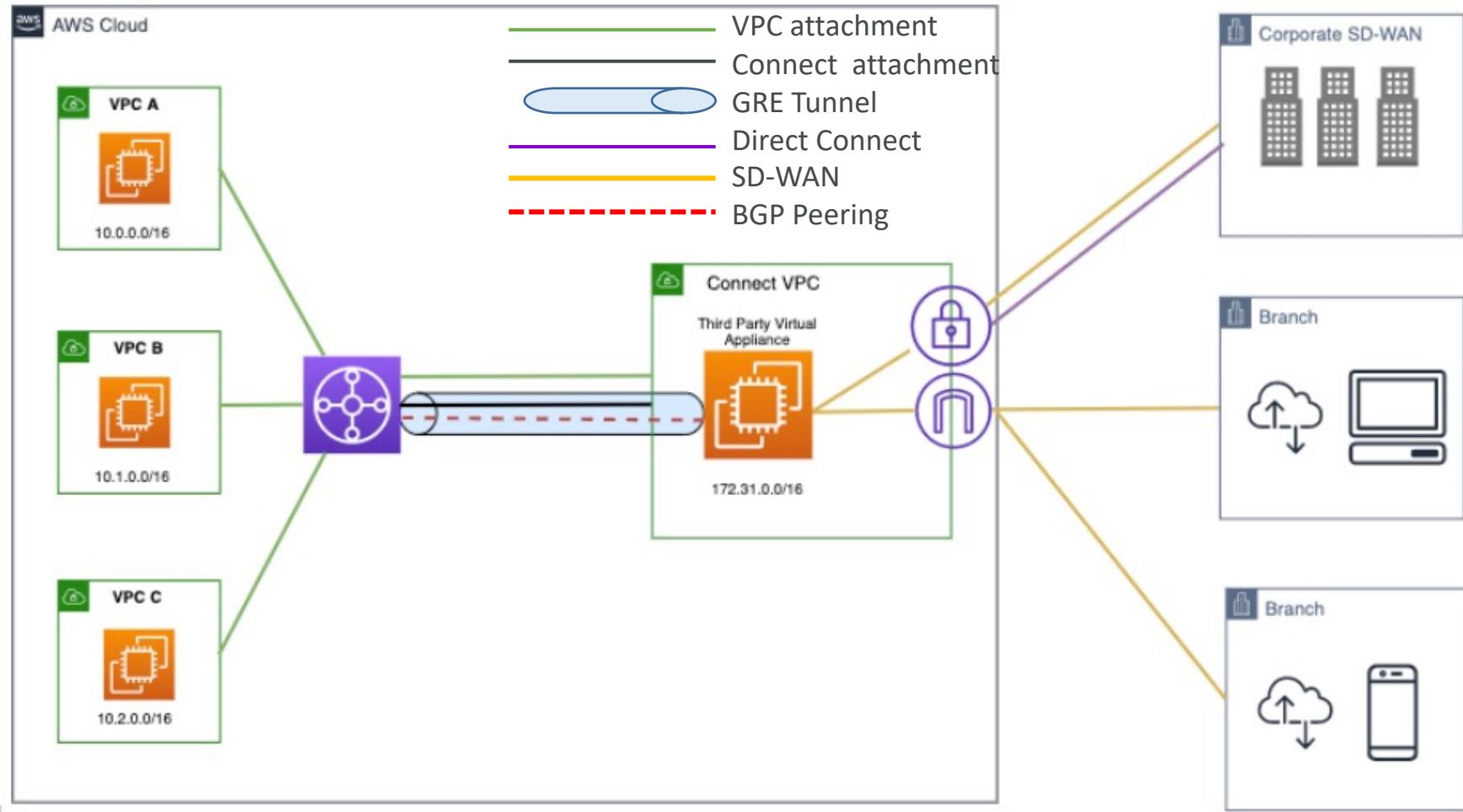
# Transit Gateway Connect Attachment

# Transit Gateway – Connect attachment

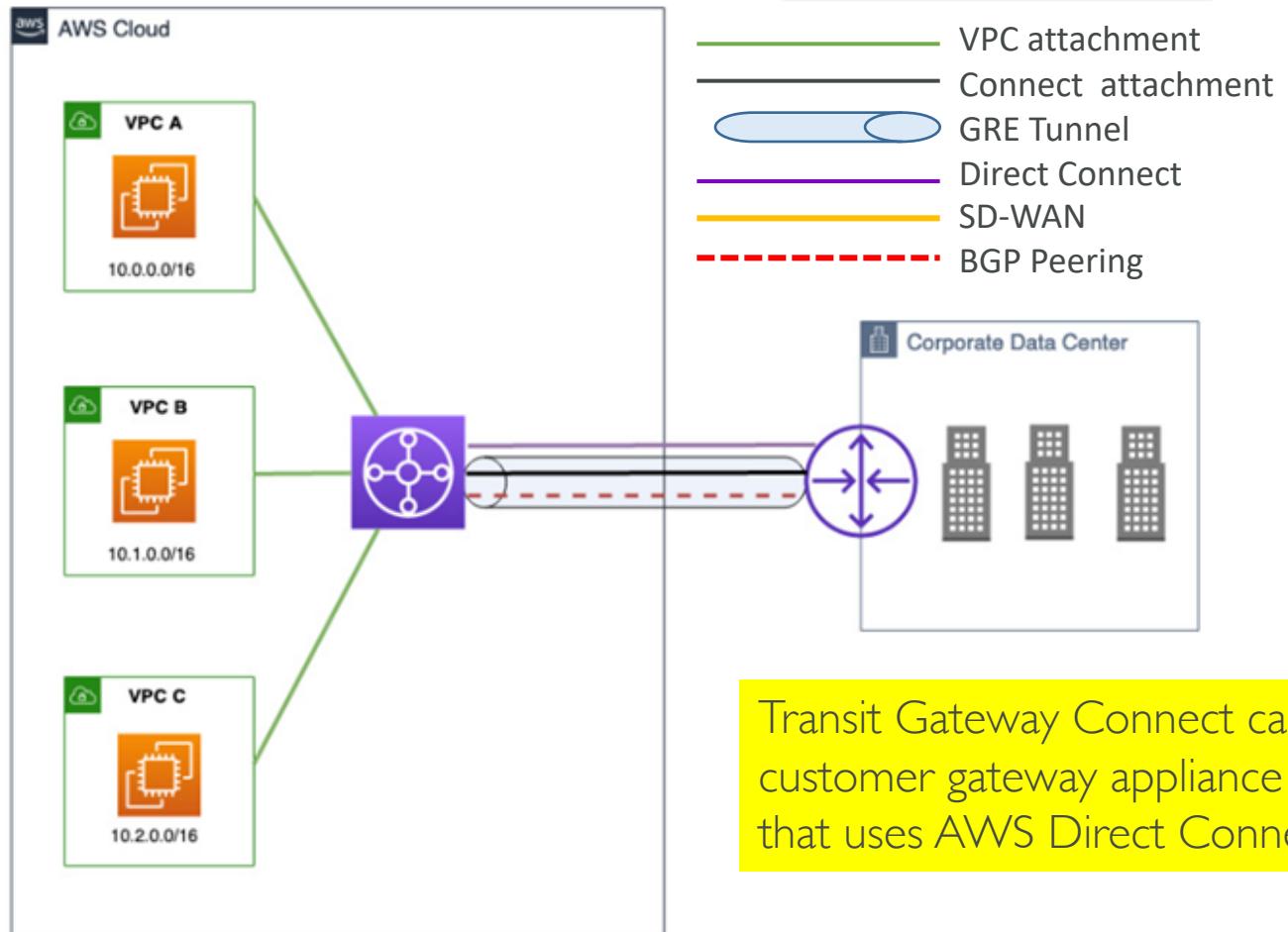
- You can create a transit gateway Connect attachment to establish a connection between a transit gateway and third-party virtual appliances (such as SD-WAN appliances) running in a VPC
- A Connect attachment uses an existing VPC or AWS Direct Connect attachment as the underlying transport mechanism.
- Supports Generic Routing Encapsulation (GRE) tunnel protocol for high performance, and Border Gateway Protocol (BGP) for dynamic routing.



# Transit Gateway Connect attachment over the VPC transport attachment



# Transit Gateway Connect attachment over the DX transport attachment

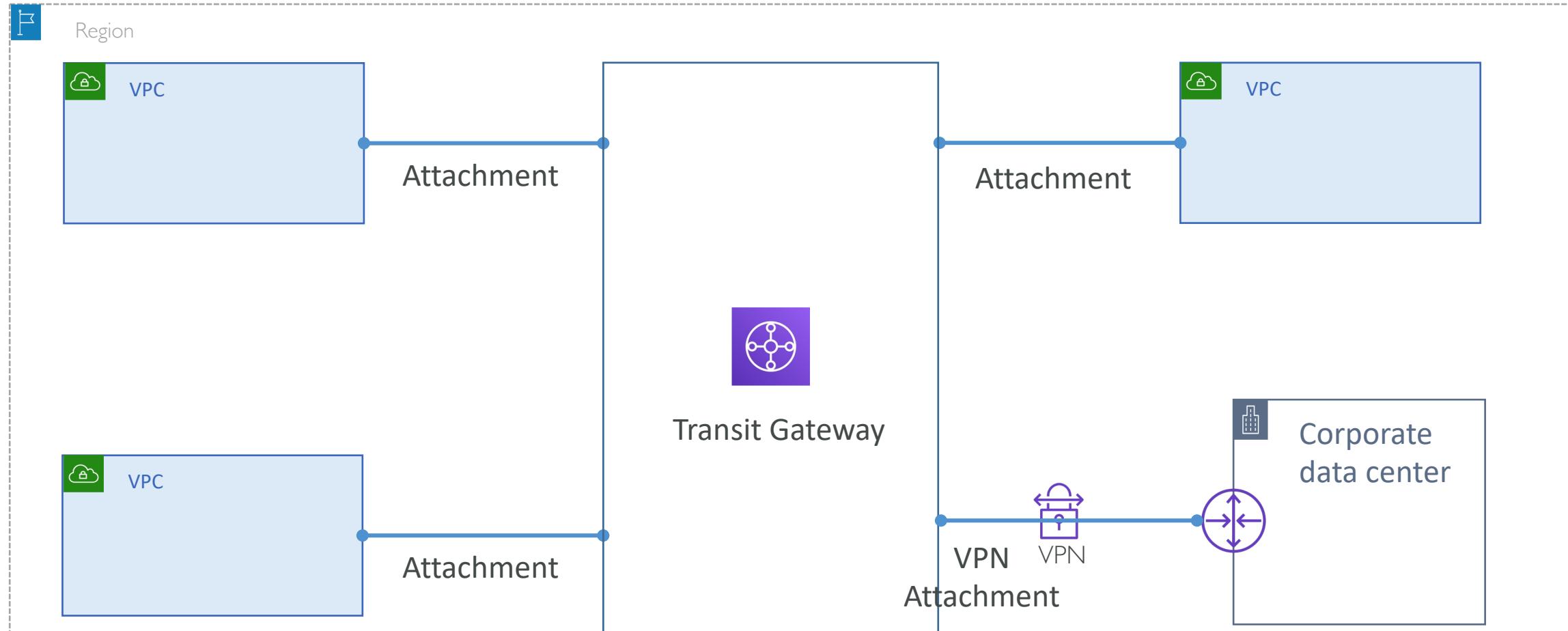


# Transit Gateway Connect attachment

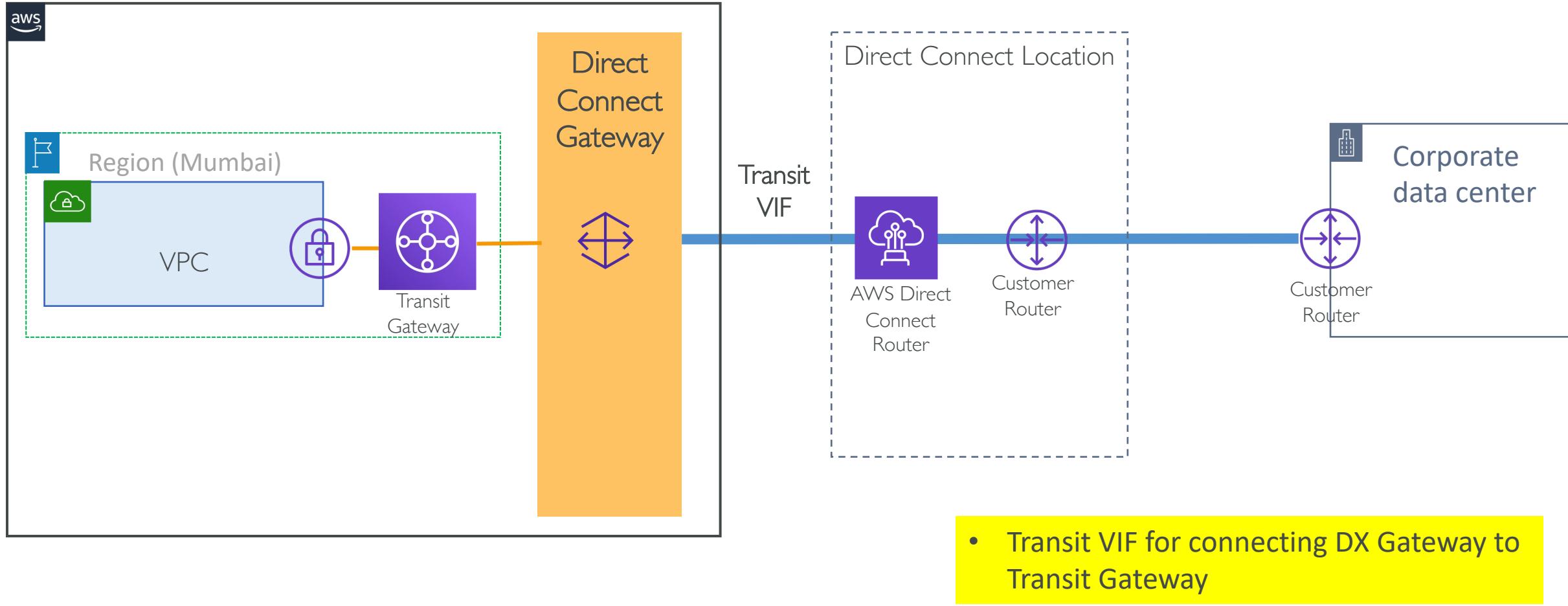
- Connect attachments do not support static routes. BGP is a minimum requirement for Transit Gateway Connect.
- Transit Gateway Connect supports a maximum bandwidth of 5 Gbps per GRE tunnel. Bandwidth above 5 Gbps is achieved by advertising the same prefixes across multiple Connect peer (GRE tunnel) for the same Connect attachment.
- A maximum of four Connect peers are supported for each Connect attachment there by providing total 20 Gbps bandwidth per connection

# Transit Gateway Hybrid connectivity options

# Hybrid connectivity – Transit Gateway with VPN attachment



# Hybrid connectivity – Transit Gateway with Direct Connect

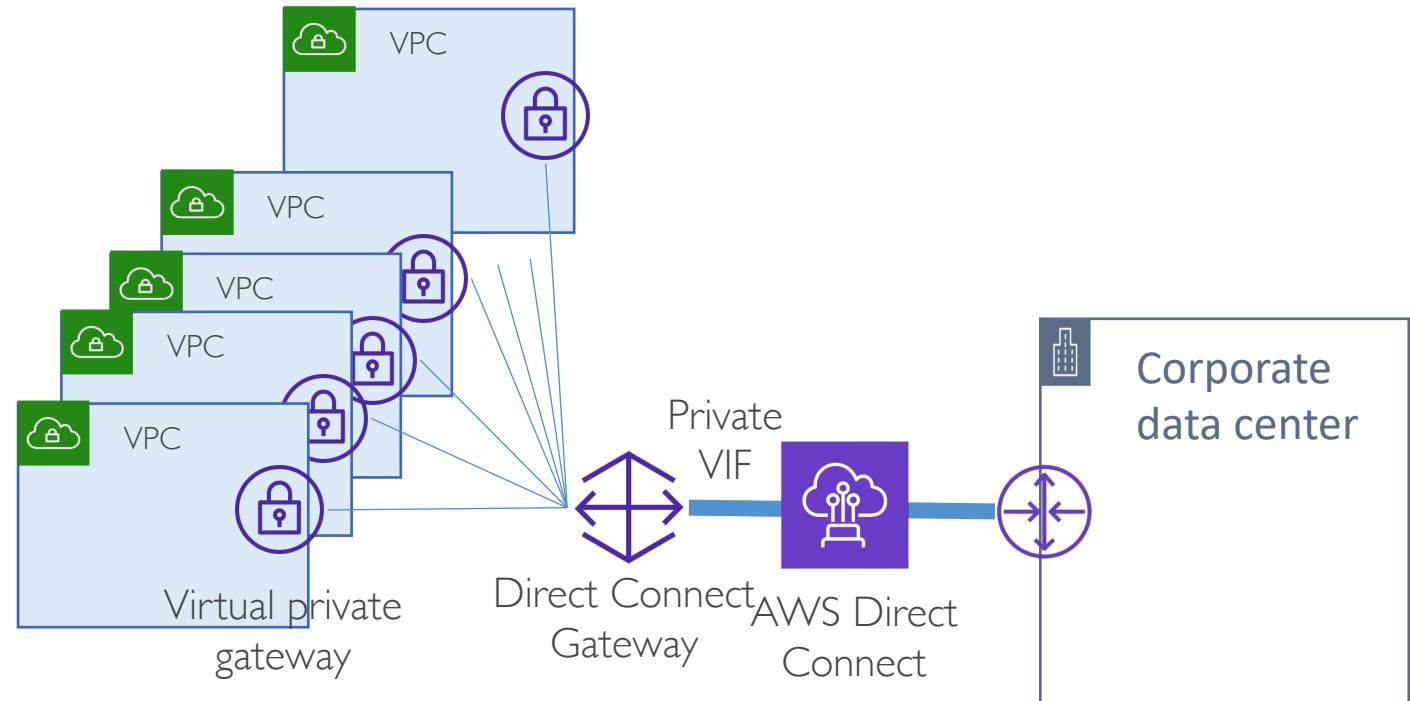


# Transit Gateway & Direct Connect

# Option I – Direct Connect Gateway associated with Transit Gateway

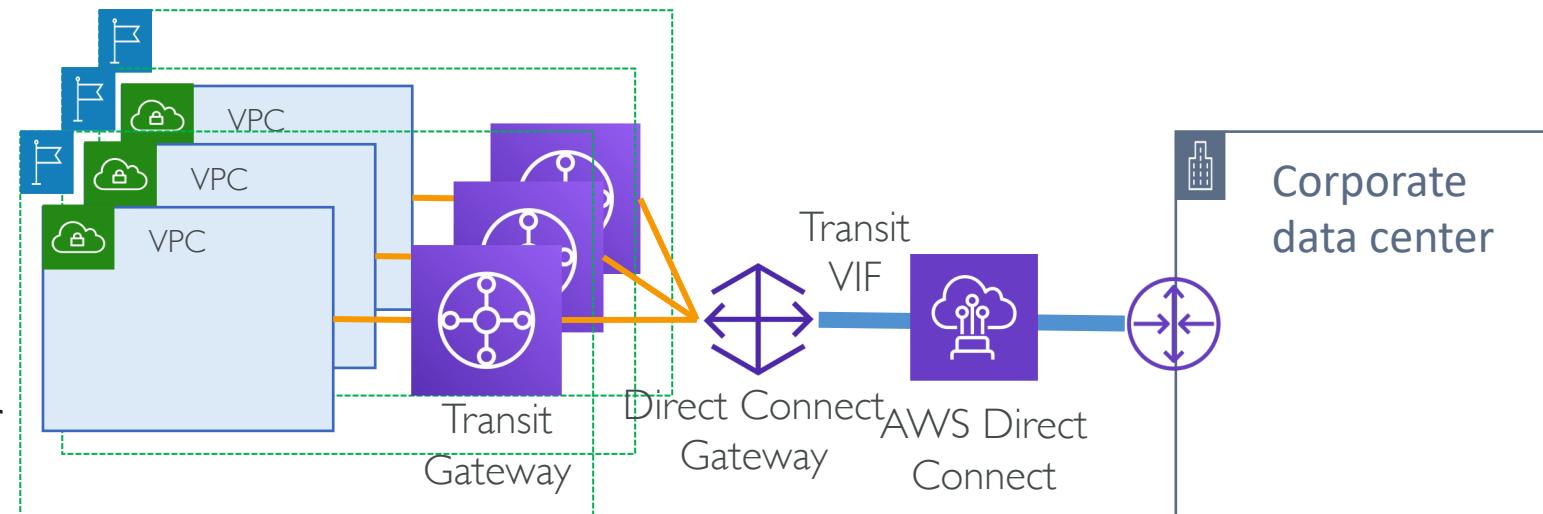
## Without Transit Gateway

- Using Direct Connect Gateway over a Private VIF
- Allows connecting only 10 VPCs



# Option I – Direct Connect Gateway associated with Transit Gateway

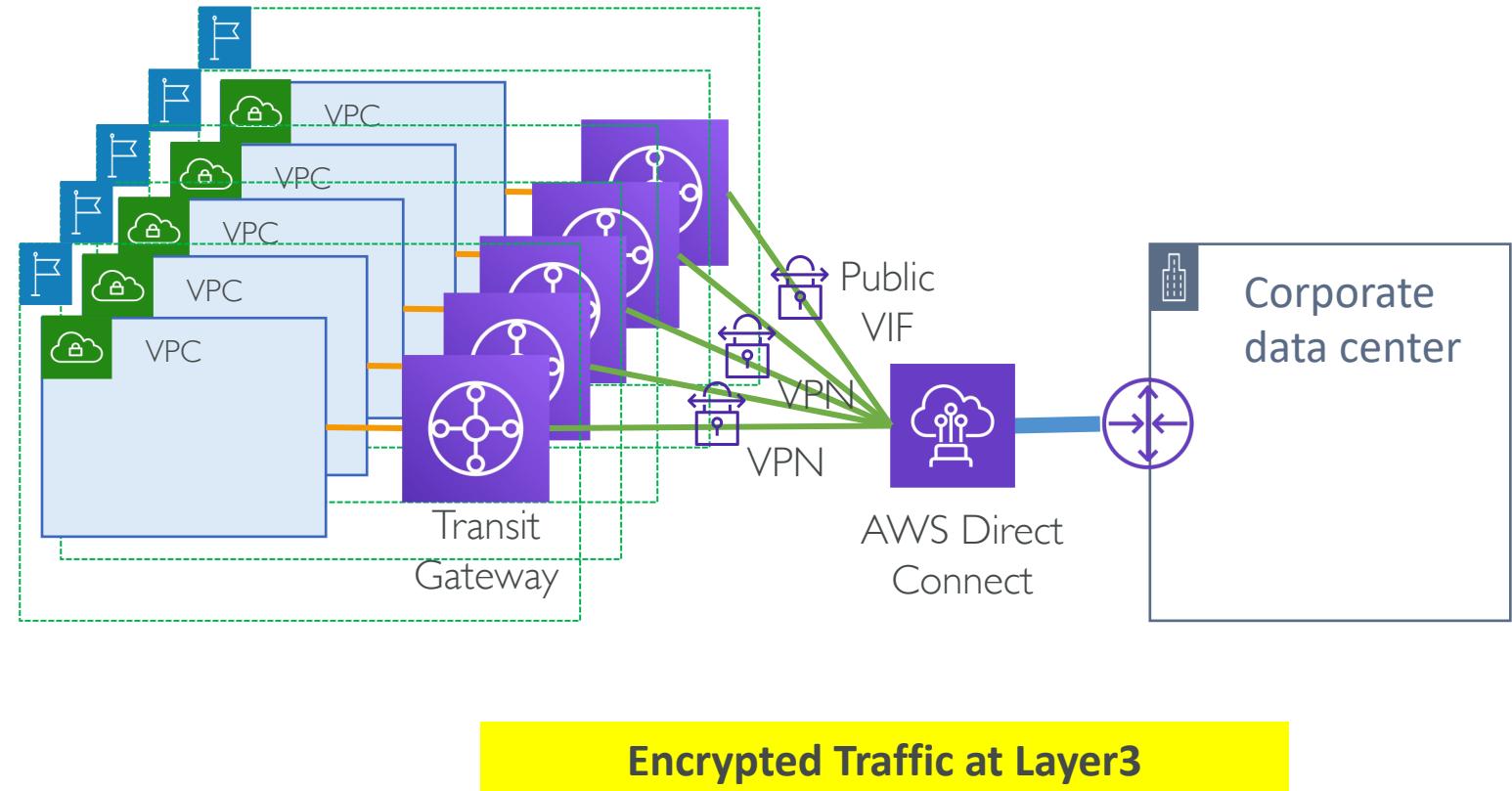
- Connect Direct Connect Gateway to Transit Gateway using Transit VIF
- Transit VIF is available on Dedicated or hosted Connection > 1Gbps
- Single Transit VIF allows connecting DX Gateway to up to 3 Transit Gateways across AWS regions and accounts
- You can advertise only 20 prefixes per Transit Gateway to on-premises router and 100 prefixes from on-premises to Transit Gateway over a Transit VIF
- Additional charge for Transit Gateway attachment and data processing



Simple, scalable, Multi-region, Multi-account

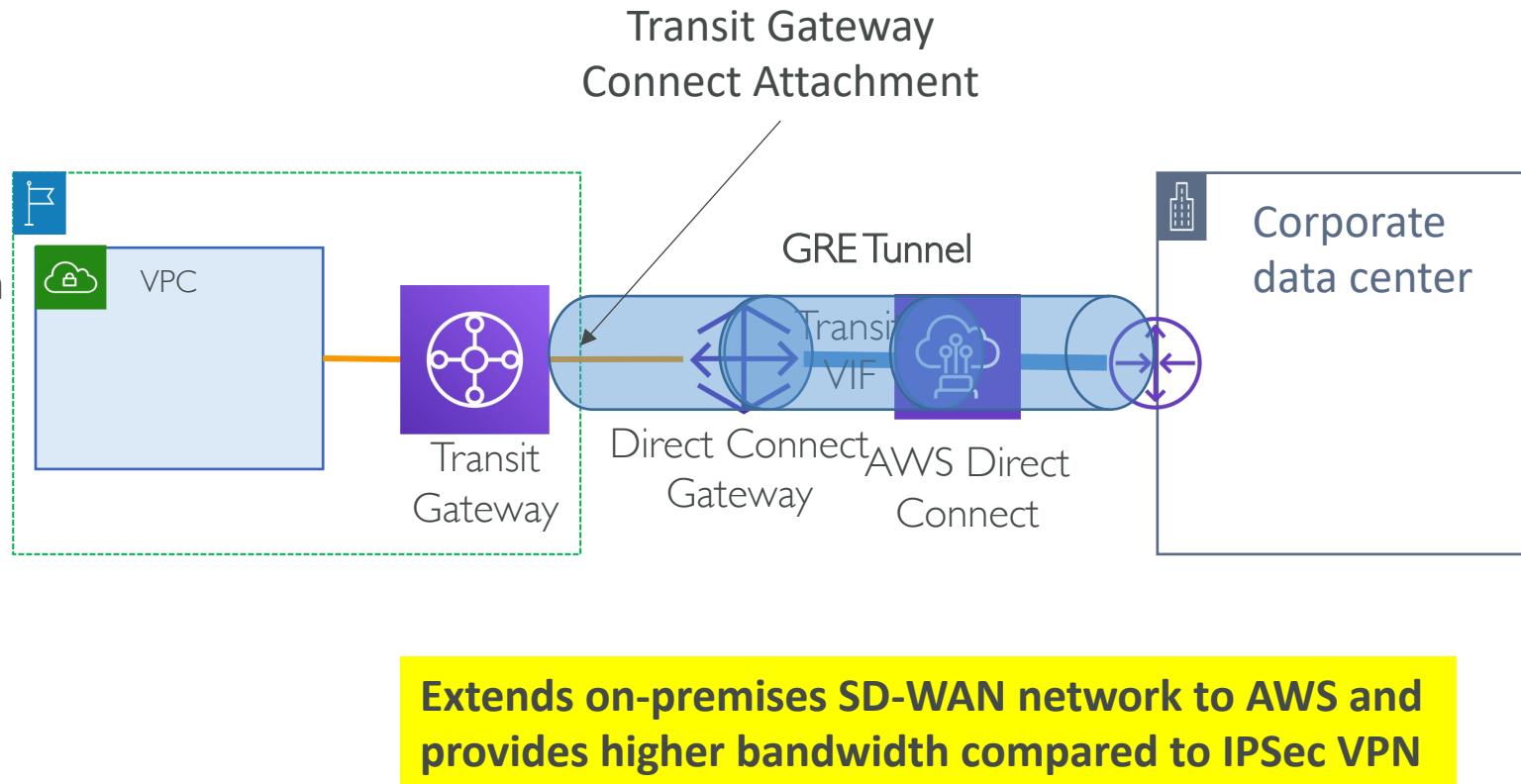
# Option 2 – VPN connection to Transit Gateway over a DX Public VIF

- IPSec VPN connection to Transit Gateway Public IPs over a Direct Connect Public VIF
- Create as many VPN connections to as many Transit Gateways in any AWS region and account
- **Encrypts the traffic at Layer3**



# Option 3 – GRE tunnels to Transit Gateway over a transit VIF

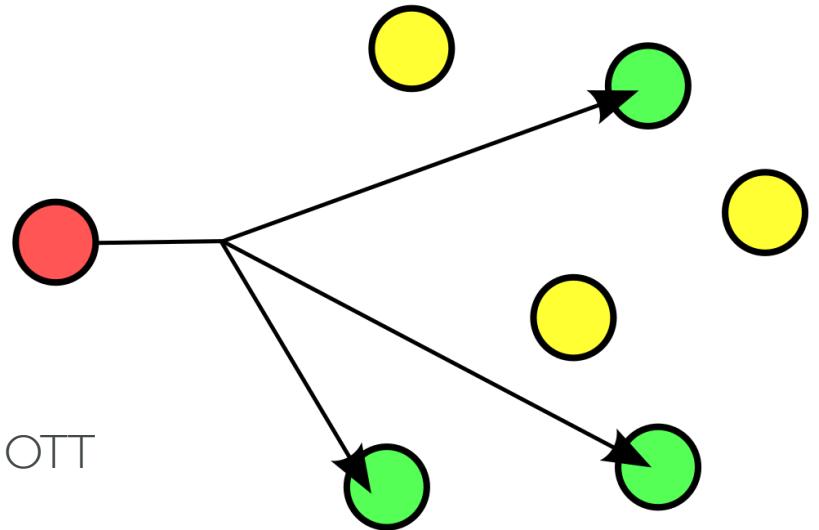
- Connect Direct Connect Gateway to Transit Gateway using Transit VIF
- Create GRE tunnel between Transit Gateway and on-premises router
- Supports BGP
- No need to setup IPsec VPNs between SD-WAN network virtual appliances and Transit Gateway
- Provides higher bandwidth (20 Gbps) compared to a VPN connection (1.25 Gbps) per connection



# Multicast with Transit Gateway

# Multicast

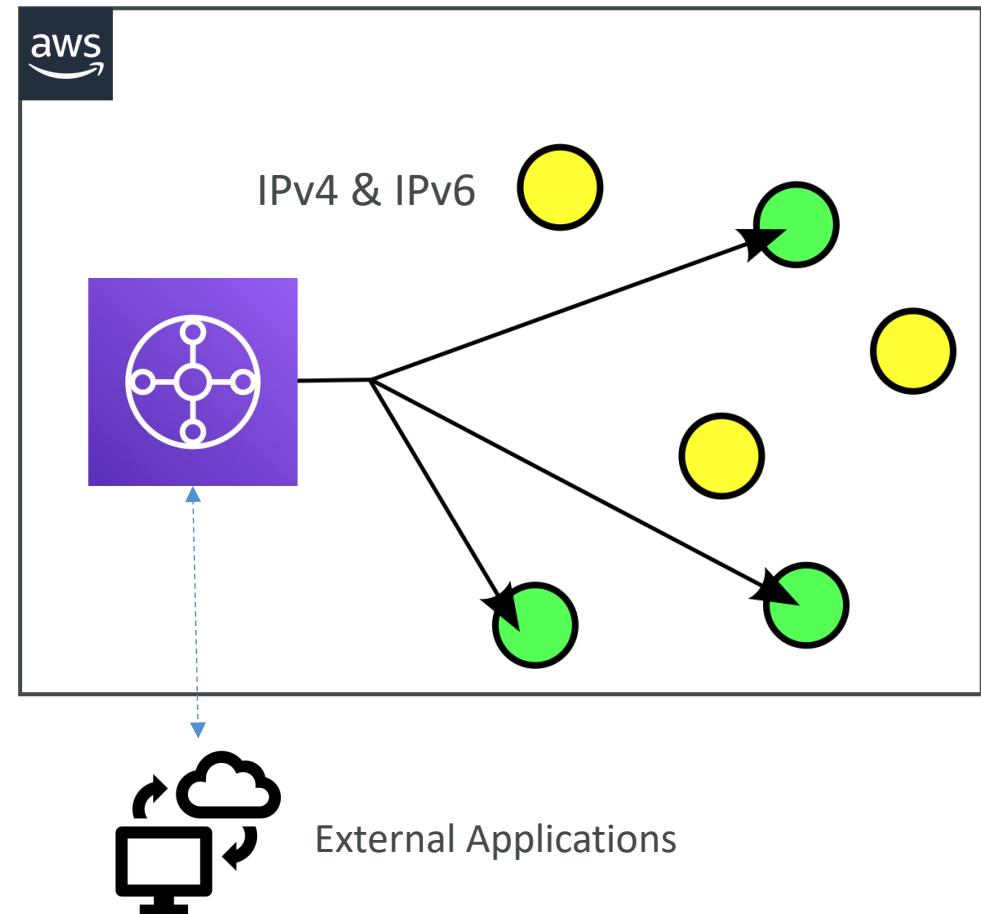
- Multicast is a communication protocol used for delivering a single stream of data to multiple receiving computers simultaneously.
- Single/multiple sources and destinations
- Destination is a multicast group address:
  - Class D - 224.0.0.0 to 239.255.255.255
- Connectionless UDP based transport
- One way communication
- Examples: Sending email to the email-list, Conference call / Group chat, OTT platforms / TV Media, Stock exchange transaction updates
- Multicast components
  - Multicast Domain
  - Multicast Group and member
  - Multicast source and receivers
  - Internet Group Management Protocol (IGMP)



<https://en.wikipedia.org/wiki/Multicast>

# Multicast with Transit Gateway

- Enable Transit Gateway for Multicast services while creating the transit gateway
- Supports IPv4 & IPv6 IP addressing
- Supports hybrid integration with external applications
- Supports both static (API based) and Dynamic group membership through IGMP (supports IGMPv2)



# Multicast considerations for TGW

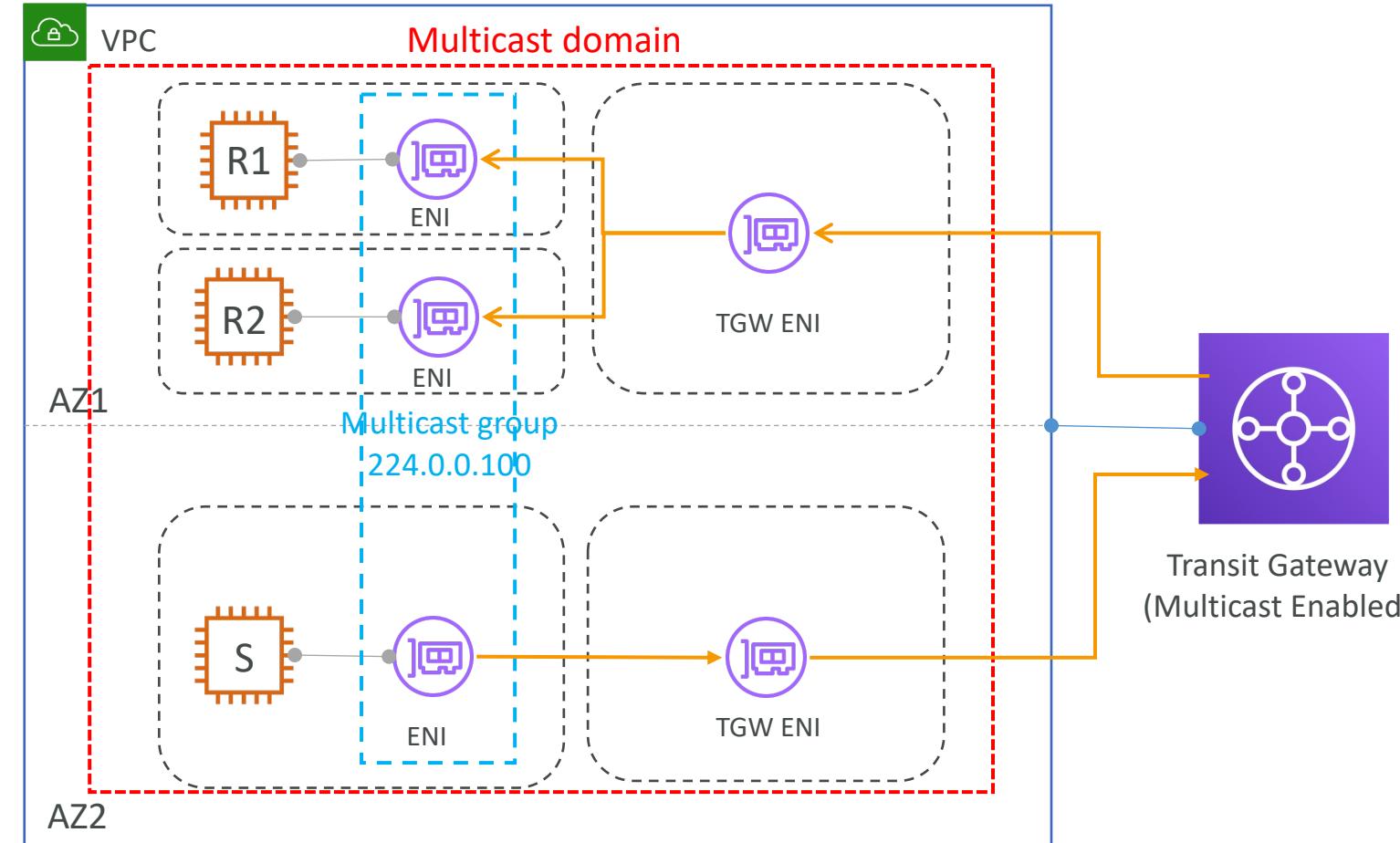
- Transit Gateway supports routing multicast traffic between subnets of attached VPCs.
- A subnet can only be in one multicast domain.
- Hosts (ENIs) in the subnet can be part of one or more multicast groups within the Multicast domain.
- Multicast group membership is managed using the VPC Console or the AWS CLI, or IGMP.
- IGMPv2 support attribute determines how hosts join the multicast group. Members send JOIN or LEAVE IGMP message.
- Transit gateway issues an IGMPv2 QUERY message to all members every two minutes. Each member sends an IGMPv2 JOIN message in response, which is how the members renew their membership.
- Members that do not support IGMP must be added or removed from the group using the Amazon VPC console or the AWS CLI.
- **Igmpv2Support** attribute determines how group members join or leave a multicast group. When this attribute enabled, members can send JOIN or LEAVE messages.
- **StaticSourcesSupport** multicast domain attributes determine whether there are static multicast sources for the group.

# Multicast considerations for TGW

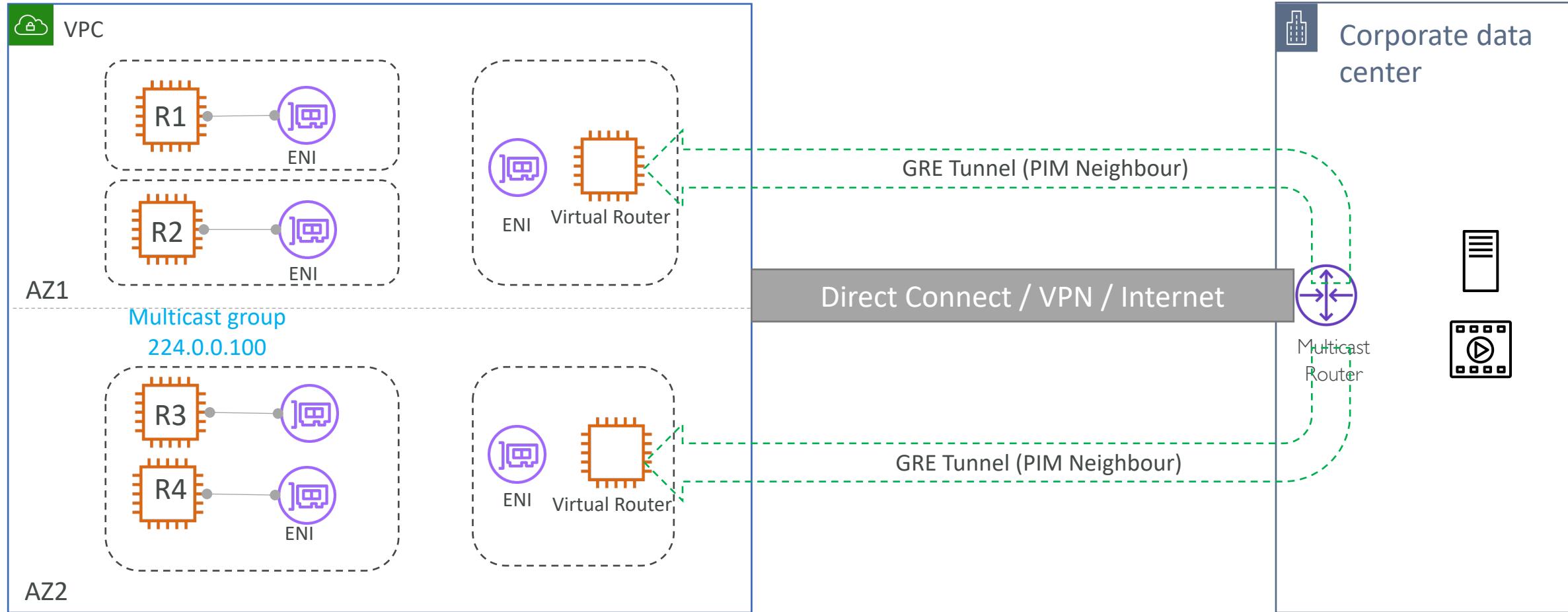
- A non-Nitro instance cannot be a multicast sender. If you use a non-Nitro instance as a receiver, you must disable the Source/Destination check
- Multicast routing is not supported over AWS Direct Connect, Site-to-Site VPN, TGW peering attachments, or transit gateway Connect attachments.
- The source IP of these IGMP query packets is 0.0.0.0/32, destination IP is 224.0.0.1/32 and the protocol is IGMP(2).
- Security group configuration on the IGMP hosts (instances), and any ACLs configuration on the host subnets must allow these IGMP protocol messages.

# Multicast traffic in a VPC

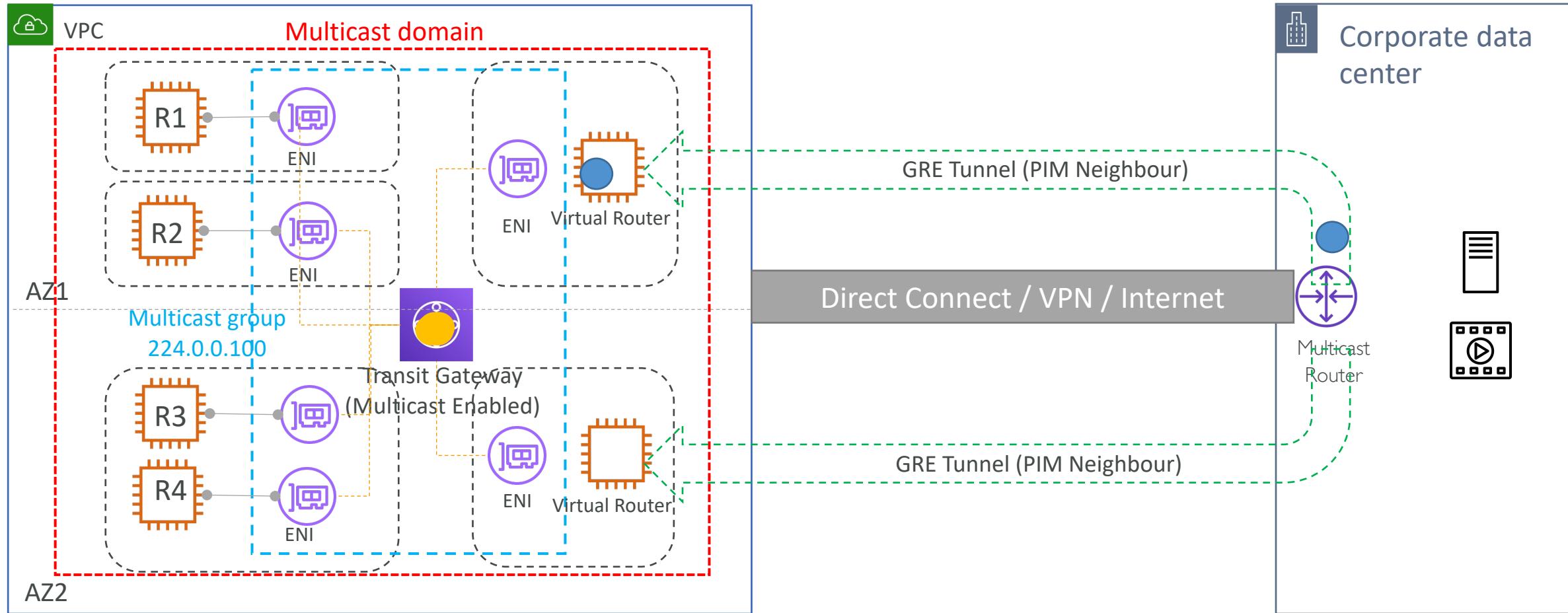
- Create multicast domain and add participating subnets
- Create multicast group and associate group membership IP (e.g. 224.0.0.100)
- Configure the group membership statically using CLI/SDK or dynamically using IGMPv2
- Send traffic from source to multicast group IP
- All members in the group receive the multicast traffic
- Similar network flow works in case of multi-vpc and multi-account architecture



# Integrating external multicast services

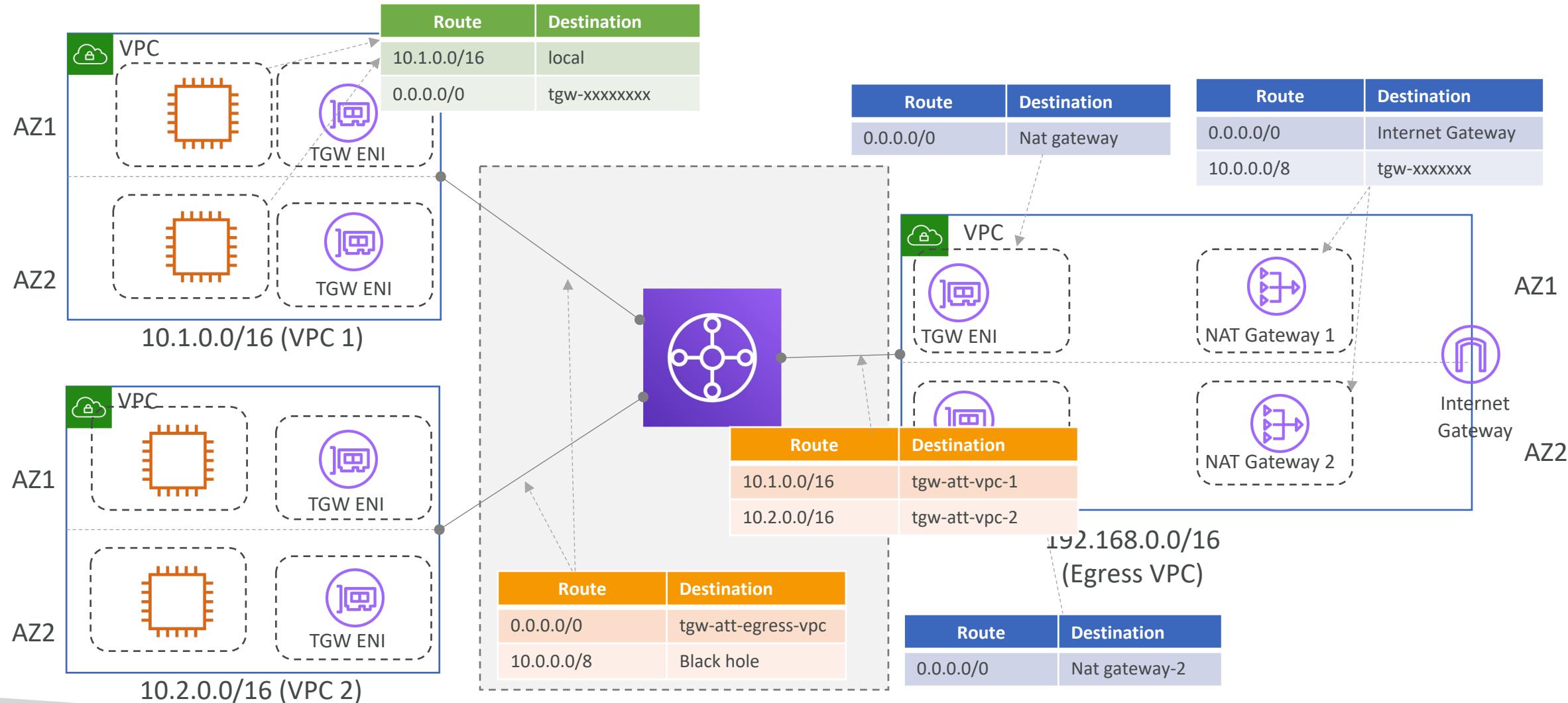


# Integrating external multicast services

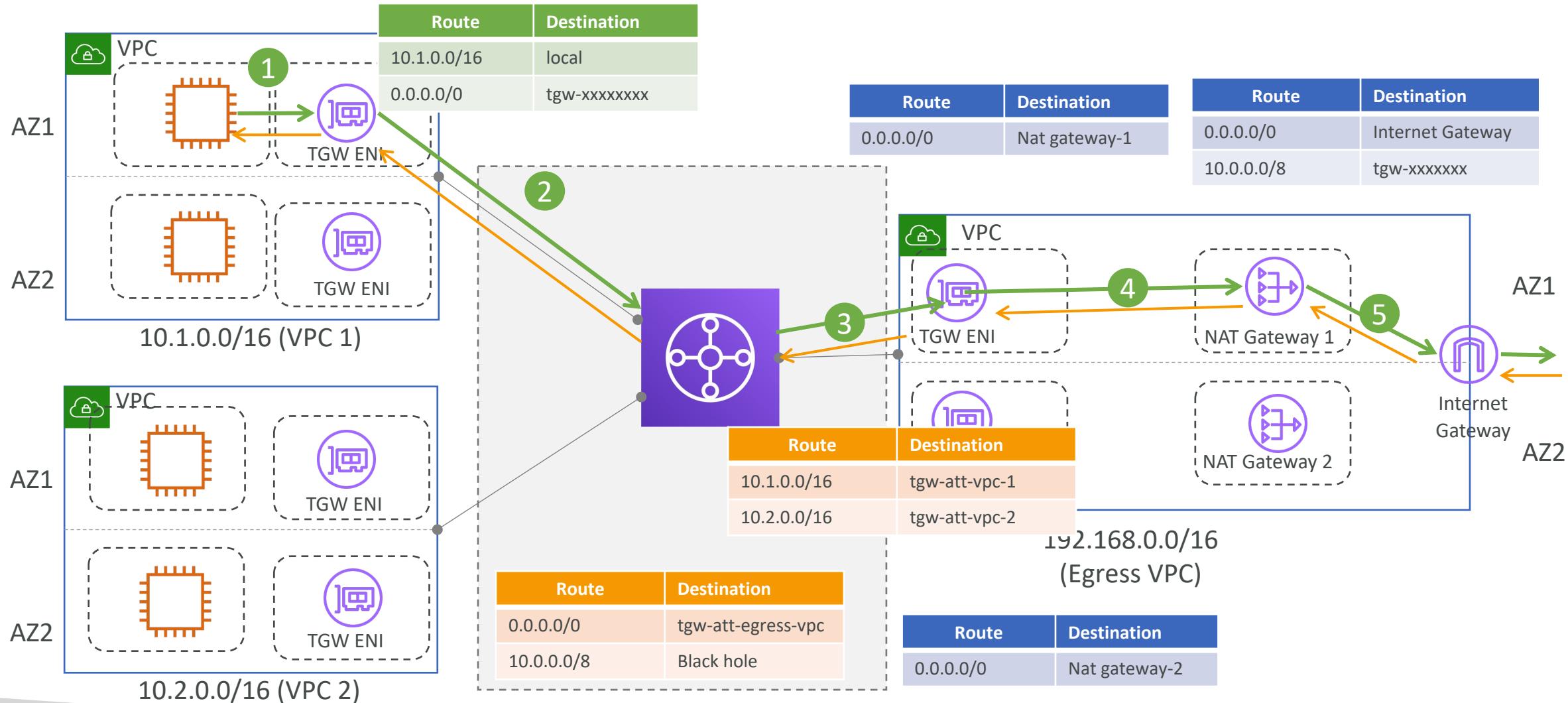


# Transit Gateway – Egress NAT Gateway

# Centralized egress with NAT gateway



# Centralized egress with NAT gateway



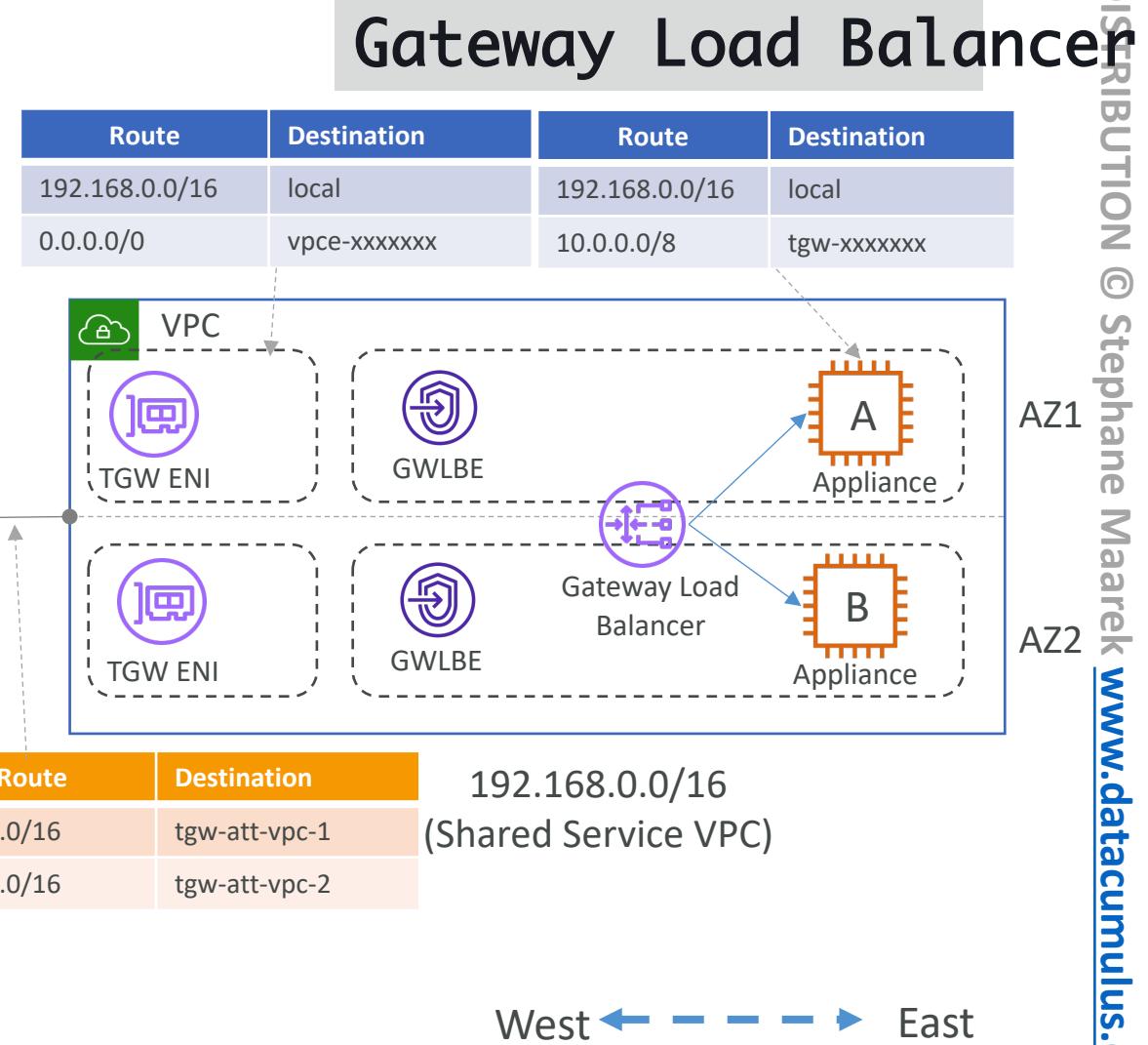
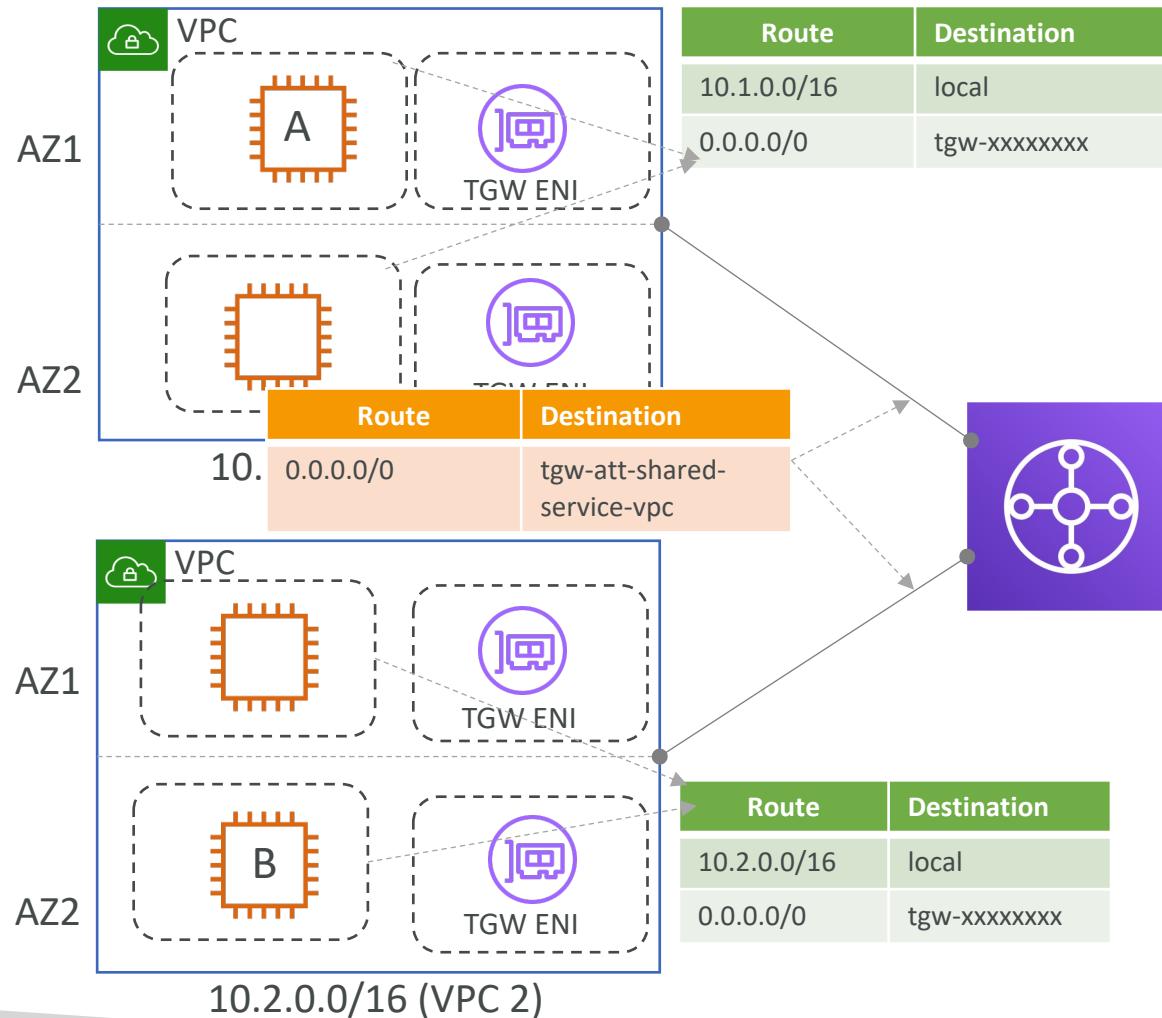
# Centralized egress with NAT gateway

Important to know:

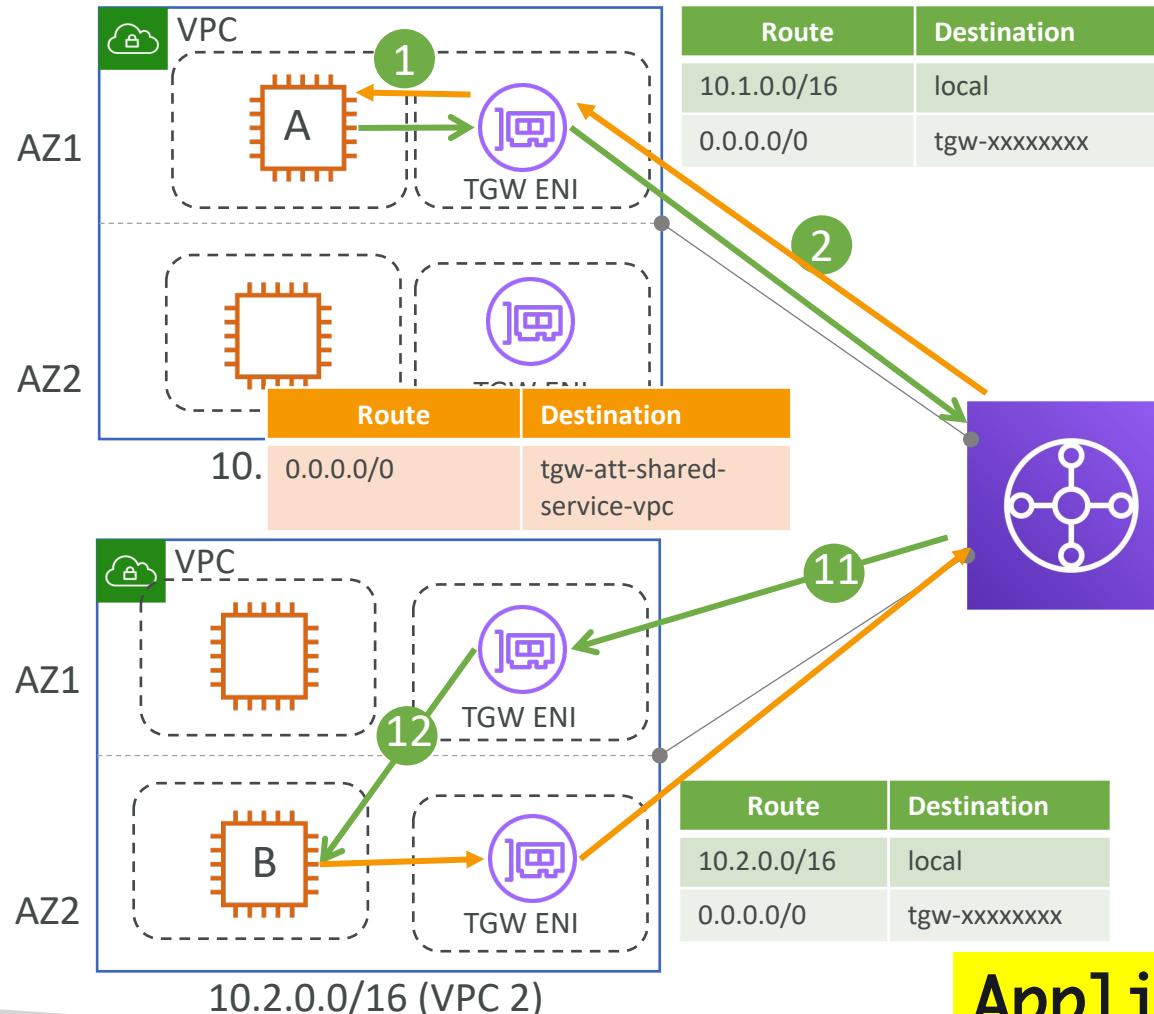
- Use NAT gateway's in each AZ for high availability.
- If one Availability Zone entirely fails, the Transit Gateway endpoint along with NAT gateway in that Availability Zone will fail, and all traffic will flow via the Transit Gateway and NAT gateway endpoints in the other Availability Zone.
- A NAT gateway can support up to 55,000 simultaneous connections to each unique destination. From a throughput standpoint, NAT gateway can scale from 5 Gbps to 45 Gbps.
- Blackhole routes in the Transit Gateway route tables to restrict inter-VPC traffic
- This architecture doesn't necessarily save the cost because instead of per VPC NAT Gateway charge (e.g. \$0.045/hr + \$0.045/GB) it adds Transit Gateway attachment & data processing charge (\$0.05/hr per VPC attachment + \$0.02/GB)

# Transit Gateway – Centralized inspection with Gateway Load Balancer

# Centralized inspection with AWS GWLB

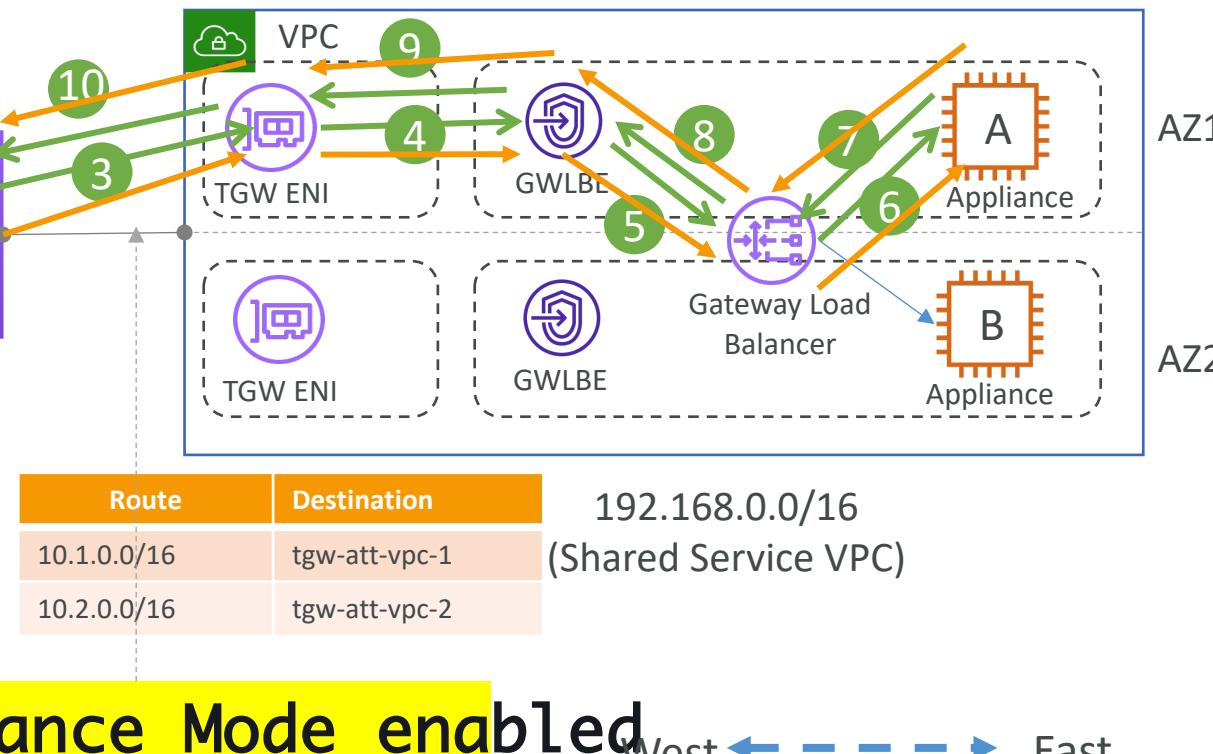


# Centralized inspection with AWS GWLB

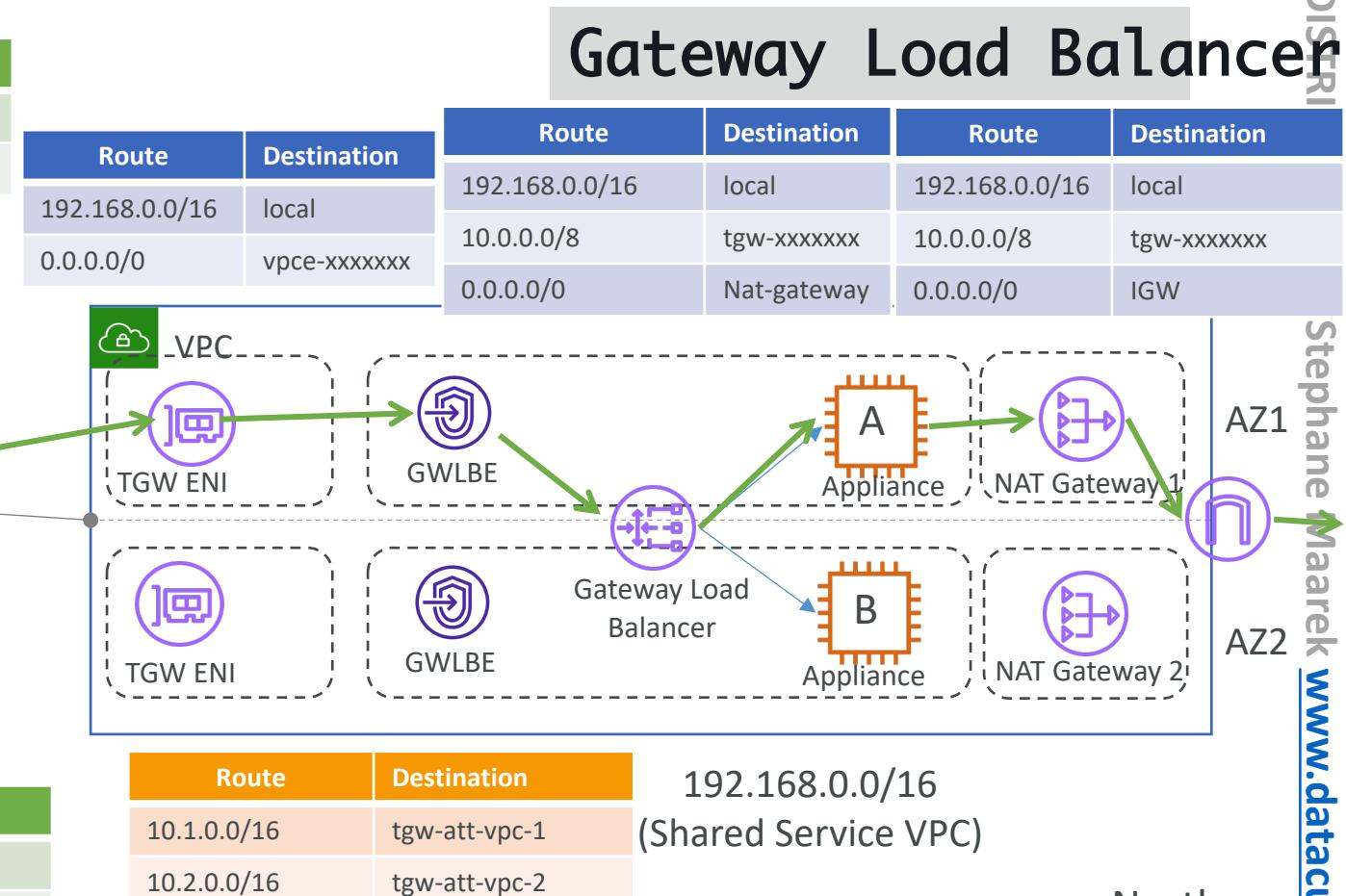
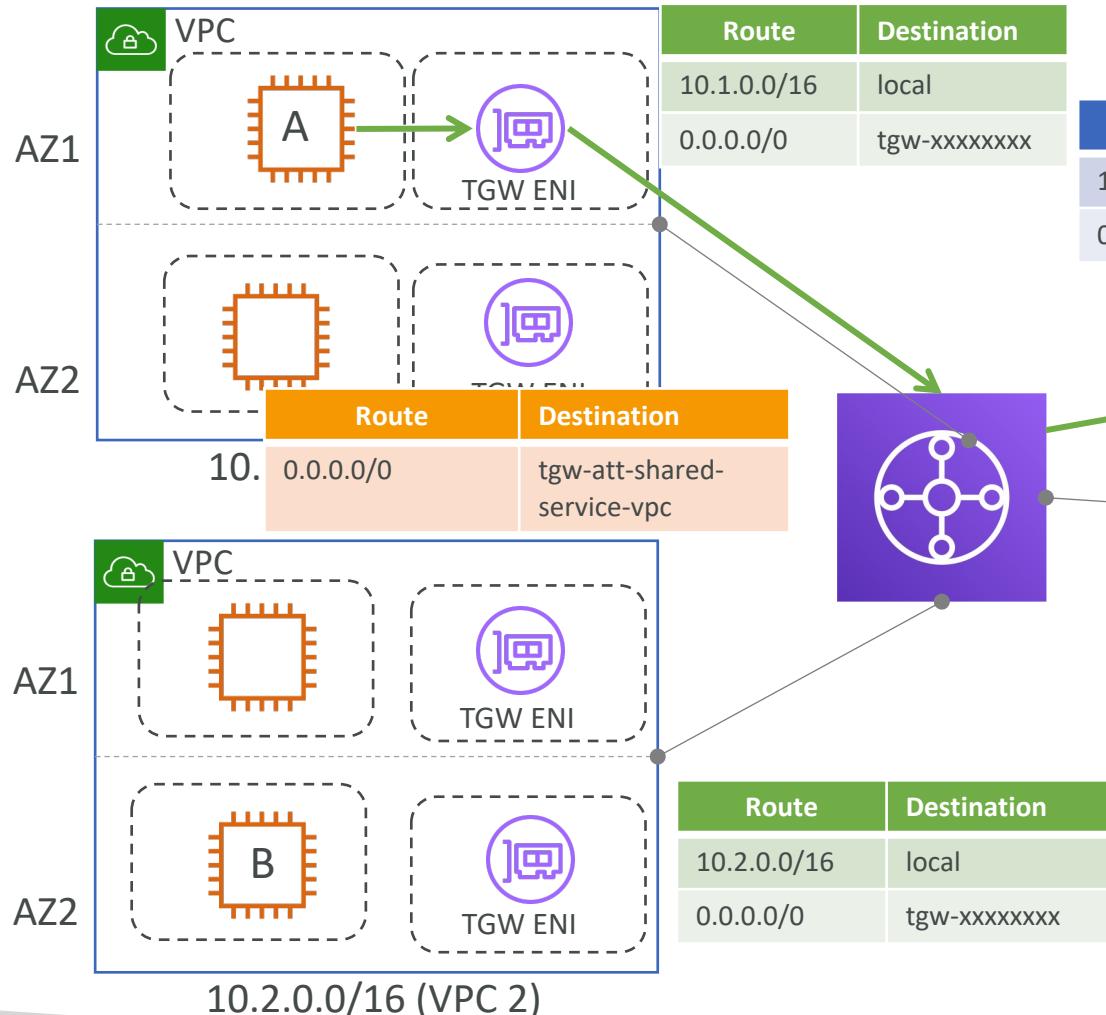


## Gateway Load Balancer

| Route          | Destination  | Route          | Destination |
|----------------|--------------|----------------|-------------|
| 192.168.0.0/16 | local        | 192.168.0.0/16 | local       |
| 0.0.0.0/0      | vpce-xxxxxxx | 10.0.0.0/8     | tgw-xxxxxxx |



# Centralized inspection with AWS GWLB



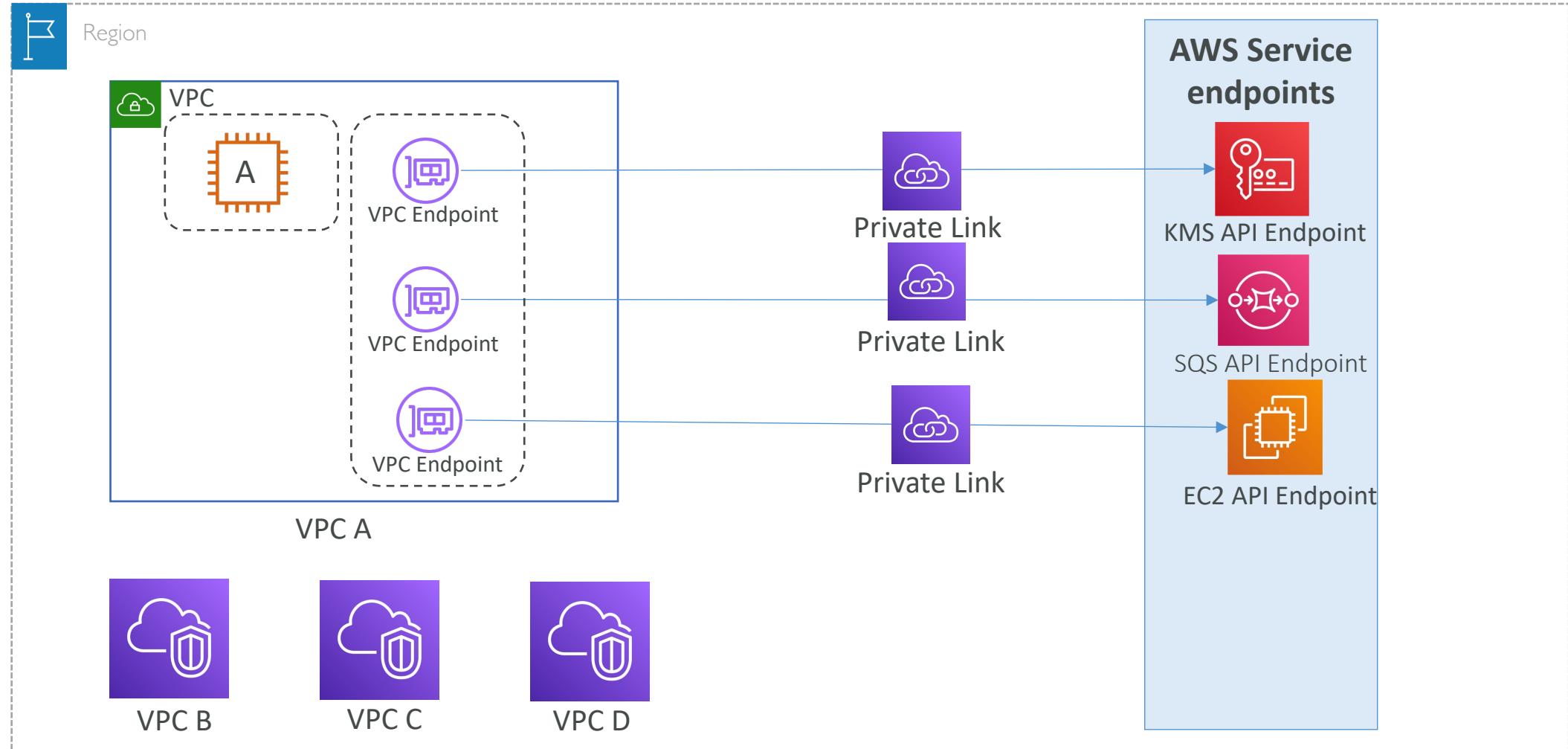
# Centralized inspection with AWS GWLB

## Important to know:

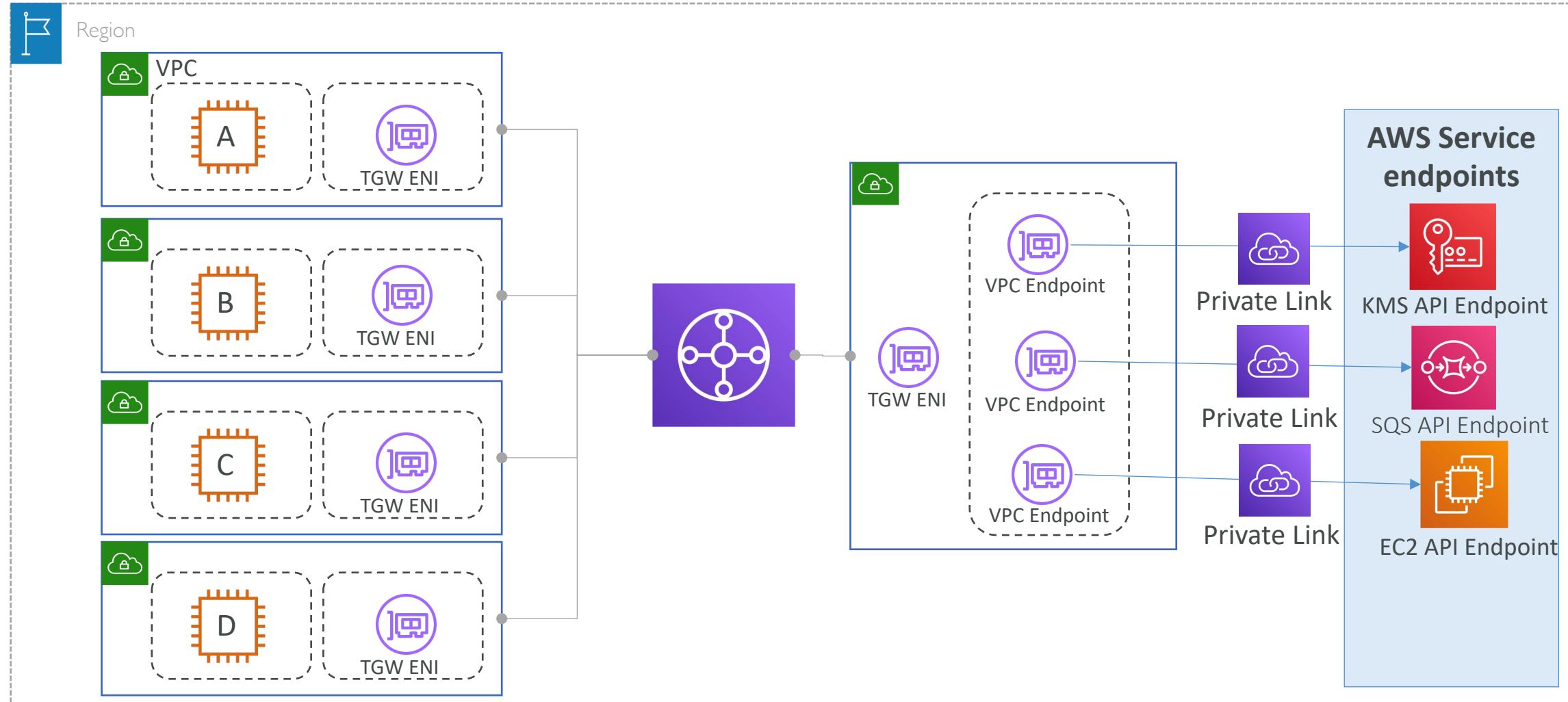
- Using AWS PrivateLink, GWLB Endpoint routes traffic to GWLB. Traffic is routed securely over Amazon network without any additional configuration.
- GWLB encapsulates the original IP traffic with a GENEVE header and forwards it to the appliance over UDP port 6081.
- GWLB uses 5-tuples or 3-tuples of an IP packet to pick an appliance for the life of that flow. This creates session stickiness to an appliance for the life of a flow required for stateful appliances like firewalls.
- This combined with Transit Gateway appliance mode, provides session stickiness irrespective of source and destination AZ.
- .. more about Gateway Load Balancer in a separate section

# Transit Gateway – Centralized VPC interface endpoints

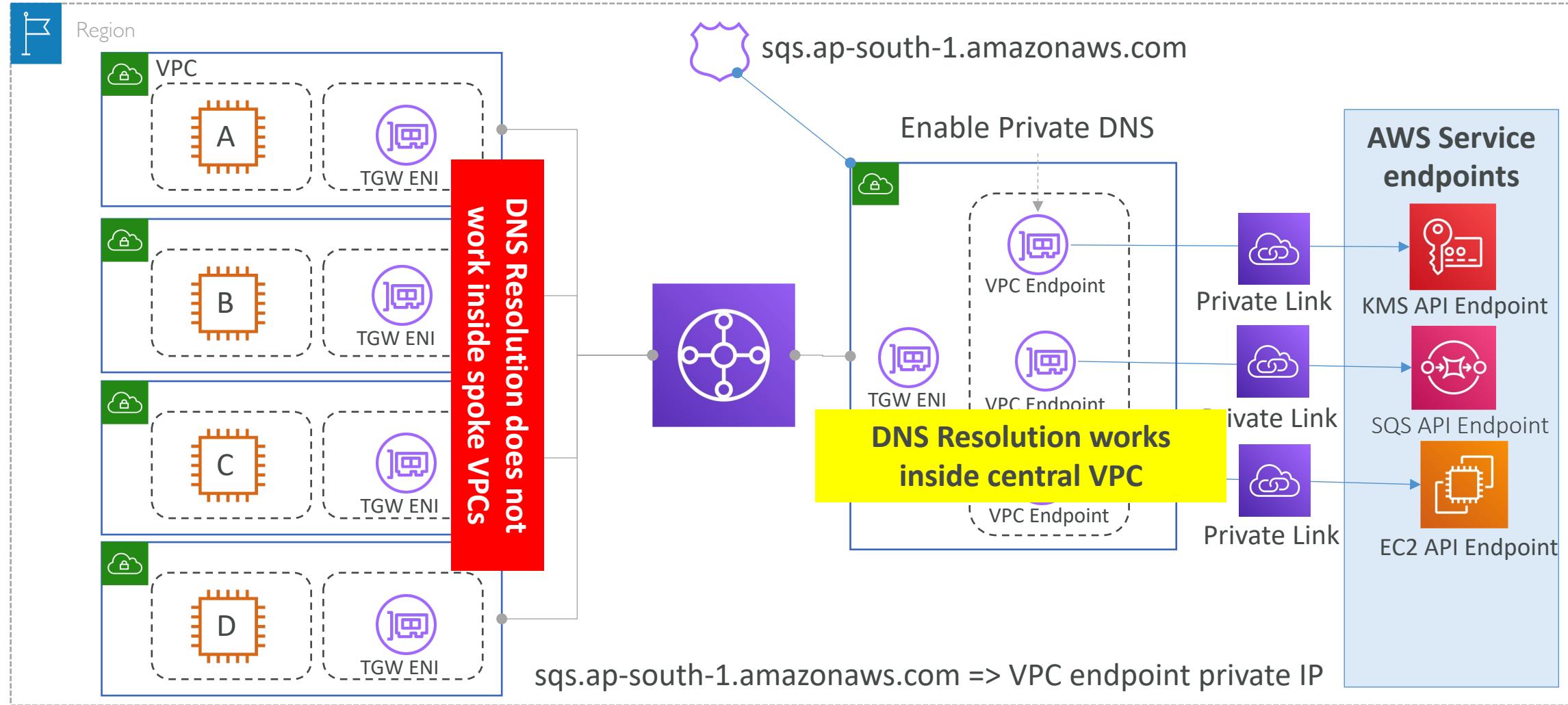
# Interface endpoints



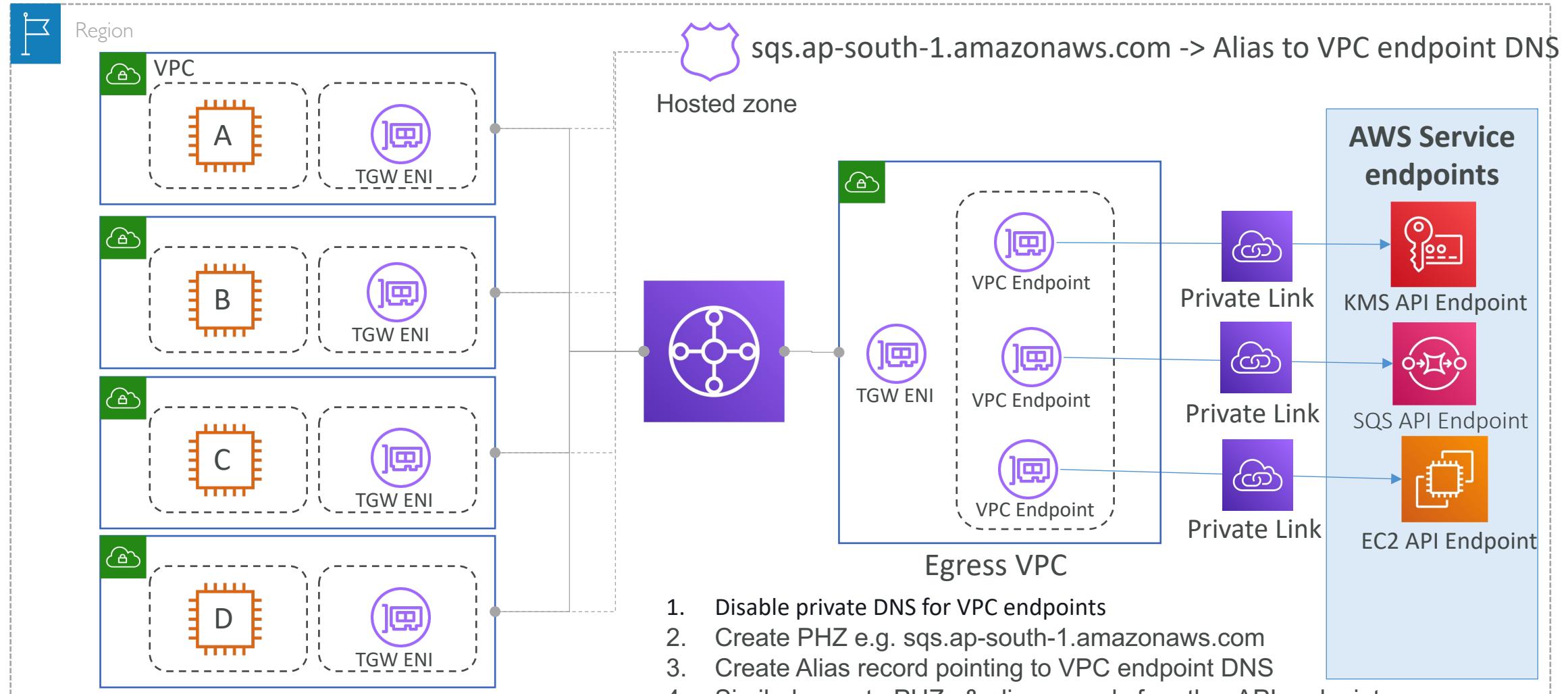
# Centralized VPC interface endpoints



# What about AWS service DNS resolution?



# DNS resolution solution



# Centralized VPC interface endpoints

Important to know:

- VPC interface endpoints provides the regional and AZ level DNS endpoints
- The regional DNS endpoint will return the IP addresses for all the AZ endpoints
- In order to save the inter-AZ data transfer cost from Spoke VPC to Hub VPC, you can use the AZ specific DNS endpoints. Selection has to be done by the client.

```
[cloudshell-user@ip-10-0-112-90 ~]$ nslookup vpce-0b098a7d7394c553e-vzodek17.ec2.ap-south-1.vpce.amazonaws.com
Server:      127.0.0.1
Address:     127.0.0.1#53
[REDACTED]

Non-authoritative answer:
Name:   vpce-0b098a7d7394c553e-vzodek17.ec2.ap-south-1.vpce.amazonaws.com
Address: 10.10.1.230
Name:   vpce-0b098a7d7394c553e-vzodek17.ec2.ap-south-1.vpce.amazonaws.com
Address: 10.10.0.137
```

# Centralized VPC interface endpoints

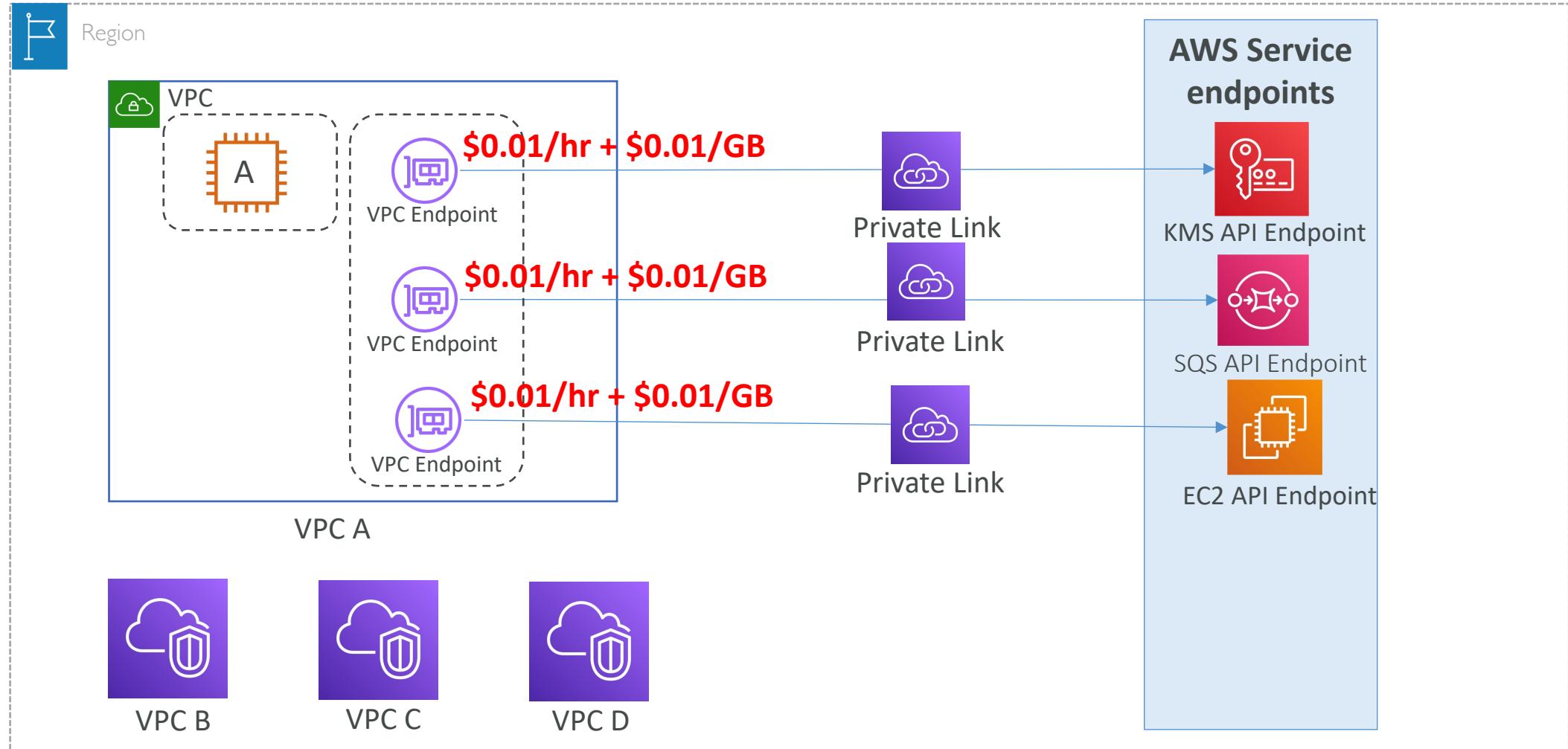
Important to know:

- VPC interface endpoints provides the regional and AZ level DNS endpoints
- The regional DNS endpoint will return the IP addresses for all the AZ endpoints
- In order to save the inter-AZ data transfer cost from Spoke VPC to Hub VPC, you can use the AZ specific DNS endpoints. Selection has to be done by the client.

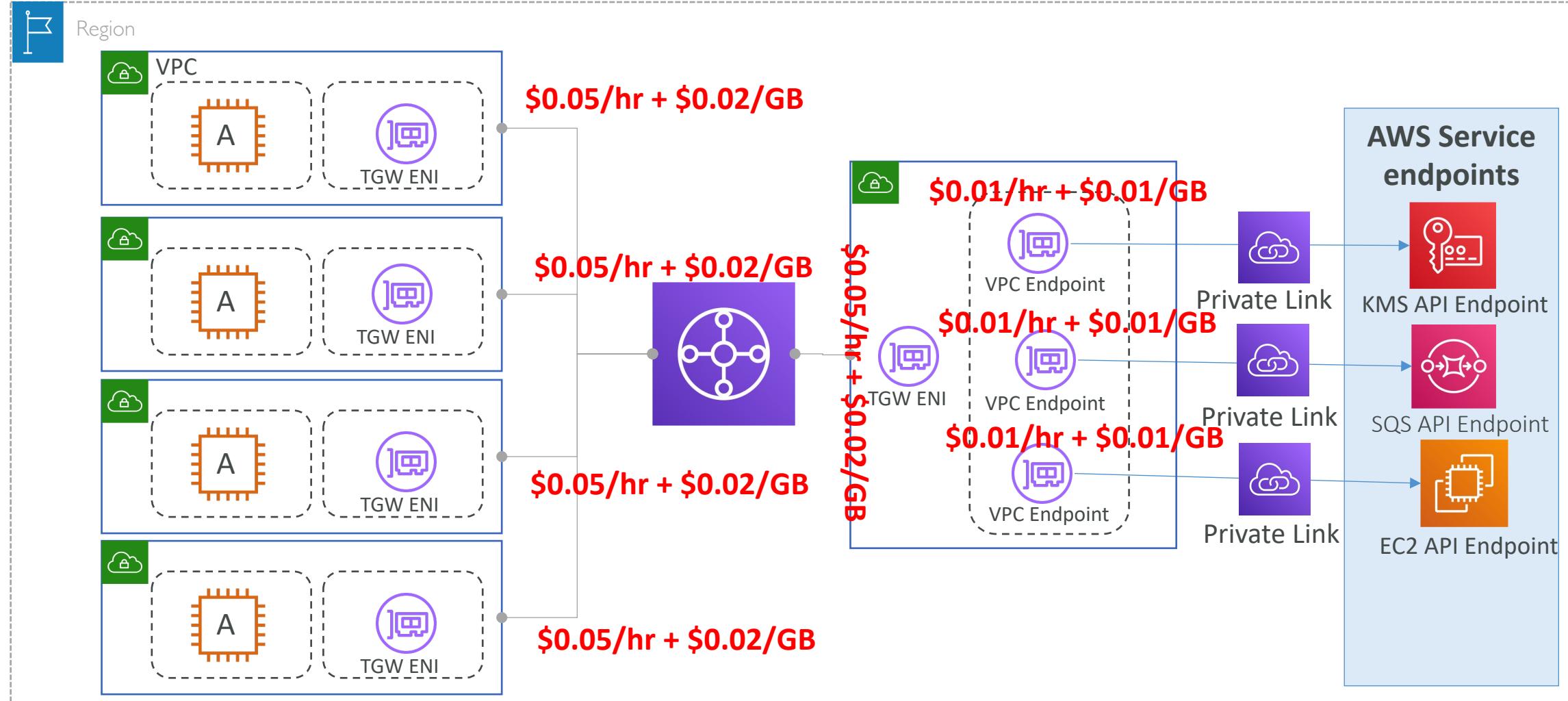
```
[cloudshell-user@ip-10-0-112-90 ~]$ nslookup vpce-0b098a7d7394c553e-vzodek17-ap-south-1a.ec2.ap-south-1.vpce.amazonaws.com
Server:      127.0.0.1
Address:    127.0.0.1#53

Non-authoritative answer:
Name:  vpce-0b098a7d7394c553e-vzodek17-ap-south-1a.ec2.ap-south-1.vpce.amazonaws.com
Address: 10.10.0.137
```

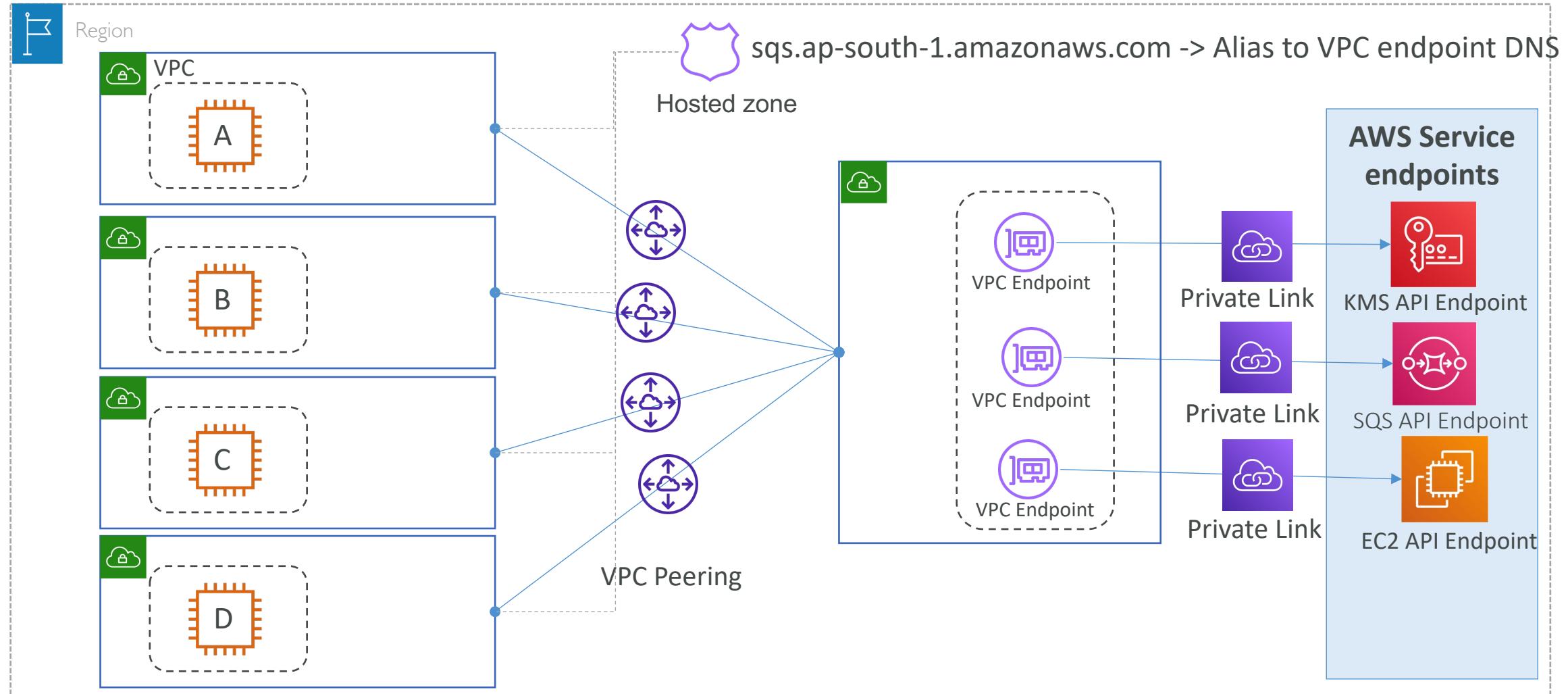
# Let's apply the cost lens..



# Cost of centralized VPC interface endpoints



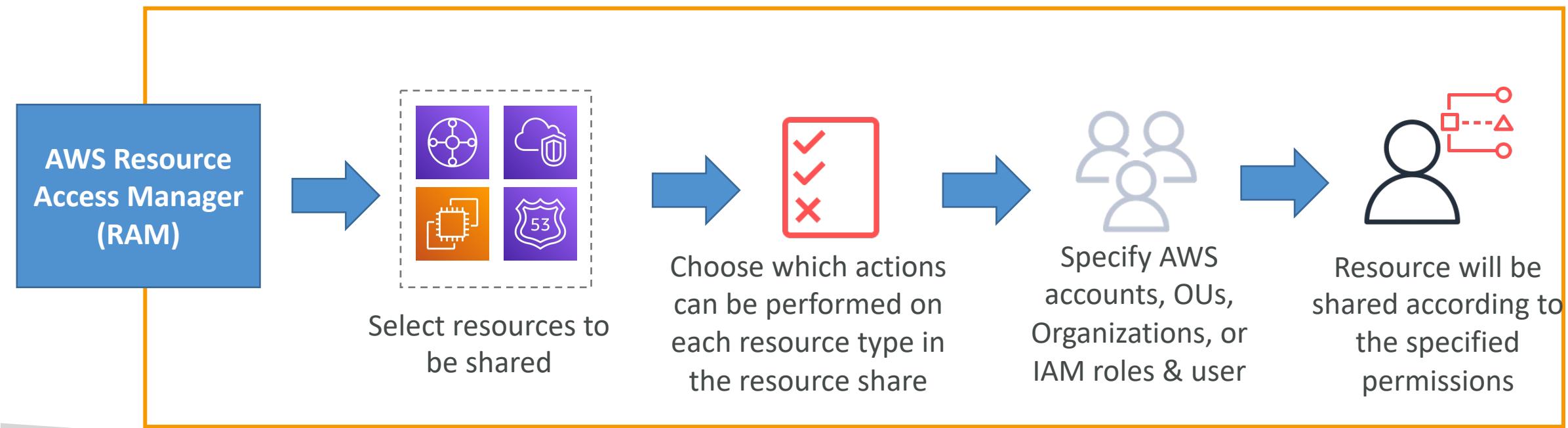
# Using VPC peering instead of Transit Gateway



# Transit Gateway Sharing

# Sharing Transit Gateway

- Use AWS Resource Access Manager (RAM) to share a transit gateway for **VPC attachments** across accounts or across your organization in AWS Organizations



# Important to know about shared transit gateway

- An AWS Site-to-Site VPN attachment must be created in the same AWS account that owns the transit gateway.
- An attachment to a Direct Connect gateway can be in the same AWS account as the Direct Connect gateway, or a different account from the Direct Connect gateway.
- When a transit gateway is shared with the AWS account, that AWS account cannot create, modify, or delete transit gateway route tables, or route table propagations and associations.
- When the transit gateway and the attachment entities are in different accounts, use the Availability Zone ID to uniquely and consistently identify the Availability Zone. For example, `use1-az1` is an AZ ID for the us-east-1 Region and maps to the same location in every AWS account.

# VPC Peering vs Transit Gateway

# VPC Peering vs Transit Gateway

|                                   | VPC Peering  | Transit Gateway   |
|-----------------------------------|--|---|
| Architecture                      | One-One connection – Full Mesh   | Hub and Spoke with multiple attachments   |
| Hybrid Connectivity               | Not supported  | Supported hybrid connectivity via VPN and DX  |
| Complexity                        | Simple for fewer VPCs, Complex as number of VPCs increase  | Simple for any number of VPCs and hybrid network connectivity                             |
| Scale                             | 125 peering / VPC  | 5000 attachments / TGW  |
| Latency                           | Lowest   | Additional Hop  |
| Bandwidth                         | No limit   | 50 Gbps / attachment  |
| Ref Security Group                | Supported  | Not supported   |
| Subnet Connectivity               | For all subnets across AZs   | Only subnets within the same AZ in which TGW attachment is created                        |
| Transitive Routing e.g IGW access | Not supported  | Supported   |
| TCO                               | Lowest – Only Data transfer cost (free within same AZ, \$0.01 across AZs in each direction, \$0.02 across regions) | Per attachment cost + Data transfer cost (e.g N. Virginia: \$0.05 per hour + \$0.02 / GB) |

# TGW – Good to know

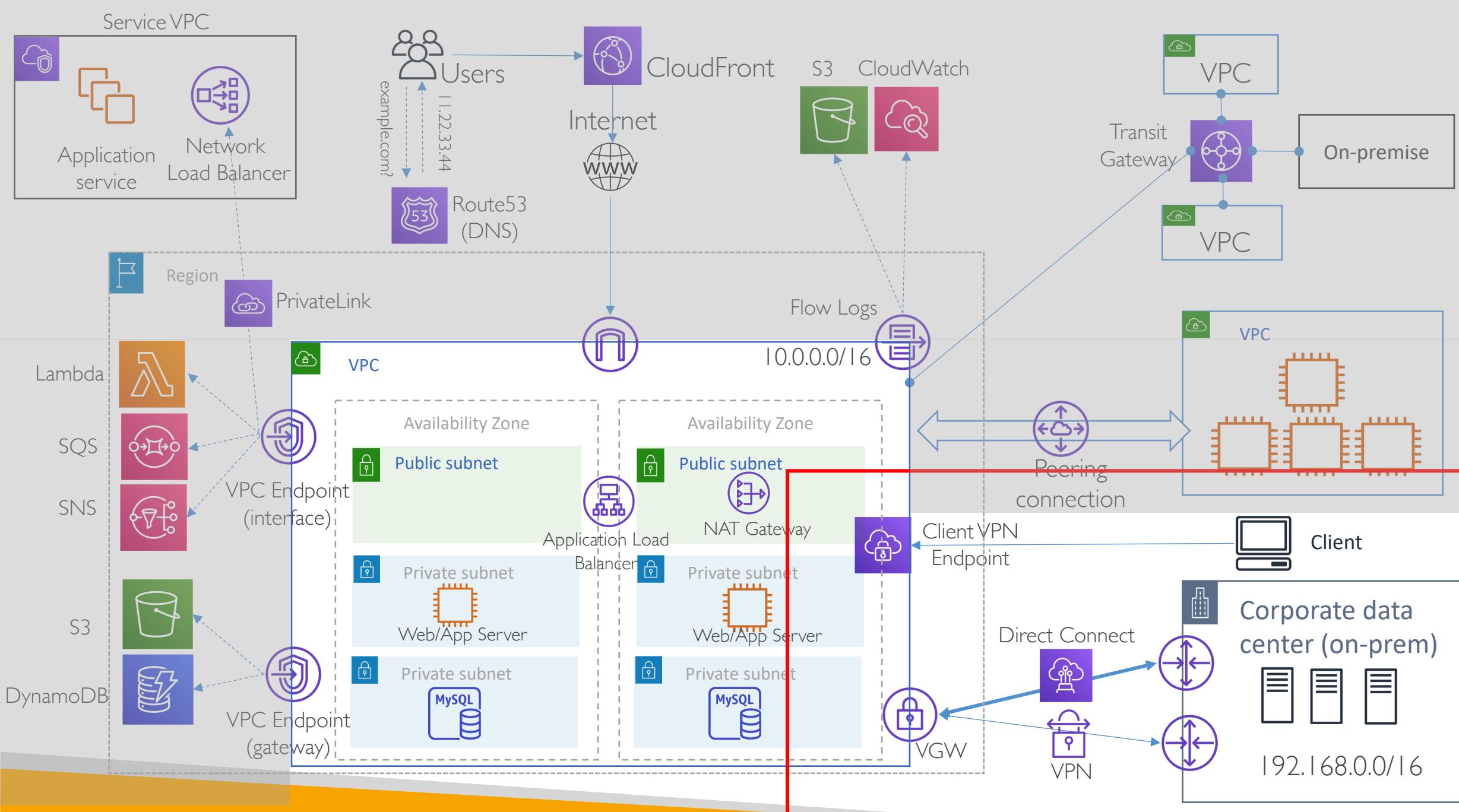
- DNS Resolution is supported for all VPC's attached to the TGW
- Supports resolving 'public' DNS names to Private IP's for EC2
- TGW can be shared using Resource Access Manager (RAM) across AWS accounts
- Billed per hour, per attachment
- Data processing charges apply for each gigabyte sent from an Amazon VPC or AWS Site-to-Site VPN to the AWS Transit Gateway
- Bandwidth at Transit Gateway is limited to 1.25 Gbps per VPN tunnel. With ECMP Transit Gateway can support up to 50 Gbps total VPN bandwidth

# TGW – Good to know

- Transit Gateway supports up to 5000 attachments
- Transit gateway supports an MTU of 8500 bytes for traffic between VPCs, AWS Direct Connect, Transit Gateway Connect, and peering attachments
- Traffic over VPN connections can have an MTU of 1500 bytes.

# Hybrid Network in AWS

AWS VPN and Direct Connect



# Networking Basics – OSI Layer

# OSI Refresher

7 Application

**End User Layer:** HTTP, FTP, SSH, DNS

6 Presentation

**Syntax Layer:** SSL, IMAP, MPEG, Encryption/Compression

5 Session

**Authentication, Authorization, Session Mgmt:** APIs, Sockets, WinSock

4 Transport

**End to End Connections:** TCP, UDP

3 Network

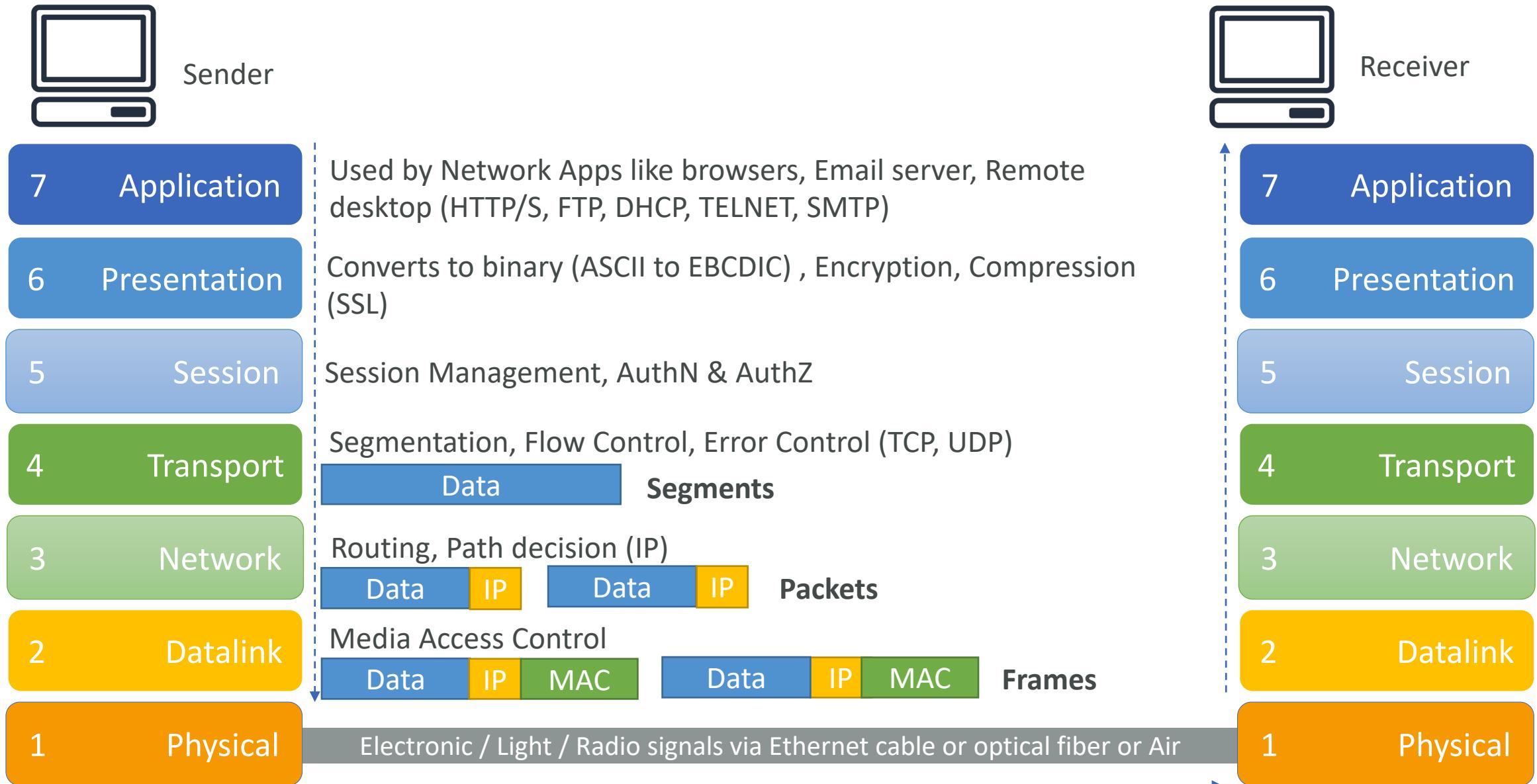
**Packets:** IP, ICMP, IPSec

2 Datalink

**Frames:** Ethernet, PPP, Switch, Bridge

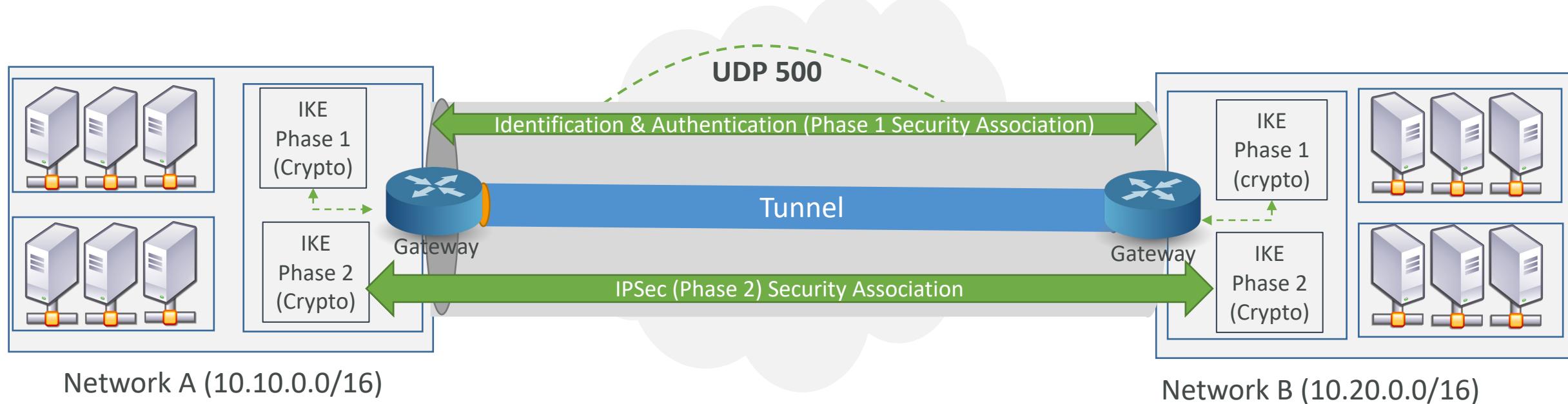
1 Physical

**Physical Structure:** Coax, Fiber, Wireless, Hubs, Repeaters



# How IPSec VPN works?

# Let's see how IPSec VPN works



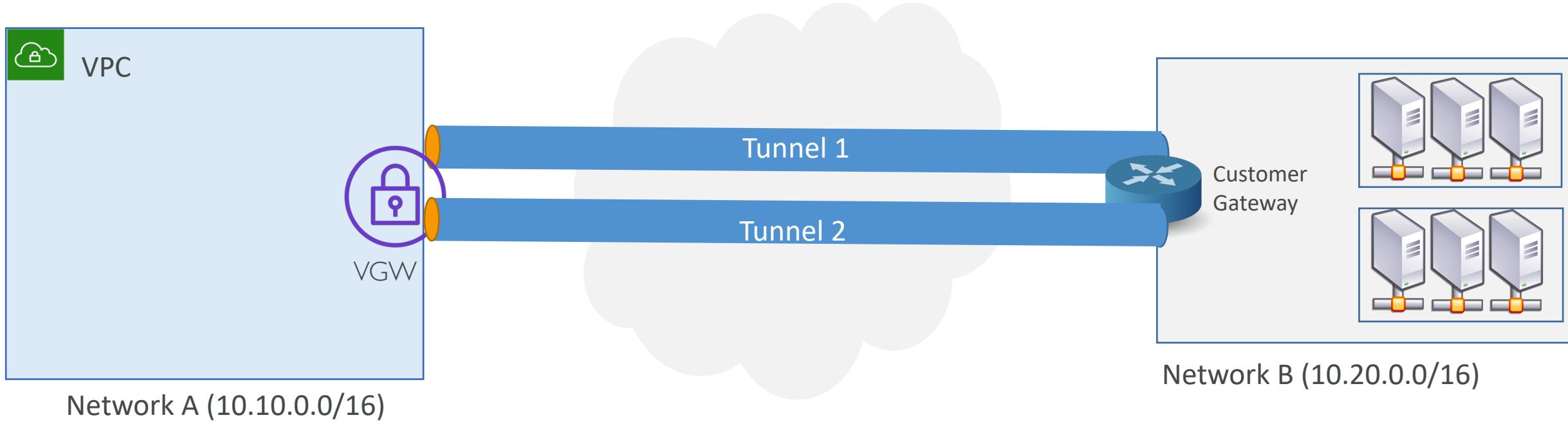
## IKE Phase 1 (IKE SA)

- Decide IKEv1 or IKEv2
- Exchange Credentials (Pre-shared key or Certs)
- Decide on NAT-T, Dead Peer Detection, Keep-Alive
- Auth type – SHA-2, SHA-1, MD5
- Encryption – DES, 3DES, AES
- SA Life – Phase 1 SA expiration (Typically 8 hrs)
- DH Group - 2, 14-24

## IKE Phase 2 (IPSec SA)

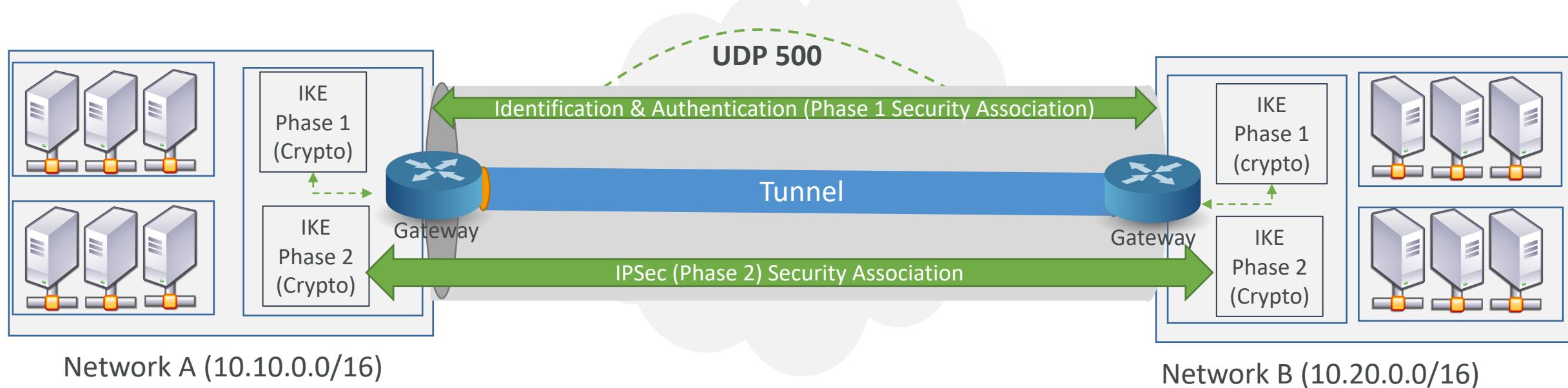
- Security Protocol – AH or ESP (ESP is better)
- Auth type – SHA-2, SHA-1, MD5
- Encryption – DES, 3DES, AES
- SA Life – Phase 2 SA expiration (Typically 1 hr)
- DH Group - 2, 5, 14-24 (If using PFS)

# AWS IPSec VPN



- Virtual Private Gateway (VGW) is used on AWS side of the VPN Connection
- Customer Gateway (CGW) is used on the customer side of the VPN connection
- There are 2 Tunnels for High Availability (total 4 SAs)

# Let's see how IPSec VPN works



## IKE Phase 1 (IKE SA)

- Decide IKEv1 or IKEv2
- Exchange Credentials (Pre-shared key or Certs)
- Decide on NAT-T, Dead Peer Detection, Keep-Alive
- Auth type – SHA-2, SHA-1, MD5
- Encryption – DES, 3DES, AES
- SA Life – Phase 1 SA expiration
- DH Group - 2, 14-24

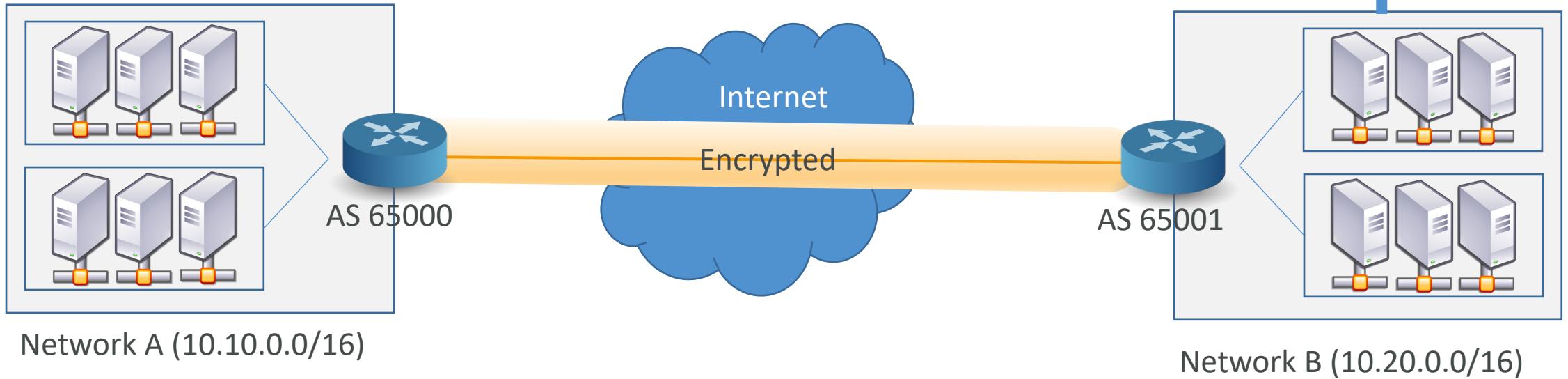
## IKE Phase 2 (IPSec SA)

- Decide whether to use PFS – Prevent using Phase 1 keys
- Security Protocol – AH or ESP (ESP is better)
- Auth type – SHA-2, SHA-1, MD5
- Encryption – DES, 3DES, AES
- Force Key Expiration – Typically 8 hours
- DH Group - 2, 5, 14-24

# VPN Routing – Static vs Dynamic

# AS – Autonomous Systems

- Routers controlled by one entity in a network
- Assigned by IANA for Public ASN (0-65525)
- Private ASN (64512-65534)



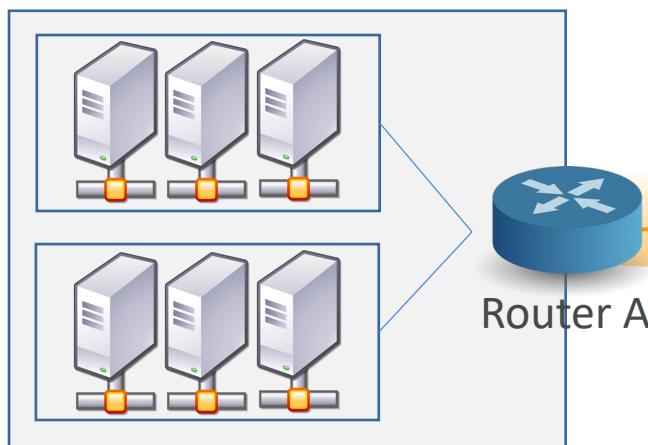
# Static Routing

Network A

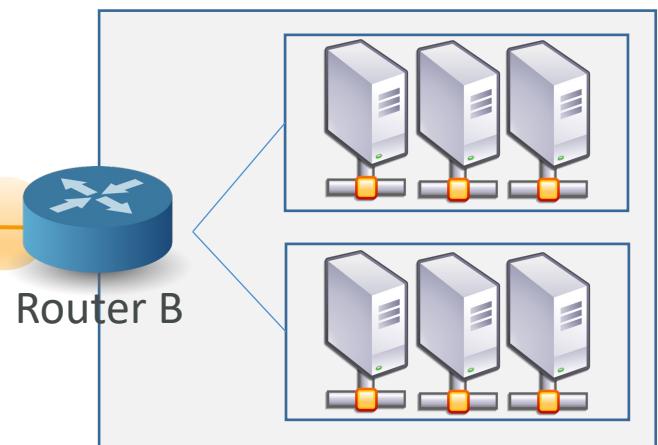
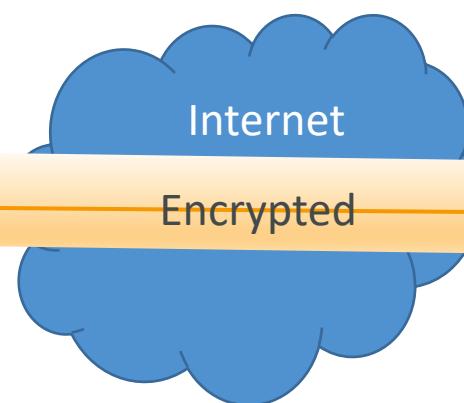
| Destination         | Target          |
|---------------------|-----------------|
| 10.10.0.0/16        | Local           |
| <b>10.20.0.0/16</b> | <b>Router B</b> |

Network B

| Destination  | Target   |
|--------------|----------|
| 10.20.0.0/16 | Local    |
| 10.10.0.0/16 | Router A |



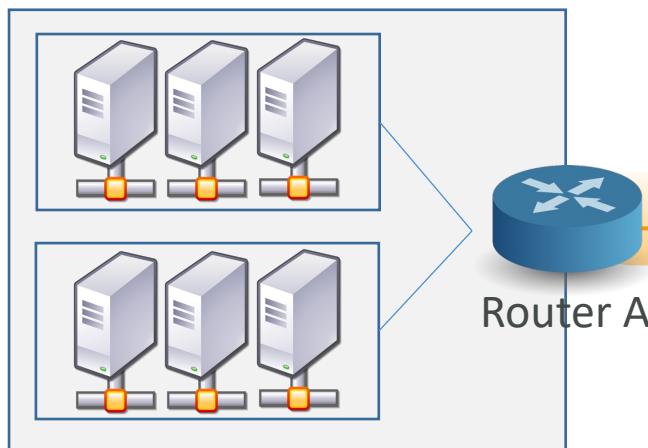
Network A (10.10.0.0/16)



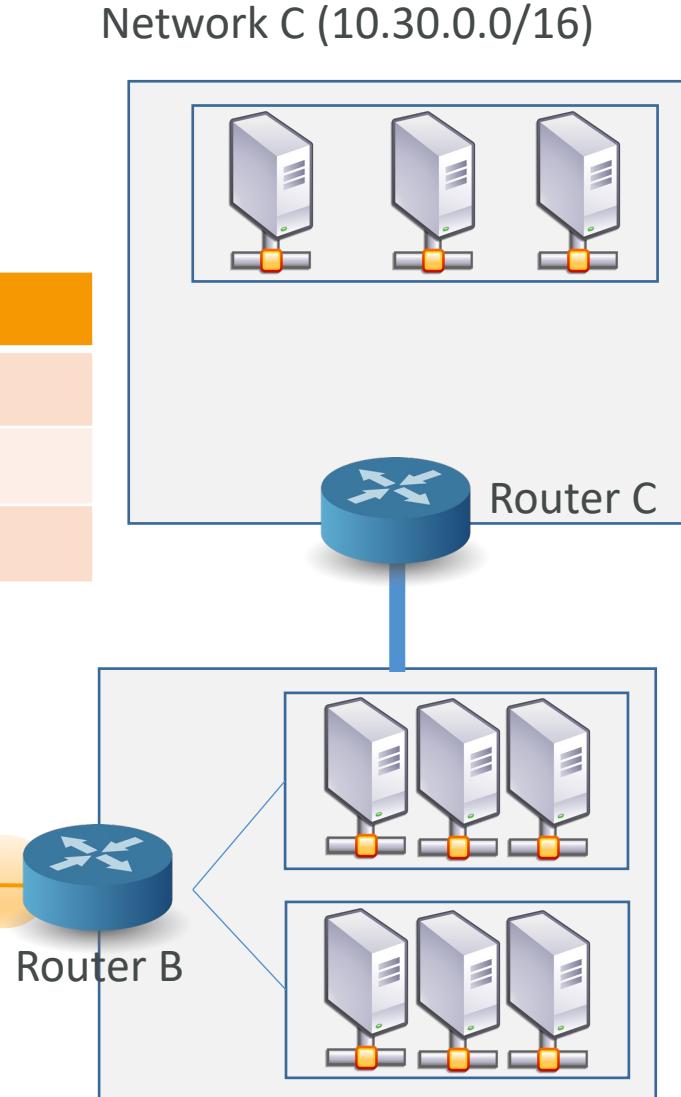
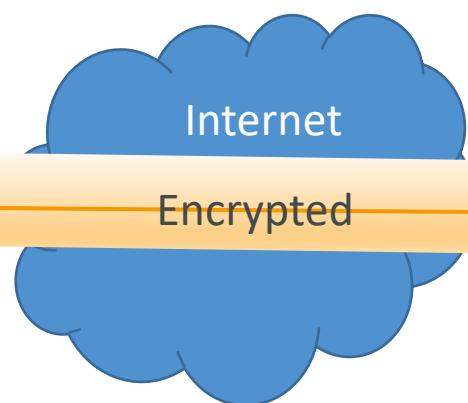
Network B (10.20.0.0/16)

# Static Routing

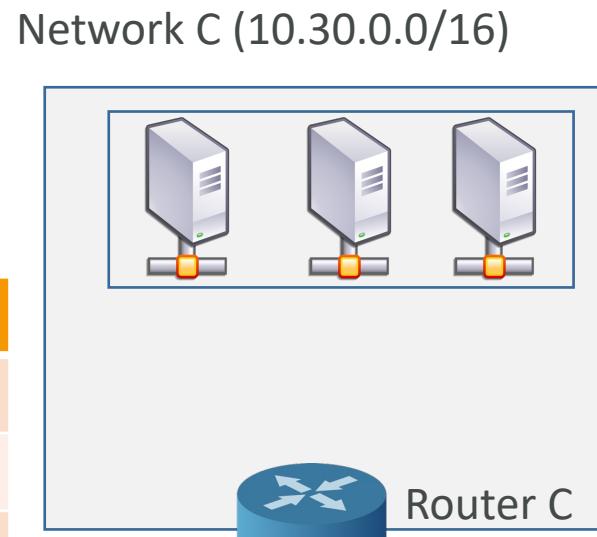
| Network A           |                 | Network B    |          |
|---------------------|-----------------|--------------|----------|
| Destination         | Target          | Destination  | Target   |
| 10.10.0.0/16        | Local           | 10.20.0.0/16 | Local    |
| <b>10.20.0.0/16</b> | <b>Router B</b> | 10.10.0.0/16 | Router A |
|                     |                 | 10.30.0.0/16 | Router C |



Network A (10.10.0.0/16)



Network B (10.20.0.0/16)

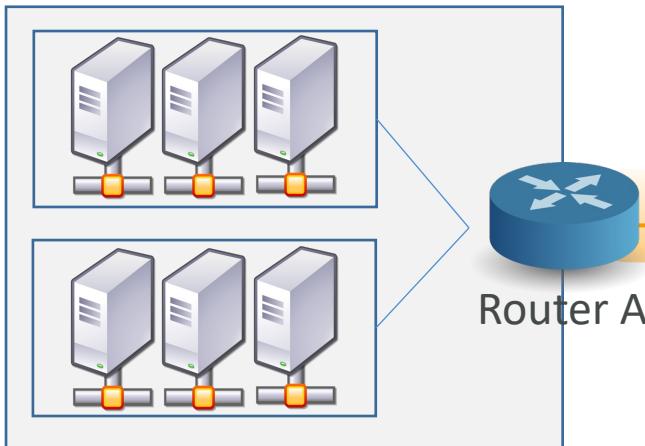


# Static Routing

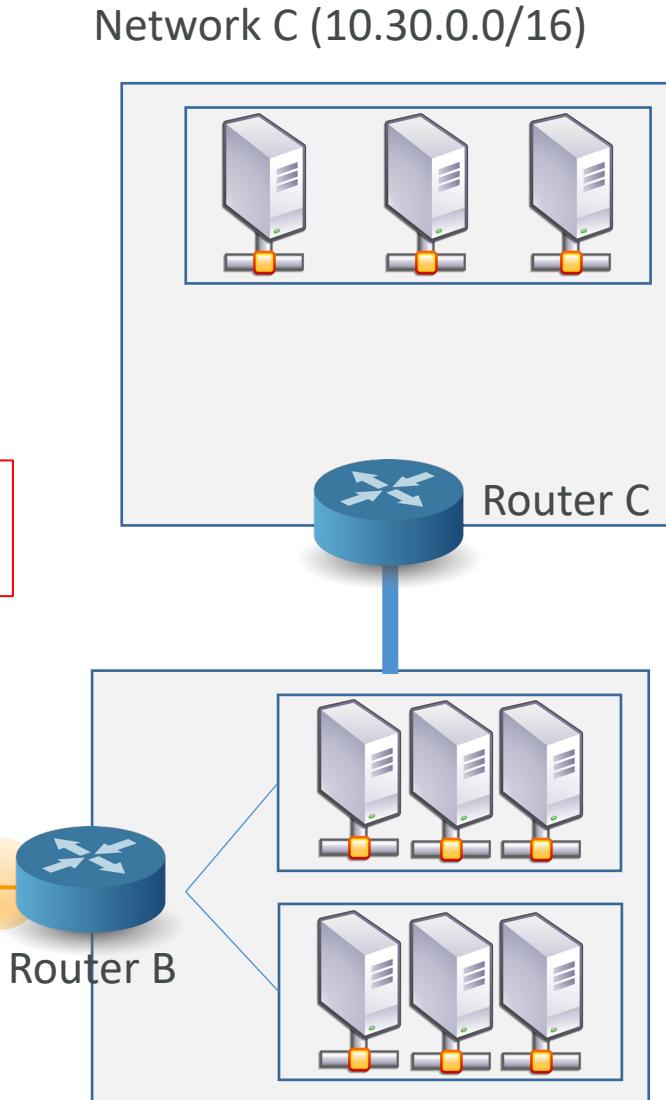
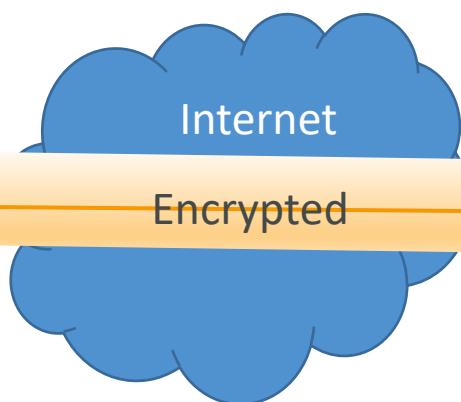
Network A

| Destination  | Target   |
|--------------|----------|
| 10.10.0.0/16 | Local    |
| 10.20.0.0/16 | Router B |

In static routing we need to add this route explicitly



Network A (10.10.0.0/16)



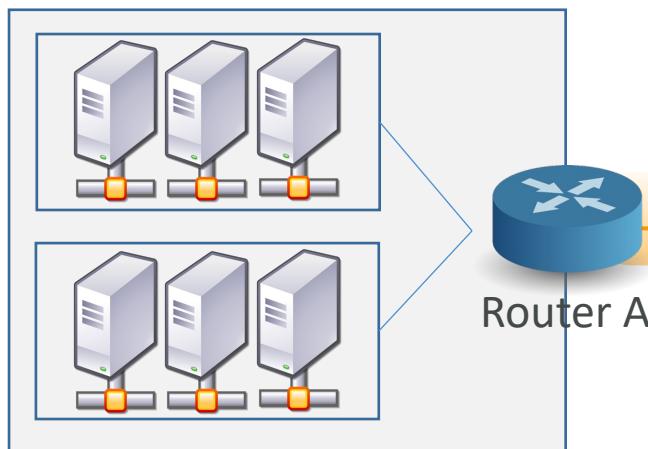
Network C (10.30.0.0/16)

# Dynamic Routing

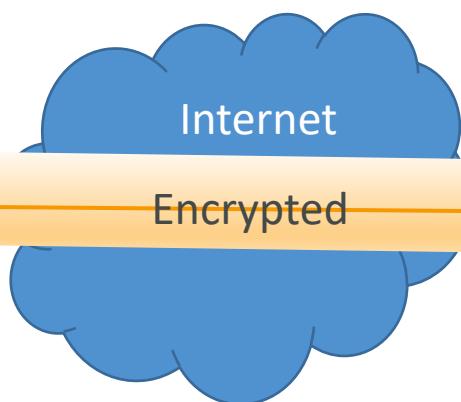
Network A

| Destination         | Target          |
|---------------------|-----------------|
| 10.10.0.0/16        | Local           |
| <b>10.20.0.0/16</b> | <b>Router B</b> |

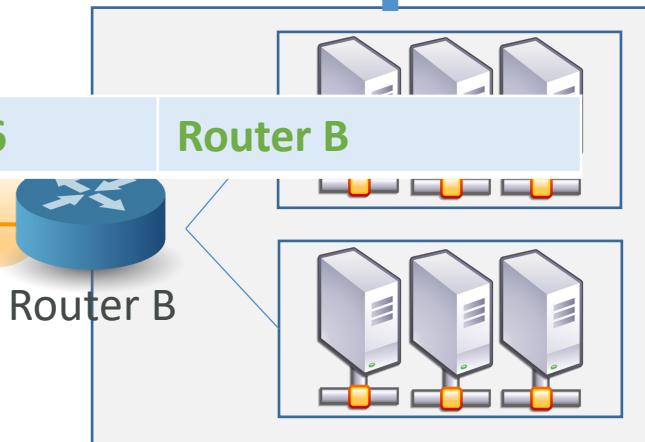
In dynamic routing the routes get propagated automatically



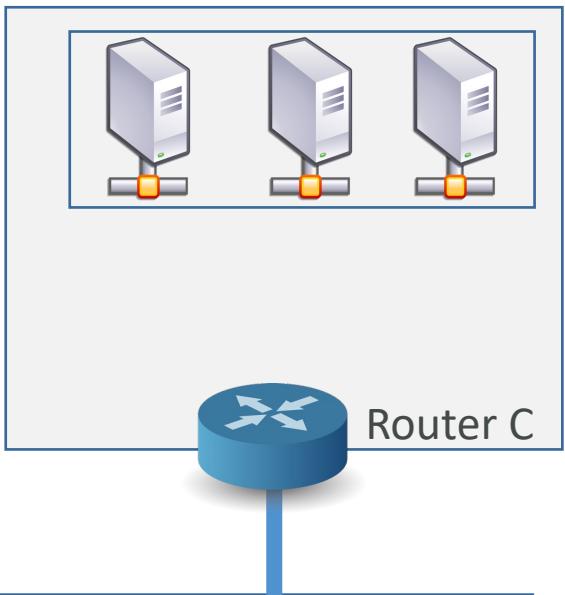
Network A (10.10.0.0/16)



**10.30.0.0/16**



Network B (10.20.0.0/16)



Network C (10.30.0.0/16)

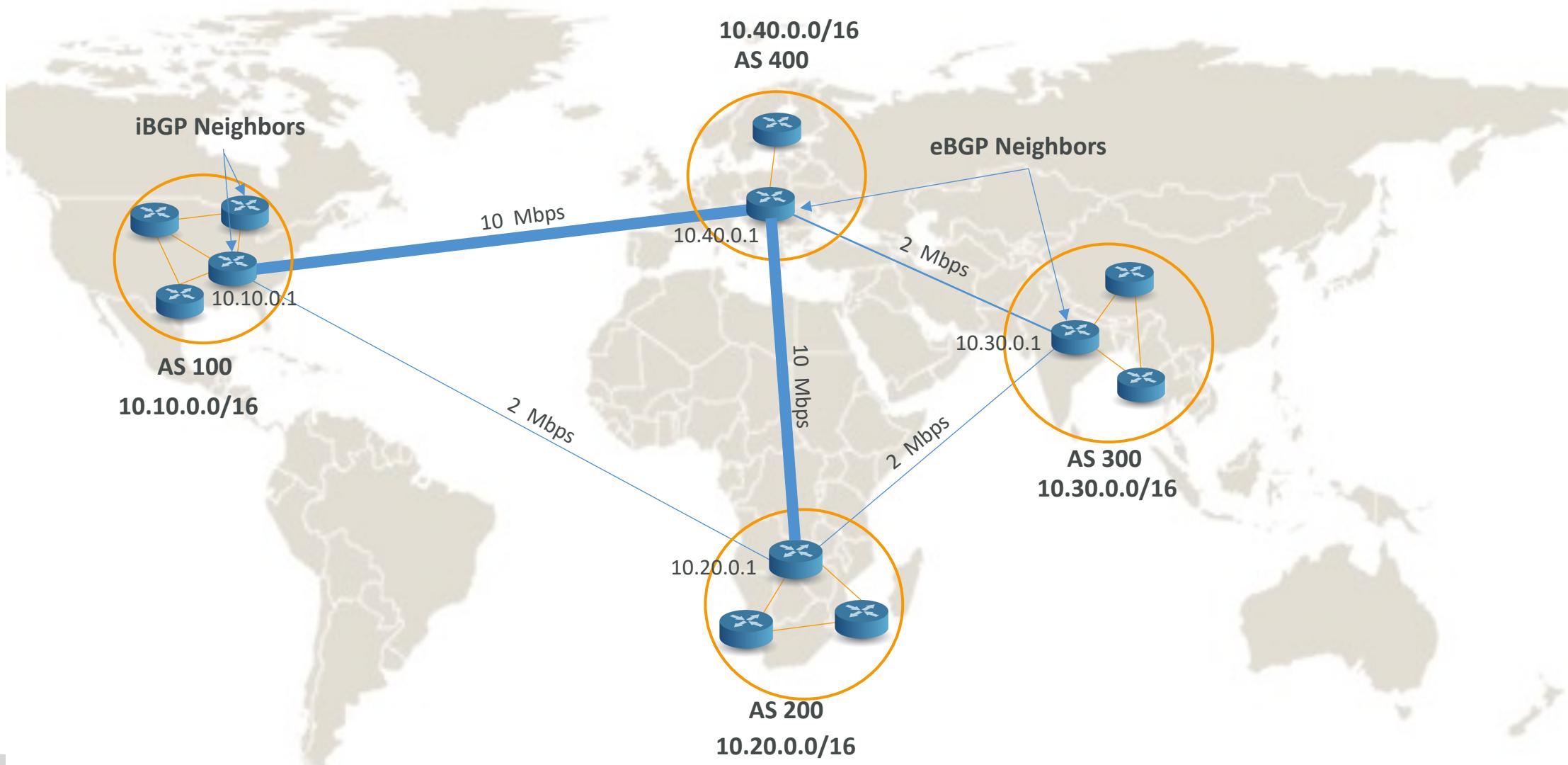
# Dynamic Routing using Border Gateway Protocol (BGP)

# BGP – Border Gateway Protocol

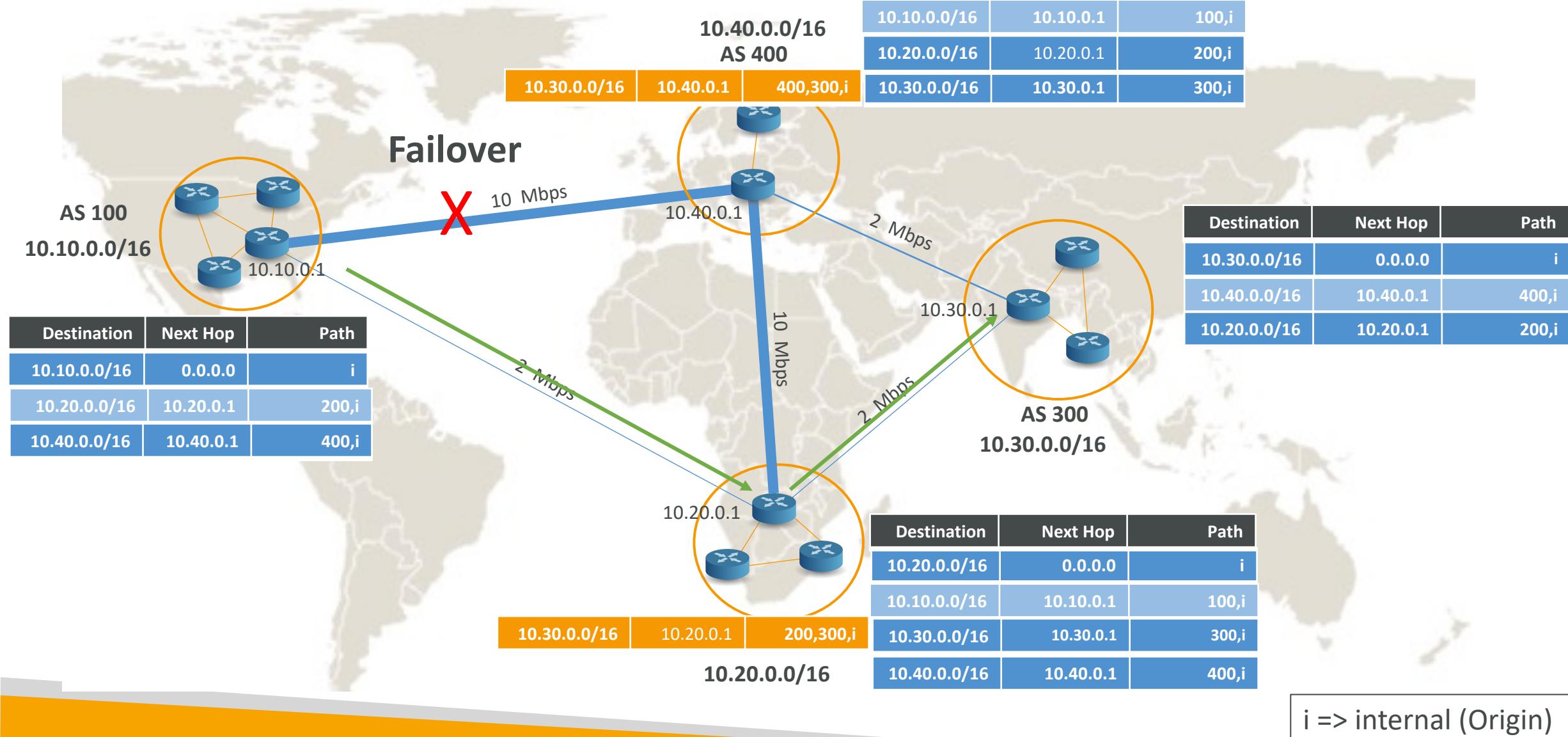
- Dynamic Routing using **Path-Vector** protocol where it exchanges the best path to a destination between peers or AS (ASPATH)
- iBGP – Routing within AS
- eBGP – Routing between AS's
- Routing decision is influenced by
  - Weight (Cisco routers specific – works within AS)
  - ASPATH – Series of AS's to traverse the path (Works between AS)
  - Local Preference LOCAL\_PREF (Works within AS)
  - MED – Multi-Exit Discriminator (Works between AS)

The current version of BGP is version 4 (BGP4), which was published as [RFC 4271](#) in 2006

# How BGP works



# How BGP works

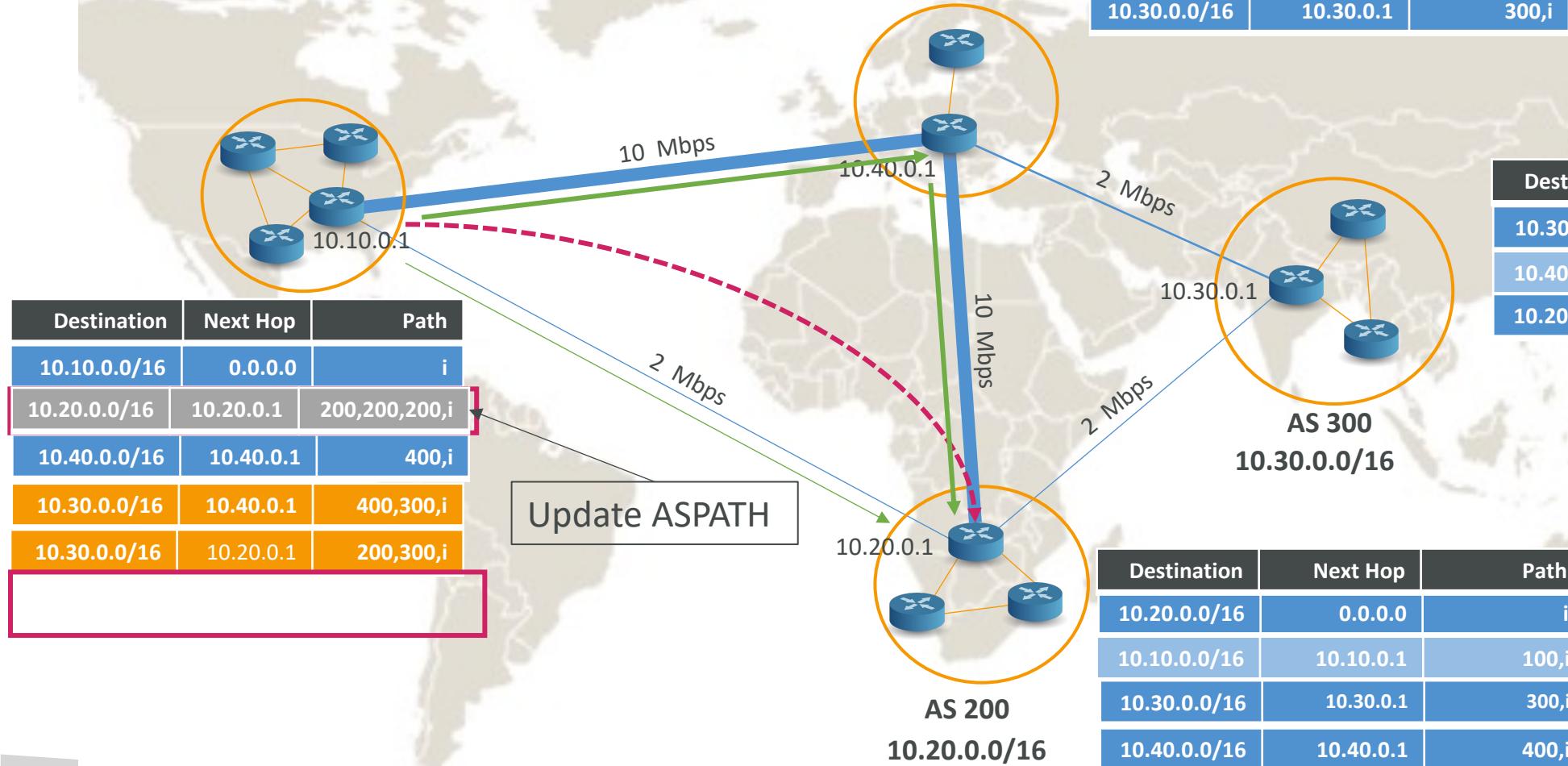


# How BGP works

## ASPATH

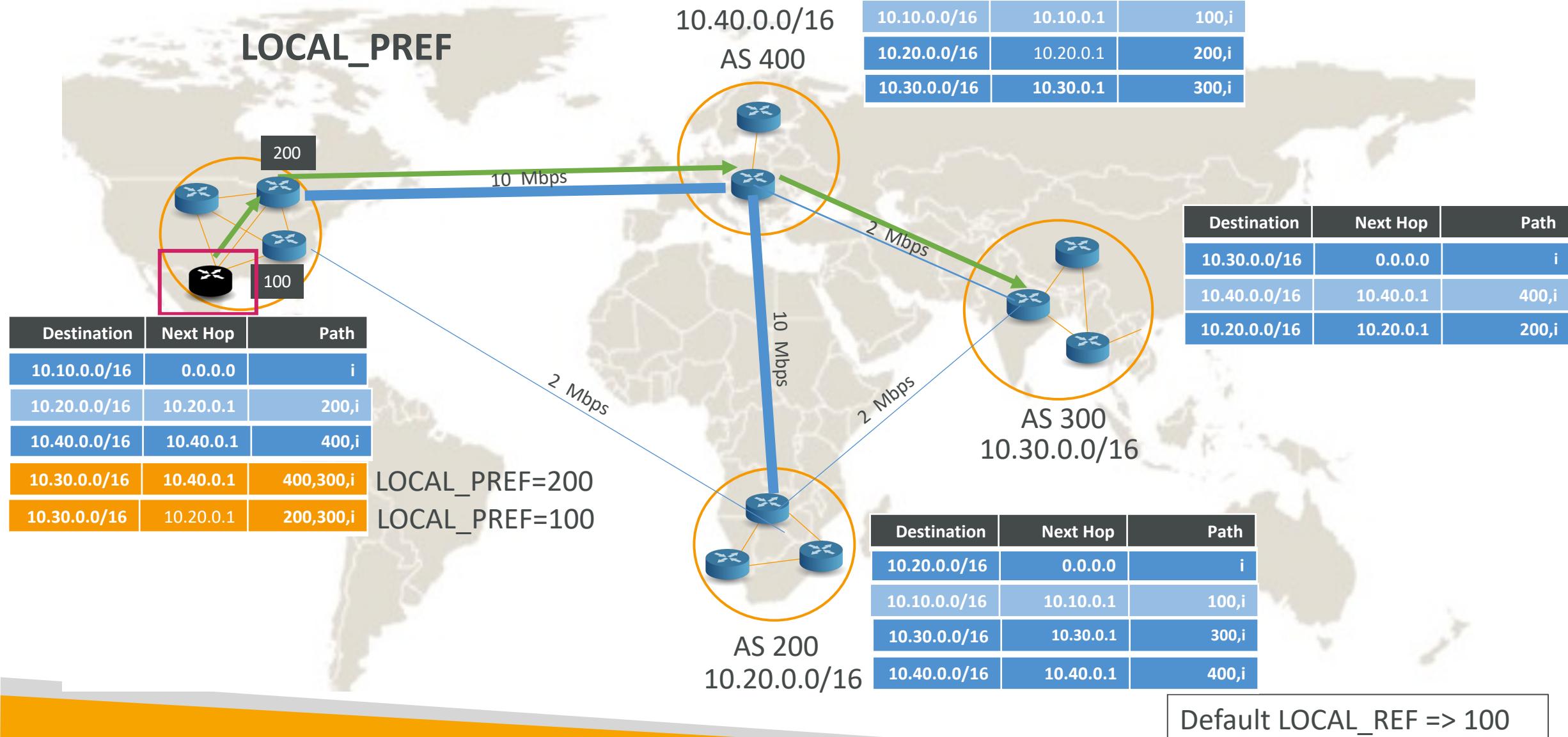
**10.40.0.0/16**  
 10.20.0.0/16 | 10.40.0.1 | 400,200,i

| Destination  | Next Hop  | Path  |
|--------------|-----------|-------|
| 10.40.0.0/16 | 0.0.0.0   | i     |
| 10.10.0.0/16 | 10.10.0.1 | 100,i |
| 10.20.0.0/16 | 10.20.0.1 | 200,i |
| 10.30.0.0/16 | 10.30.0.1 | 300,i |



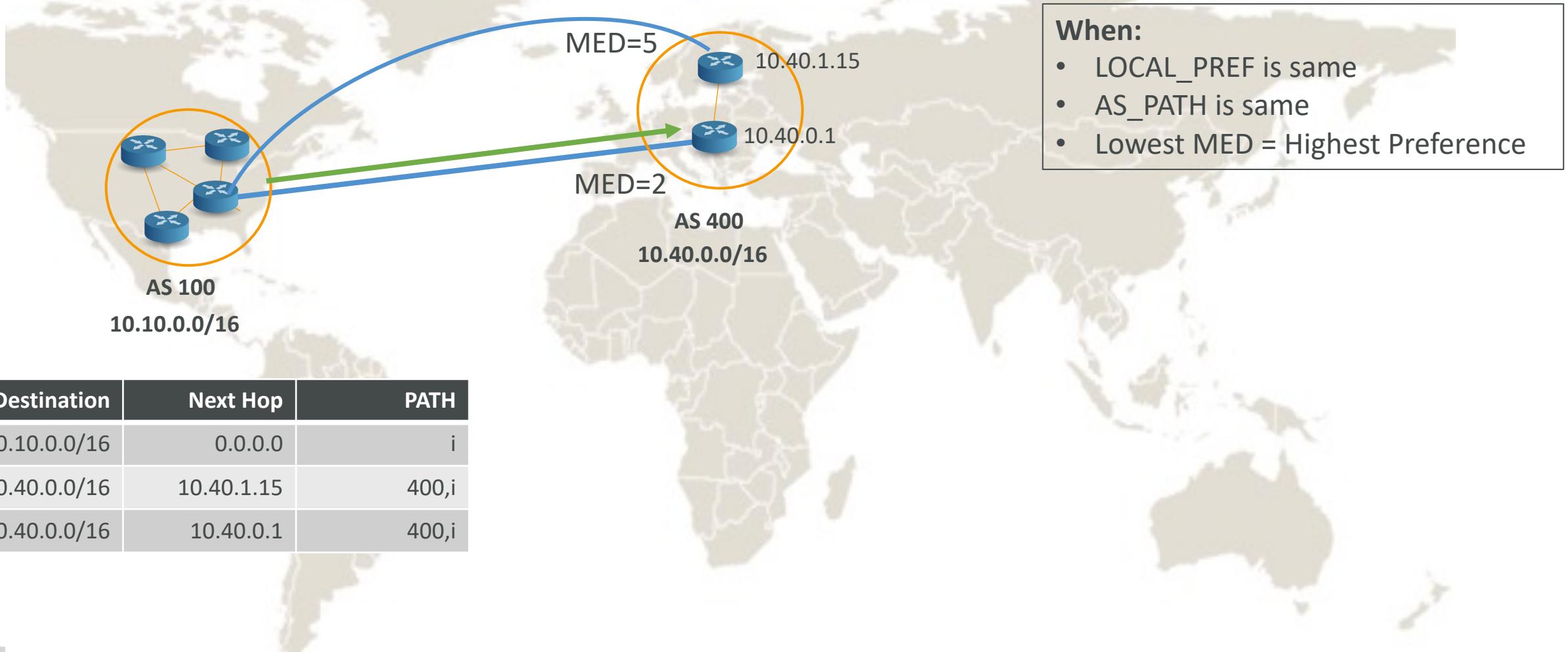
# How BGP works

## LOCAL\_PREF



# How BGP works

## MED – Multi Exit Discriminator



# Quick recap of BGP Route selection order and attributes

- Highest Weight
  - Cisco routers specific attribute
  - Works within the AS and set by the originating router
  - Not exchanged between external BGP routers
- Highest local preference
  - Choose the outbound external BGP path
  - Set by the local AS for internal BGP routers
  - Not exchanged between external BGP routers
  - Default value 100
- Shorted AS Path
  - AS Path can be modified using AS path prepend to make it appear longer
- Lowest Multi Exit Discriminator (MED)
  - Used when there are multiple paths between two AS's
  - MED is exchanged **between** autonomous systems

# AWS Site-to-Site VPN

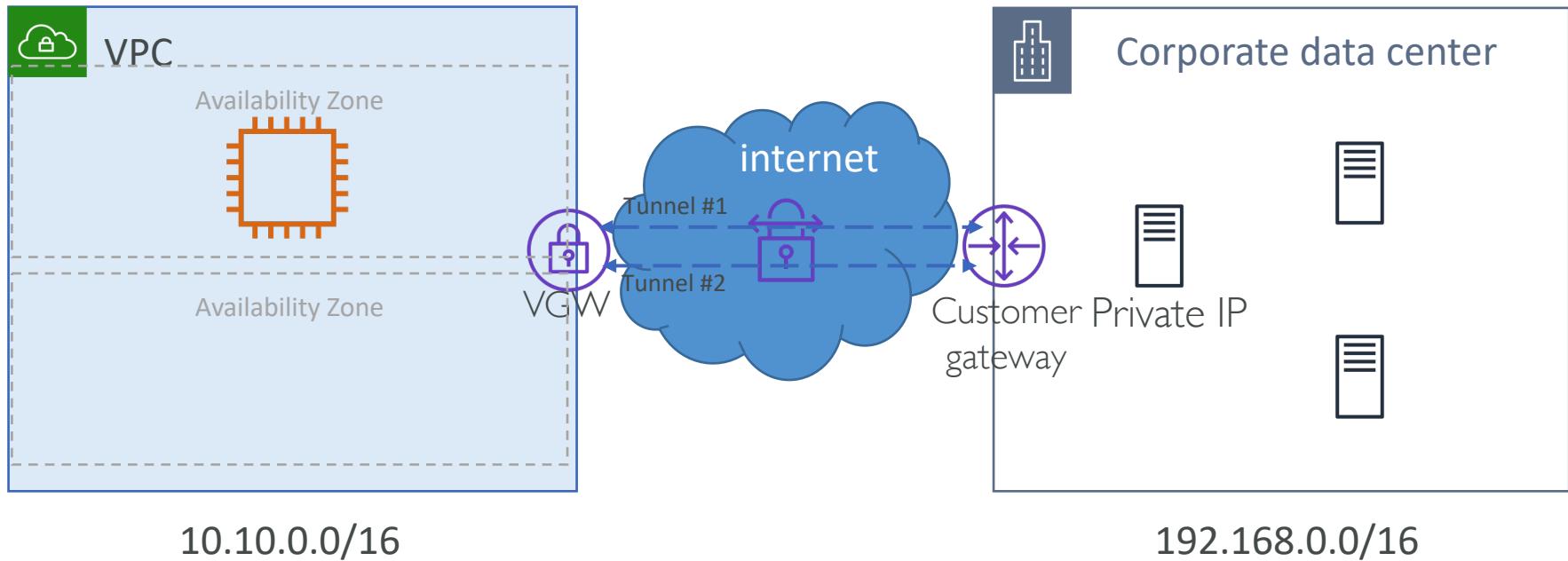


# VPN Basics

- VPN allows hosts to communicate privately over an untrusted intermediary network like internet, in **encrypted form**
- AWS supports Layer 3 VPN (not Layer 2)
- VPN has 2 forms – Site to Site VPN and Client to Site VPN
  - Site to Site VPN connects 2 different networks.
  - Client to Site VPN connects the client device like laptop to the private network
- VPN types
  - IPSec (IP Security) VPN which is supported by AWS managed VPN
  - Other VPNs like GRE and DMVPN are not supported by AWS managed VPN.

# How Site-to-Site VPN works in AWS

- VPN connection terminated at Virtual Private Gateway (VGW) on AWS end
- VGW creates 2 Tunnel endpoints in different AZs for High Availability



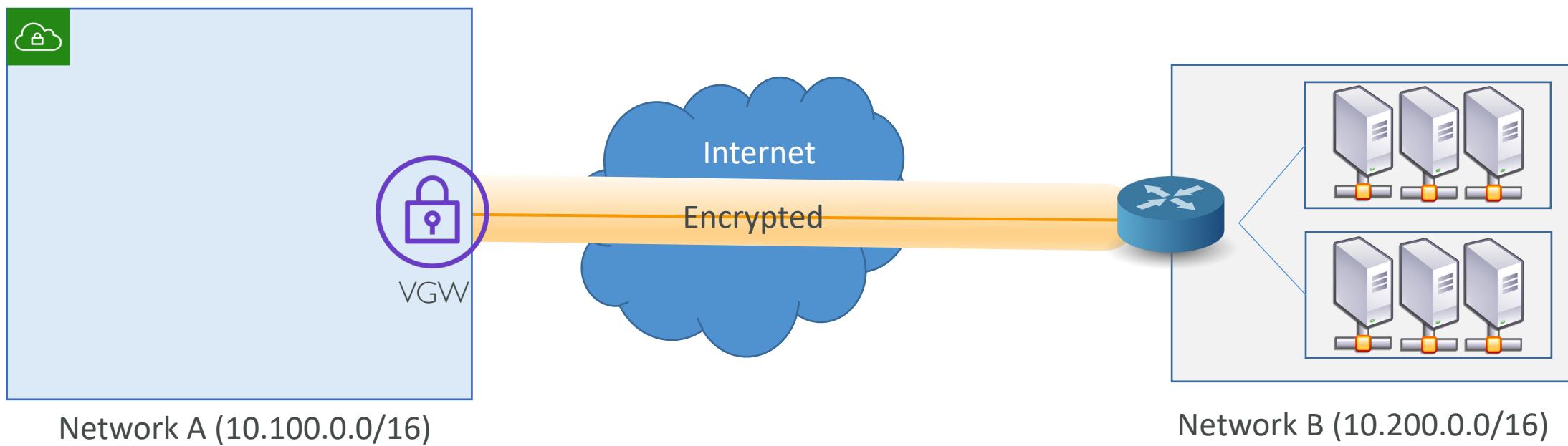


# Virtual Private Gateway (VGW)

- Managed gateway endpoint for the VPC
- Only one VGW can be attached to VPC at a time
- VGW supports both Static Routing and Dynamic routing using Border gateway protocol (BGP)
- For BGP, you can assign the private ASN (Autonomous System Number) to VGW in the range of 64512 to 65534
- If you don't define ASN, AWS assigns default ASN. ASN can not be modified once assigned (default 64512)
- VGW Supports AES-256 and SHA-2 for encryption and data integrity

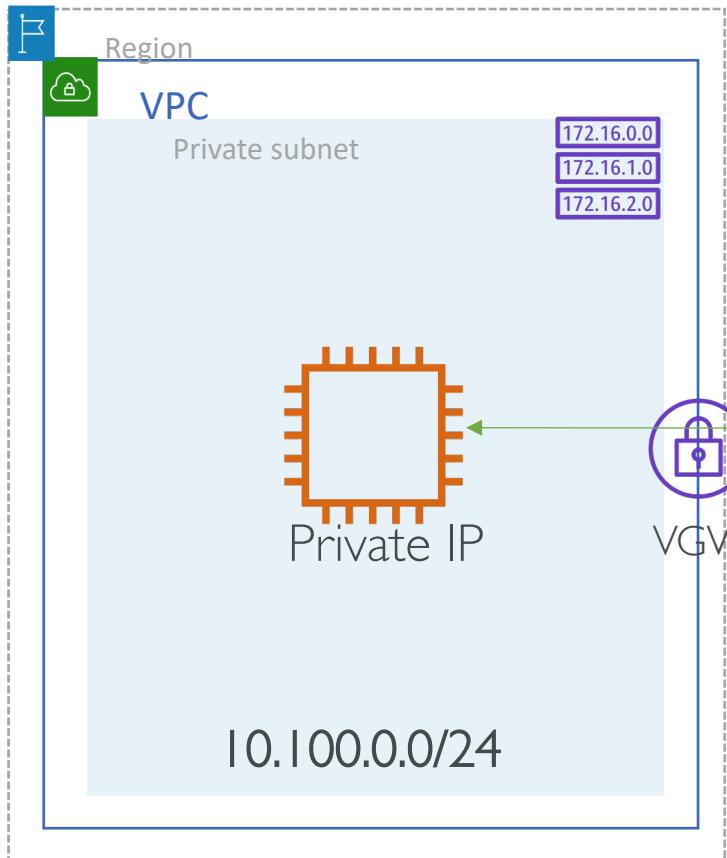
# VPN Demo

- Network A – AWS Network
- Network B – Customer Network

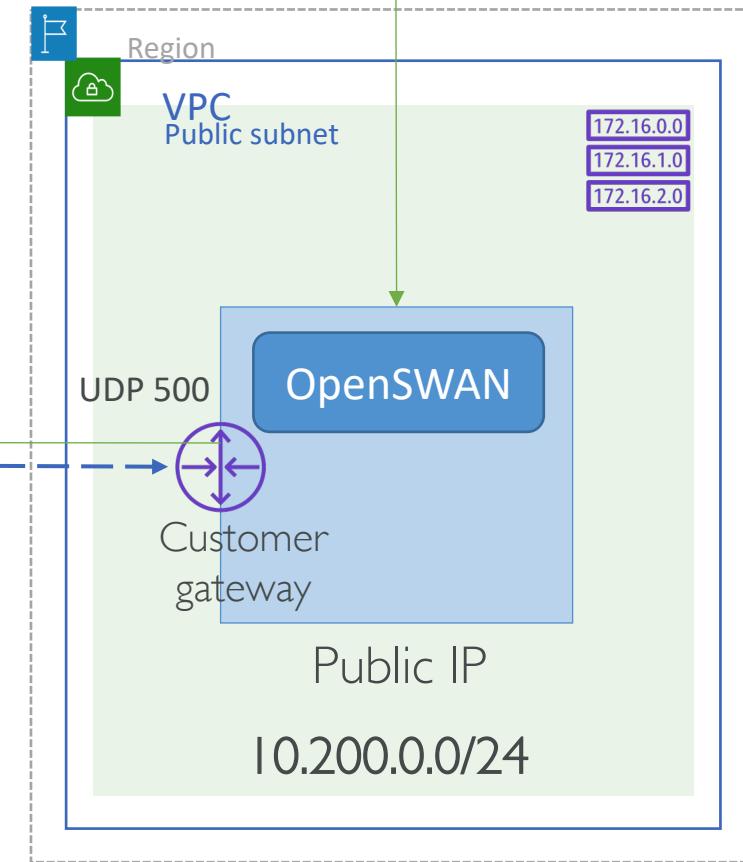


# VPN Demo

Mumbai Region



N Virginia Region



Customer Network

# Site to Site VPN Setup steps

1. Create VPN gateway on AWS side and attach to VPC
  - You can keep the ASN default or provide your own private ASN (16 bit)
2. Create Customer Gateway (by mentioning Customer side Public IP)
  - Specify routing method – Static or Dynamic
  - In case of Dynamic, you must provide ASN number
  - Specify the Public IP address on customer side
3. Create Site-to-Site VPN connection by selecting VGW and CGW
  - If you select Static routing, then provide customer side CIDR IP prefix ranges
  - Provide tunnel details like Pre shared key, Inside tunnel Ips / 30 or leave blank
4. Download the VPN configuration file as per your side of the router vendor e.g Cisco, Juniper or you can also download generic vendor file
5. Use VPN configuration file to configure your side of the VPN connection

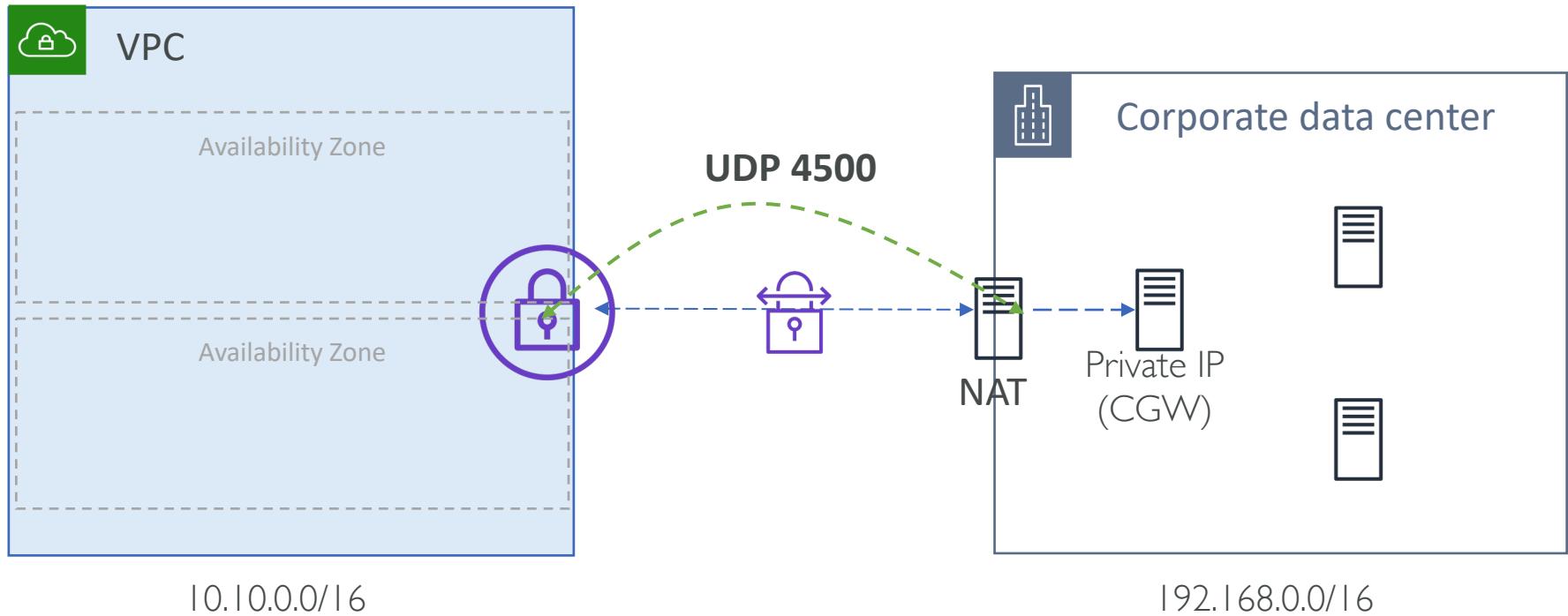
# VPN Exercise

- Download the exercise doc and set up VPN on your own to remember all the steps and options available

# VPN NAT Traversal (NAT-T)

# VPN support for NAT-Traversal (NAT-T)

- AWS VPN supports the VPN termination behind NAT on customer side
- You must open UDP port 4500 on customer side of firewall for NAT-T

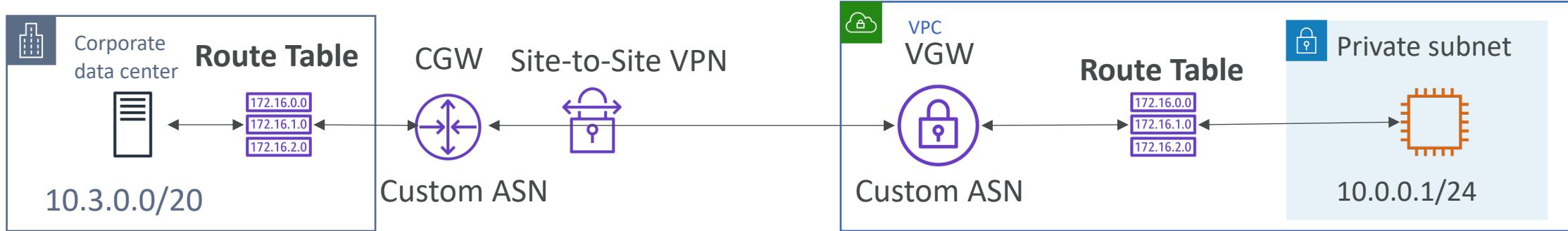


# VPN Route Propagations

# VPN Static and Dynamic routing

- In case of static routing, you must pre-define the CIDR ranges on both sides of the VPN connection. If you add new network ranges on either sides, the routing changes are not propagated automatically
- In case of Dynamic routing, both the ends learns the new network changes automatically. On AWS side, the new routes are automatically propagated in the route tables.
- AWS route table can not have more than 100 propagated routes. Hence you need to make sure you don't publish more than 100 routes from on-premises network
- To avoid this situation, you may think to consolidate network ranges into larger CIDR range

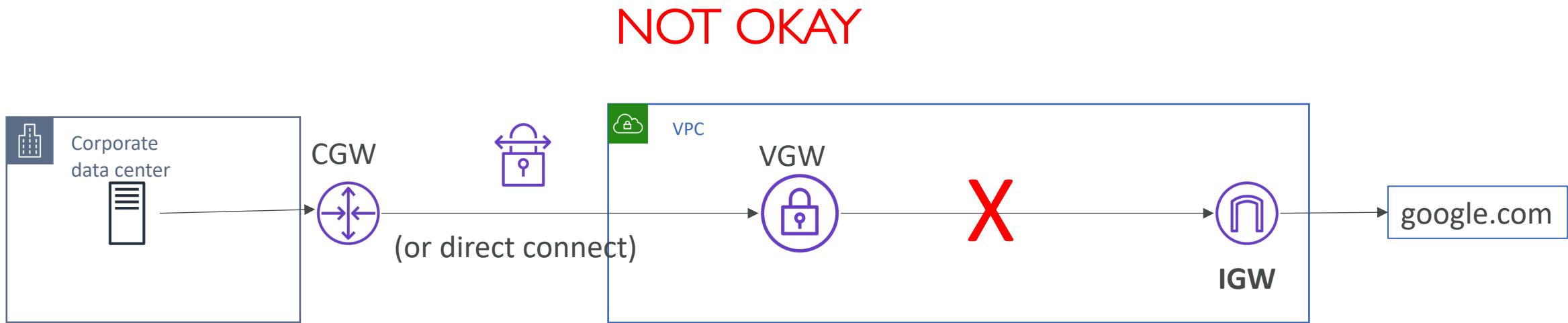
# Route Propagation in Site-to-Site VPN



- **Static Routing:**
  - Create static route in corporate data center for 10.0.0.1/24 through the CGW
  - Create static route in AWS for 10.3.0.0/20 through the VGW
- **Dynamic Routing (BGP):**
  - Uses BGP (Border Gateway Protocol) to share routes automatically (eBGP for internet)
  - We don't need to update the routing tables, it will be done for us dynamically
  - Just need to specify the ASN (Autonomous System Number) of the CGW and VGW

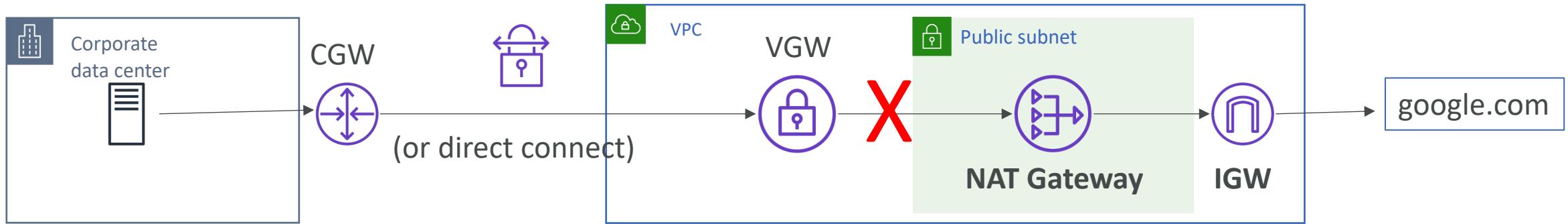
# VPN Transitive Routing

# Site to Site VPN and Internet Access



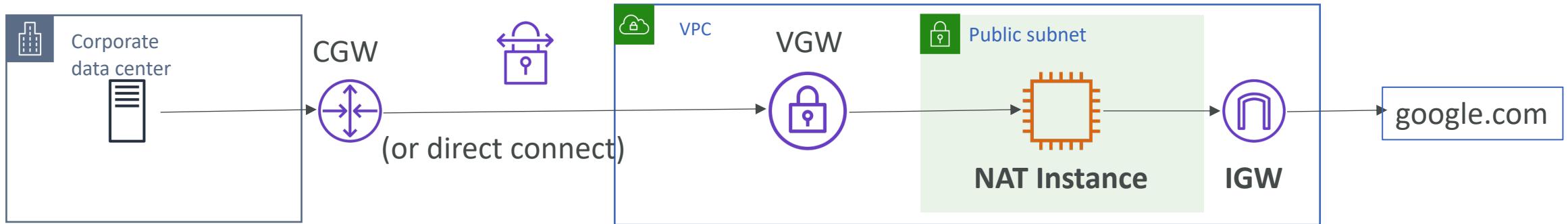
# Site to Site VPN and Internet Access

**NOT OKAY** (blocked by NAT Gateway restrictions)

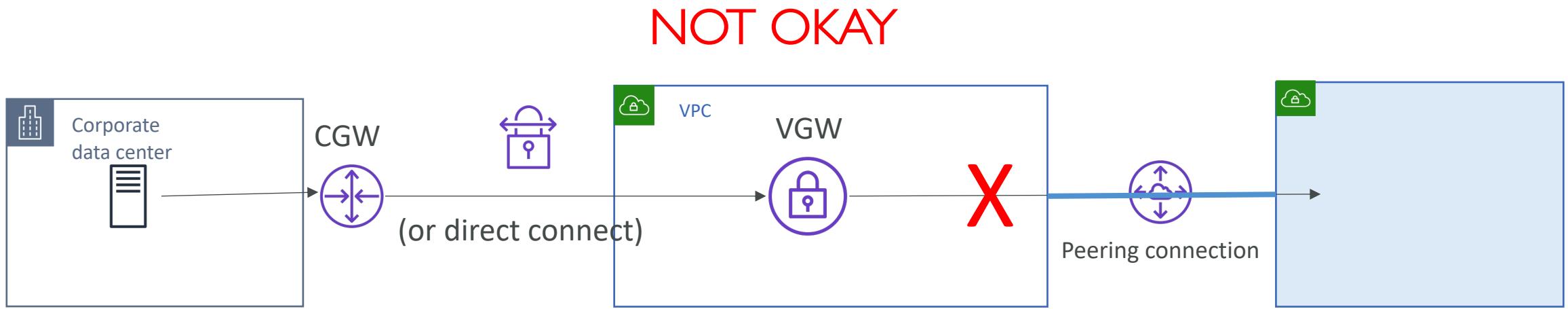


# Site to Site VPN and Internet Access

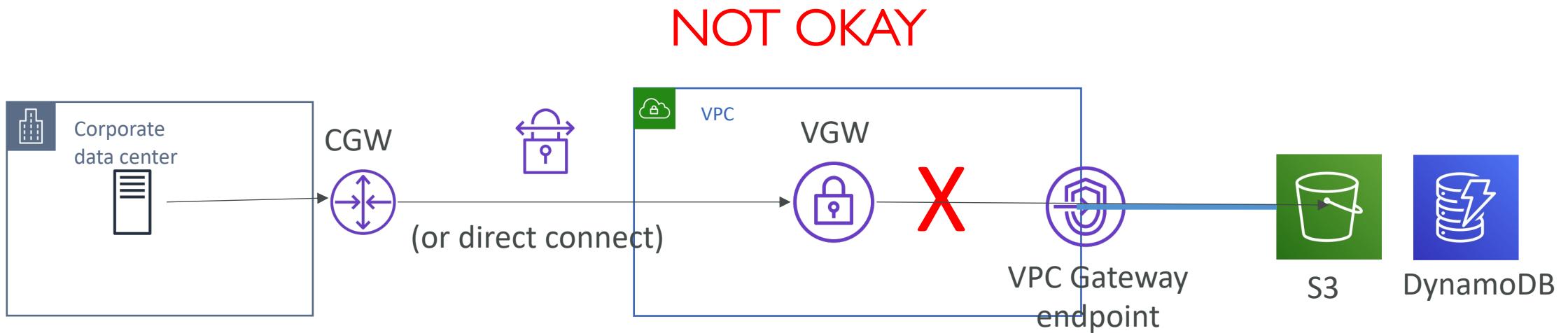
OKAY (self managed NAT Instance – more control)



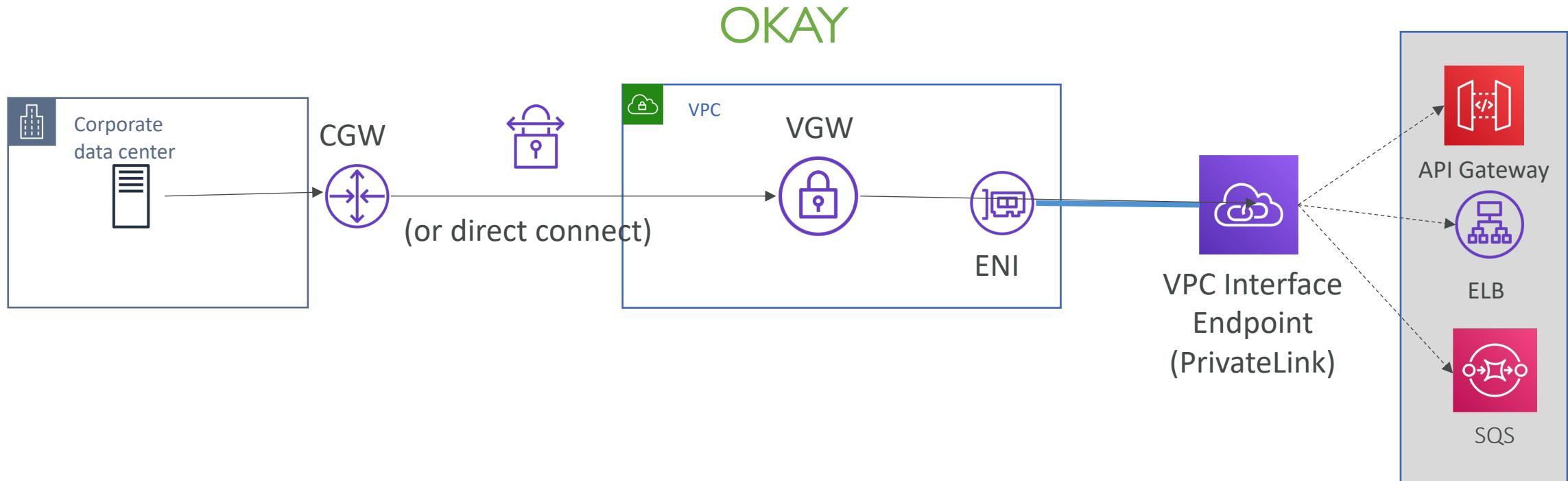
# Site to Site VPN and VPC Peering



# Site to Site VPN and VPC gateway endpoint

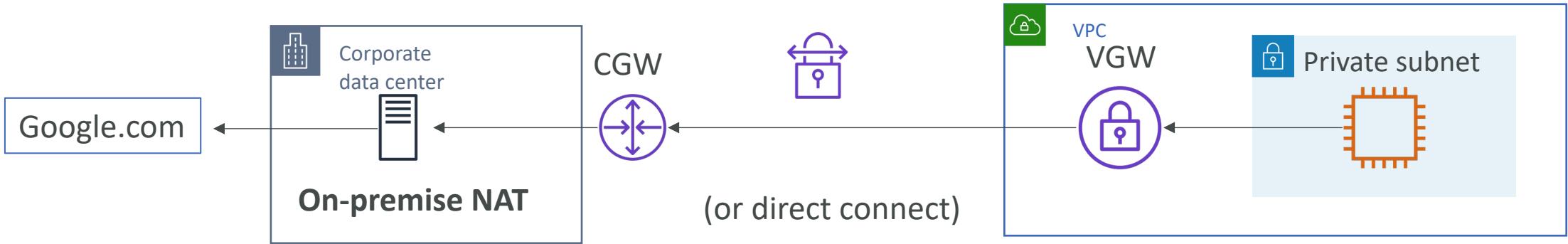


# Site to Site VPN and VPC Interface endpoint



# Site to Site VPN and on-premises Internet Access

OKAY (alternative to NAT Instances / Gateway)



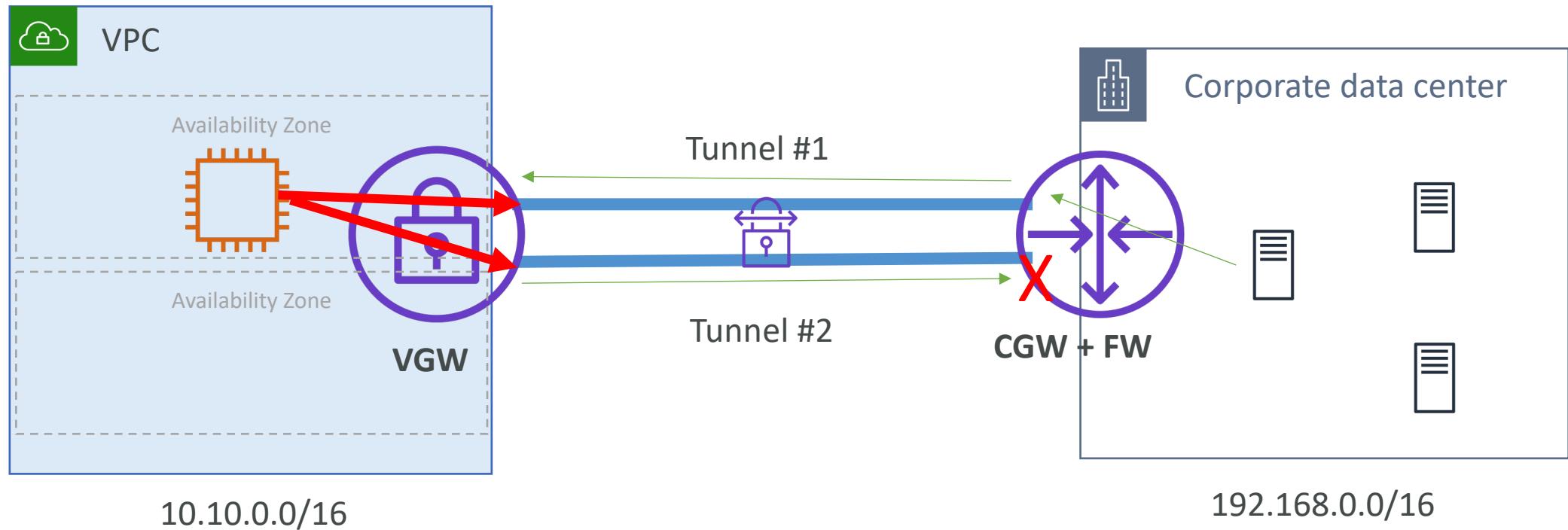
# Summary

- From on-premises to AWS via Virtual Private Gateway:
  - You **can not** access Internet through VPC attached Internet Gateway
  - You **can not** access Internet through NAT Gateway in Public subnet
  - You **can not** access peered VPC resources through VPC peering connection via the AWS VGW
  - You **can not** access S3, DynamoDB via the VPC gateway endpoint
  - You **can** access AWS services endpoint e.g API gateway, SQS and customer endpoint services (powered by Privatelink) via VPC interface endpoint
  - You **can** access Internet through NAT EC2 instance in Public subnet
- From AWS to on-premises via customer gateway
  - You can access Internet and other network endpoints based on routing rules setup on CGW in on-premises network

# VPN Tunnels and Routing Active/Active and Active/Passive

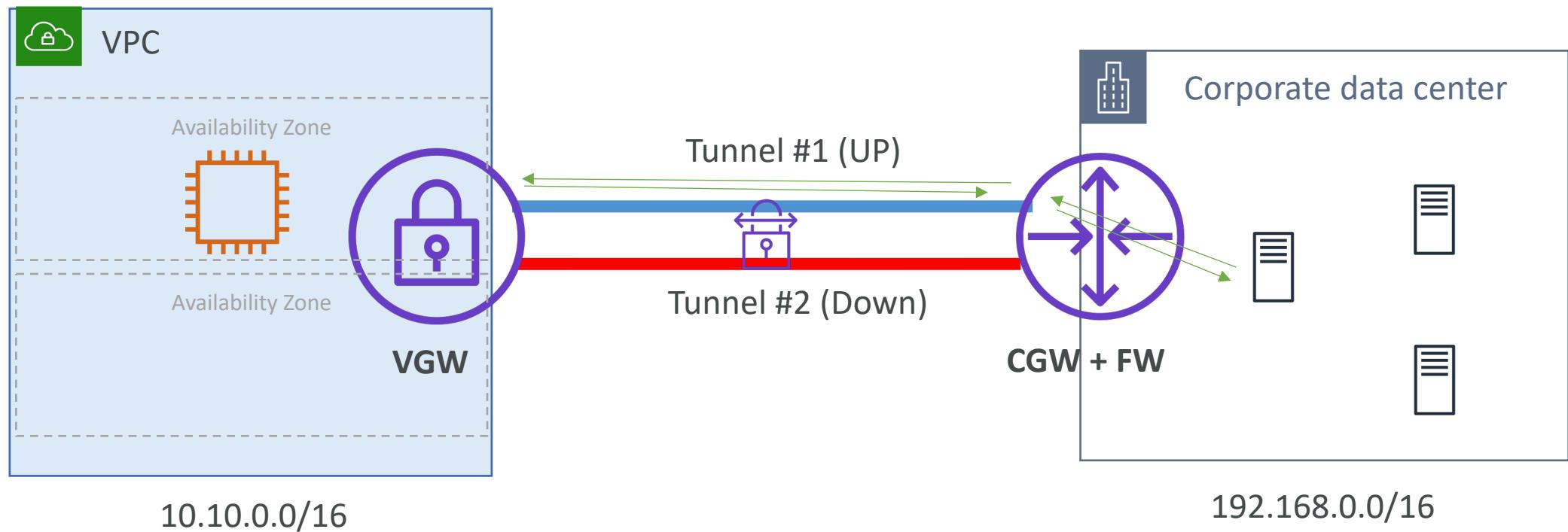
# Static Routing - Active/Active Tunnels

- Active/Active tunnel may cause **Asymmetric routing** and Asymmetric routing should be enabled on the CGW
- For traffic originating from AWS, one of the tunnel is selected randomly



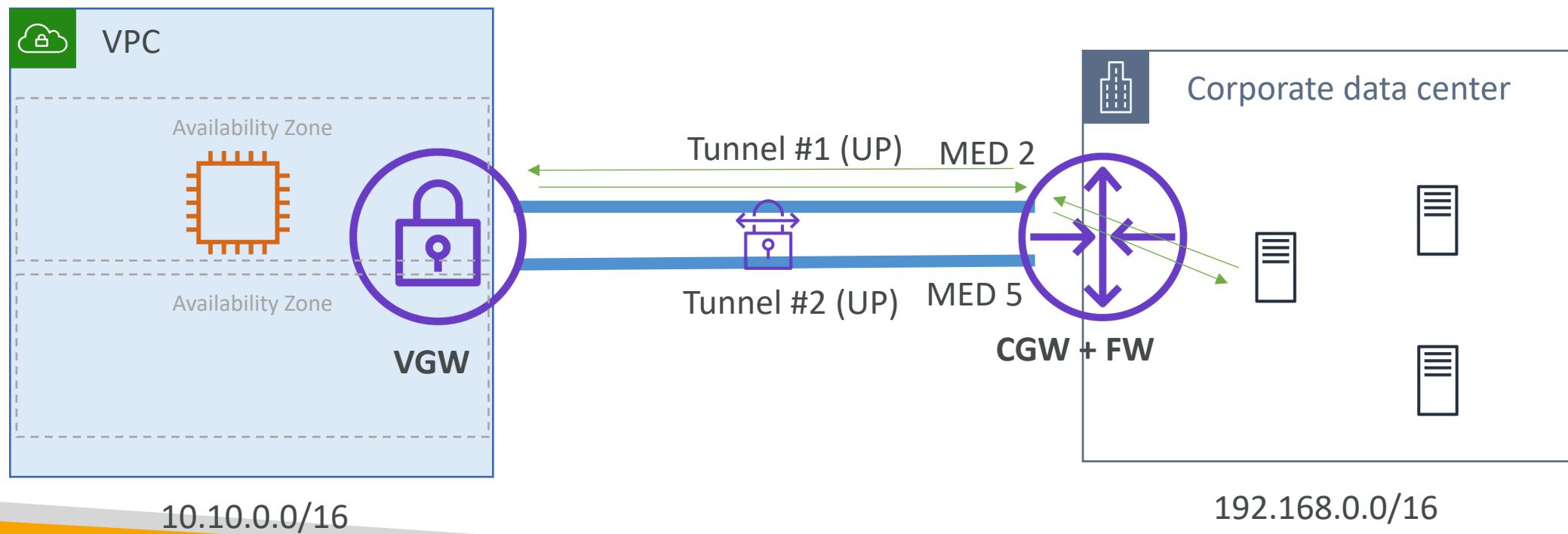
# Static Routing - Active/Passive Tunnels

- When only one tunnel is UP, its used for traffic in both the directions
- In case one tunnel goes down, another tunnel should be brought up from CGW end



# Dynamic Routing - Active/Active Tunnels

- Advertise a more specific prefix on the tunnel
- For matching prefixes where each VPN connection uses BGP, use AS PATH
- When the AS PATHs are the same length, use multi-exit discriminators (MEDs)
- The path with the lowest MED value is preferred.



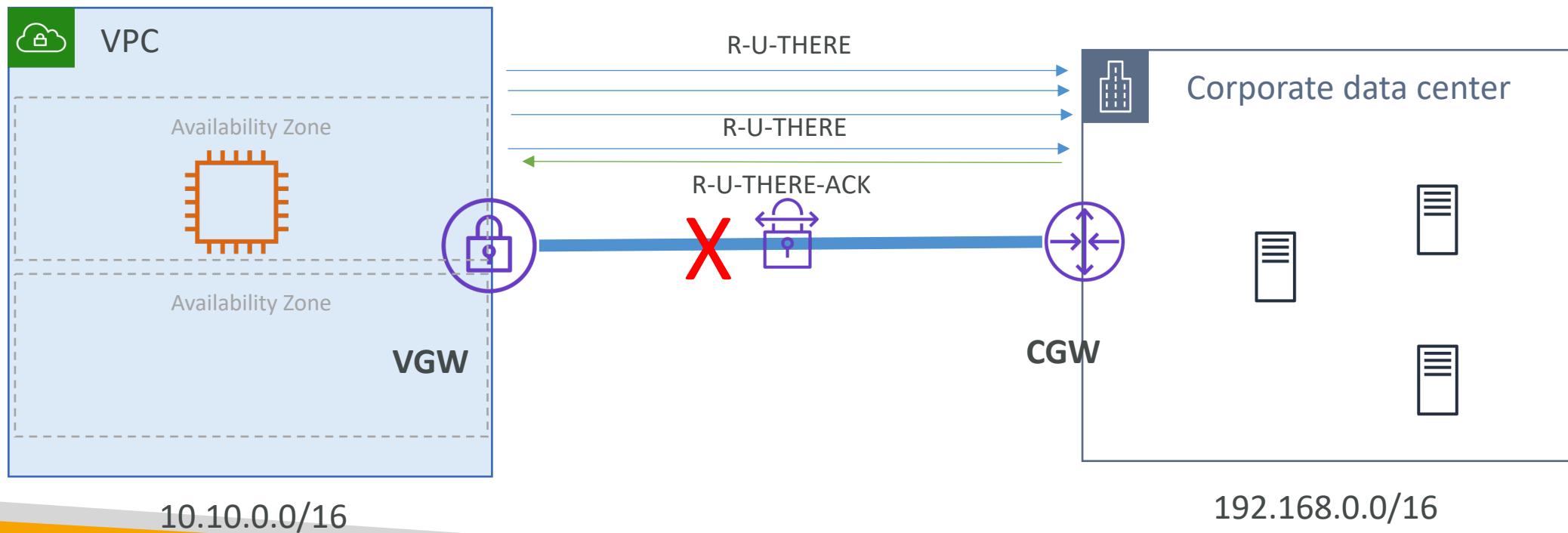
# Summary

- In case of static routing active-active tunnel mode
  - AWS randomly select the tunnel for traffic originating from AWS.
  - Make sure that Asymmetric routing is enabled on CGW
- In case of static routing active-passive tunnel mode
  - The tunnel which is UP is used for the traffic in both the directions
- In case of dynamic routing (BGP) active-active tunnel mode you can influence the tunnel preference by
  - Advertising more specific prefix over preferred tunnel
  - Advertising shorter AS PATH over the preferred tunnel
  - Setting lower MED values over the preferred tunnel

# VPN Dead Peer Detection (DPD)

# Dead peer detection

- Dead Peer Detection (DPD) is a method to detect liveliness of IPSec VPN connection
- VPN peers can decide during “IKE Phase I” if they want to use DPD
- If DPD is enabled AWS continuously (every 10 sec) sends DPD (R-U-THERE) message to customer gateway and waits for the R-U-THERE-ACK. If there is no response to consecutive 3 requests, then DPD times out.



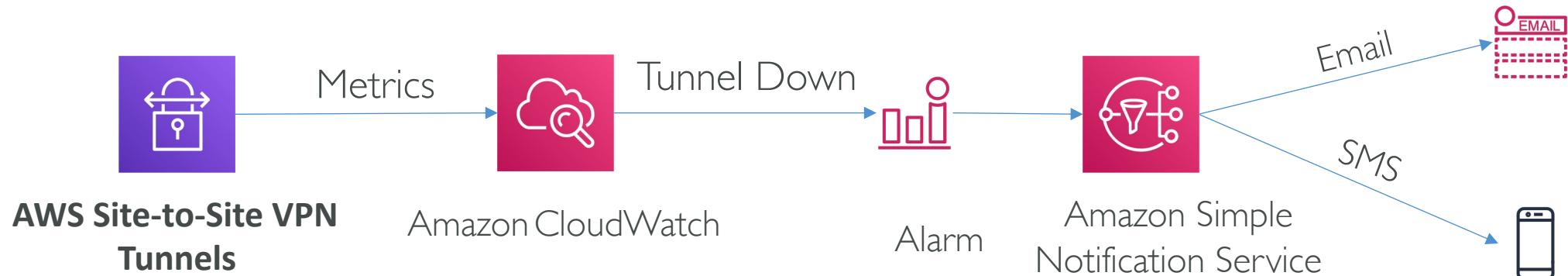
# Dead Peer Detection

- By default, when DPD occurs, the gateways delete the security associations. During this process, the alternate IPsec tunnel is used if possible.
- Default DPD timeout value is 30 sec which can be set higher than 30 seconds.
- DPD uses UDP 500 or UDP 4500 to send DPD messages
- DPD timeout actions:
  - Clear (default action) - End the IPSec IKE session, stop the tunnel and clear the routes
  - Restart - AWS Restarts the IKE Phase I
  - None – Take no action
- Customer router must support DPD when using Dynamic Routing (BGP)
- VPN tunnel can still be terminated due to inactivity or idle connection. You can set up appropriate idle timeout or setup a host which sends ICMP (ping) requests every say 5-10 seconds

# VPN Monitoring

# VPN Monitoring with CloudWatch

- **TunnelState:** The state of the tunnel. 0 indicates DOWN and 1 indicates UP. Value between 0 and 1 indicates that at least 1 tunnel is not UP.
- **TunnelDataIn:** The bytes received through the VPN tunnel
- **TunnelDataOut:** The bytes sent through the VPN tunnel



# VPN Monitoring with Health Dashboard

- AWS Site-to-Site VPN automatically sends notifications to the AWS Personal Health Dashboard (PHD)
- Tunnel endpoint replacement notification
- Single Tunnel VPN notification (when one of the tunnel is down for more than one hour in a day)

The screenshot shows the AWS Personal Health Dashboard interface. The top navigation bar includes the AWS logo, Services dropdown, Edit dropdown, and a bell icon with a red dot. The left sidebar has links for Personal Health Dashboard and Dashboard (which is selected). The main area is titled 'Dashboard' and displays three summary boxes: '5 Open issues Past 7 days', '4 Scheduled changes', and '3 Other n Past 7 days'. Below these is a section titled 'Changes to your AWS infrastructure that are scheduled to occur.' with a 'Filter by tags or attributes' button. A table lists four scheduled events:

|   | Event                           | Status   | Region/AZ      | Start time                  |
|---|---------------------------------|----------|----------------|-----------------------------|
| 1 | VPN emergency maintenanc...     | Upcoming | us-west-1      | December 1, 2016 at 10:3... |
| 2 | Elasticache redis maintenanc... | Upcoming | us-west-1      | December 3, 2016 at 10:0... |
| 3 | EC2 dedicated host power m...   | Upcoming | us-east-1      | December 5, 2016 at 8:00... |
| 4 | EC2 instance retirement sch...  | Upcoming | ap-northeast-1 | December 4, 2016 at 2:00... |

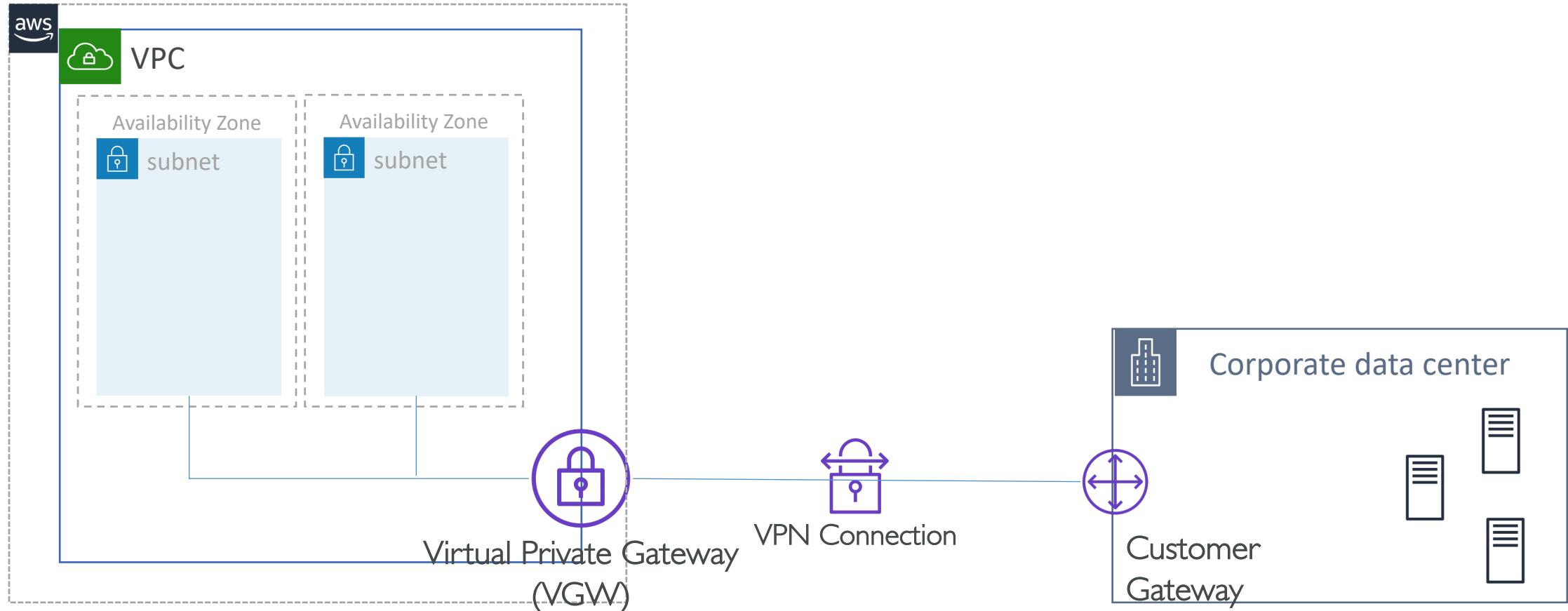
# AWS Site-to-Site VPN architectures

# AWS Site-to-Site VPN architectures

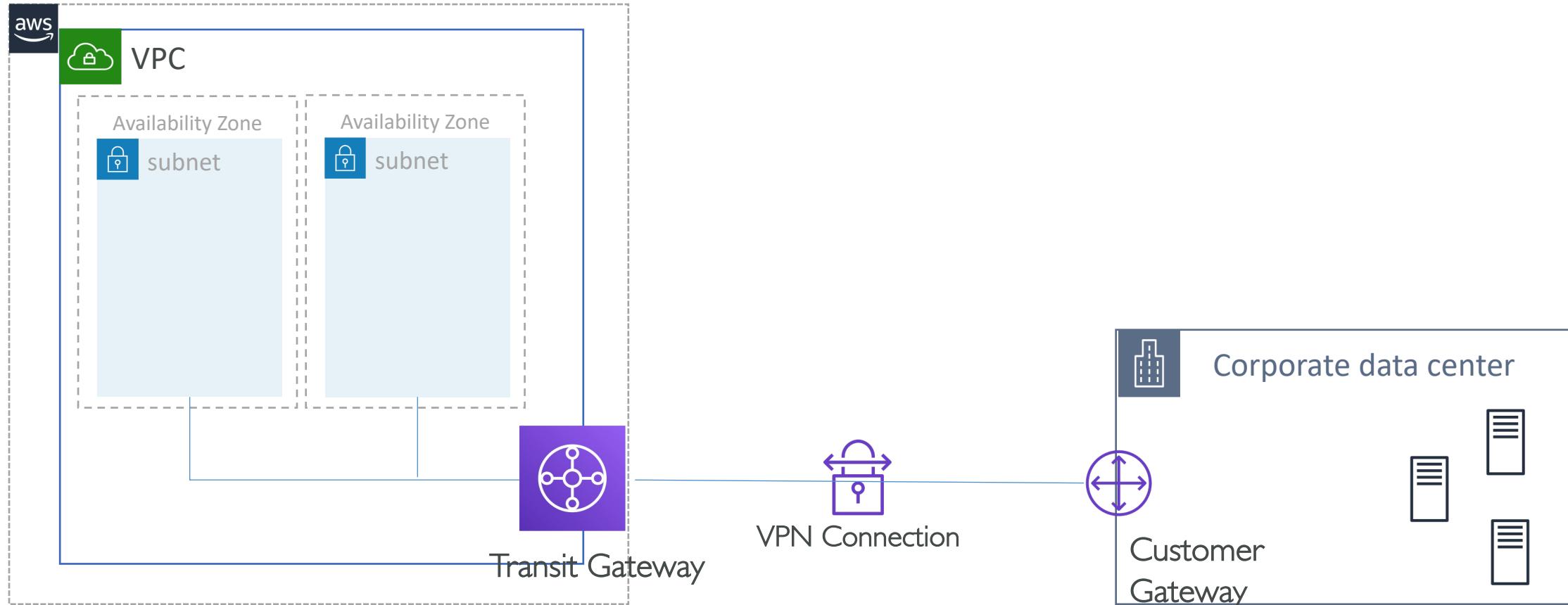
- Single Site-to-Site VPN connection
  - With Virtual Private Gateway (VGW)
  - With Transit Gateway (TGW)
- Multiple Site-to-Site VPN connections to branch offices
- Redundant VPN connections for High Availability

[Refer: https://docs.aws.amazon.com/vpn/latest/s2svpn/s2s-vpn-user-guide.pdf](https://docs.aws.amazon.com/vpn/latest/s2svpn/s2s-vpn-user-guide.pdf)

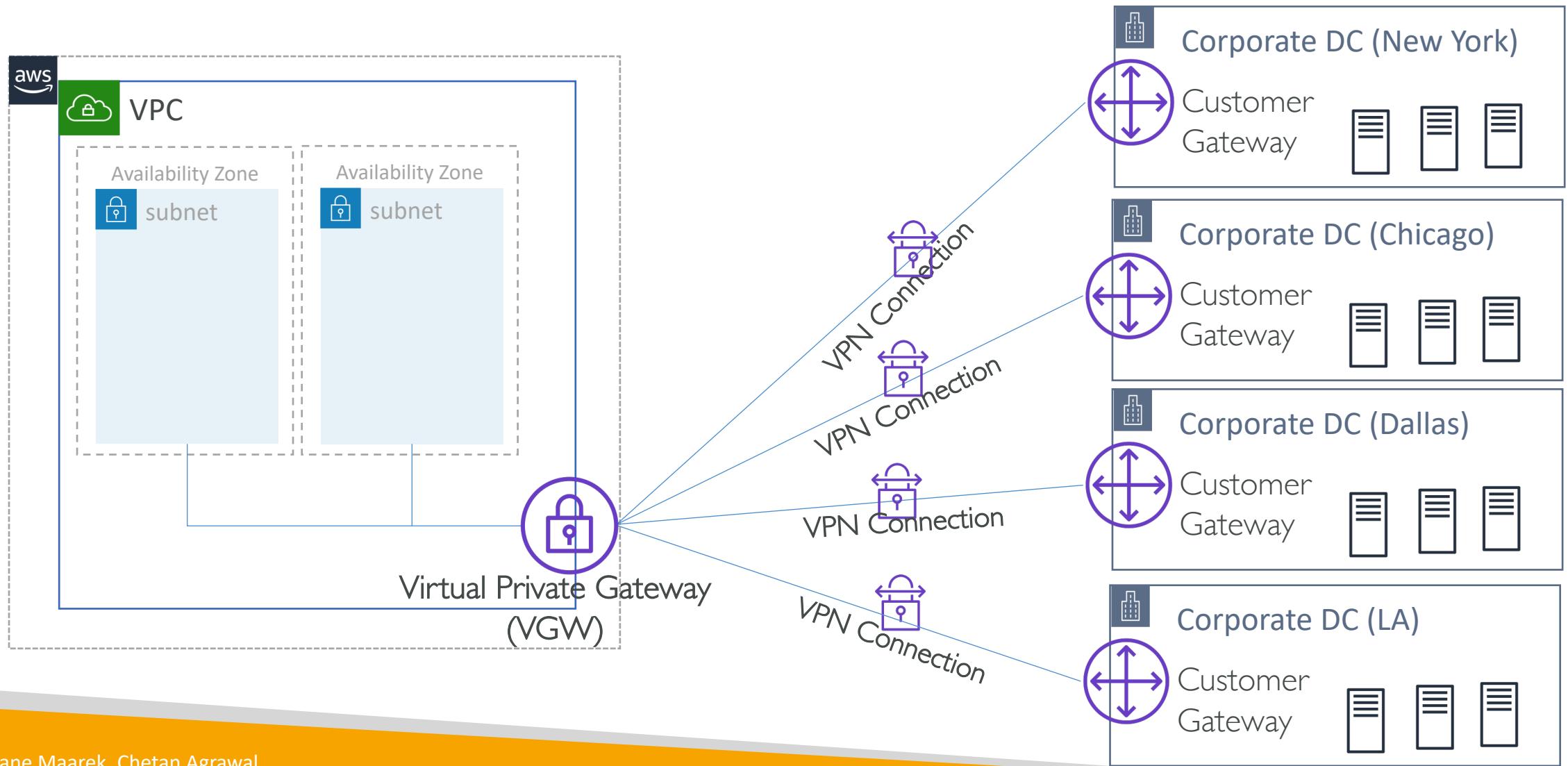
# Single Site-to-Site VPN Connection using VGW



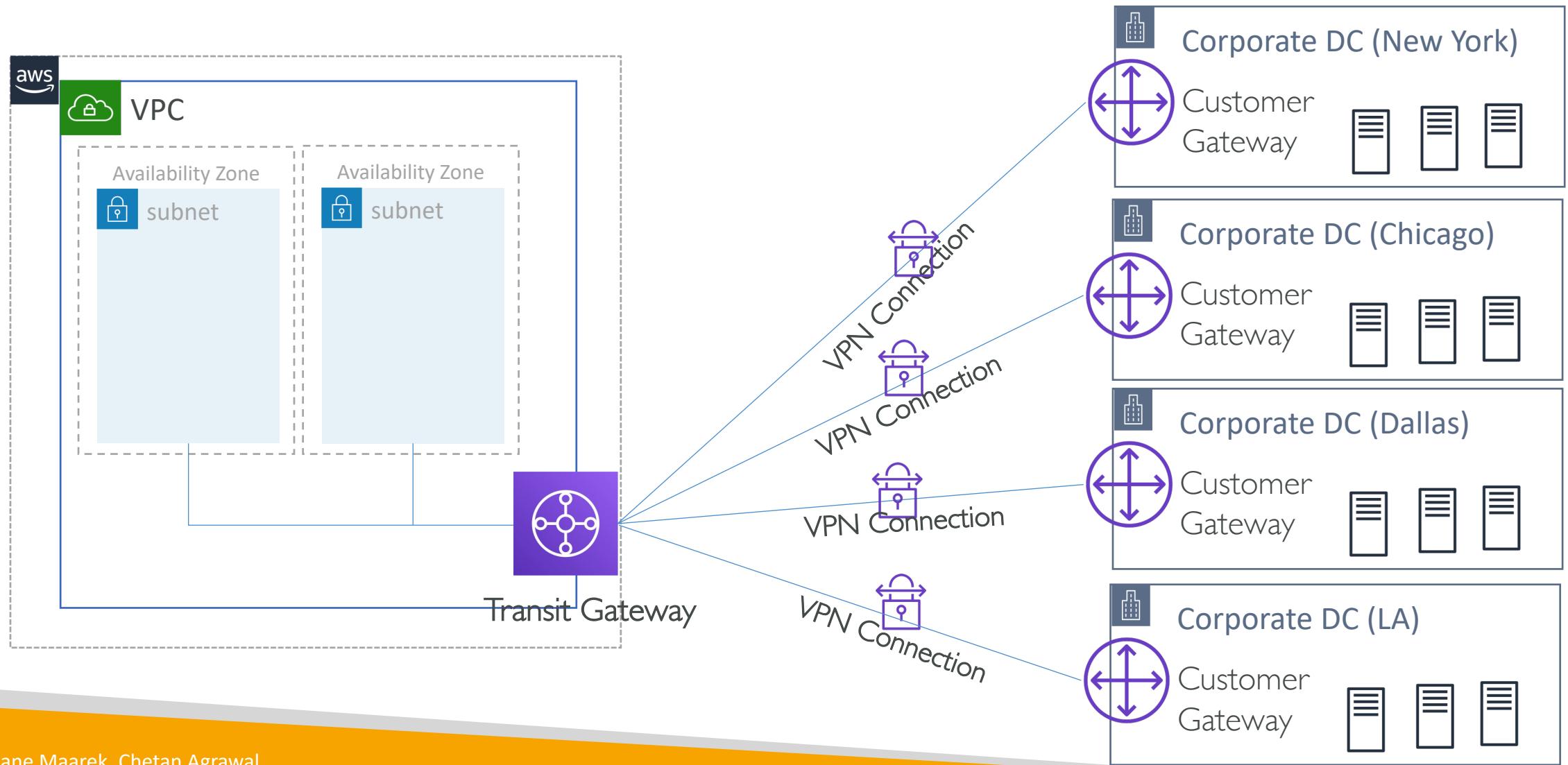
# Single Site-to-Site VPN Connection using Transit Gateway (TGW)



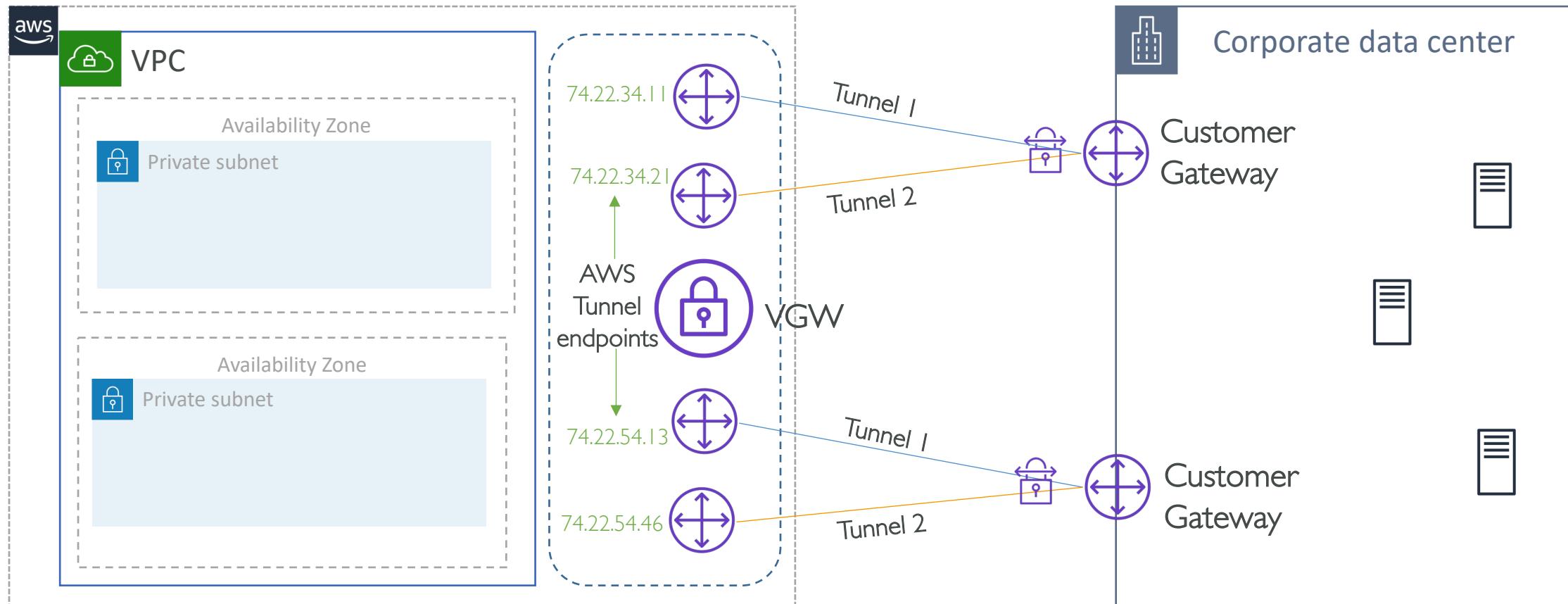
# Multiple Site-to-Site VPN Connections using VGW



# Multiple Site-to-Site VPN Connections using TGW

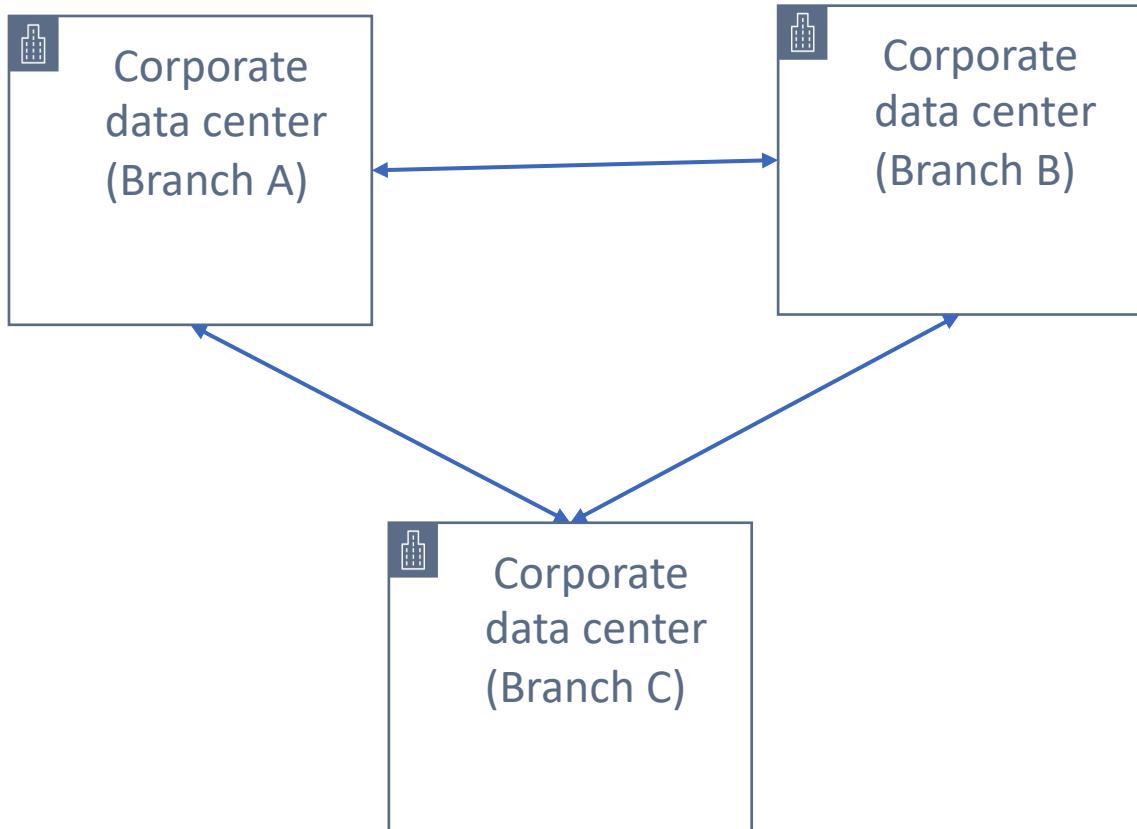


# Redundant VPN connections for HA



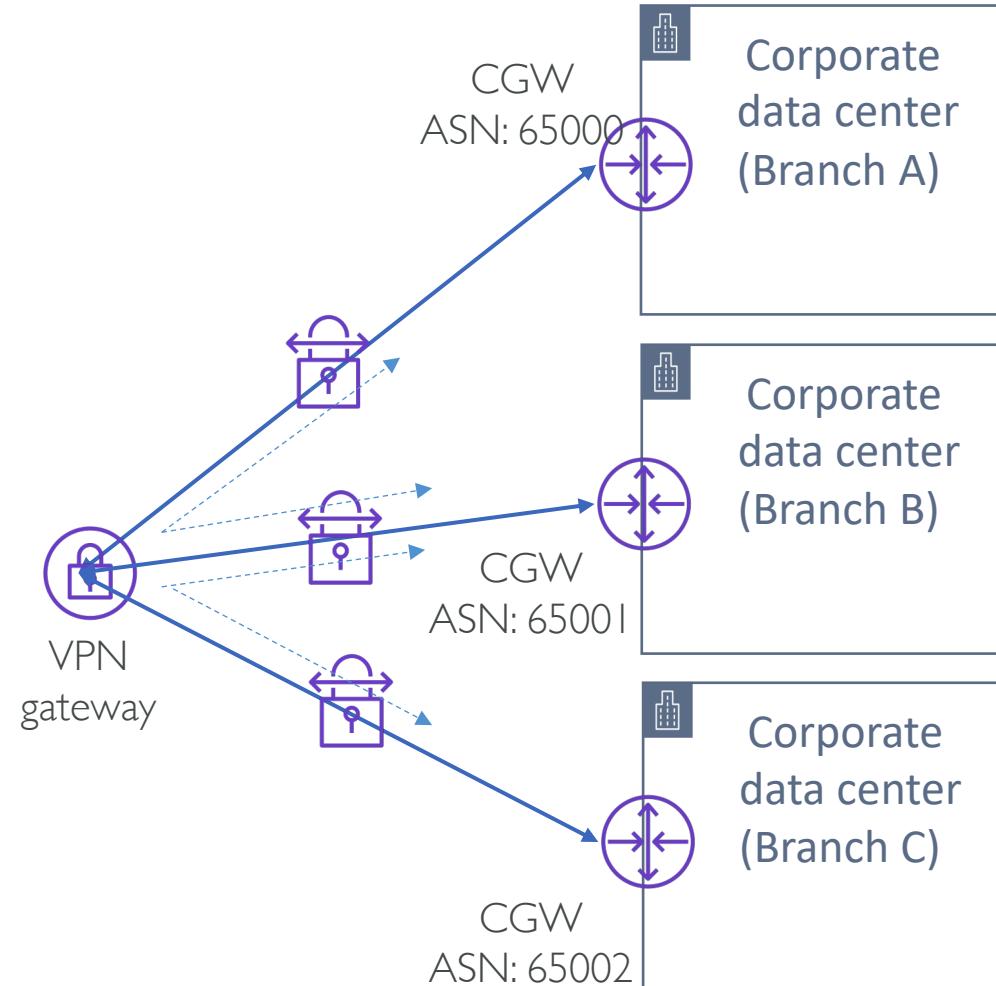
# AWS Site-to-Site VPN Cloud Hub

# VPN CloudHub – Routing between multiple customer sites



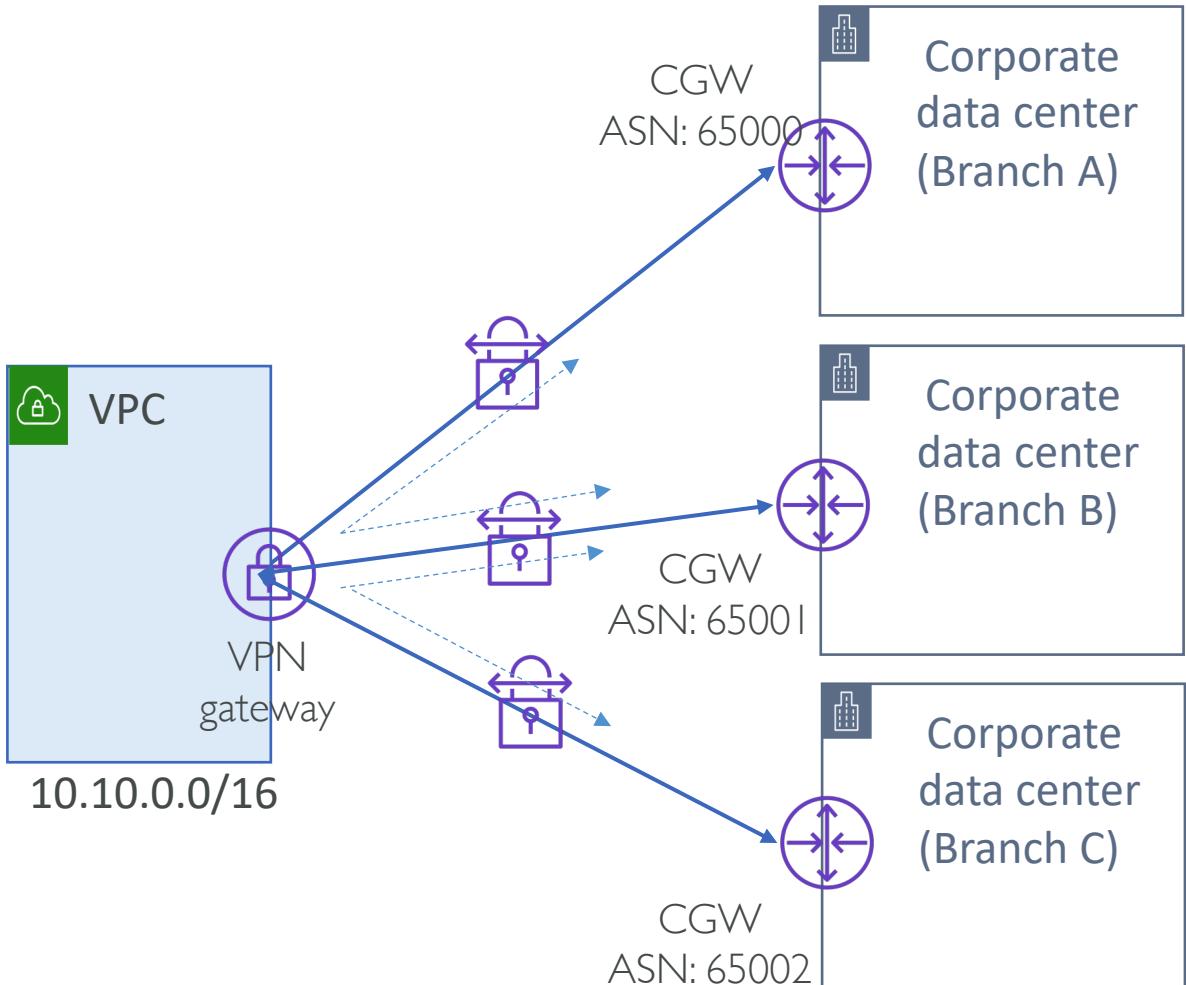
# VPN CloudHub – Routing between multiple customer sites

- Using VPN Gateway in detached mode
- Each customer gateway must have unique BGP ASN with dynamic routing
- Sites must not have overlapping IP ranges
- Connect up to 10 Customer Gateways
- Can serve as failover connection between on-premises locations



# VPN CloudHub – Routing between multiple customer sites

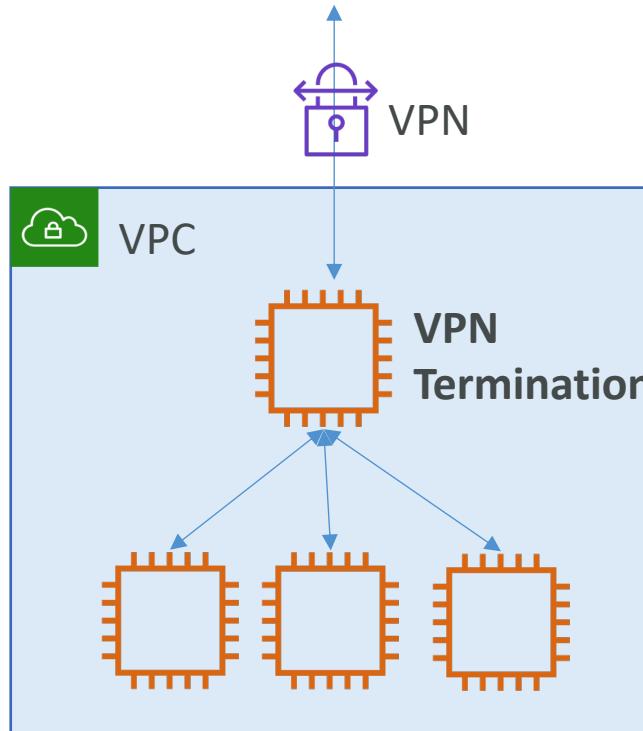
- VPN gateway can also be attached to VPC to enable communication



# Amazon EC2 based VPN

**EC2 based VPN**

# VPN termination on EC2



## Why would you do that?

- You want to use different VPN protocol than IPSec e.g General Routing Encapsulation (GRE) or Dynamic MultiPoint VPN (DMVPN)
- You are using overlapping CIDRs
- You want to enable transitive routing on AWS side
- You want to have special features on VPN termination endpoints such as Advanced Threat Protection
- When bandwidth required > 1.25 Gbps

# Considerations for EC2 based VPN

- To enable routing of traffic, make sure that the **source/destination check is disabled on the VPN termination EC2 instance** and IP forwarding is enabled at the operating system level.
- In case of hardware failure, EC2 will be recovered automatically if EC2 auto recovery has been setup using CloudWatch status check
- Bandwidth limitations between EC2 instances:
  - Up to 5 Gbps outside VPC
- AWS does not provide or maintain third party software VPN appliances
  - Partner VPN Solutions on AWS Marketpalce: Cisco, Juniper, Palo Alto You may get a highly available & self-healing setup with their solution

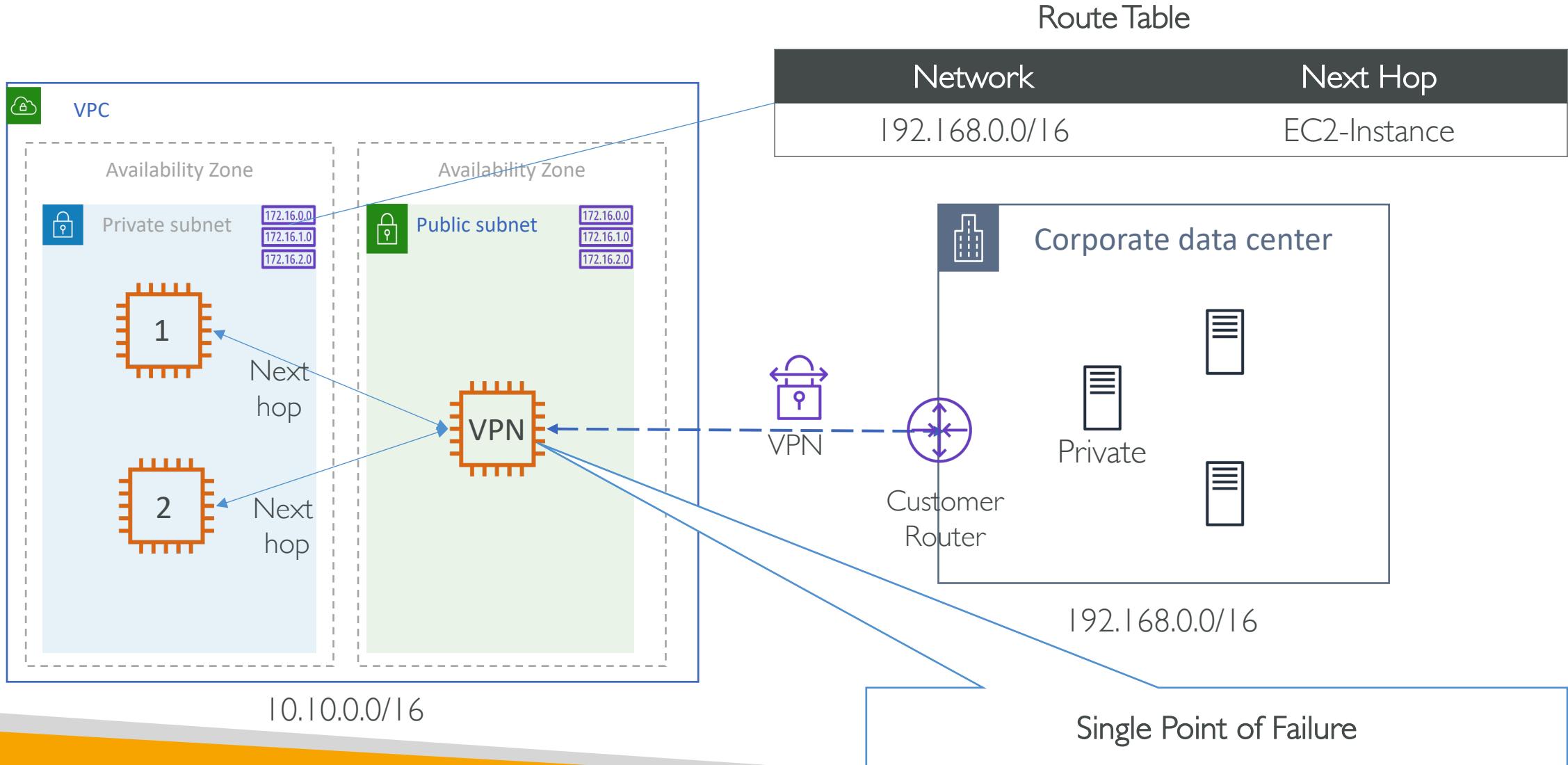
# Considerations for EC2 based VPN

- Vertical scaling:
  - Increase instance size for increased performance
- Horizontal scaling
  - You can have the architecture which provides horizontal scaling (covered in next topic)
- IPsec as a protocol doesn't function through a Network Load Balancer due to non-Transmission Control Protocol (TCP) (IPSec protocol 50)

# Amazon EC2 based VPN – High Availability

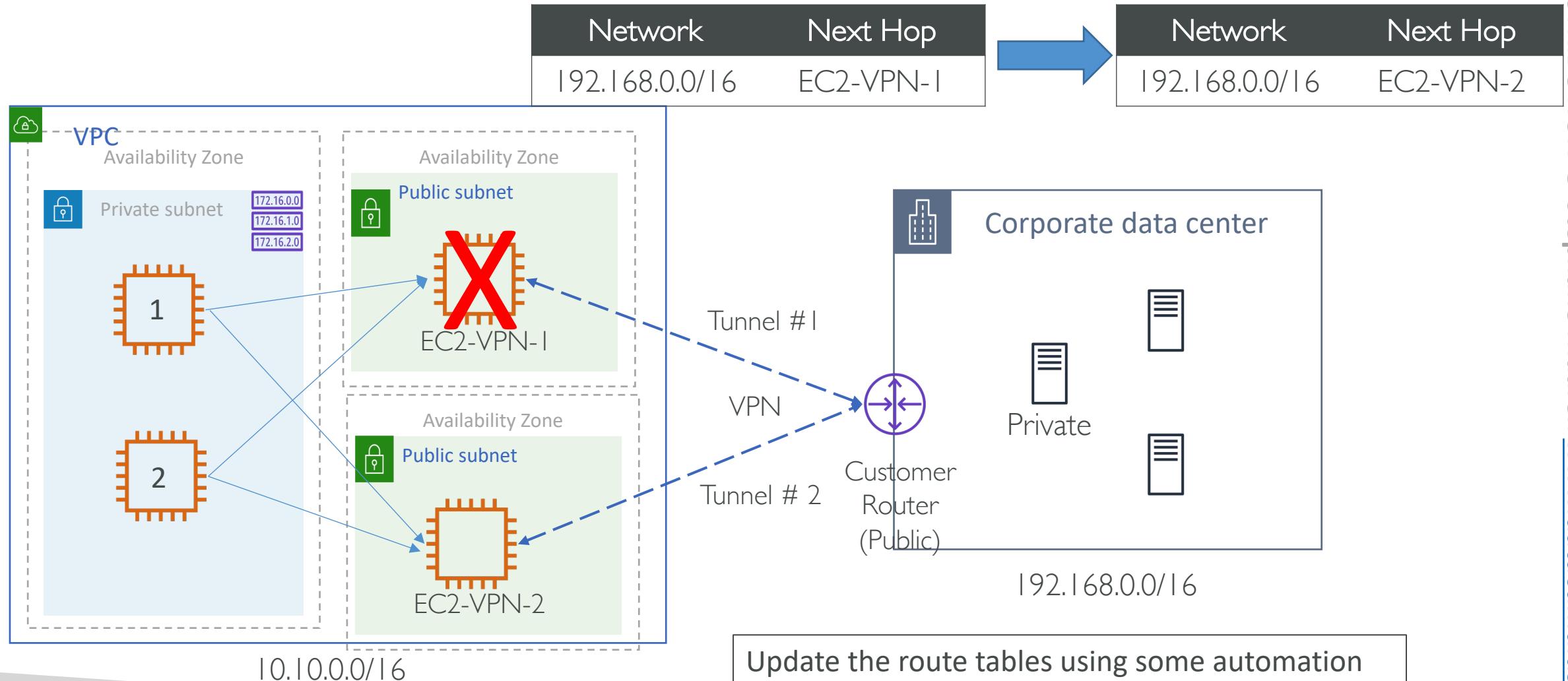
**EC2 based VPN**

# VPN termination on EC2



**EC2 based VPN**

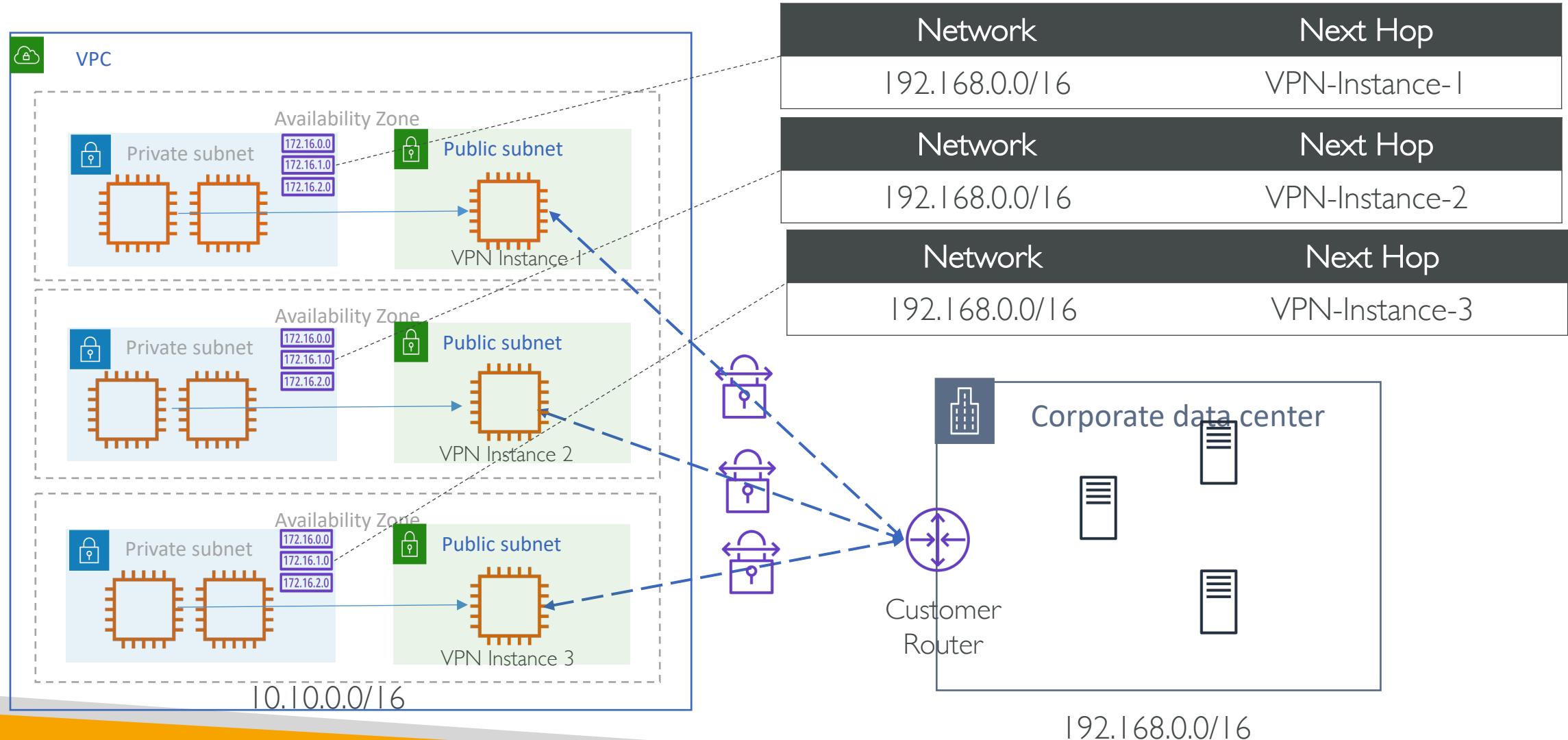
# HA solution with EC2 based VPN



# Amazon EC2 based VPN – Horizontal Scaling

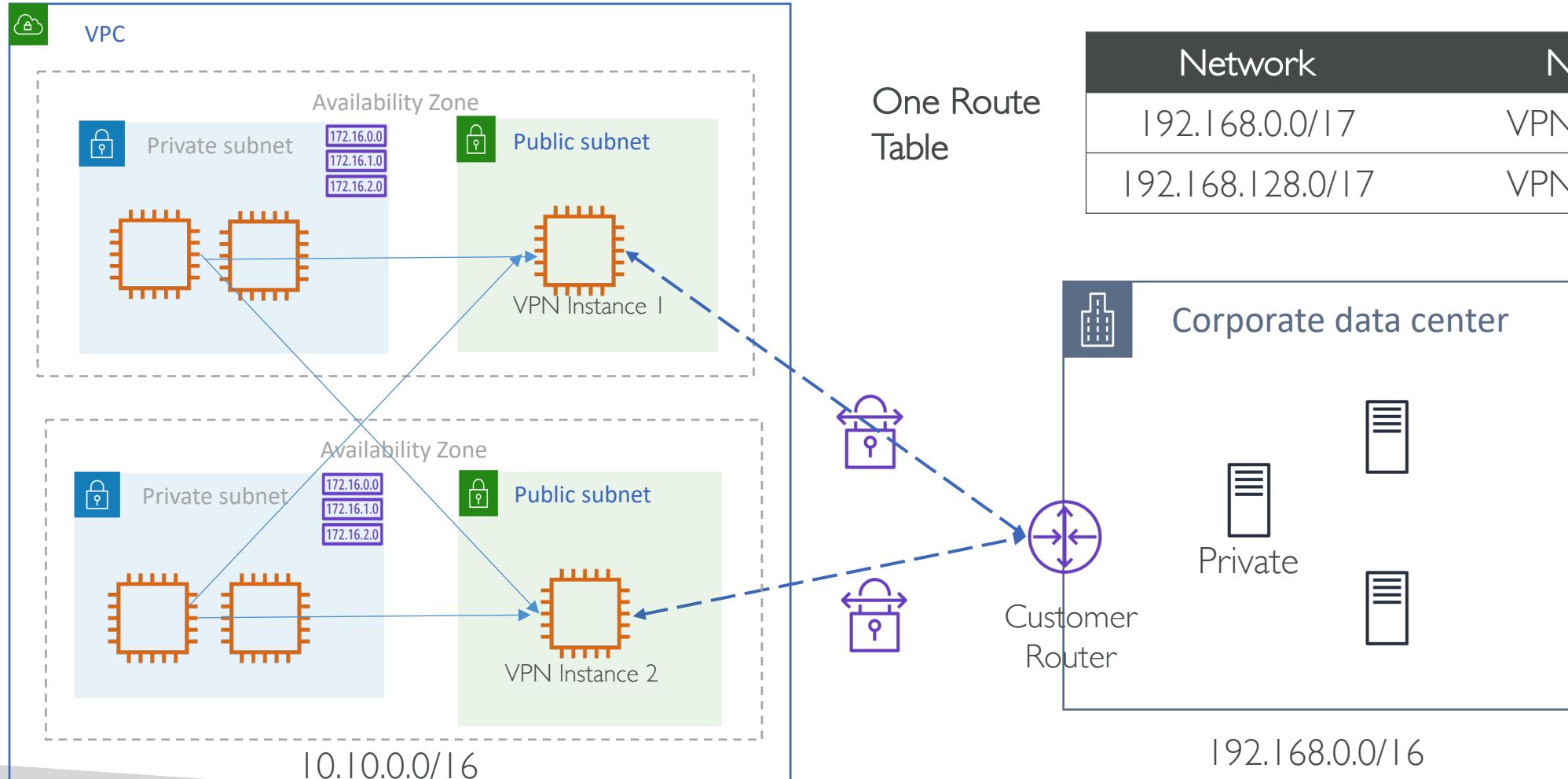
EC2 based VPN

# Horizontal scaling – VPN EC2 per subnet



EC2 based VPN

# Horizontal scaling – Split traffic



# VPN scenarios

# VPN design pattern scenarios

- I. On-premises network needs to connect to AWS VPC with moderate throughput requirement (up to 1 Gbps)

Solution:

- Use AWS managed VPN (VGW)
- VGW supports bandwidth of 1.25 Gbps.
- Multiple VPN connections to the same Virtual Private Gateway are bound by an aggregate throughput limit of 1.25 Gbps.
- For AWS Direct Connect connection on a Virtual Private Gateway, the throughput is bound by the Direct Connect physical port itself.

# VPN design pattern scenarios

2. on-premises network needs to connect to AWS VPC with very high throughput requirement (> 2 Gbps)

Solutions:

1. Direct connect should be the first choice.
2. Terminate VPN on EC2 instance. You should also use multiple EC2 instances for high availability and aggregated throughput.

# VPN design pattern scenarios

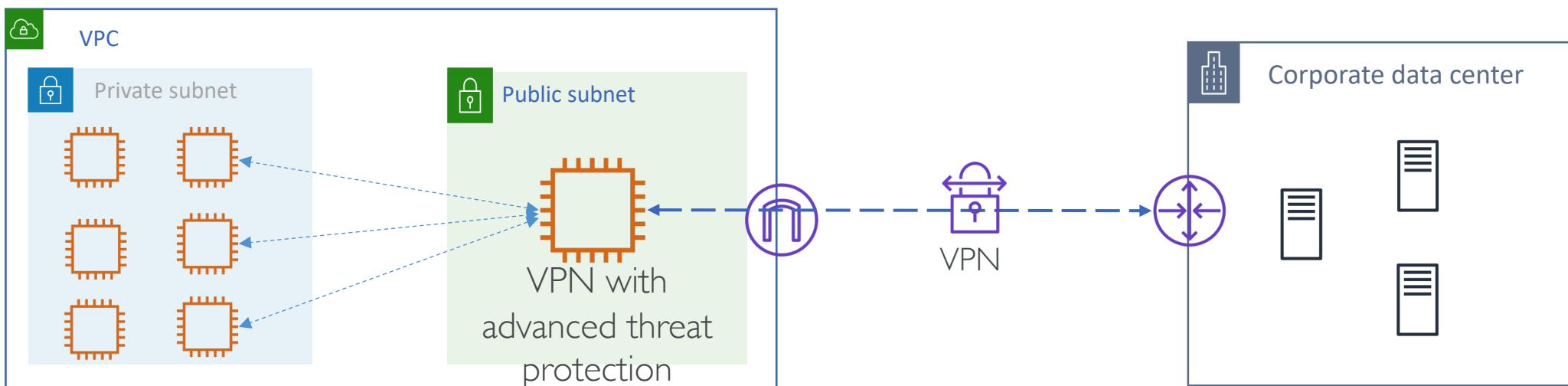
3. You want to configure VPN using non-IPSec protocol

Solution: Terminate VPN on EC2 instance as it would support all possible protocols like DMVPN and GRE.

# VPN design pattern scenarios

4. You want to have advanced threat protection on your VPN termination endpoint

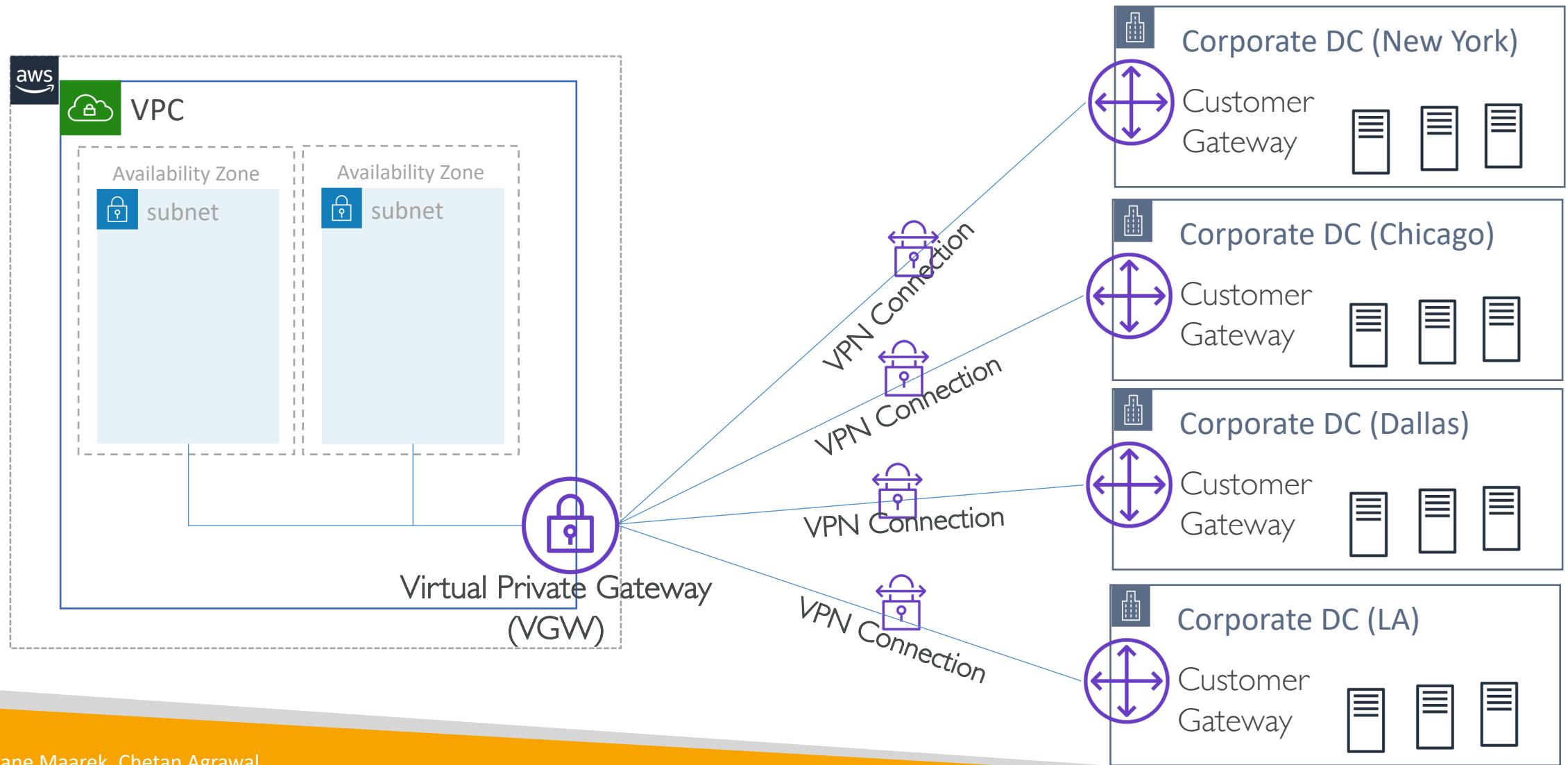
Solution: Terminate VPN on EC2 instance and install required threat protection software on EC2. You may also look at AWS Marketplace EC2 based solution.



# Transit VPC

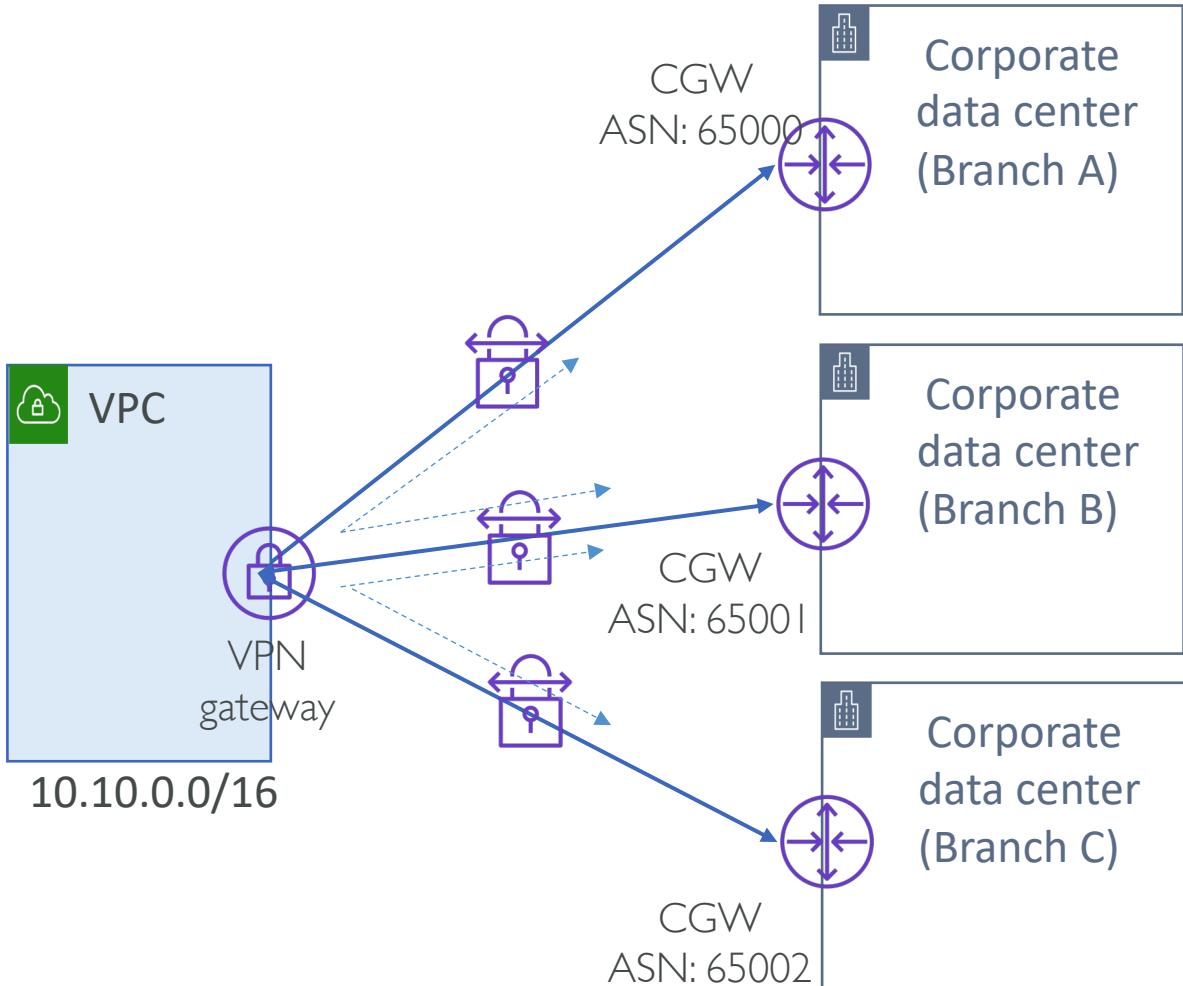
VPN architectures that we already looked at ..

# Multiple Site-to-Site VPN Connections using VGW

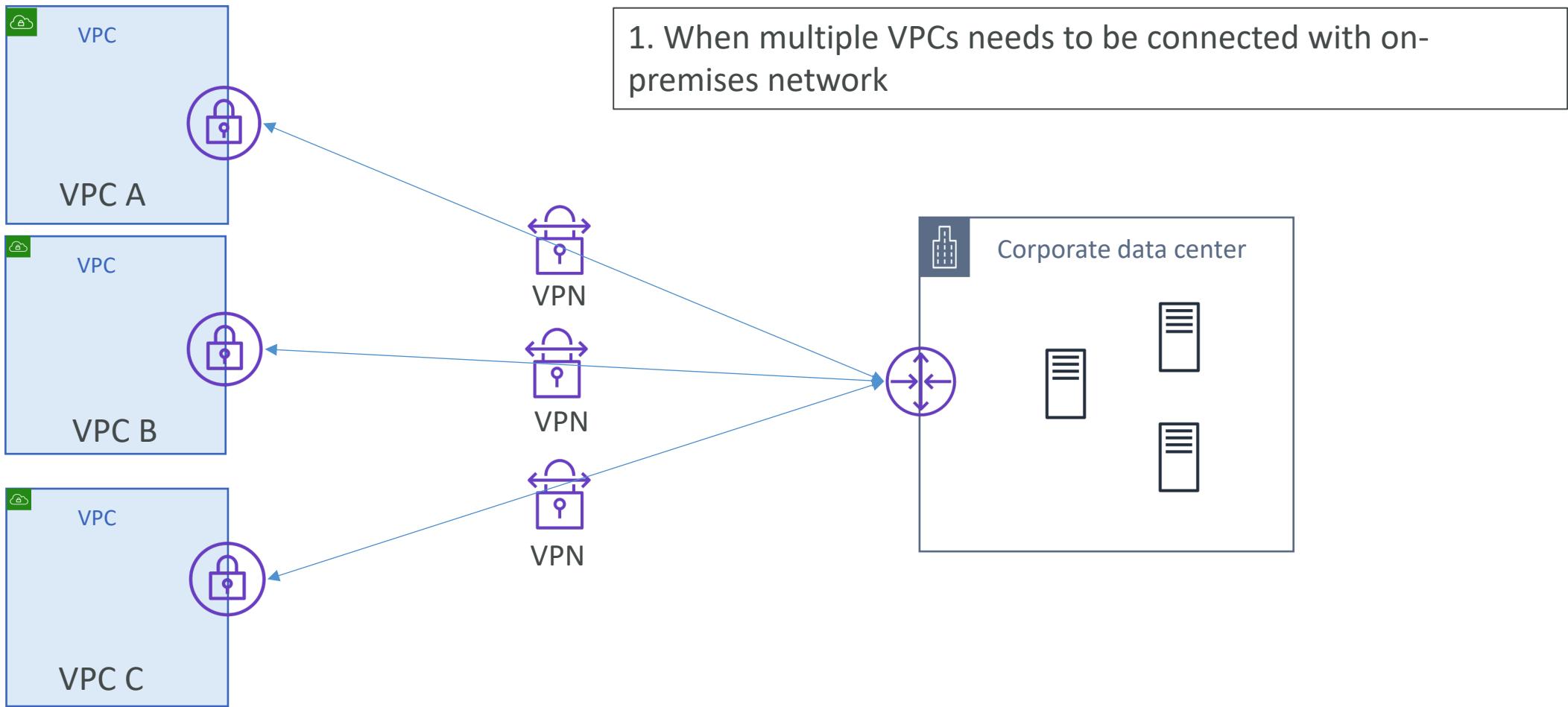


# VPN CloudHub – Routing between multiple customer sites

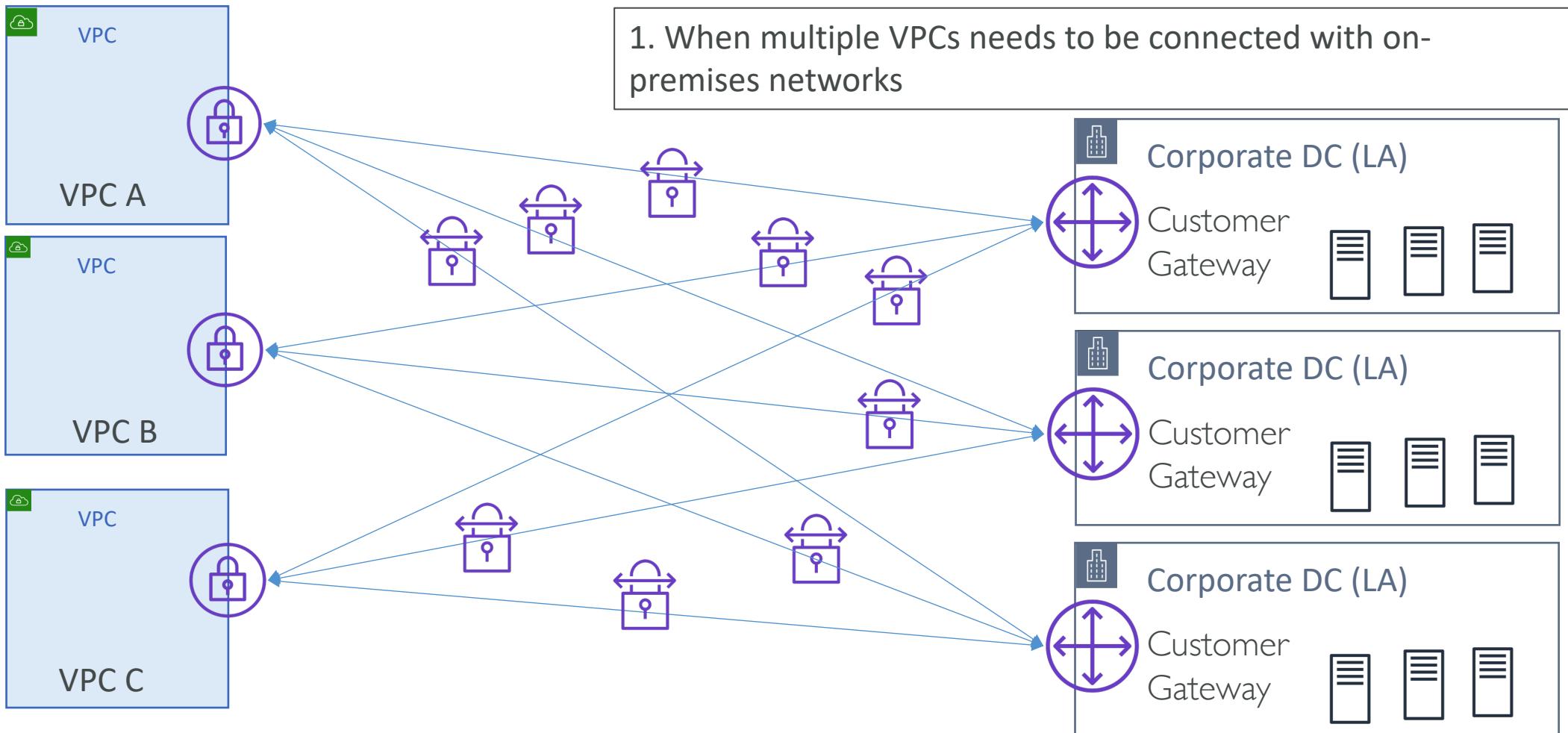
- VPN gateway can also be attached to VPC to enable communication



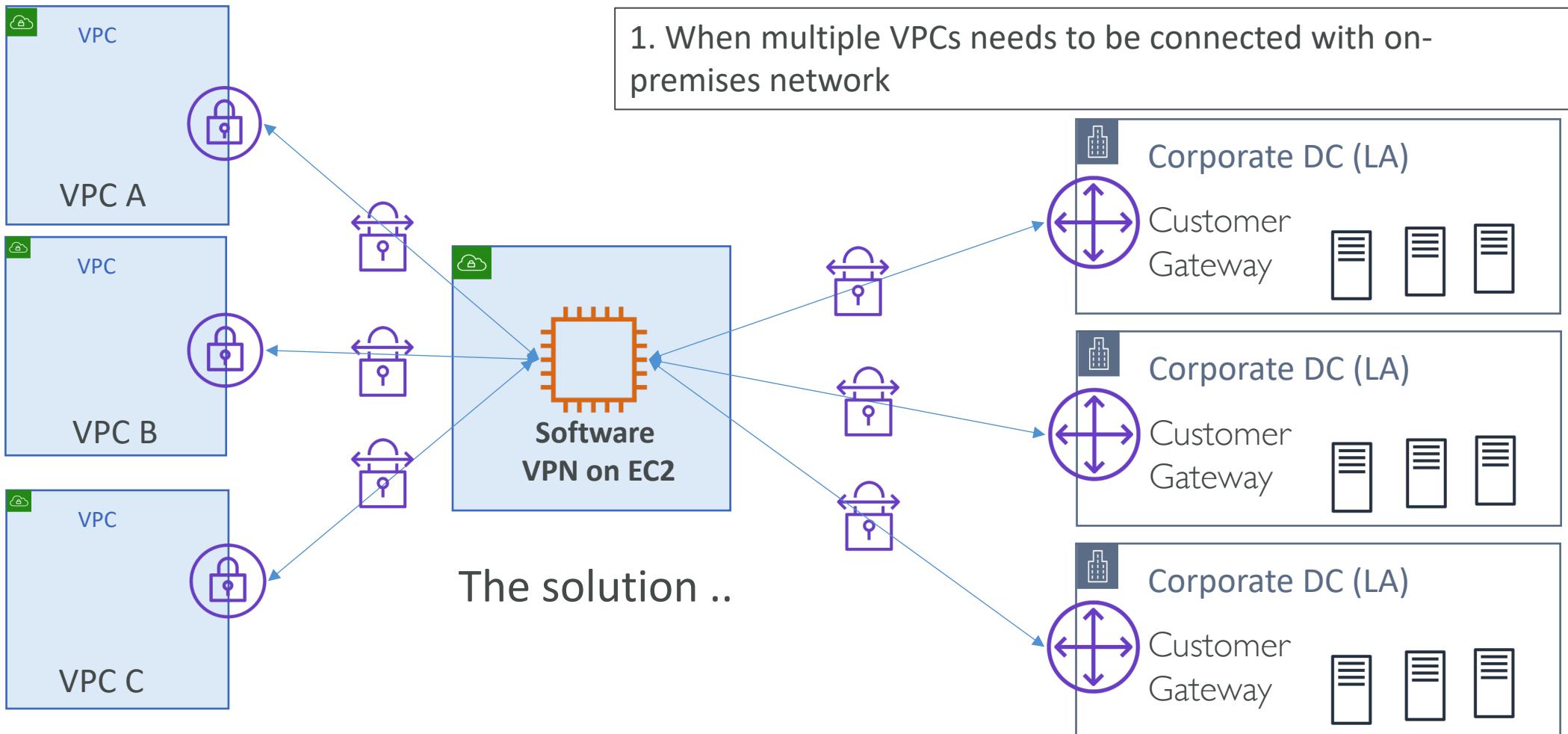
# Transit VPC – What is it and why it is required?



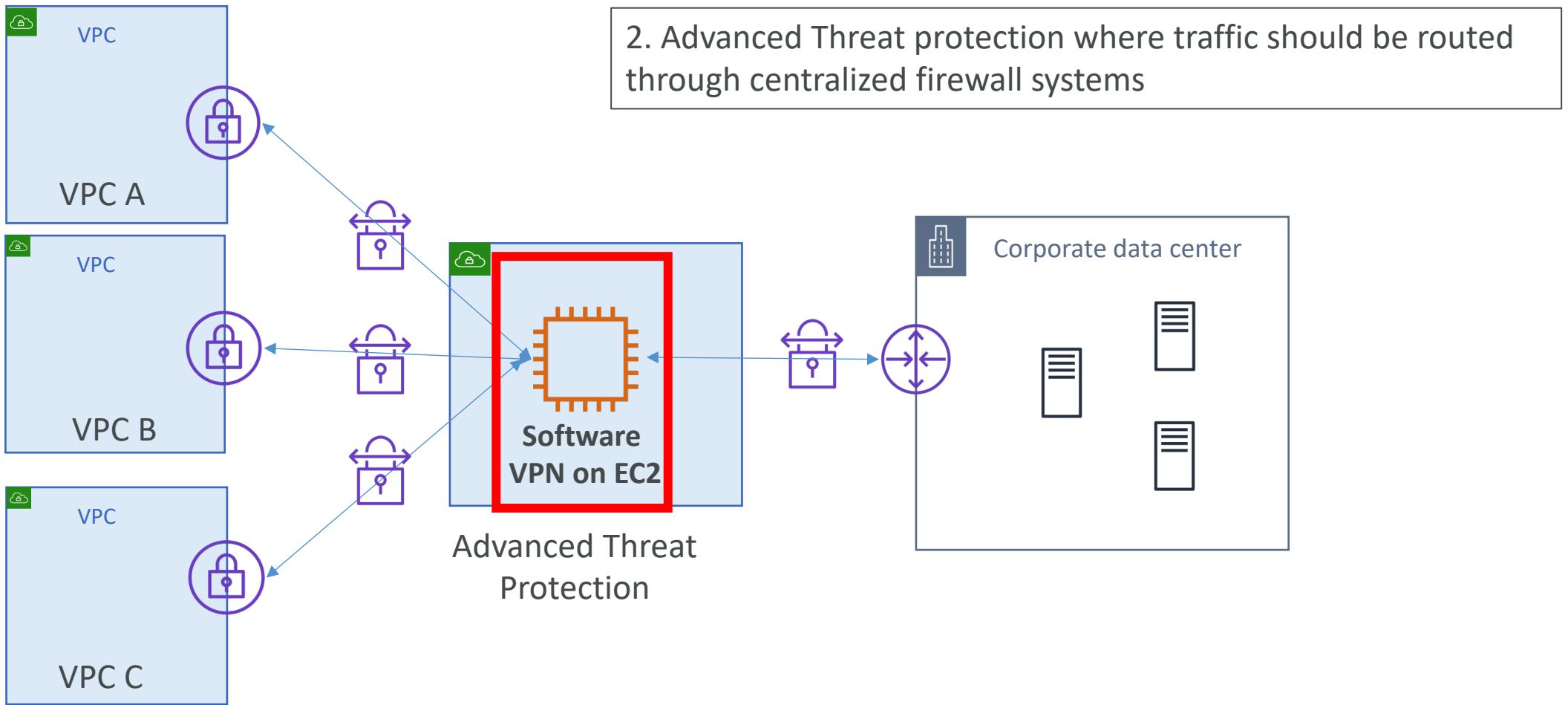
# Transit VPC – What is it and why it is required?



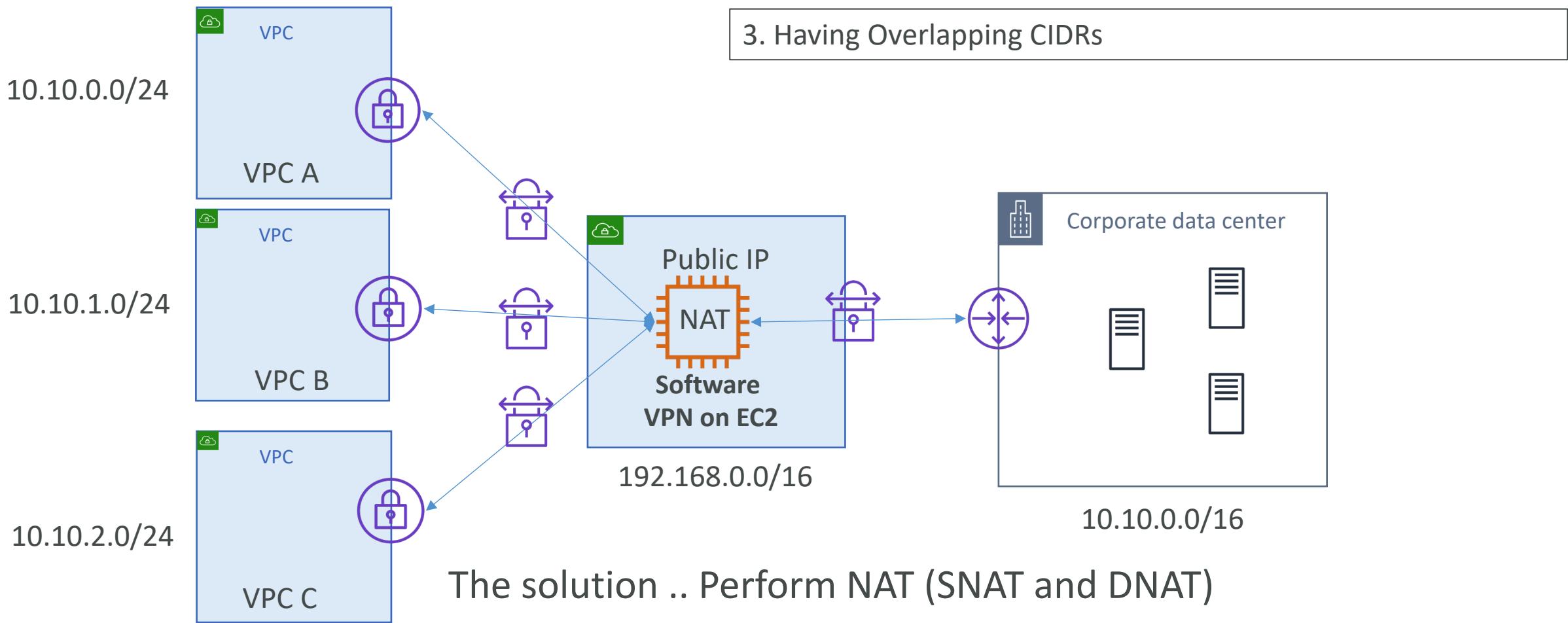
# Transit VPC – What is it and why it is required?



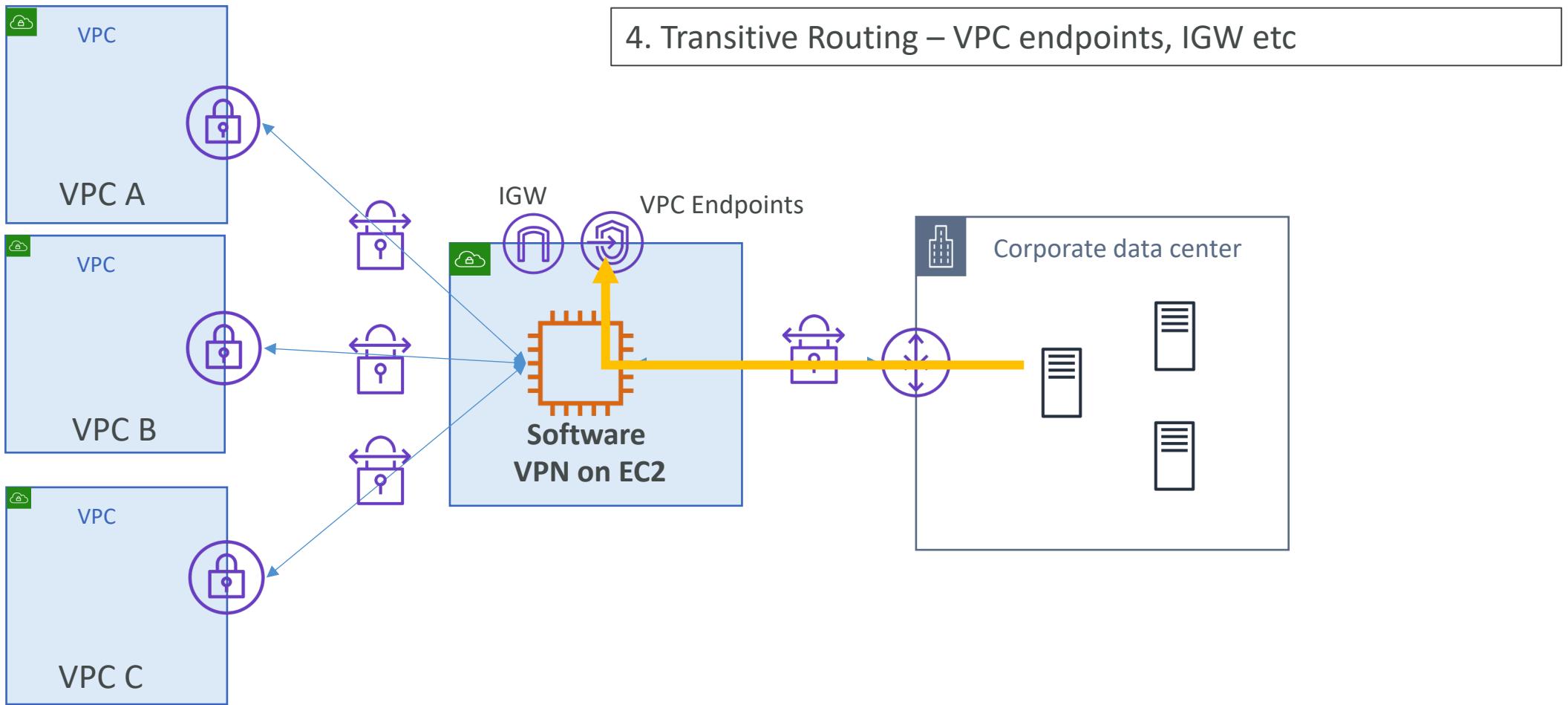
# Transit VPC – What is it and why it is required?



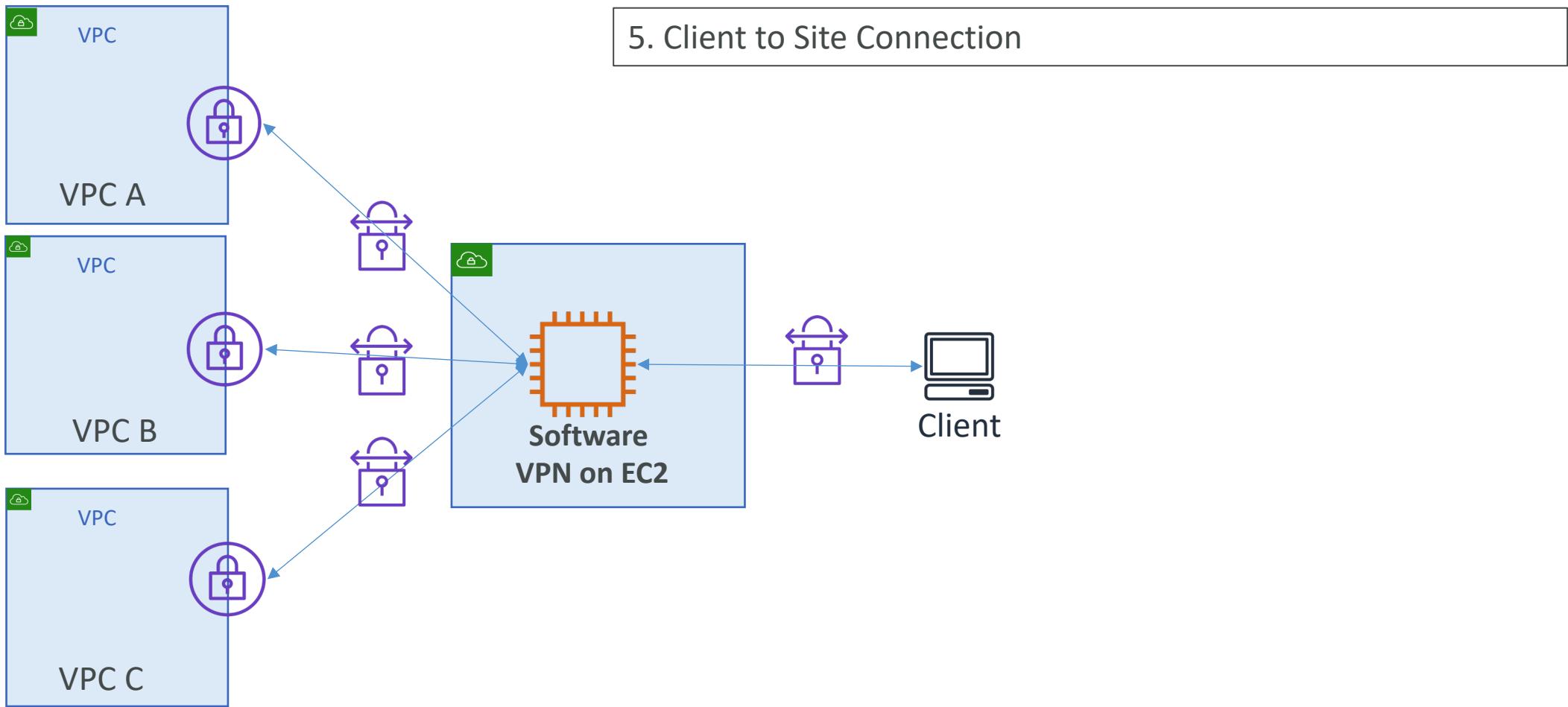
# Transit VPC – What is it and why it is required?



# Transit VPC – What is it and why it is required?



# Transit VPC – What is it and why it is required?

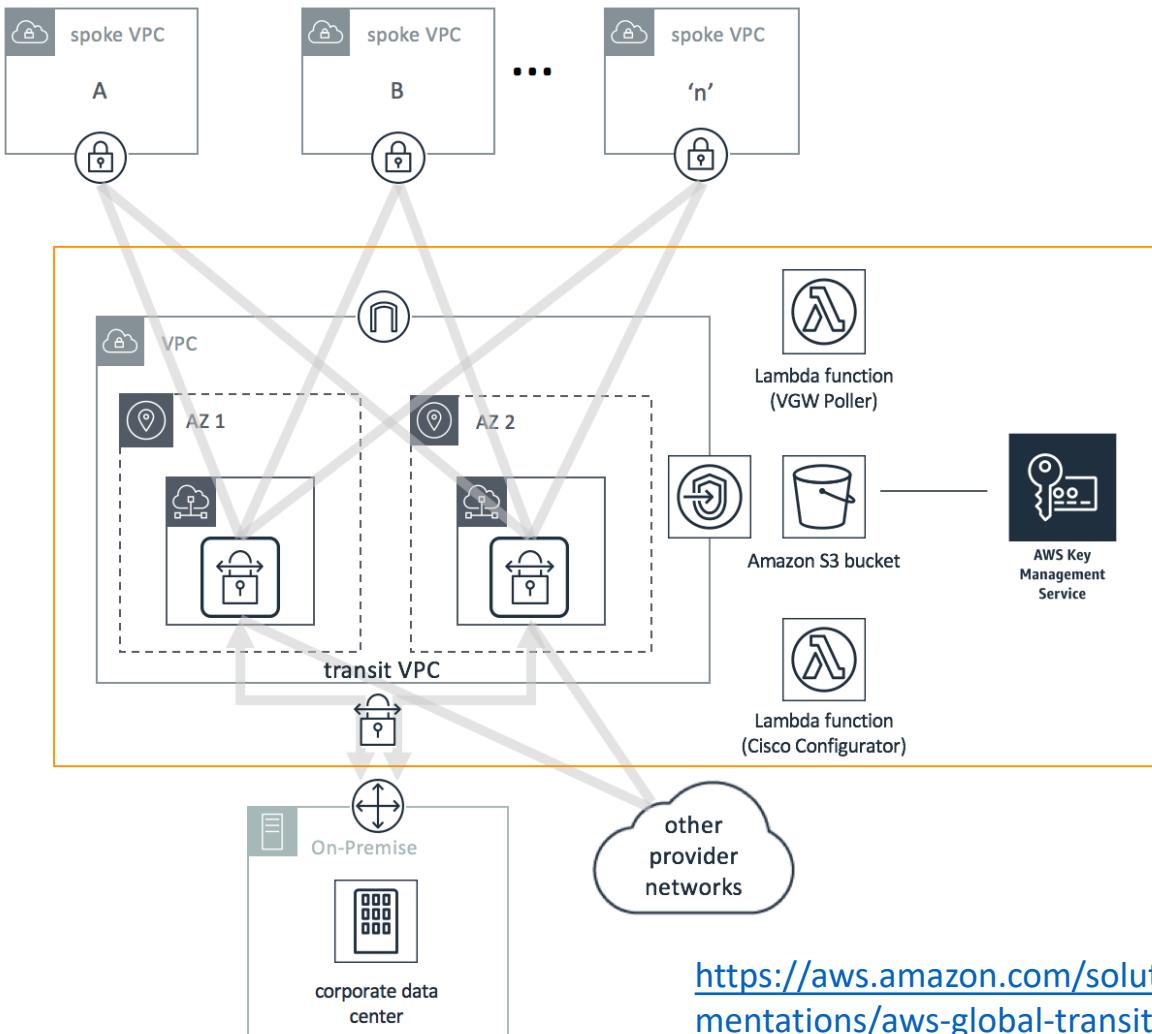


# Transit VPC scenarios

1. When multiple VPCs needs to be connected with on-premises network
2. Advanced Threat protection where traffic should be routed through centralized firewall systems
3. When you want to connect to on-premises network having overlapping CIDRs. Transit VPC acts as a NAT translating IPs to different range to enable communication.
4. Allow access for remote networks (Spoke VPC or On-premise network) to endpoints hosted in Transit Hub VPC
5. Client-to-Site VPN where client devices can connect to the Transit VPC EC2 instance by establishing VPN connection

# Transit VPC architecture for Transit Hub

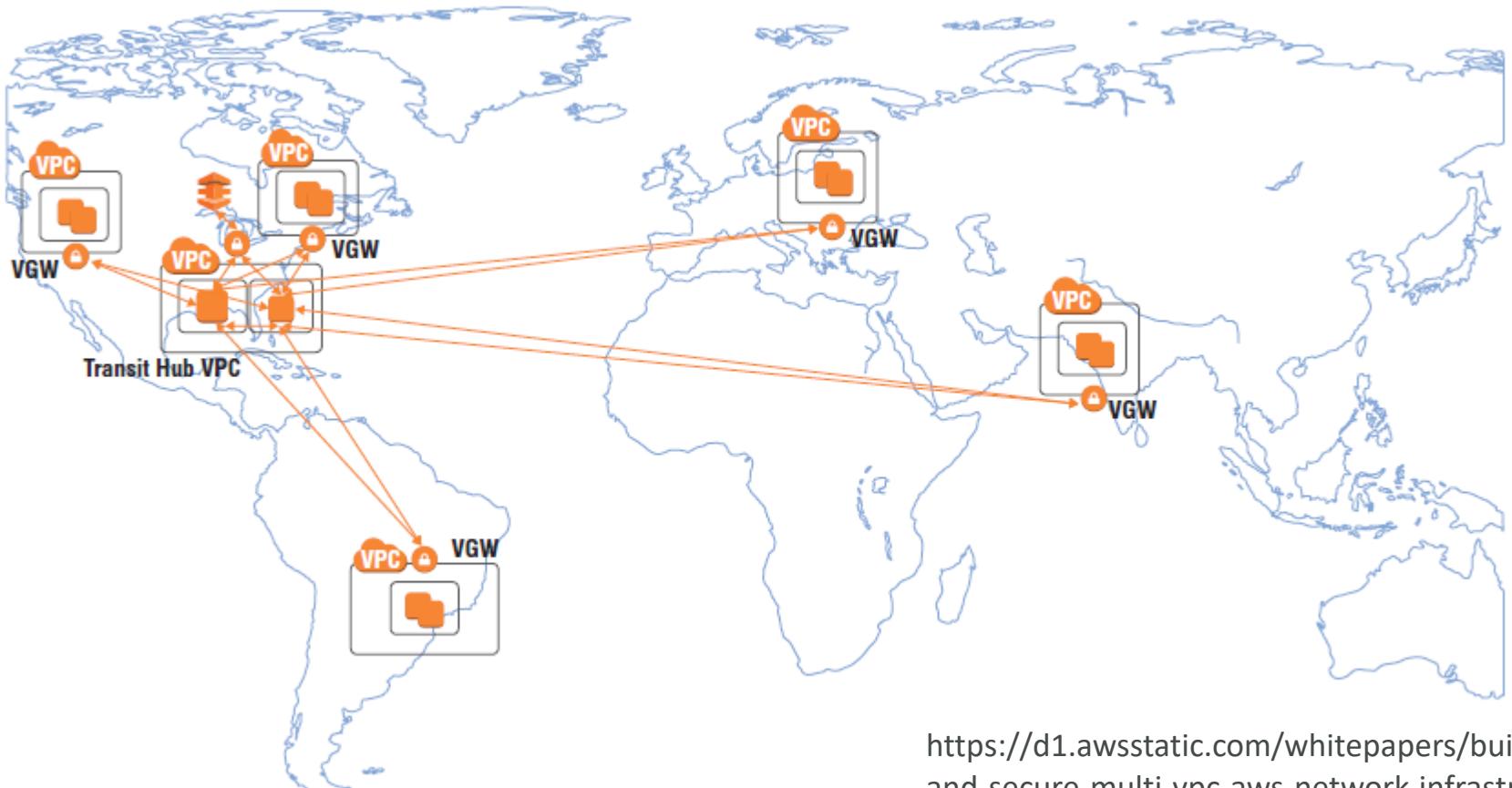
- Hub/Transit VPC: EC2 with VPN software (PaloAlto, Avitarix, CheckPoint etc), multi AZ for HA
- Each Spoke VPC has a VGW as their VPN termination
- On-premises establishes a VPN connection to the transit hub
- Possibility to use Direct Connect instead
- This architecture allows Full Mesh for communication between VPCs, On-premise VPN and AWS Direct Connect connection



<https://aws.amazon.com/solutions/implementations/aws-global-transit-network/>

# Transit VPC scenarios

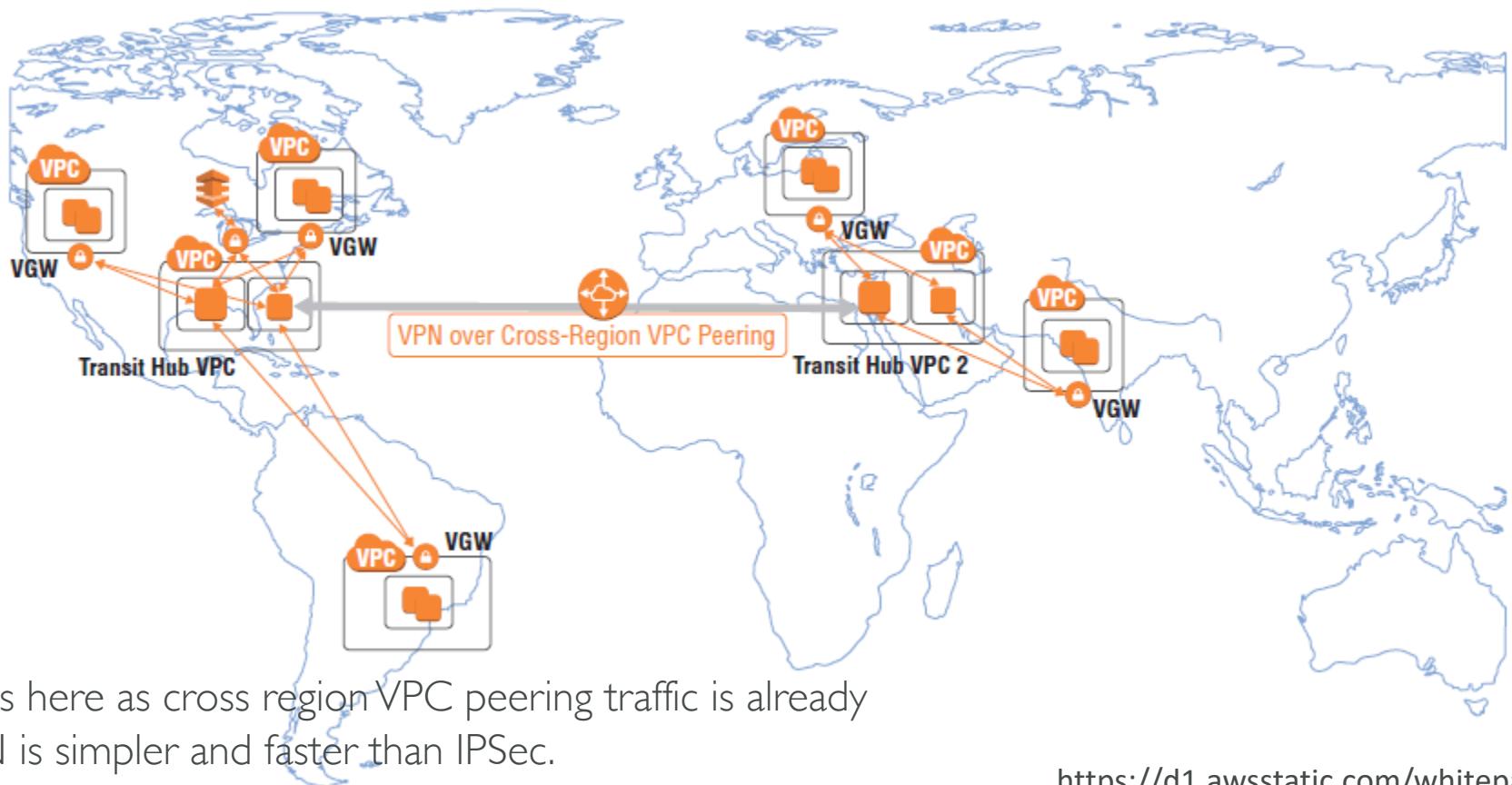
Global VPN infrastructure with single Transit Hub



<https://d1.awsstatic.com/whitepapers/building-a-scalable-and-secure-multi-vpc-aws-network-infrastructure.pdf>

# Transit VPC scenarios

Global VPN infrastructure with Transit Hubs in each region

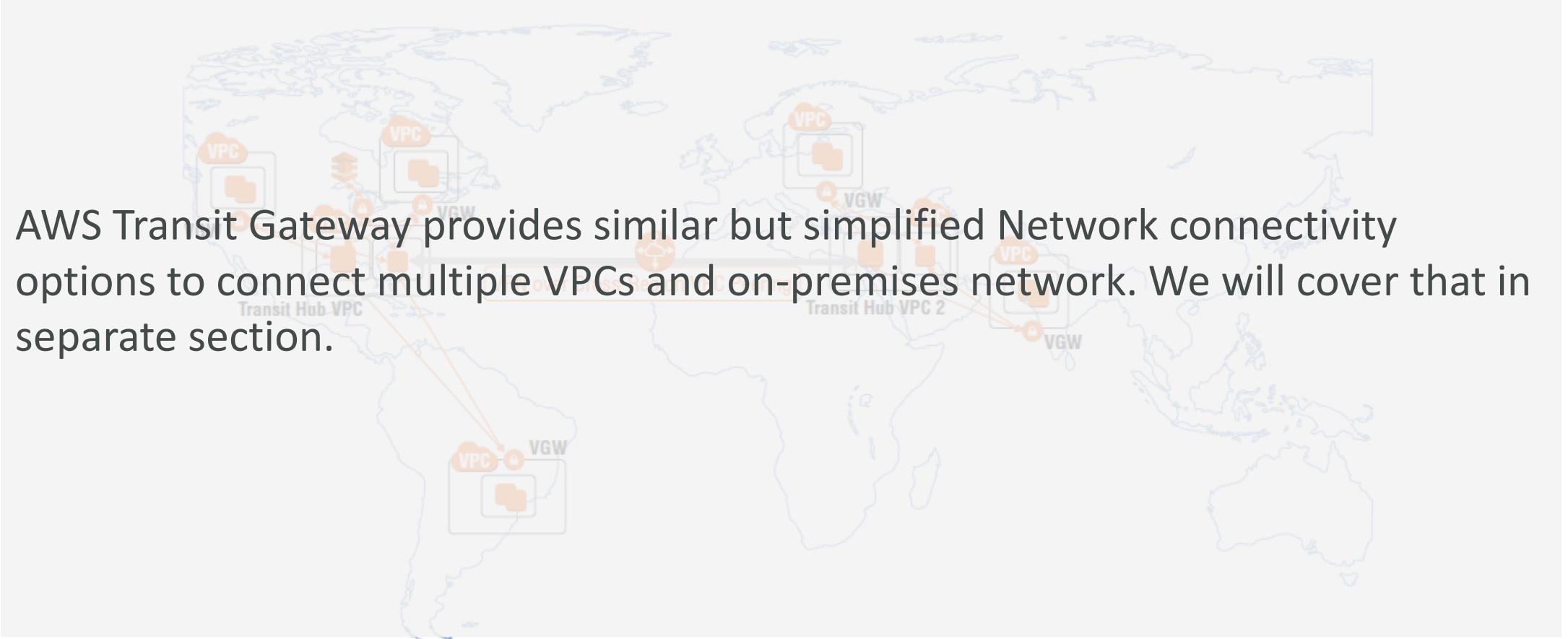


May use GRE tunnels here as cross region VPC peering traffic is already encrypted. GRE VPN is simpler and faster than IPSec.

<https://d1.awsstatic.com/whitepapers/building-a-scalable-and-secure-multi-vpc-aws-network-infrastructure.pdf>

# Transit VPC scenarios

Global VPN infrastructure with Transit Hubs in each region



# Site to Site VPN Summary

- VPN comes into 2 categories – Site to Site and Client to Site
- AWS has fully managed VPN solution for Site-to-Site VPN. On AWS side, VPN terminates at Virtual Private Gateway (VGW)
- AWS supports IPSec Site-to-Site (S2S) VPN
- AWS S2S VPN creates 2 tunnels for high availability
- After successful VPN setup, the traffic should be initiated from the customer end. **VGW never initiates the traffic. (was true until Feb 2021)**
- The VGW has the aggregate bandwidth limit of 1.25 Gbps
- AWS S2S VPN supports static and dynamic routing using BGP
- You can not publish more than 100 routes towards VGW. You can consolidate CIDRs in case you are hitting the limit

# Exam Essentials

- AWS managed VPN supports only IPSec protocol
- You can not create VPN connection between 2 VPCs using VGWs at both the end of the connection. This is because VGW never initiates the VPN traffic. Traffic should be initiated from customer side of VPN.
- You can monitor the tunnel status with AWS CloudWatch metrics like TunnelState, TunnelDataIn, TunnelDataOut
- AWS managed VPN does not support transitive routing
- For Client to Site VPN, you need to terminate VPN connection on EC2 (until 2018 when AWS released Managed Client to Site VPN service)
- There are many other scenarios for which you need to terminate the AWS side of the VPN connection on EC2 instance

# AWS Site-to-Site VPN – Good to know

- IPv6 traffic is not supported for VPN connections with VGW however IPv6 is supported with VPN connection with Transit Gateway
- AWS managed VPN connection does not support Path MTU Discovery

# AWS Site-to-Site VPN - Appendix

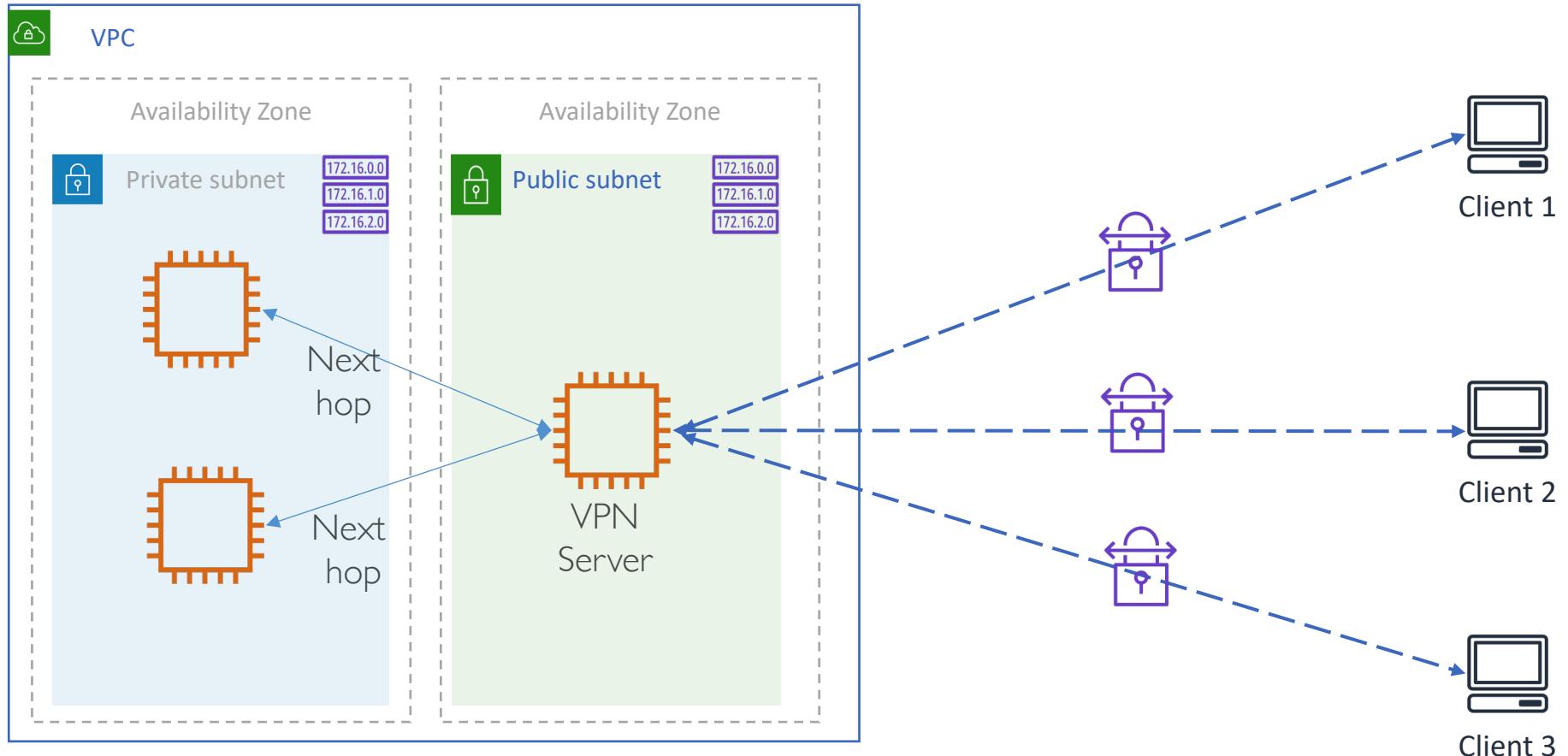
# Which customer gateway you can use for VPN?

- Customer gateway devices supporting statically-routed VPN connections must be able to:
  - Establish IKE Security Association using Pre-Shared Keys
  - Establish IPsec Security Associations in Tunnel mode
  - Utilize the AES 128-bit, 256-bit, 128-bit-GCM-16, or 256-GCM-16 encryption function
  - Utilize the SHA-1, SHA-2 (256), SHA2 (384) or SHA2 (512) hashing function
  - Utilize Diffie-Hellman (DH) Perfect Forward Secrecy in "Group 2" mode, or one of the additional DH groups we support
  - Perform packet fragmentation prior to encryption
- In addition to the above capabilities, devices supporting dynamically-routed Site-to-Site VPN connections must be able to:
  - Establish Border Gateway Protocol (BGP) peering
  - Bind tunnels to logical interfaces (route-based VPN)
  - Utilize IPsec Dead Peer Detection

# AWS Client VPN

# Client to Site VPN using EC2

(AWS Managed Client VPN was launched in Dec 2018)



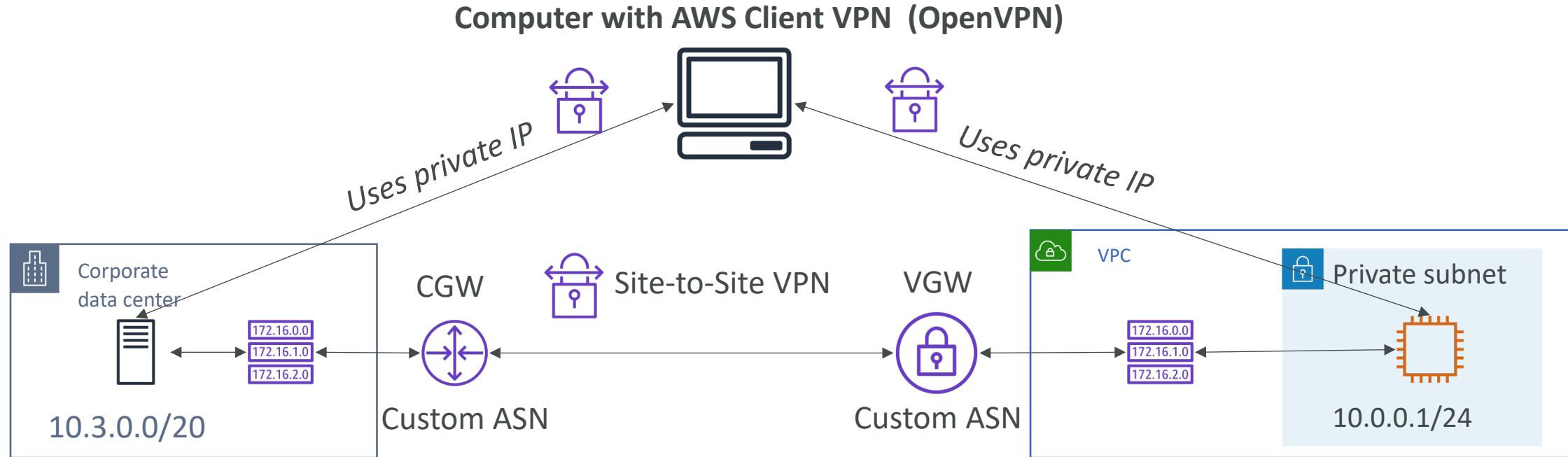
# Client to Site VPN features

- AWS has a managed Client VPN (so far out of scope for the exam)
- Or setup Client VPN on EC2:
  - AWS Marketplace: Use pre-baked AMI with VPN software and license. Example vendors are Cisco, Aviatrix, Palo Alto, Sophos etc
  - Manual installation on EC2: OpenSwan, StrongSwan
- For high availability pointers:
  - Multiple EC2 instances with NLB
  - Optionally use DNS load balancing (client side) instead of NLB
- You may also have to implement Split tunneling at client side

# AWS Client VPN

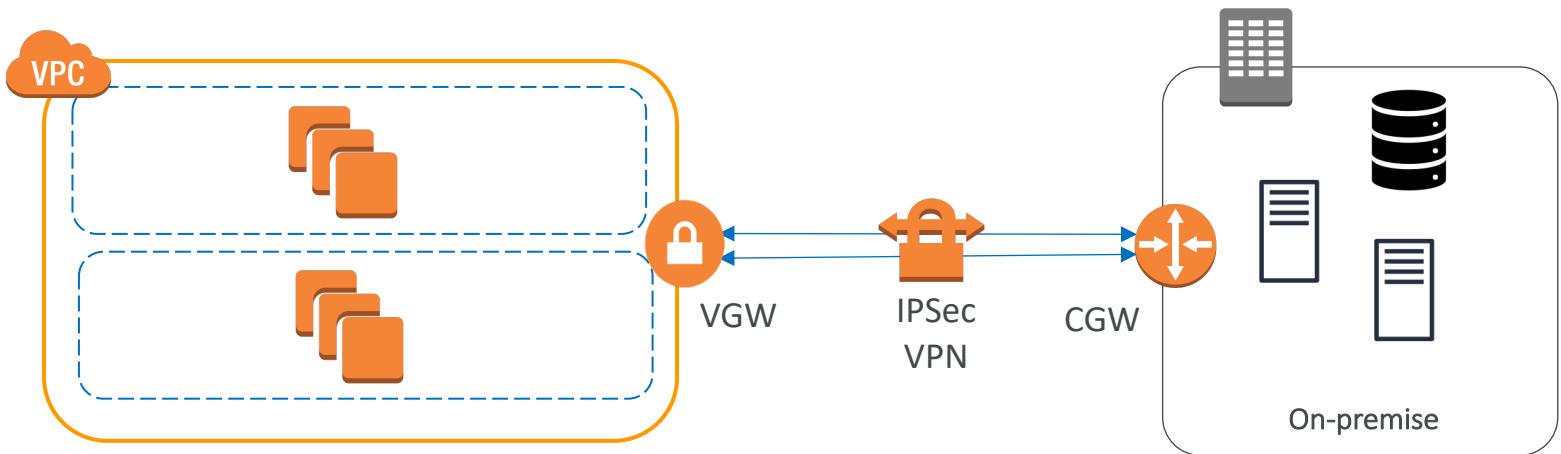


- Connect from your computer using OpenVPN to your private network in AWS and on-premise

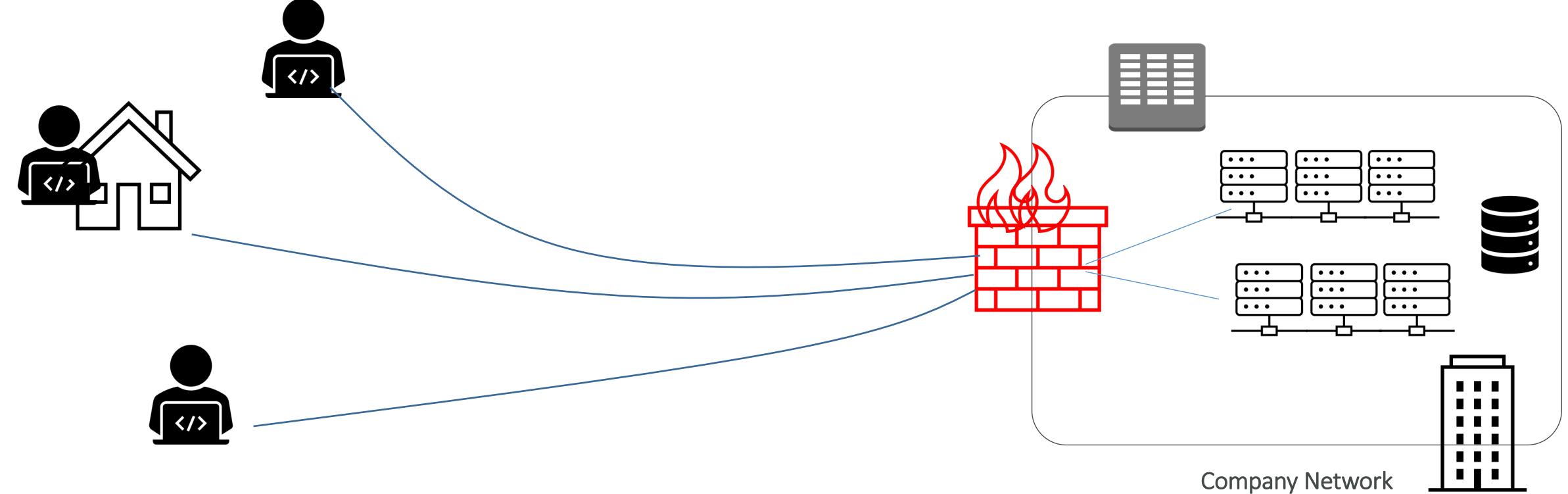


# Site to Site VPN

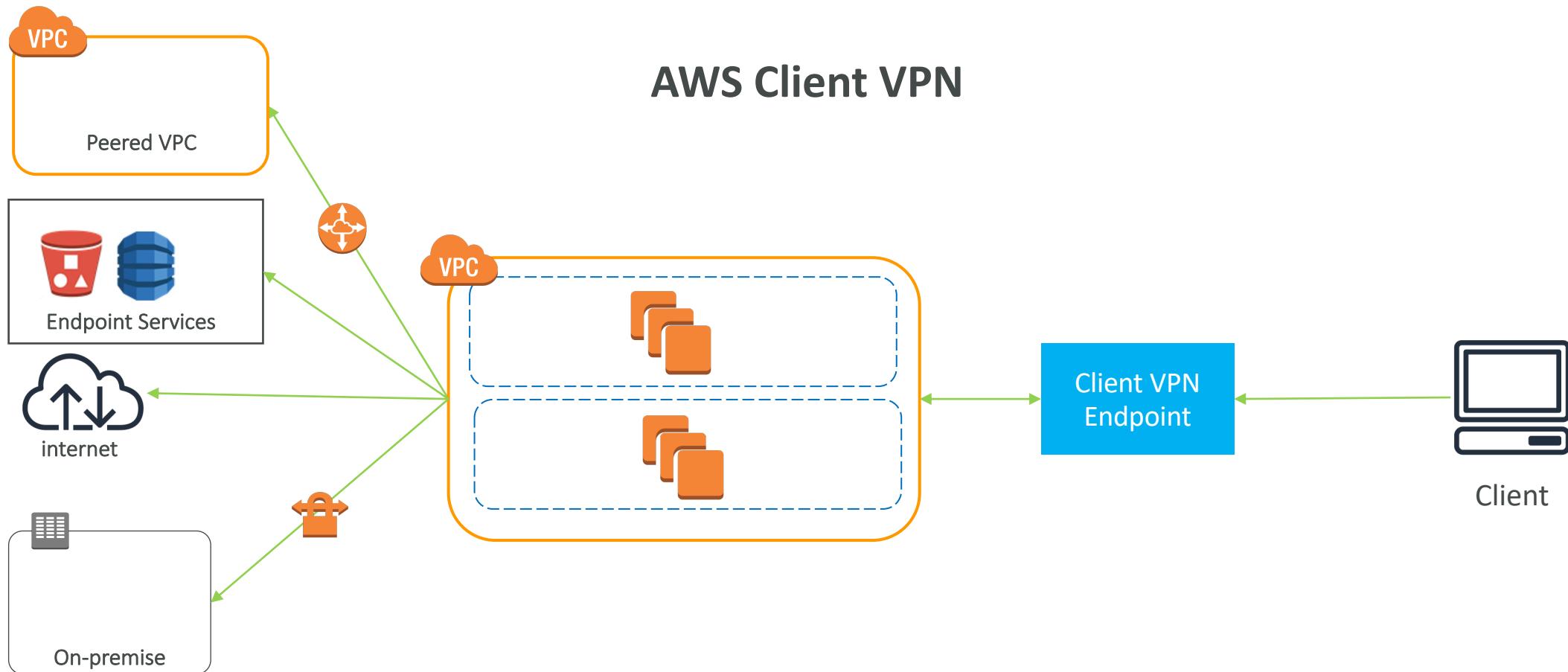
## Site to Site VPN



# Client to Site VPN



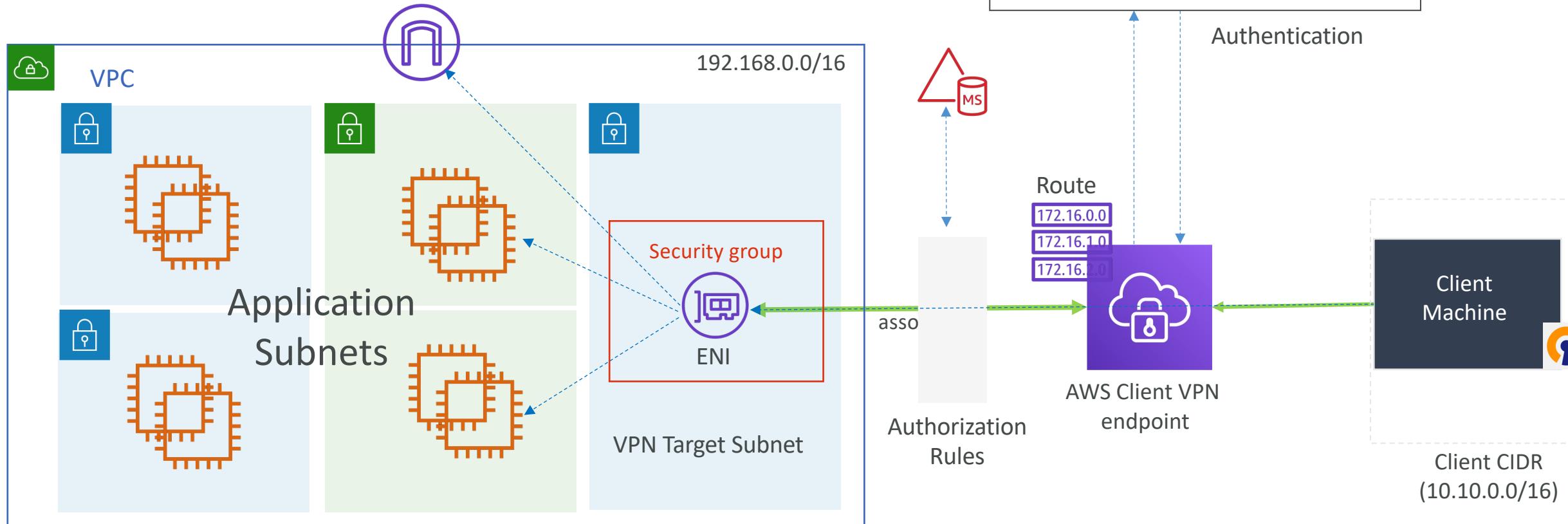
# Client to site VPN



# AWS Client VPN Components

- VPC
- Target Network Subnet
- Client VPN Endpoint
- Route
- Authorization Rules
- Client CIDR range
- Client VPN Network Interfaces
- Authentication (Mutual, AD based, SAML based federated)
- Authorization (Security groups, Authorization groups)
- Client

# AWS Client VPN components

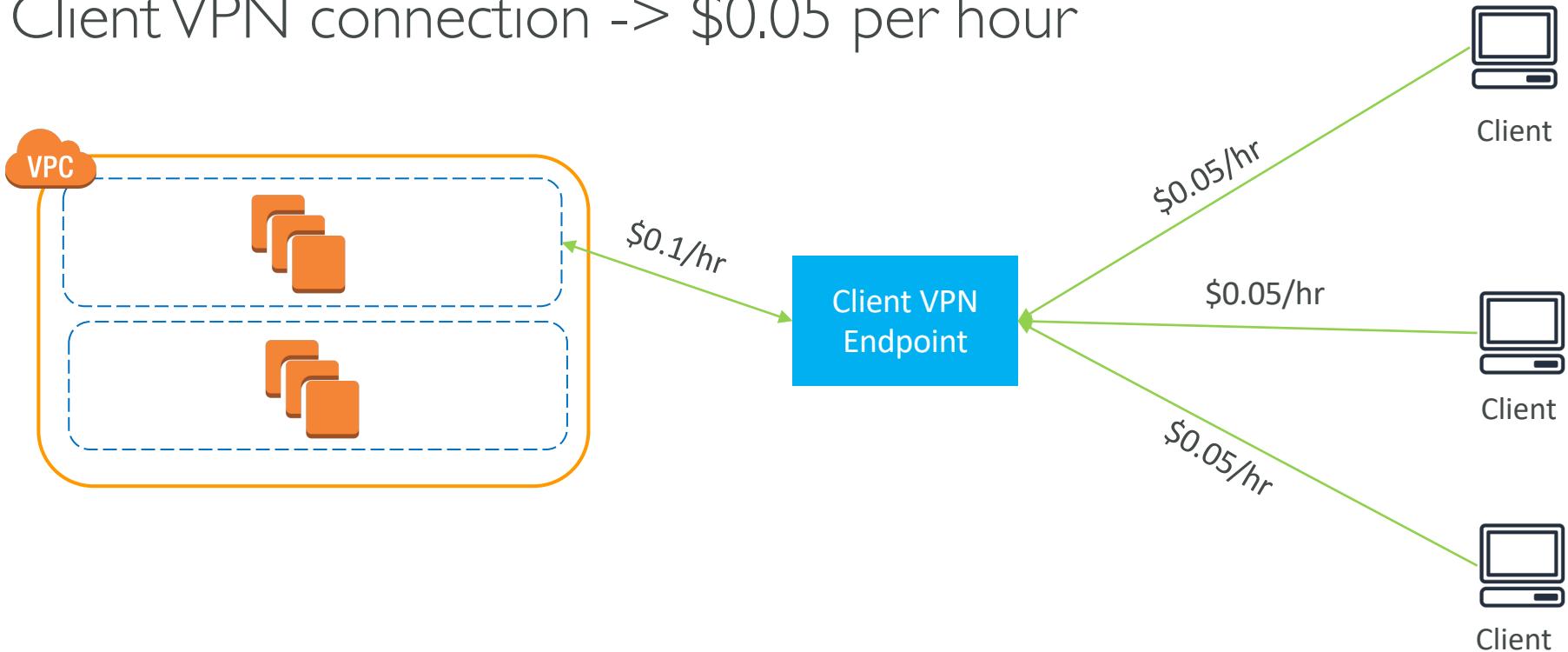


# AWS Client VPN Limitations

- Client CIDR ranges cannot overlap with the local CIDR of the VPC in which the associated subnet is located
- Client CIDR ranges cannot overlap with any routes manually added to the Client VPN endpoint's route table
- Client CIDR ranges must have a block size between /22 and /12
- The client CIDR range cannot be changed after you create the Client VPN endpoint
- You cannot associate multiple subnets from the same Availability Zone with a Client VPN endpoint
- A Client VPN endpoint does not support subnet associations in a dedicated tenancy VPC
- ClientVPN supports IPv4 traffic only

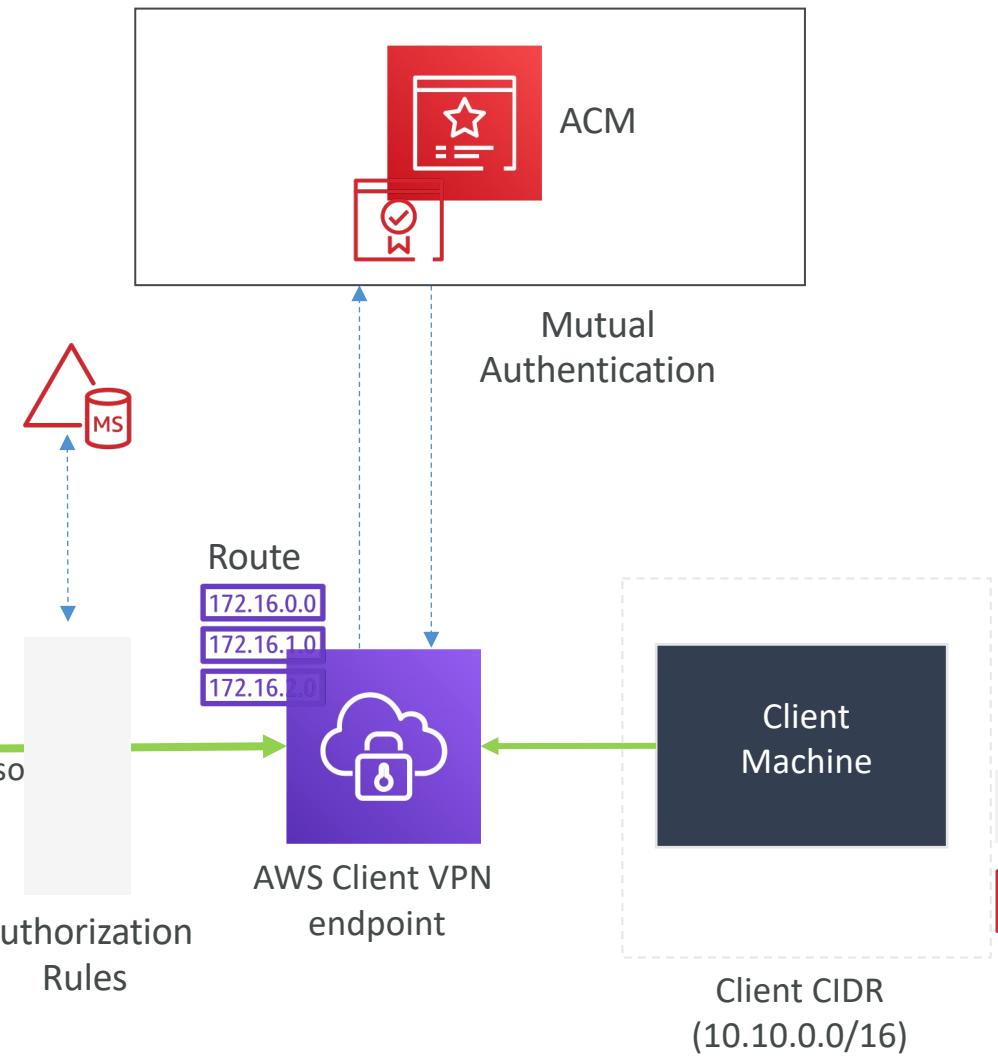
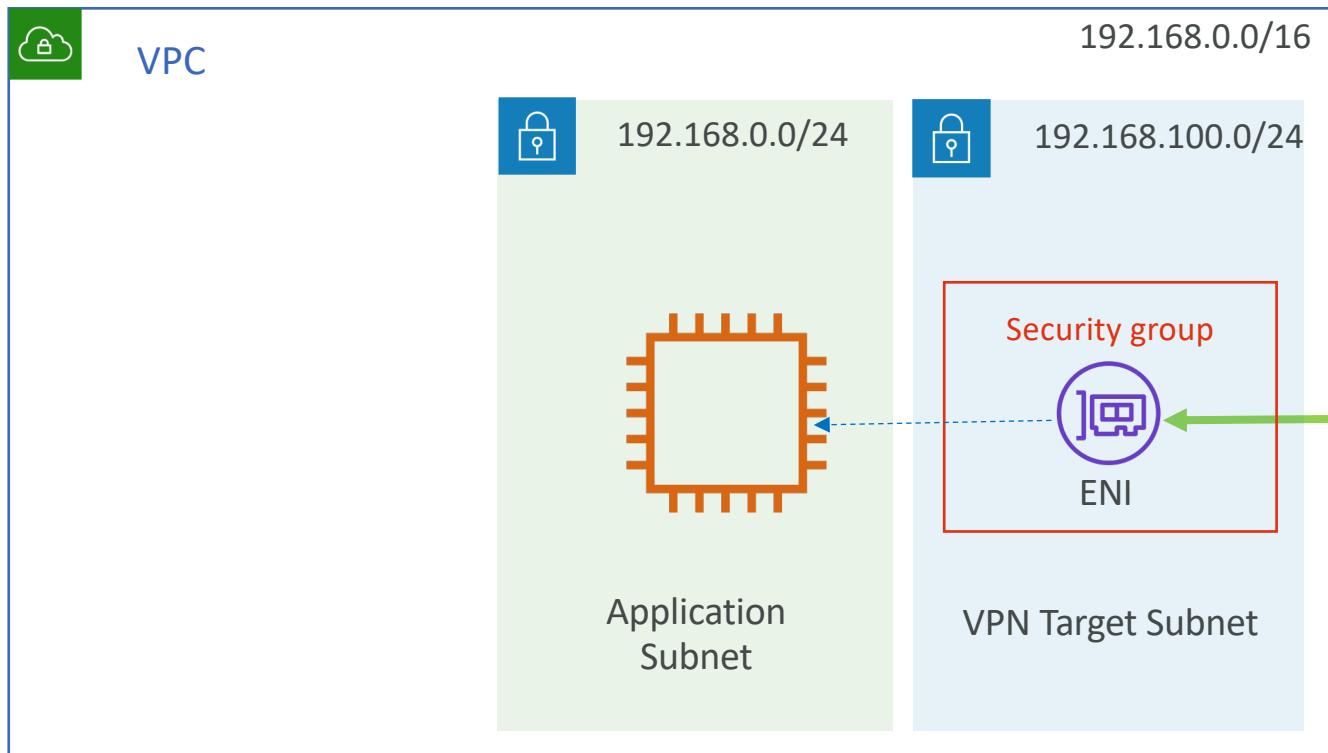
# AWS ClientVPN Pricing (N.Virginia region)

- AWS ClientVPN endpoint association -> \$0.10 per hour
- AWS ClientVPN connection -> \$0.05 per hour



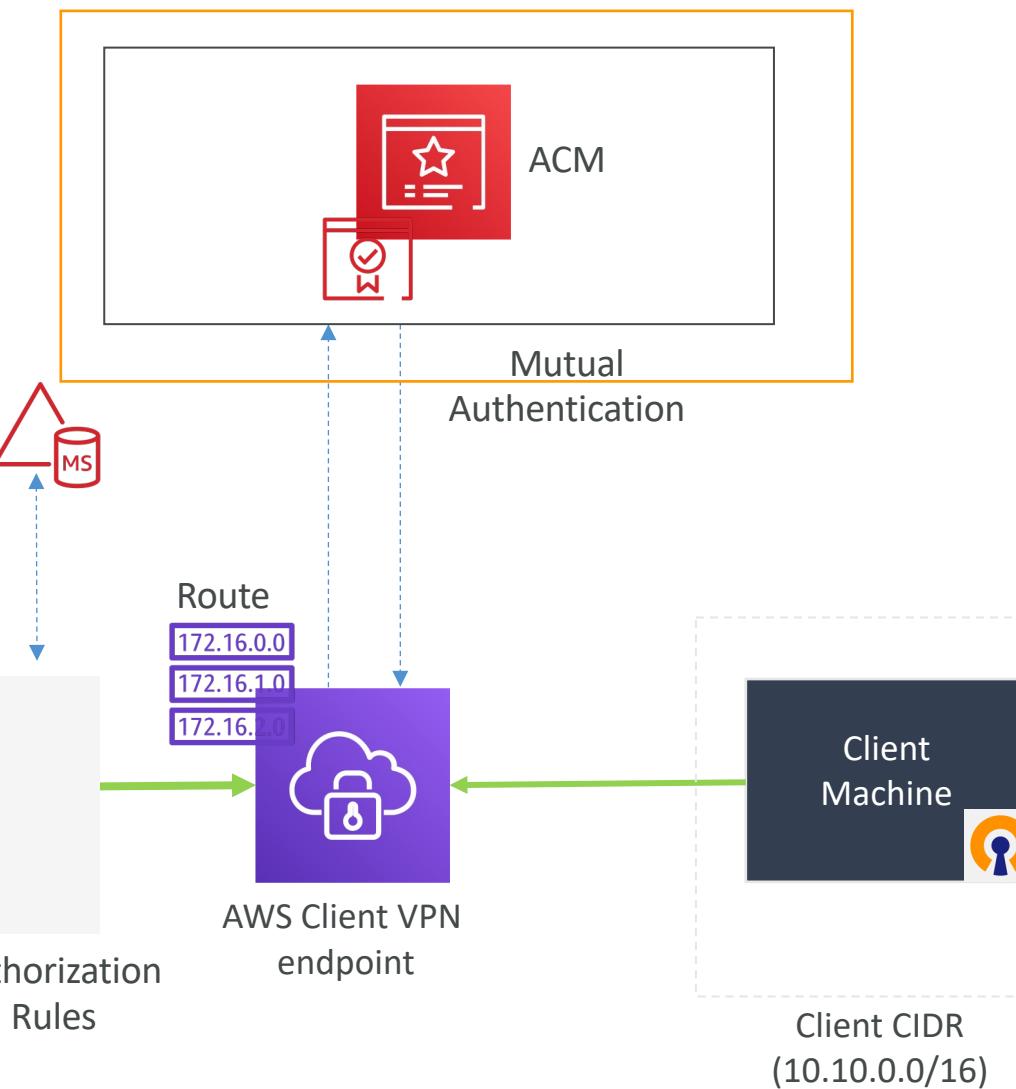
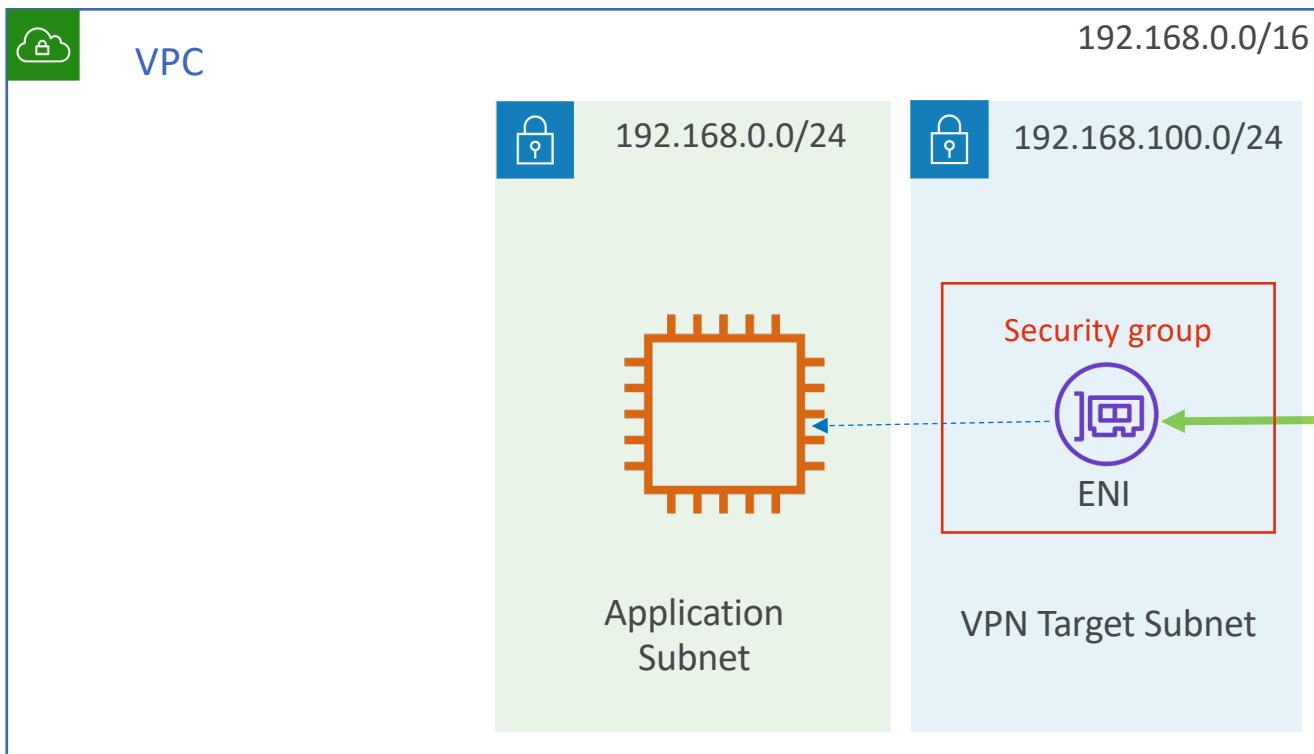
# How to setup AWS Client VPN

# Architecture



## 1. Create certificates/keys for Mutual Authentication

- Create Server and Client Certificates and keys
- Upload to ACM



<https://docs.aws.amazon.com/vpn/latest/clientvpn-admin/client-authentication.html#mutual>

# Create Server and Client certificates and keys

Run below commands from your workstation (Should have AWS CLI)

1. Clone the easy-rsa repo

```
$ git clone https://github.com/OpenVPN/easy-rsa.git
$ cd easy-rsa/easyrsa3
```

2. Initialize PKI environment

```
$ ./easyrsa init-pki
```

3. Create new Certification Authority (CA)

```
$ ./easyrsa build-ca nopass
```

4. Generate the server certificate and key

```
$ ./easyrsa build-server-full server nopass
```

# Create Server and Client certificates and keys

5. Generate the client certificate and key

```
$ ./easyrsa build-client-full client1.domain.tld nopass
```

6. Copy server and client certificates and keys to one directory

```
$ mkdir ~/demo  
$ cp pki/ca.crt ~/demo/  
$ cp pki/issued/server.crt ~/demo/  
$ cp pki/private/server.key ~/demo/  
$ cp pki/issued/client1.domain.tld.crt ~/demo/  
$ cp pki/private/client1.domain.tld.key ~/demo/  
$ cd ~/demo
```

# Create Server and Client certificates and keys

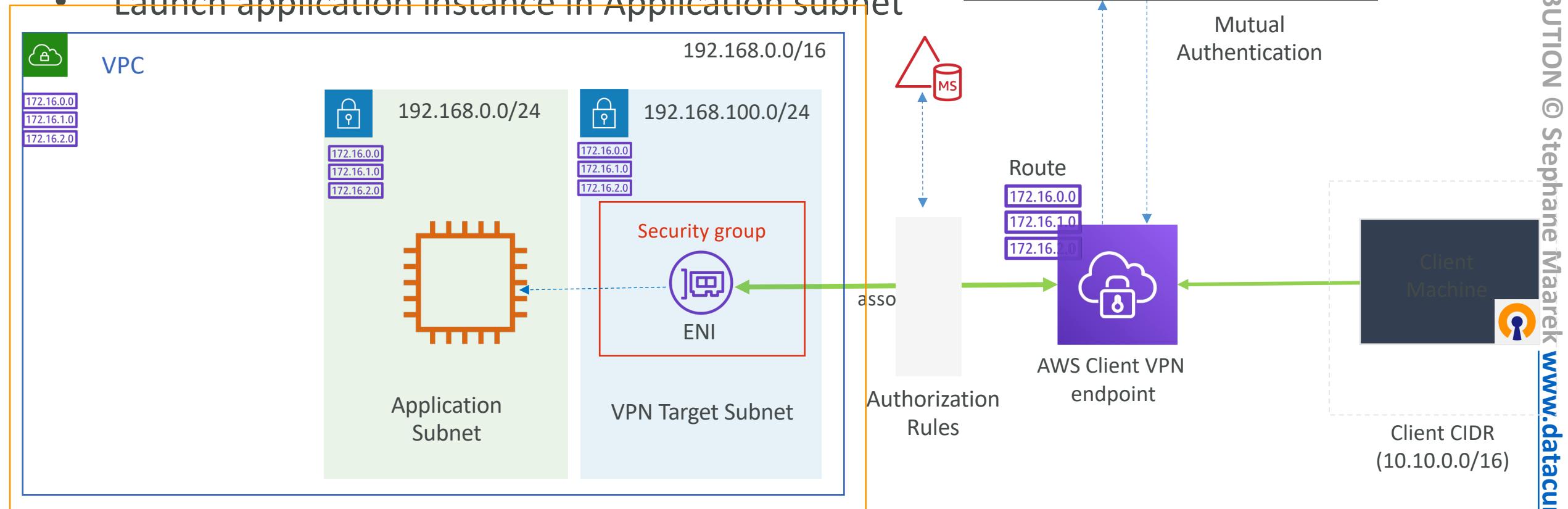
## 7. Upload the certificate and keys to ACM

```
$ aws acm import-certificate --certificate fileb://server.crt --private-key fileb://server.key --certificate-chain fileb://ca.crt --region ap-south-1
```

```
$ aws acm import-certificate --certificate fileb://client1.domain.tld.crt --private-key fileb://client1.domain.tld.key --certificate-chain fileb://ca.crt --region ap-south-1
```

## 2. Setup VPC

- Create VPC and 2 Subnets (private) and route tables
- Create security group for VPN Target subnet
- Launch application instance in Application subnet



# Steps to setup VPC

1. Create VPC (name=demo) with CIDR 192.168.0.0/16
2. Create private subnet “demo-app-l” with CIDR 192.168.0.0/24
3. Create corresponding route table “demo-app-rt” with just a local route & associate with subnet “demo-app-l”
4. Create private subnet “demo-client-vpn-l” with CIDR 192.168.100.0/24
5. Create corresponding route table “demo-client-vpn-rt” with just a local route & associate with subnet “demo-client-vpn-l”
6. Create security group “demo-client-vpn-sg”
  - Do not add any inbound rules
  - All outbound should be allowed (All traffic – 0.0.0.0/0)

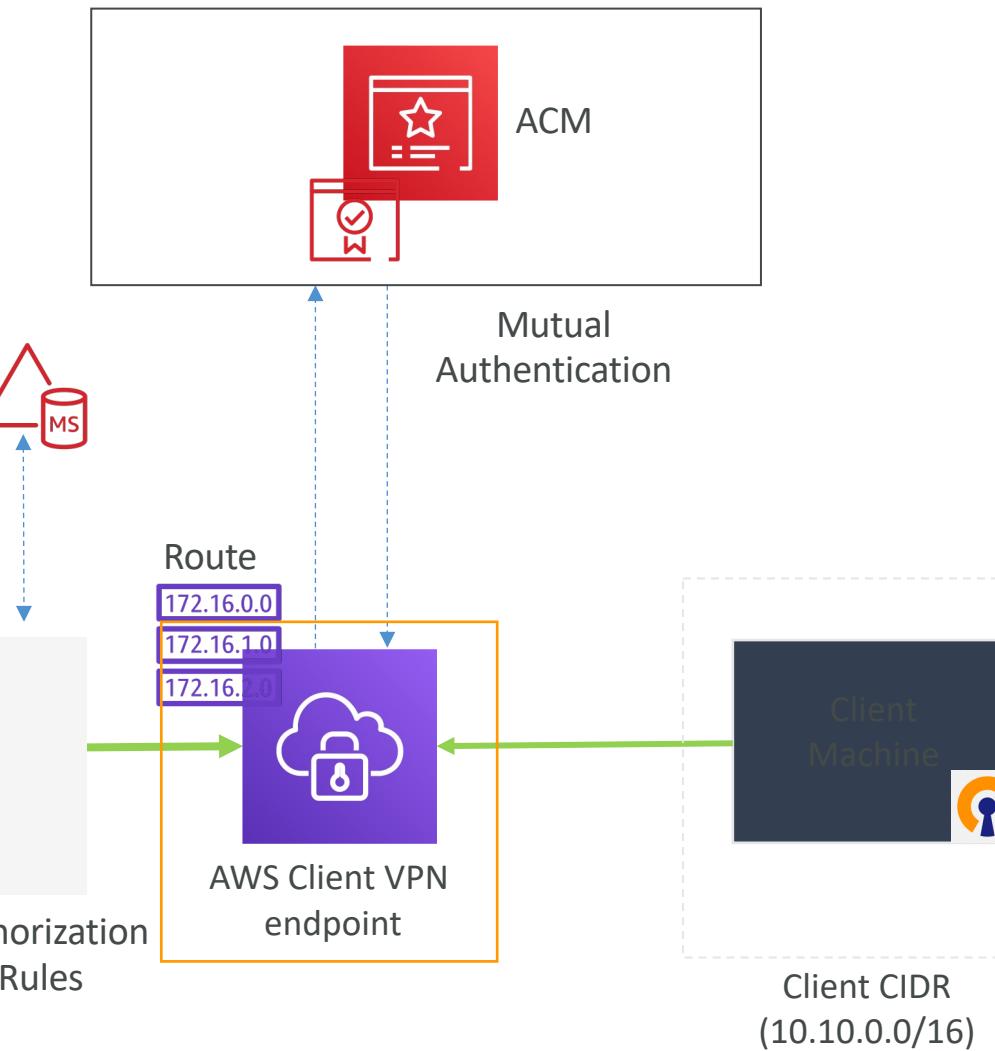
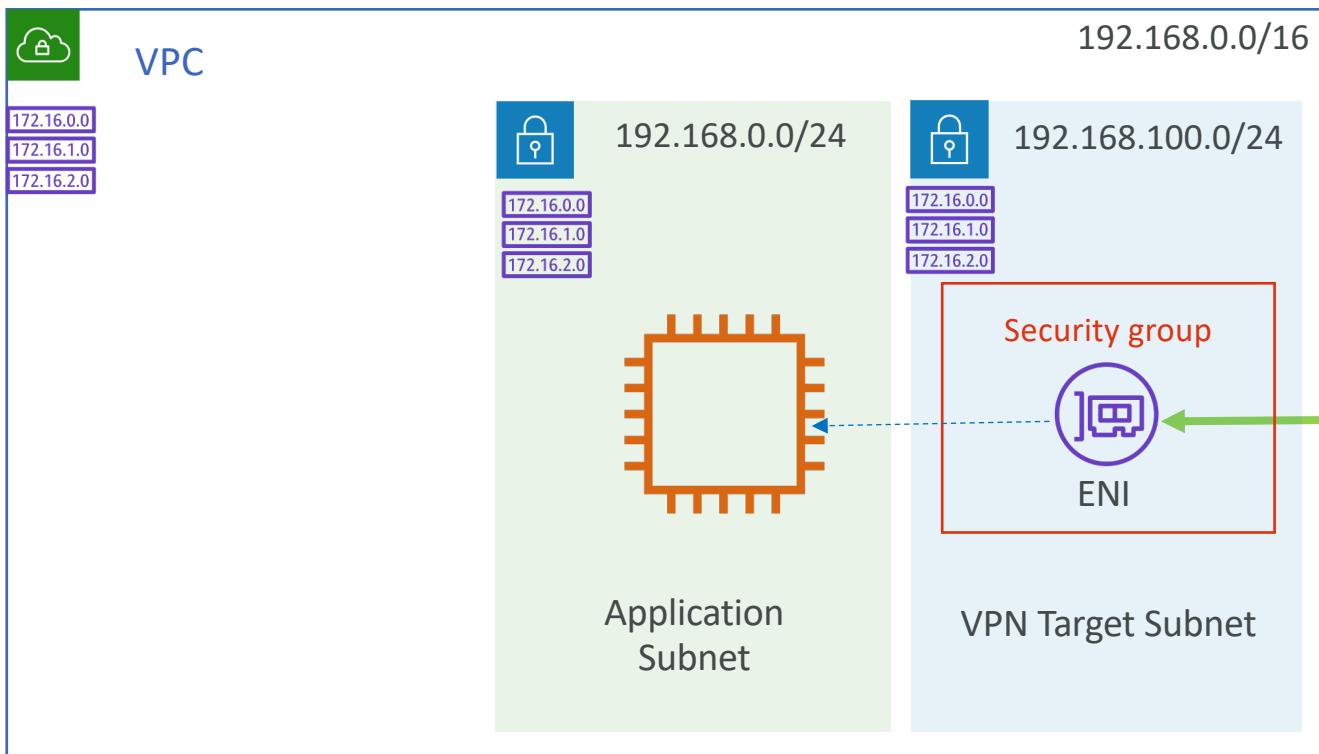
# Steps to setup VPC

7. Launch application EC2 instance in “demo-app-1” subnet

- Security group inbound rule should allow “All traffic” from security group “demo-client-vpn-sg” created in step 6

### 3. Create AWS Client VPN Endpoint

- Provide Client CIDR address (10.10.0.0/16)
- Provide ACM Server and Client Certificate
- Provide VPC and Security group details



# Steps to create Client VPN endpoint

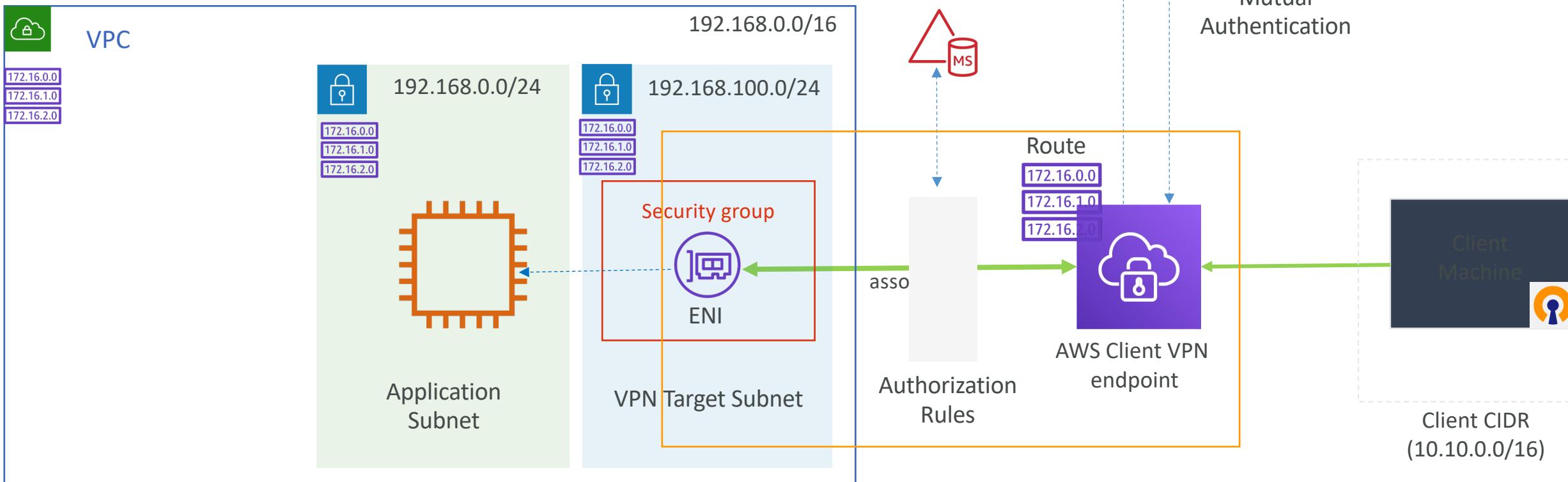
1. Provide name “demo-client-vpn-endpoint” and description
2. Client IPv4 CIDR: 10.10.0.0/16
3. Server Certificate ARN: Choose the Server Certificate created earlier
4. Authentication Options: Choose “Use Mutual Authentication”
5. Client certificate ARN: Choose the Client Certificate created earlier
6. Connection Logging: No
7. Transport Protocol:TCP
8. VPC ID: Choose “demo”VPC created in Step 2

# Steps to create ClientVPN endpoint

9. Security Group IDs: Select the “demo-client-vpn-sg” created earlier
10. VPN port: 443
11. Create ClientVPN Endpoint

## 4. Associate Target Subnet & Authorize traffic

- Target subnet (192.168.100.0/24)
- Authorize clients to access the network
- Verify the route for VPC

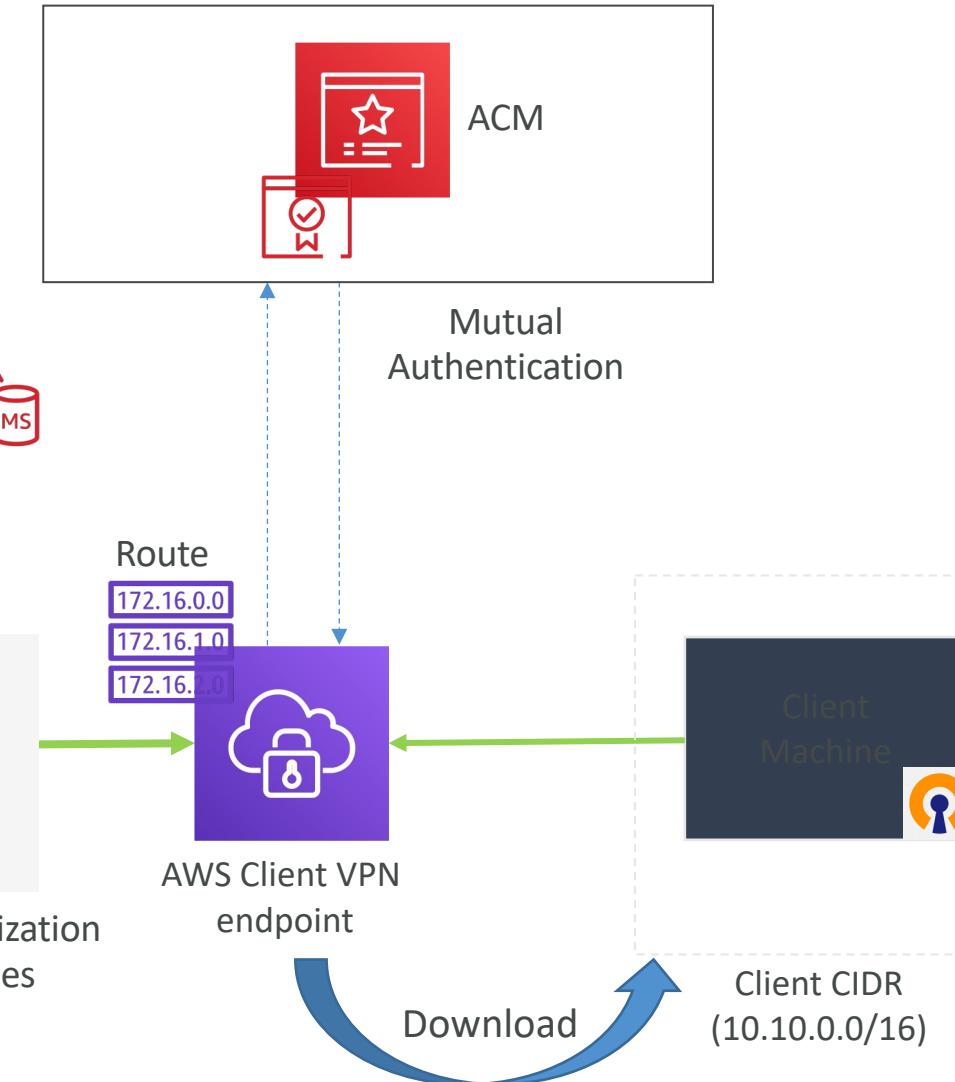
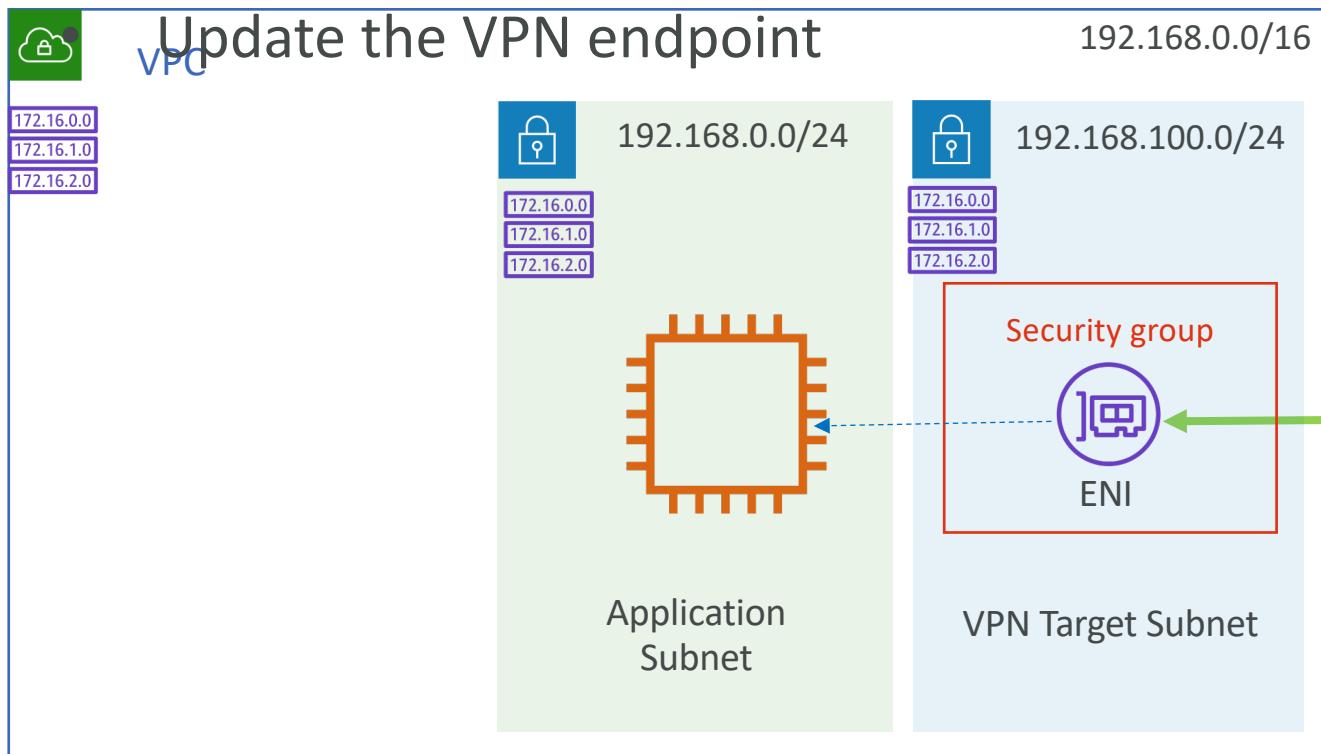


# Steps to associate Target Subnet and Authorize traffic

- Select the Client VPN endpoint created earlier
- Go to Associations and associate the target subnet “demo-client-vpn-1”
- Go to Authorizations and choose Authorize Ingress
  - For Destination Networks to enable -> Enter the VPC IP address 192.168.0.0/16
  - Grant access to -> Choose "Allow access to all users"
- Add Authorization Rule

## 5. Download and update VPN configuration file

- Download file to your local machine
- Add the Client Certificate and Key details in the file



# Steps to download and update VPN configuration file

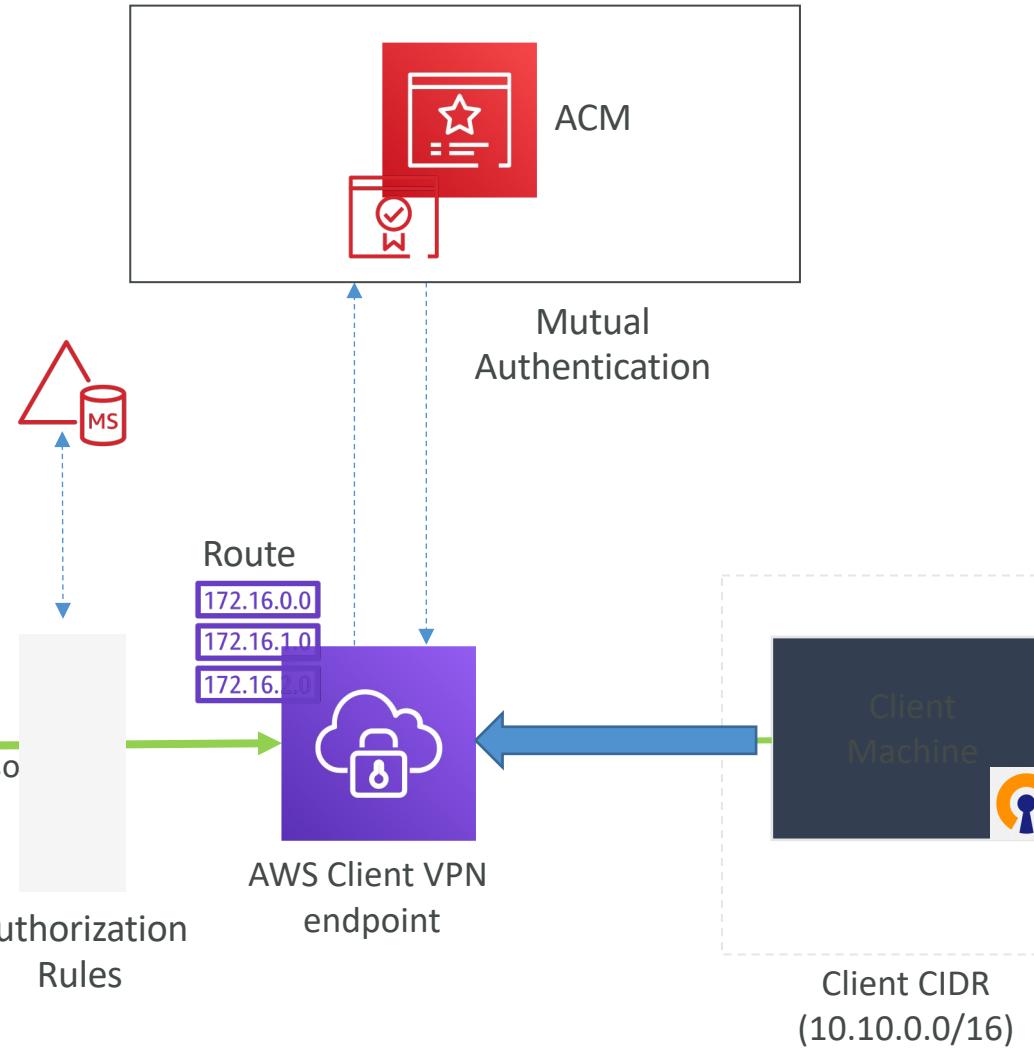
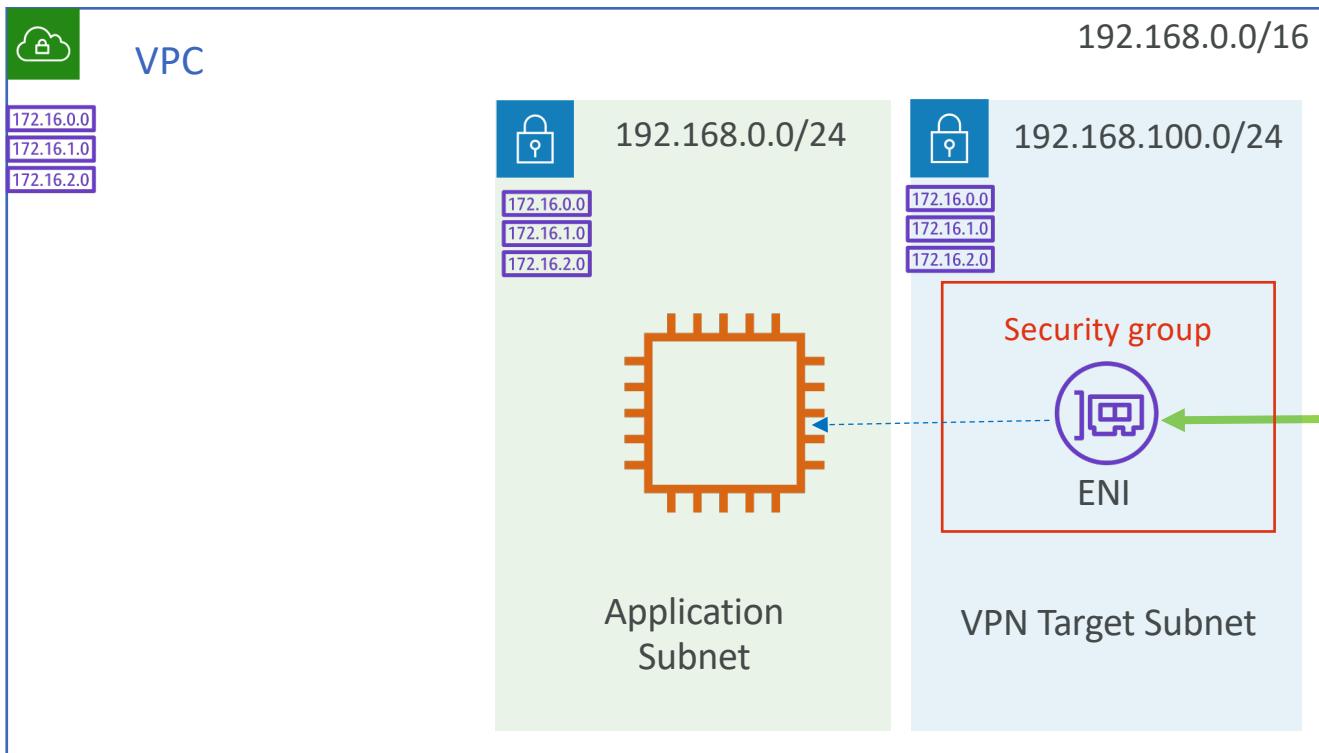
- I. Select Client VPN endpoint and “Download Client Configuration” to your local workstation
2. Copy the client certificate and client key created in Step I to any folder in local workstation
3. Open the configuration file with any editor and add following lines
  - I. cert /path/to/client1.domain.tld.crt
  2. key /path/to/client1.domain.tld.key

# Steps to download and update VPN configuration file

4. Also, modify the endpoint dns name by adding random prefix
  1. **Original:** cvpn-endpoint-0102bc4c2eEXAMPLE.prod.clientvpn.us-west-2.amazonaws.com
  2. **Modified:** xxxxxxxx.cvpn-endpoint-0102bc4c2eEXAMPLE.prod.clientvpn.us-west-2.amazonaws.com

## 6. Connect

- Import the VPN configuration file in OpenVPN client
- Connect

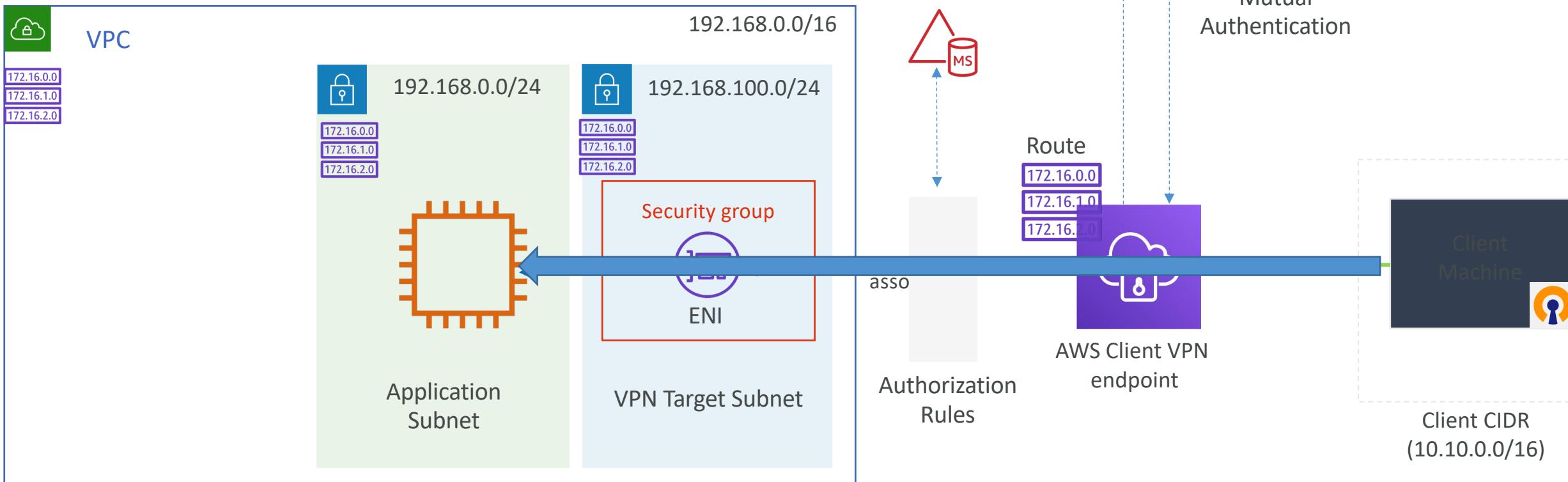


# Steps to connect

- Pre-requisite: You should download and install OpenVPN client
  - <https://openvpn.net/community-downloads/>
- Import configuration file
- Connect

## 7. Verify the connectivity

- Try to ping or ssh to Application EC2 instance from your local machine

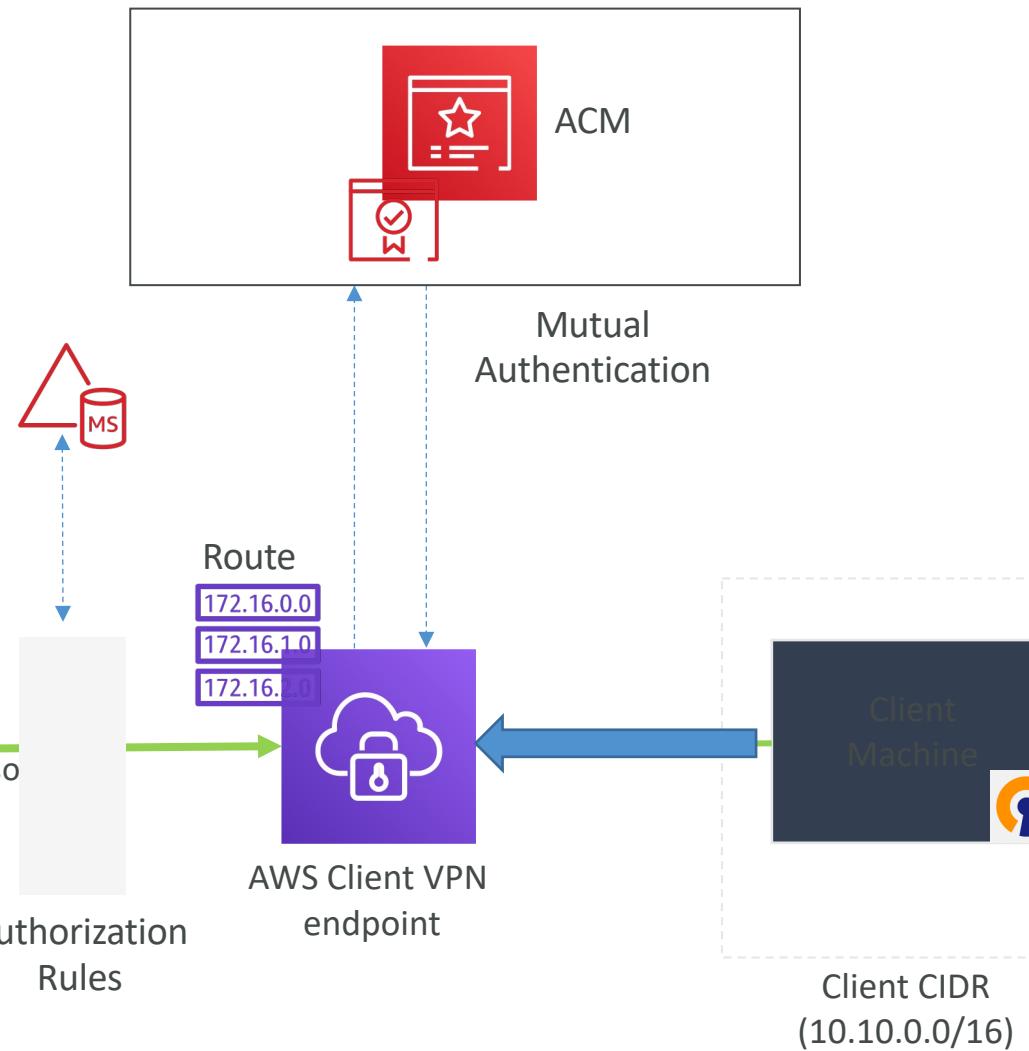
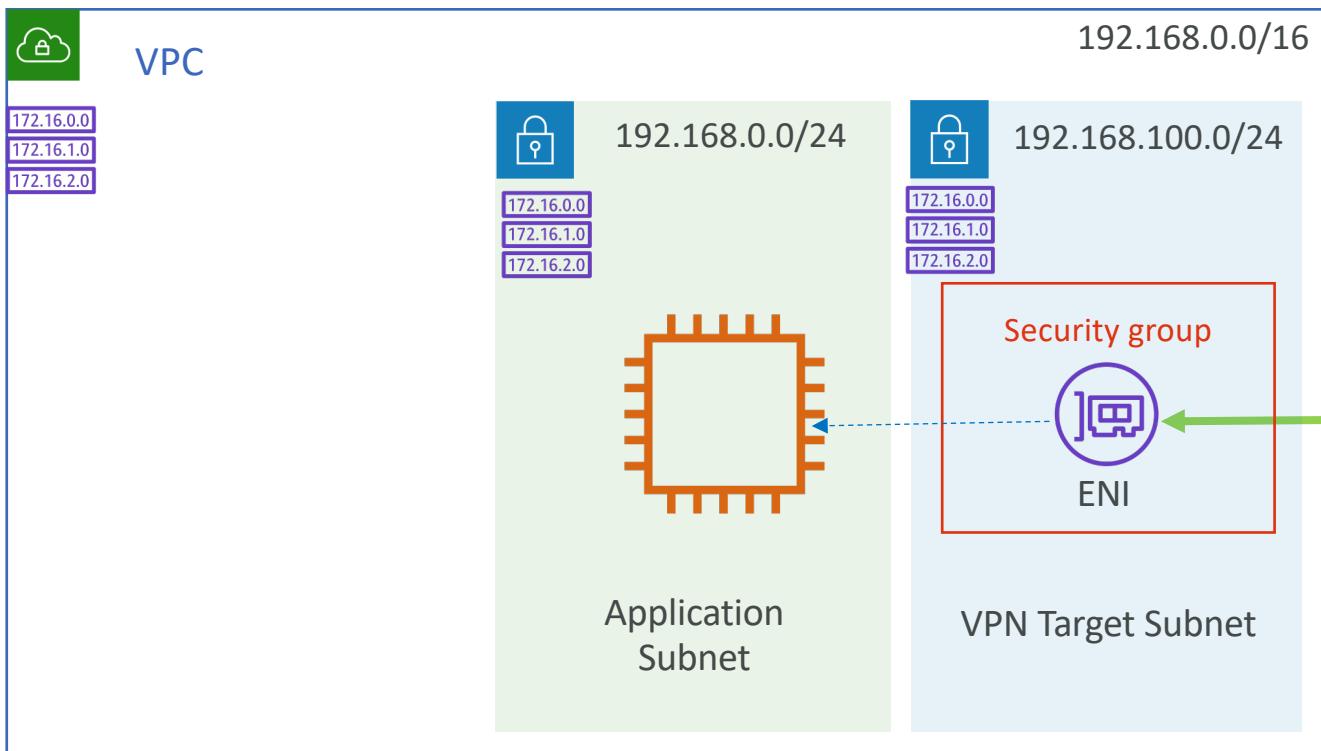


# Steps to verify the VPN connection

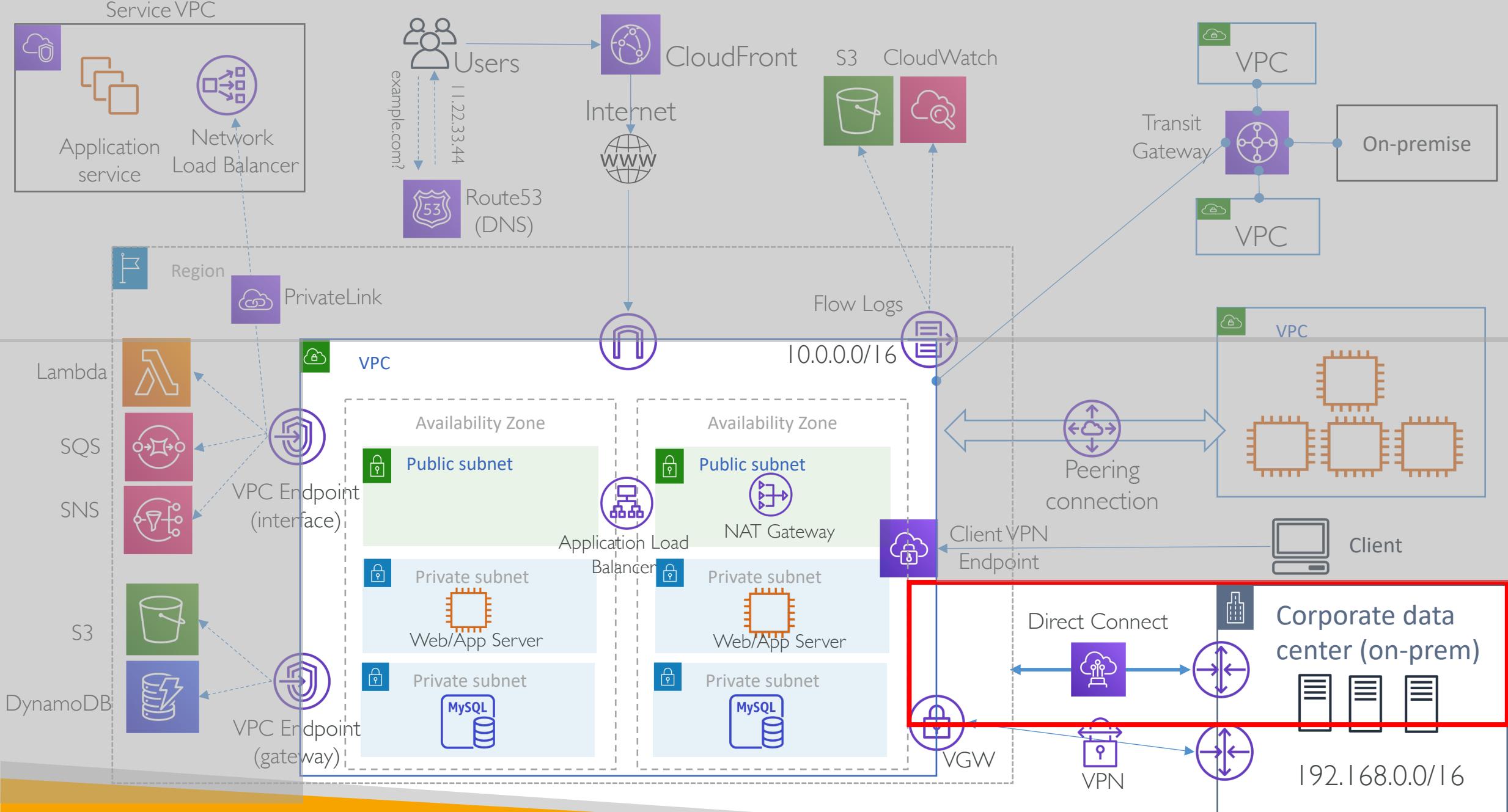
- Get the private IP address of Application EC2 instance say 192.168.0.55
- Open the command prompt from your local workstation
  - ping 192.168.0.55
- If you are using Windows workstation, also try to open SSH connection to Application instance
- Try to access now internet from your local workstation
  - Browse any website -> Does not work
  - ping amazon.com -> Does not work
- Why ?

## How to Connect to Internet?

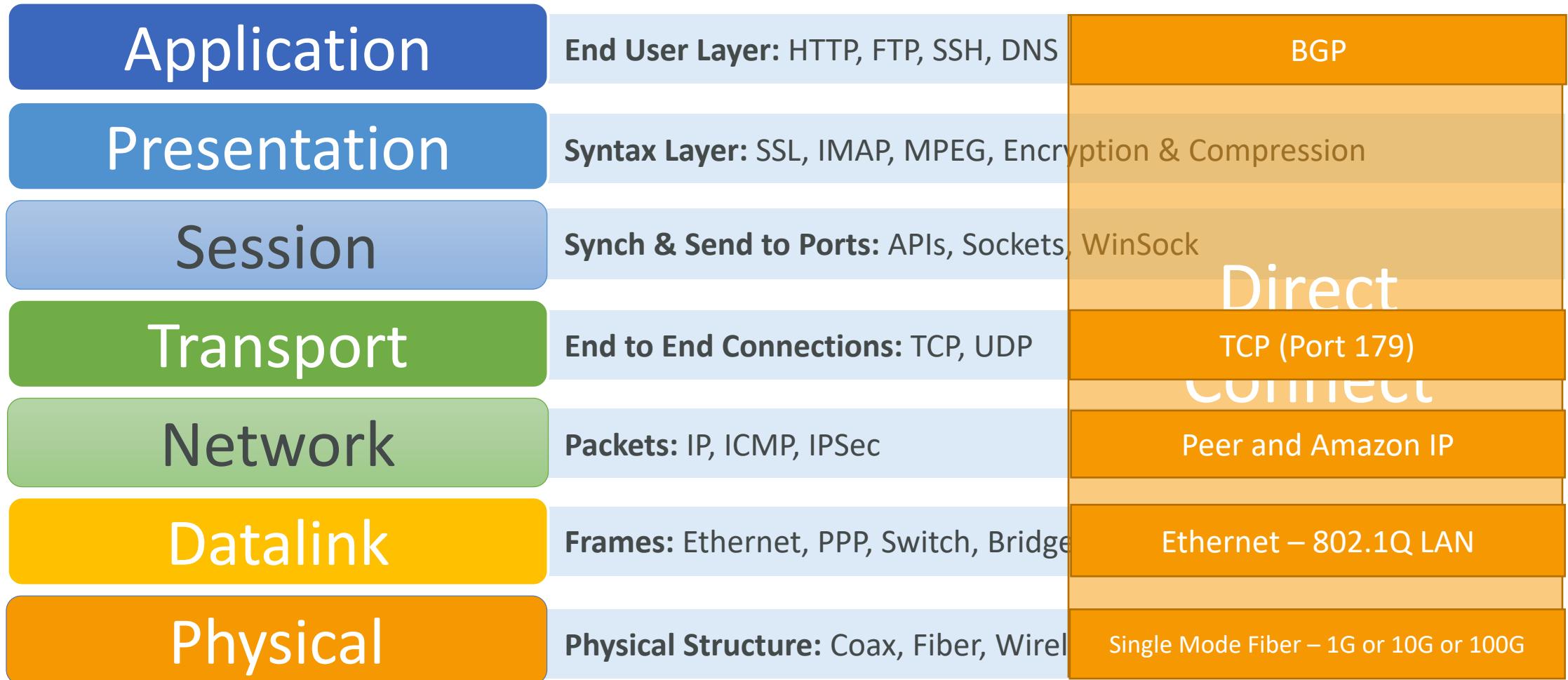
- Add IGW to your VPC
- Modify Target Subnet Route Table
- Modify Client VPN endpoint Authorization Rule
- Modify Client VPN endpoint routes



# AWS Direct Connect (DX)



# Direct Connect & OSI



# What is Direct Connect?

# AWS Direct Connect

- A dedicated network connection from on-premises to AWS
- AWS <-> DirectConnect Location <-> On-premises Data Center
- Low latency and consistent bandwidth
- Lower data transfer cost
- Access AWS Private Network (VPC) and AWS public services endpoints (e.g S3, DynamoDB)

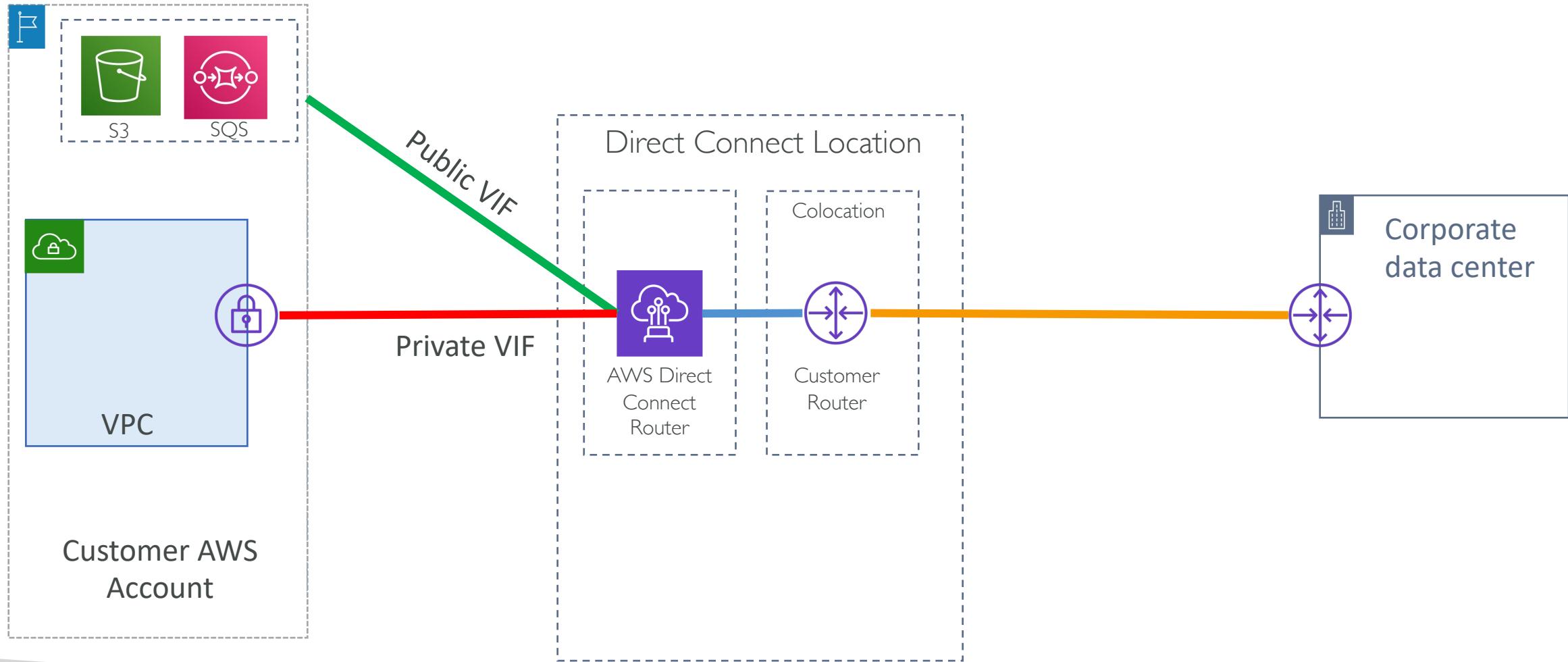




# AWS Direct Connect

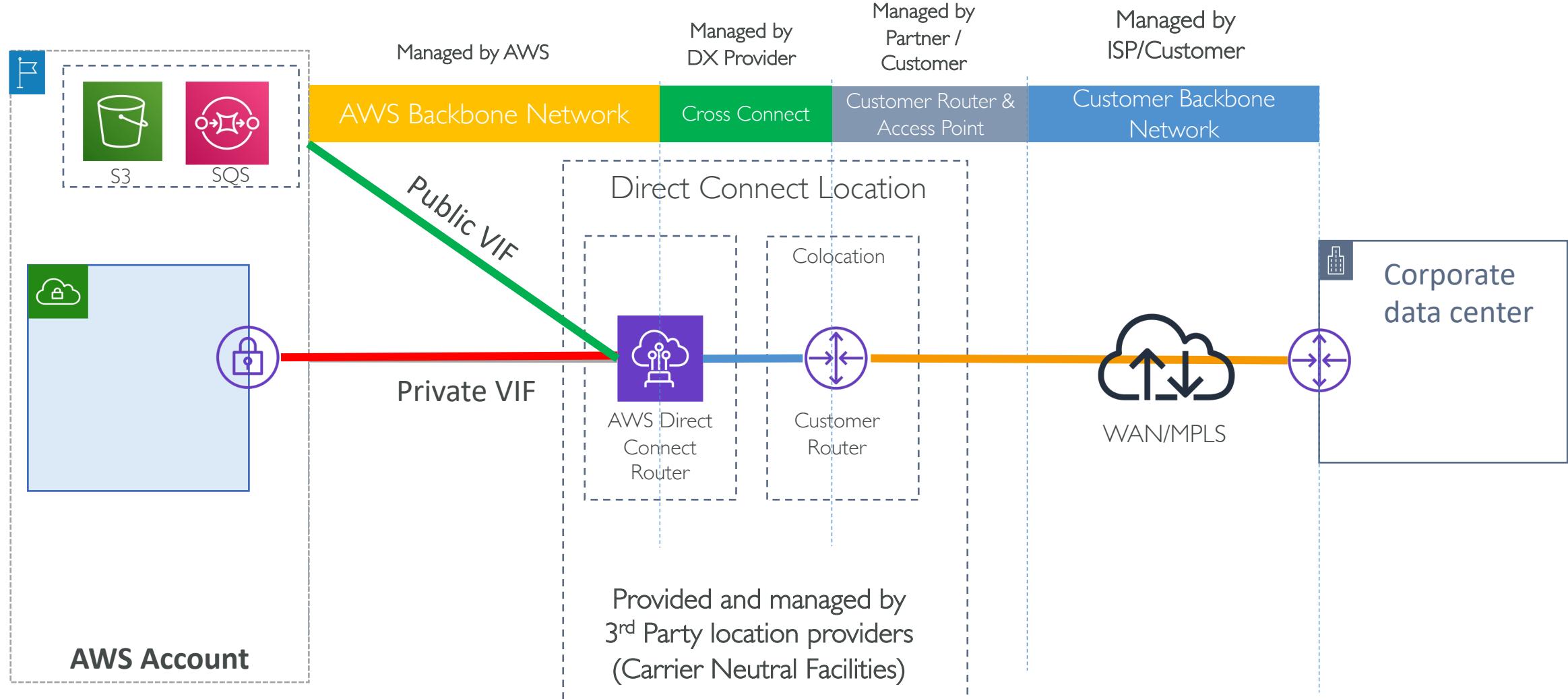
- Leverages AWS Global network backbone
- DX locations are provided by authoritative 3<sup>rd</sup> party providers.
- 108 DX locations across the world across 25 AWS regions (as of today)
- End to end provisioning time of 4-12 weeks
- Get bandwidth of 1, 10 or 100 Gbps with Dedicated connection
- Get sub-1 Gbps bandwidth (50/100/200/400/500 Mbps, 1,10 Gbps) by leveraging AWS Direct Connect APN partner

# AWS Direct Connect



# Direct Connect components

# Direct Connect components



# DX locations

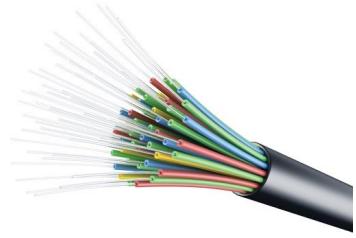
| Asia Pacific  | Canada                               | China | EU and Africa | Middle East | South America         | United States                                   |  |
|---|--------------------------------------|-------|---------------|-------------|-----------------------|---|--|
| AWS Direct Connect Location                         | Campus Location also accessible from |       |               |             | Associated AWS Region | Location-specific features                      |  |
| Cologix COL2, Columbus,<br>OH****                   |                                      |       |               |             | US East (Ohio)        | 100 Gbps available<br>100 Gbps MACsec supported |  |
| Cologix MIN3, Minneapolis, MN                       |                                      |       |               |             | US East (Ohio)        |   |  |
| CyrusOne West III, Houston, TX                      | CyrusOne West I - III, Houston       |       |               |             | US East (Ohio)        | 100 Gbps available                              |  |
| Equinix CH2, Chicago, IL                            | Equinix CH1 - CH2 & CH4, Chicago     |       |               |             | US East (Ohio)        |   |  |
| Netrality Properties 1102<br>Grand, Kansas City, MO |                                      |       |               |             | US East (Ohio)        |   |  |
| QTS, Chicago, IL                                    |                                      |       |               |             | US East (Ohio)        |   |  |
| 165 Halsey Street, Newark, NJ                       |                                      |       |               |             | US East (Virginia)    |   |  |

<https://aws.amazon.com/directconnect/locations/>

# DX network requirement, basic terminologies and concepts

# Direct Connect Network Requirements

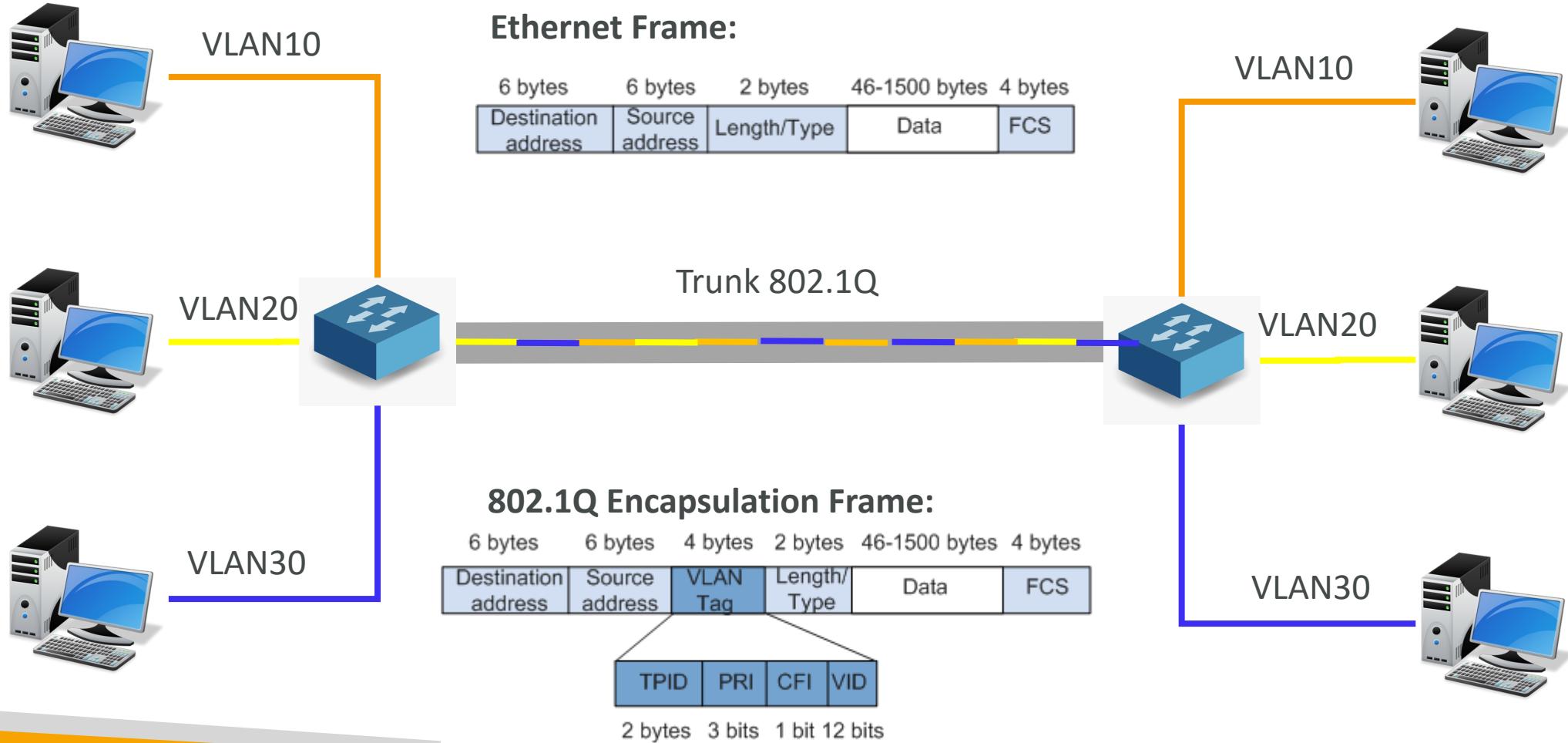
- Single-mode fiber
  - 1000BASE-LX (1310 nm) transceiver for 1 G
  - 10GBASE-LR (1310 nm) transceiver for 10 G
  - 100GBASE-LR4 for 100 G
- 802.1Q VLAN encapsulation must be supported
- Auto-negotiation for the port must be disabled
- Port speed and **full-duplex mode** must be configured manually
- End Customer Router (on-premises) must support **Border Gateway Protocol** (BGP) and BGP MD5 authentication
- (optional) **Bidirectional Forwarding Detection**



# Let's understand these terminologies & concepts

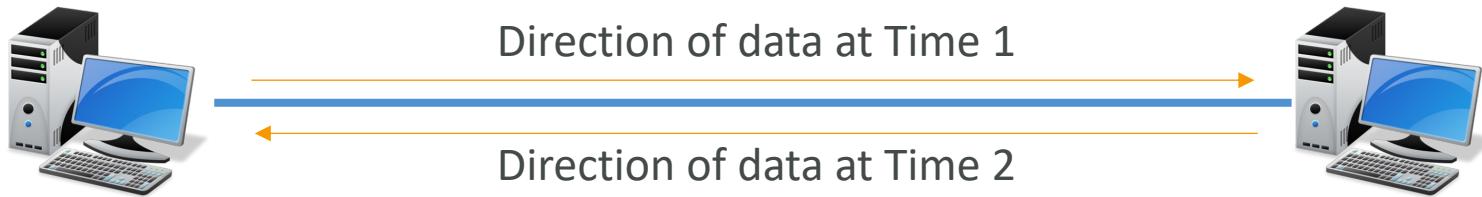
- 802.1Q Layer 2 VLAN
- Half duplex and full duplex traffic
- Auto-negotiation of ports
- ASN – Autonomous System Number (Public and Private)
- Border Gateway Protocol (BGP)
- Bidirectional Forwarding Detection (BFD)
- AWS IP Ranges (ip-ranges.json)

# 802.1Q VLAN

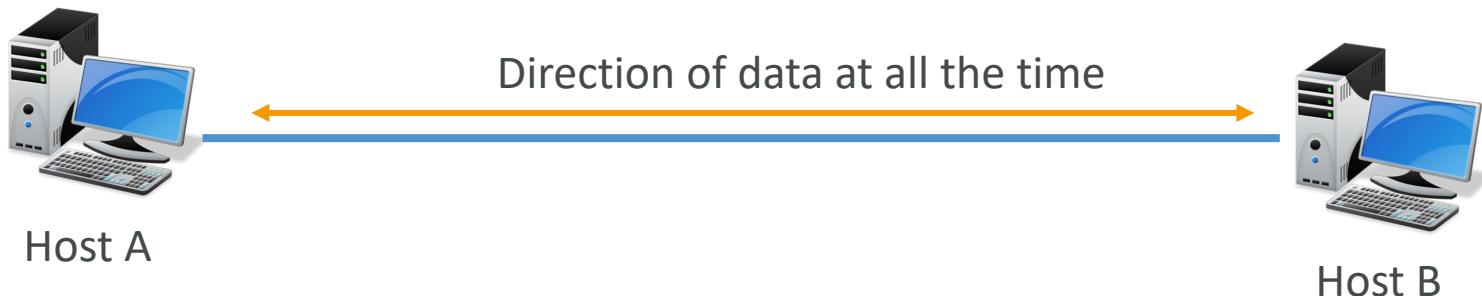


# Half Duplex and Full Duplex systems

## Half Duplex

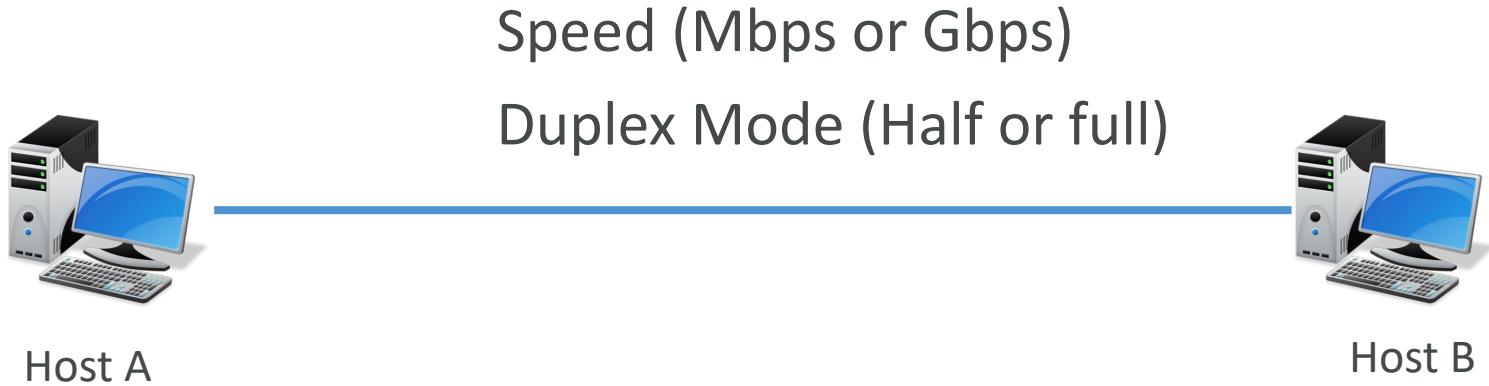


## Full Duplex



# Auto-negotiation of the ports

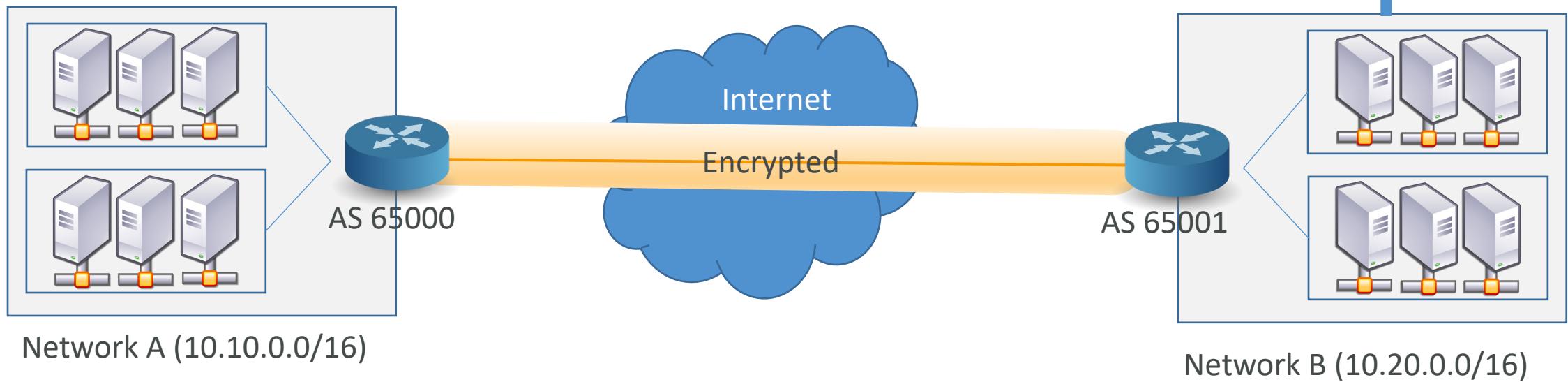
- Auto-negotiation is the ability of a network interface to automatically coordinate its own connection parameters (speed and duplex) with another network interface.
- Auto-negotiation must be enabled at both the ends for auto-negotiation of the ports to work



- For DX the Auto-negotiation for the port must be disabled.
- Port speed and full-duplex mode must be configured manually.

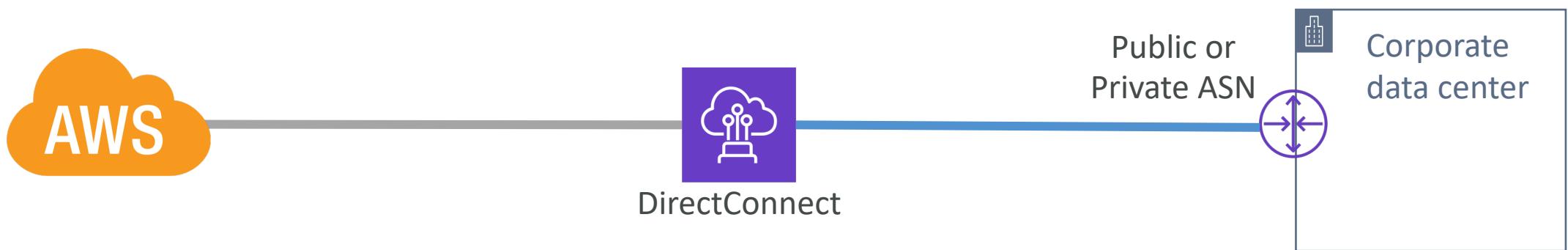
# AS – Autonomous Systems

- Routers controlled by one entity in a network
- ASN – Autonomous System Number



# AS – Autonomous Systems

- Public and Private ASN
  - Public ASNs are assigned by IANA via Regional Internet Registries (RIRs)
  - You need to use the Public ASN if you are exchanging routes over a public internet
  - You can use private ASN for **private** connection between two AS
- 2 Byte (16 bit) ASN – 0-65535
  - Public ASN is assigned by IANA (0-65525)
  - Private ASN (64512-65534)
- 4 Byte (32 bit) ASN - 0 to 4294967295
  - AWS allows 1-2147483647



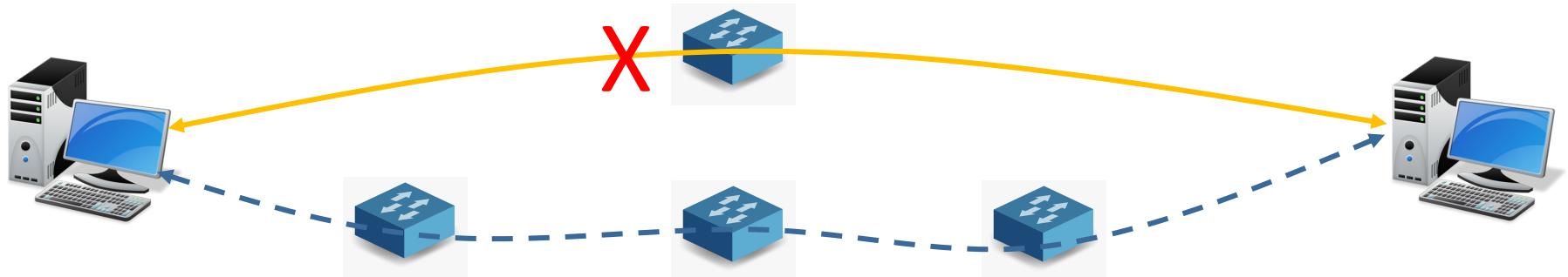
# BGP – Border Gateway Protocol

- Dynamic Routing using **Path-Vector** protocol where it exchanges the best path to a destination between peers or AS
- TCP based protocol on port 179
- iBGP – Routing within AS
- eBGP – Routing between AS's
- Routing decision is influenced by
  - Weight (Cisco routers specific – works within AS)
  - ASPATH – Series of AS's to traverse the path (Works between AS)
  - Local Preference LOCAL\_PREF (Works within AS)
  - MED – Multi-Exit Discriminator (Works between AS)

The current version of BGP is version 4 (BGP4), which was published as [RFC 4271](#) in 2006

# BFD – Bidirectional Forwarding Detection

- It's a simple Hello Network Protocol
- Lowers the network failure detection time between the neighboring peers
- BFD control-packets are transmitted and received periodically between the BFD peers (Asynchronous mode)
- Neighbors can use dynamic routing protocol or static routes (BGP/OSPF)
- Provides the failure detection in less than 1 sec



# AWS IP Ranges - ip-ranges.json

- AWS Publishes all its **current** public IP addresses in JSON format
- Download json: <https://ip-ranges.amazonaws.com/ip-ranges.json>
- You can filter the IP addresses by Region, AWS Service, IPv4 or IPv6
- Useful when you want to restrict access to only AWS services public IPs
- Get notified by subscribing to AWS SNS topic in N.Virginia region
  - Topic Name: AmazonIpSpaceChanged
  - Topic ARN: arn:aws:sns:us-east-1:806199016981:AmazonIpSpaceChanged

# AWS IP Ranges - ip-ranges.json

```
{  
  "ip_prefix": "50.16.0.0/15",  
  "region": "us-east-1",  
  "network_border_group": "us-east-1",  
  "service": "AMAZON"  
},  
{  
  "ip_prefix": "50.19.0.0/16",  
  "region": "us-east-1",  
  "network_border_group": "us-east-1",  
  "service": "AMAZON"  
},  
...
```

```
$ jq -r '.prefixes[] | select(.service=="CODEBUILD") | .ip_prefix' < ip-ranges.json  
52.47.73.72/29  
13.55.255.216/29  
52.15.247.208/29  
...
```

**AWS Services:** AMAZON | AMAZON\_APPFLOW | AMAZON\_CONNECT | API\_GATEWAY | CHIME\_MEETINGS | CHIME\_VOICECONNECTOR | CLOUD9 | CLOUDFRONT | CODEBUILD | DYNAMODB | EBS | EC2 | EC2\_INSTANCE\_CONNECT | GLOBALACCELERATOR | KINESIS\_VIDEO\_STREAMS | ROUTE53 | ROUTE53\_HEALTHCHECKS | S3 | WORKSPACES\_GATEWAYS

# Let's understand these terminologies & concepts

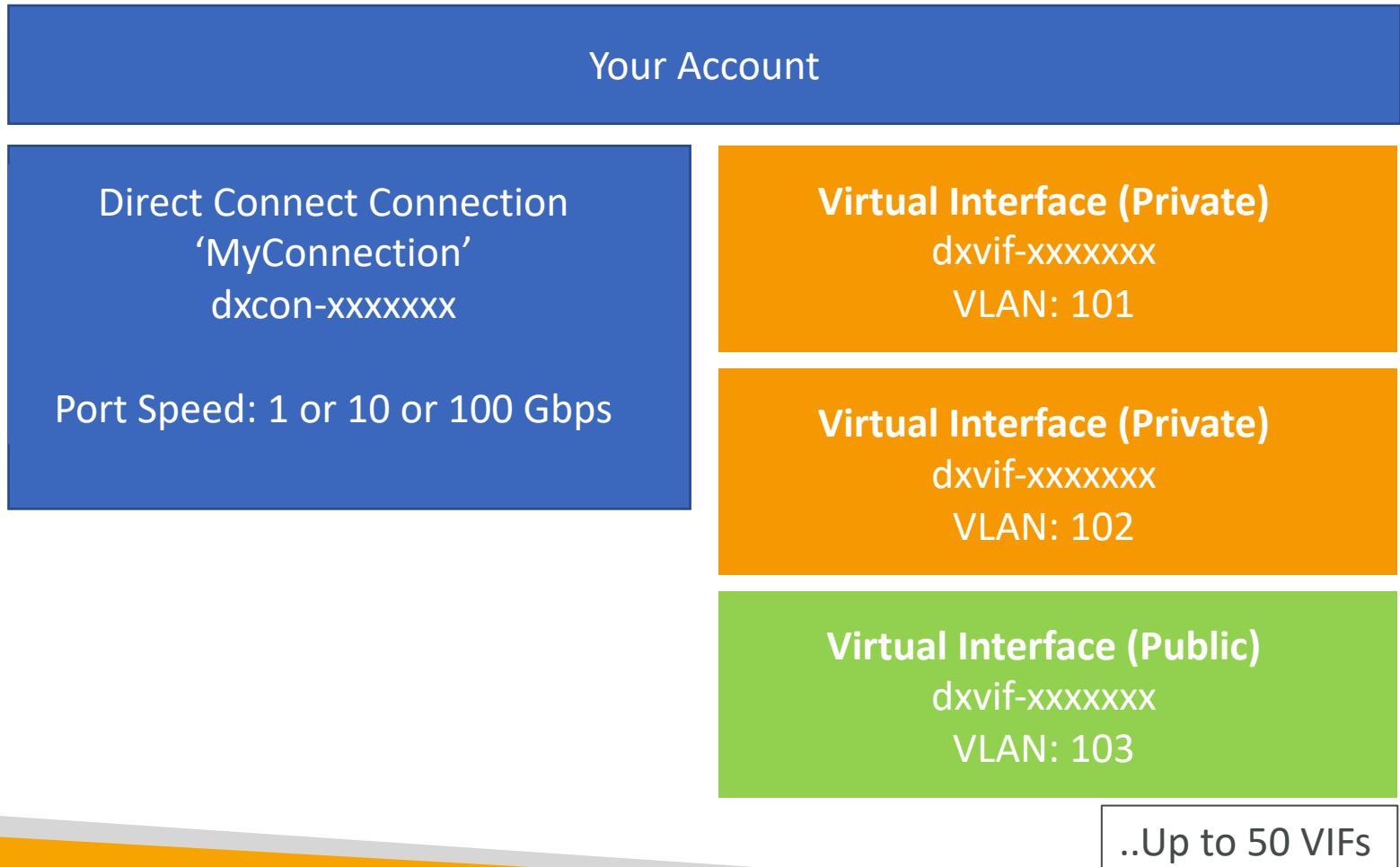
- 802.1Q Layer 2 VLAN
- Half duplex and full duplex traffic
- Auto-negotiation of ports
- ASN – Autonomous System Number (Public and Private)
- Border Gateway Protocol (BGP)
- Bidirectional Forwarding Detection (BFD)
- AWS IP Ranges (ip-ranges.json)

# DX Connection Types

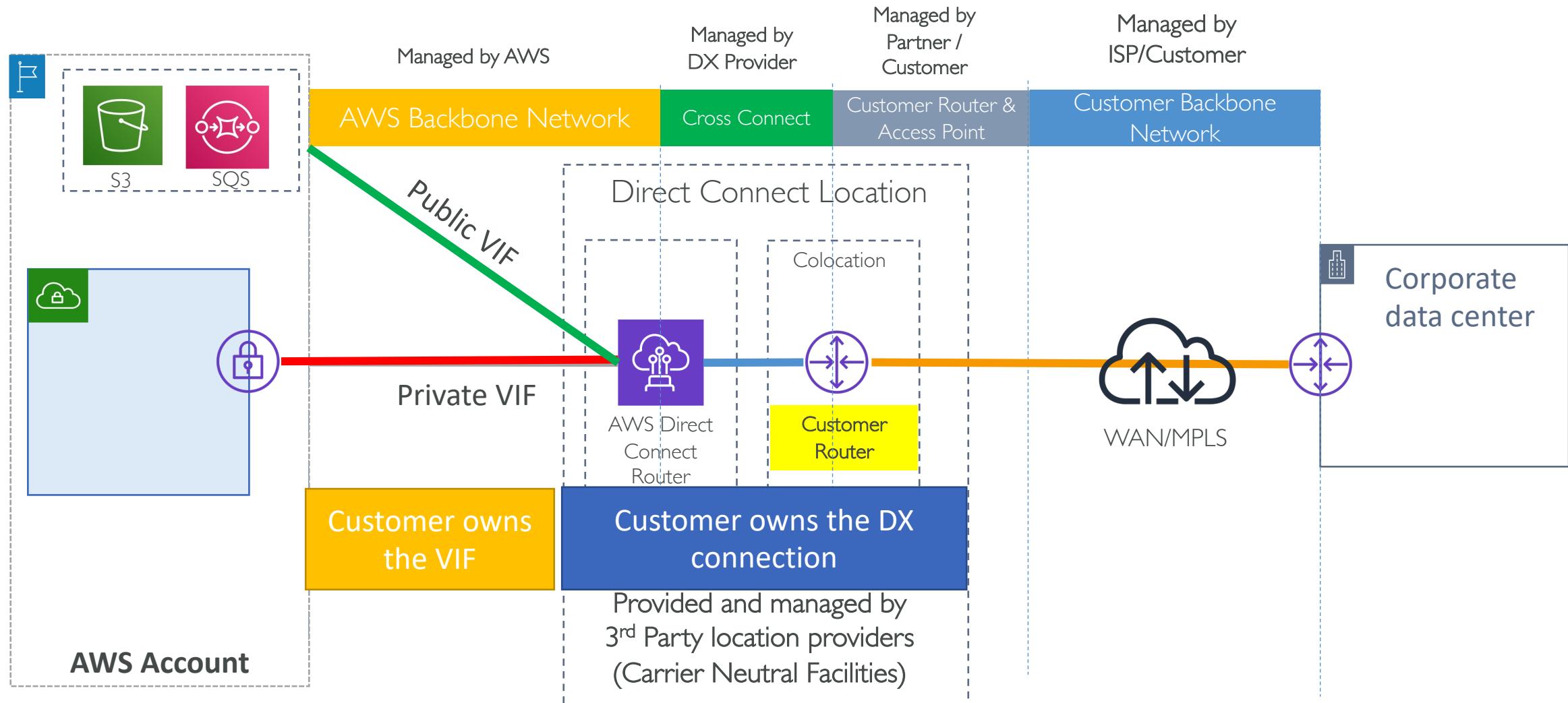
# Direct Connect – Connection Types

- **Dedicated Connections:** 1 Gbps, 10 Gbps and 100 Gbps capacity
  - Physical ethernet port dedicated to a customer
  - Request made to AWS first, then completed by AWS Direct Connect Partners
  - Can be either setup by your Network Provider or AWS Direct Connect Partner
- **Hosted Connections:**
  - 50, 100, 200, 300, 400, 500 Mbps and 1 Gbps, 2 Gbps, 5 Gbps, 10 Gbps
  - Connection requests are made via AWS Direct Connect Partners
  - 1, 2, 5, 10 Gbps available at select AWS Direct Connect Partners
  - AWS uses traffic policing on hosted connections – excess traffic is dropped

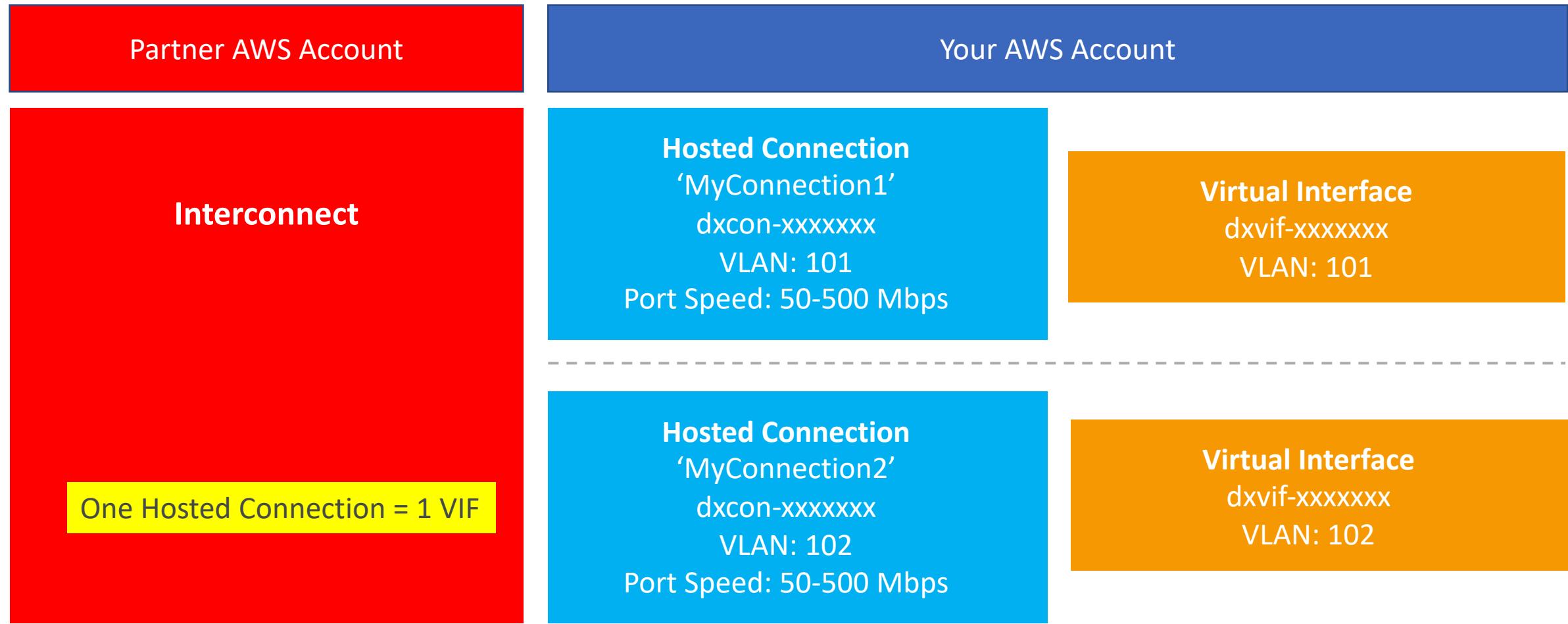
# Dedicated connection – AWS account



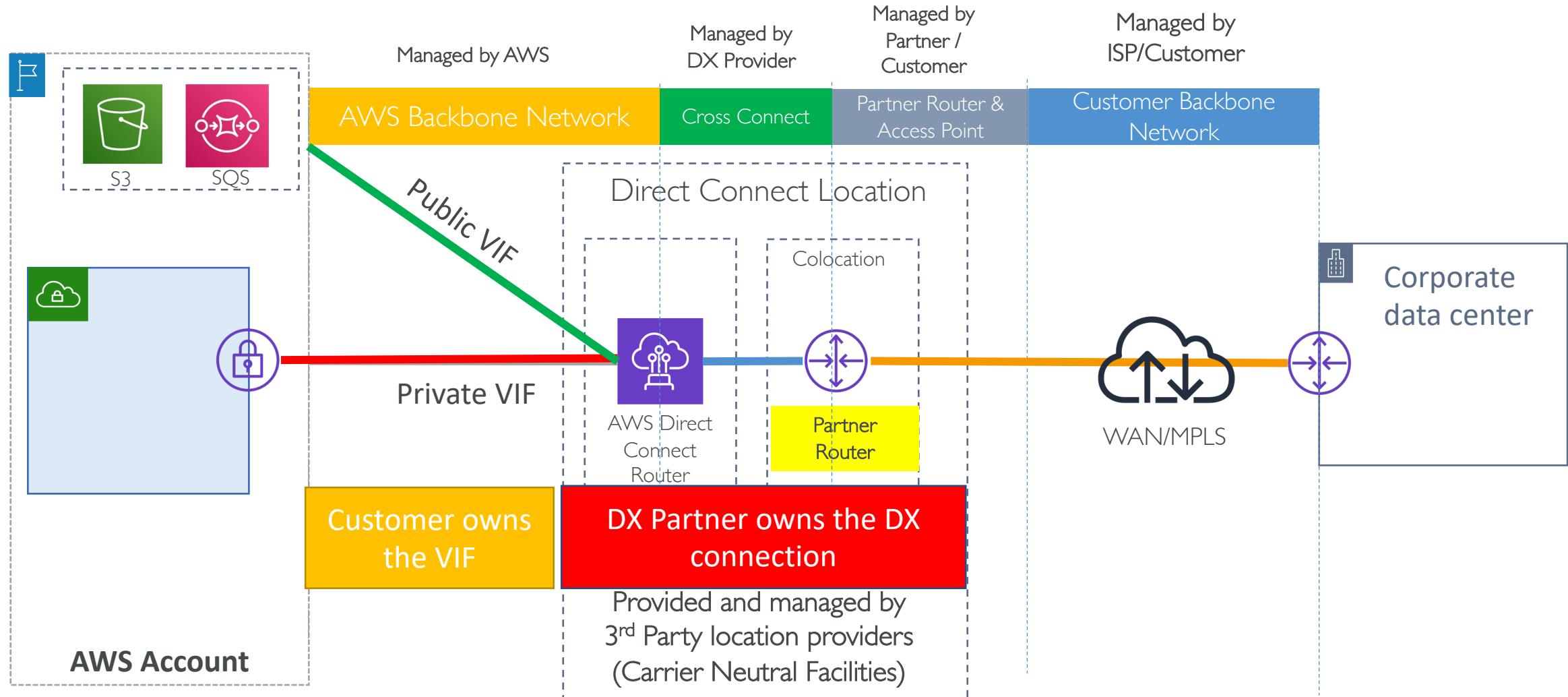
# Dedicated connection



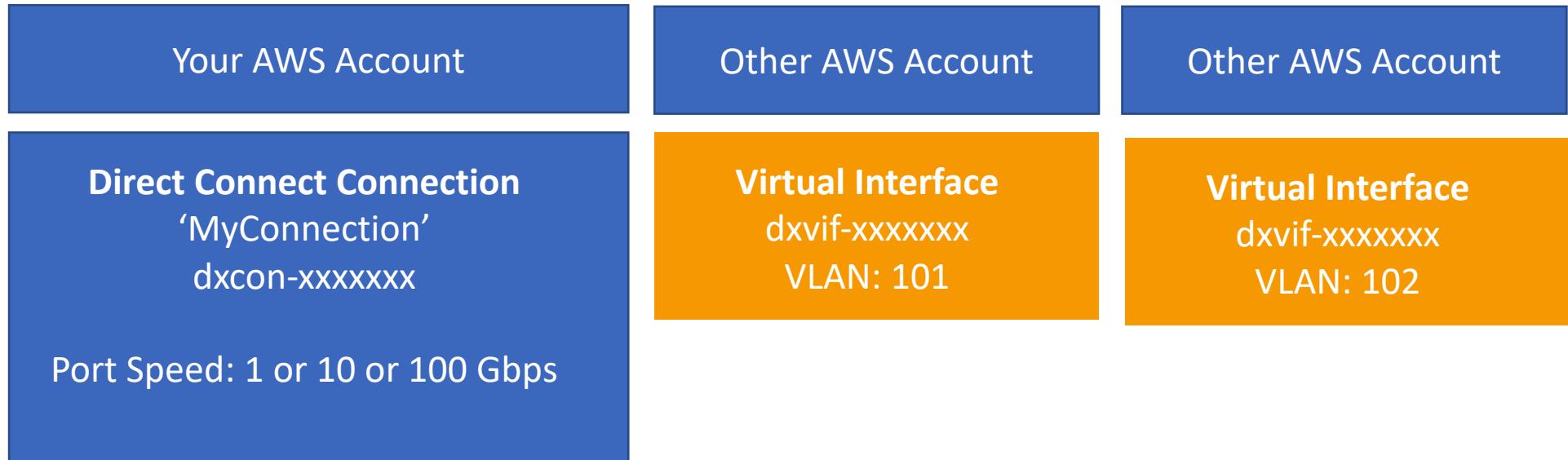
# Hosted connection – AWS account



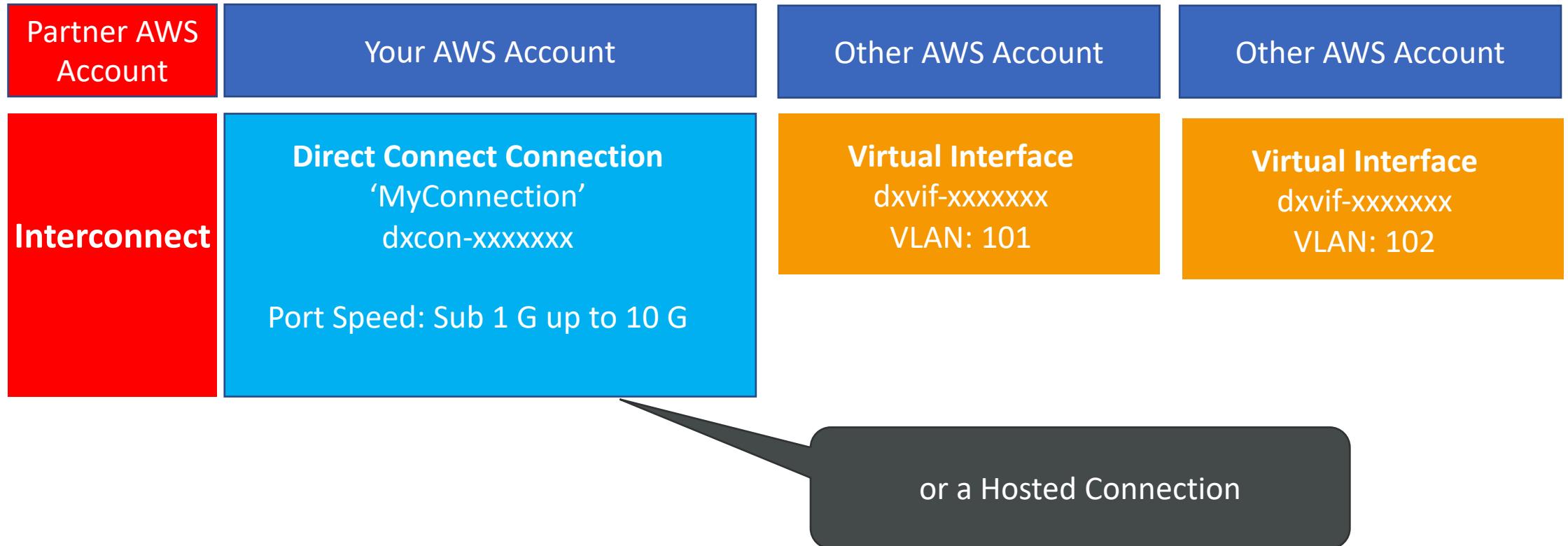
# Hosted connection



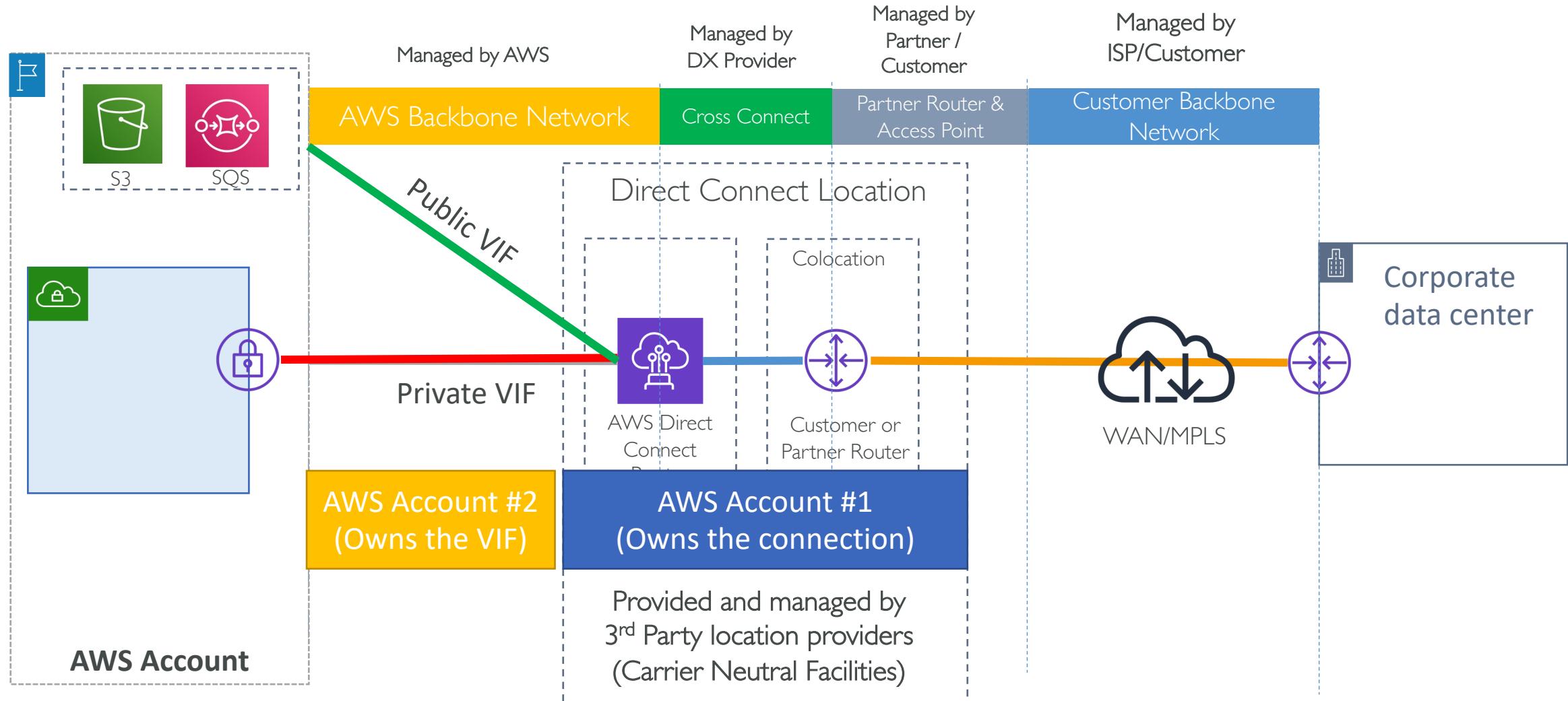
# HostedVIF – AWS account



# HostedVIF – AWS account



# Hosted VIF

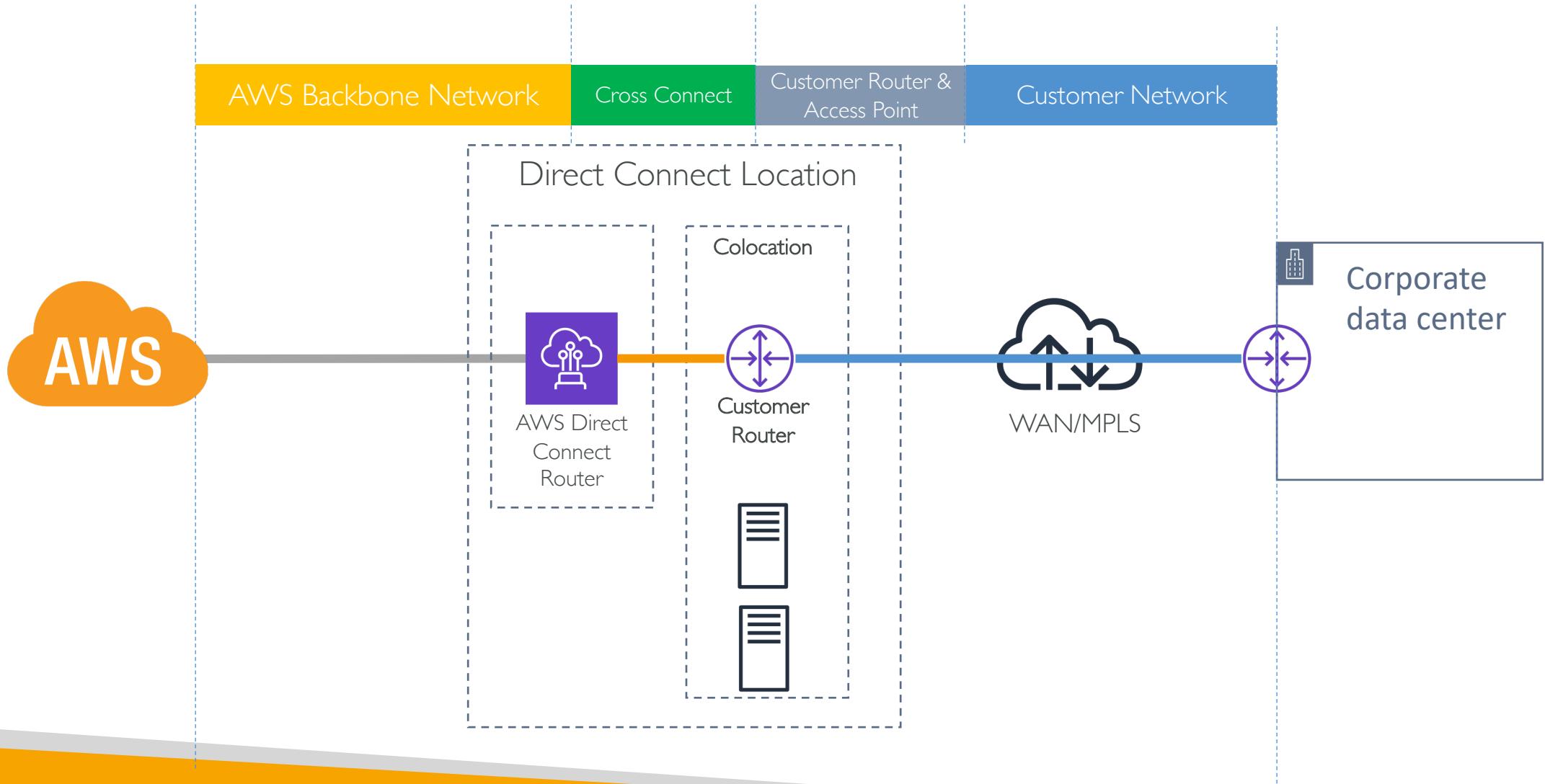


# Steps to setup Direct Connect connections

# Direct Connect – Connection Types

- **Dedicated Connections:** 1 Gbps, 10 Gbps and 100 Gbps capacity
  - Physical ethernet port dedicated to a customer
  - Request made to AWS first, then completed by AWS Direct Connect Partners
  - Can be either setup by your Network Provider or AWS Direct Connect Partner
- **Hosted Connections:**
  - 50, 100, 200, 300, 400, 500 Mbps and 1 Gbps, 2 Gbps, 5 Gbps, 10 Gbps
  - Connection requests are made via AWS Direct Connect Partners
  - 1, 2, 5, 10 Gbps available at select AWS Direct Connect Partners

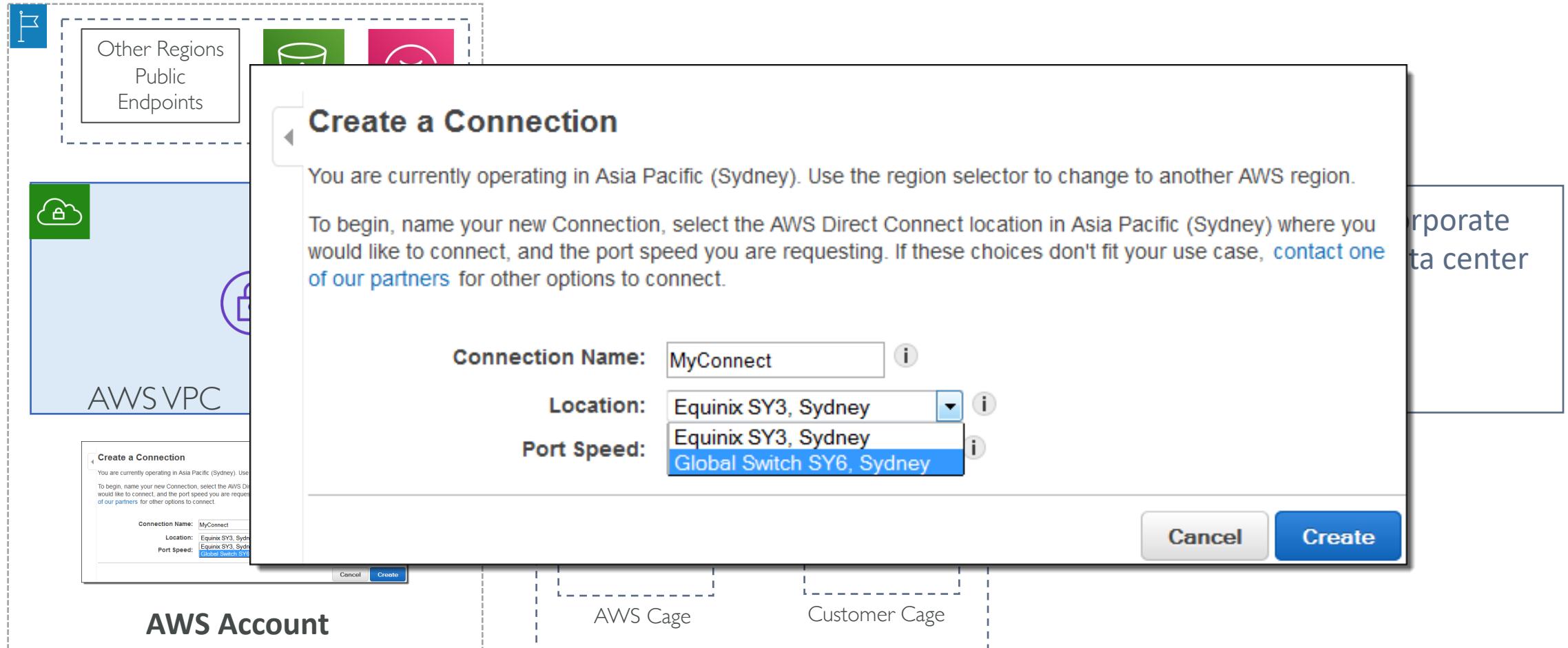
# Dedicated Connection



# Dedicated connection – Sequence of events

1. You select the AWS region, DX location and submit the connection request via AWS console or CLI or API
2. AWS provisions your port within 72 hrs and provide you with the LOA-CFA (Letter of Authorization – Connection facility assignment)
3. LOA contains the demarcation details of assigned port within the facility
4. If your organization has physical presence in DX location, then you can request for cross-connect within the facility to connect to AWS device
5. If not, you provide the copy of LOA to DX APN partner and partner places the order for cross-connect
6. After connection is up, you receive Tx/Rx optical signal at your equipment
7. Now, you can create Private or Public Virtual Interfaces to connect to your VPC or public AWS services

# Steps to setup DX Dedicated connection



# Steps to setup DX Dedicated connection

Other Regions  
Public  
Endpoints

S3 SOS

Direct Connect Location

### Letter of Authorization and Connecting Facility Assignment

AWS

Create a Connection

To begin, name your new connection. You can choose from our partners for other options.

Connect to:

Region: US East (N. Virginia) Customer Cage: Customer Cage

Customer IP Range: 10.0.0.0/16 Customer Subnet: 10.0.0.0/24

Provider IP Range: 10.0.0.0/16 Provider Subnet: 10.0.0.0/24

Router IP Range: 10.0.0.0/16 Router Subnet: 10.0.0.0/24

Issue Date: March 16, 2013

Requested By: [REDACTED]

Issued By: VADATA, INC.

Issued To: IBX - Equinix DC2

Facility - Cage Number: Equinix DC2 - 2030

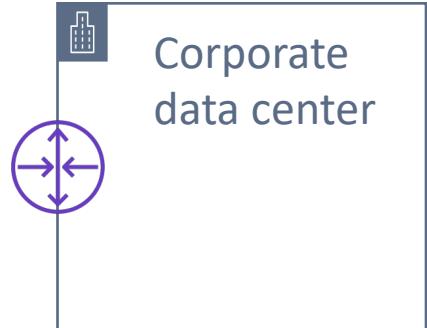
AWS Direct Connection ID: dxcon-fglbz3ny

Rack, Patch Panel, Port Number: Rack: 211  
Patch Panel: CP:0211:104714  
Strands: 13/14

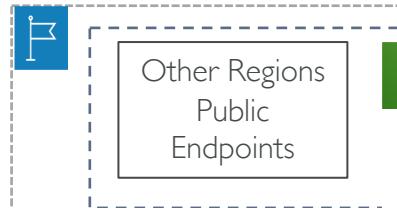
Cable Type: Single Mode Fiber

Access Ticket Number\*\*: 0020398879

Aws Account: [REDACTED]



# Steps to setup DX Dedicated connection



**Asia Pacific (Mumbai)**



Create a Connection

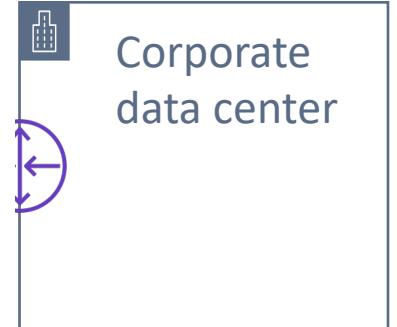
You are currently operating in Asia Pacific (Sydney). Use the r  
To begin, name your new Connection, select the AWS Direct C  
would like to connect, and the port speed you are requesting.  
of our partners for other options to connect.

|                  |                     |
|------------------|---------------------|
| Connection Name: | MyConnect           |
| Location:        | Equinix SY3, Sydney |
| Port Speed:      | 1 Gbps              |

AWS Acc...



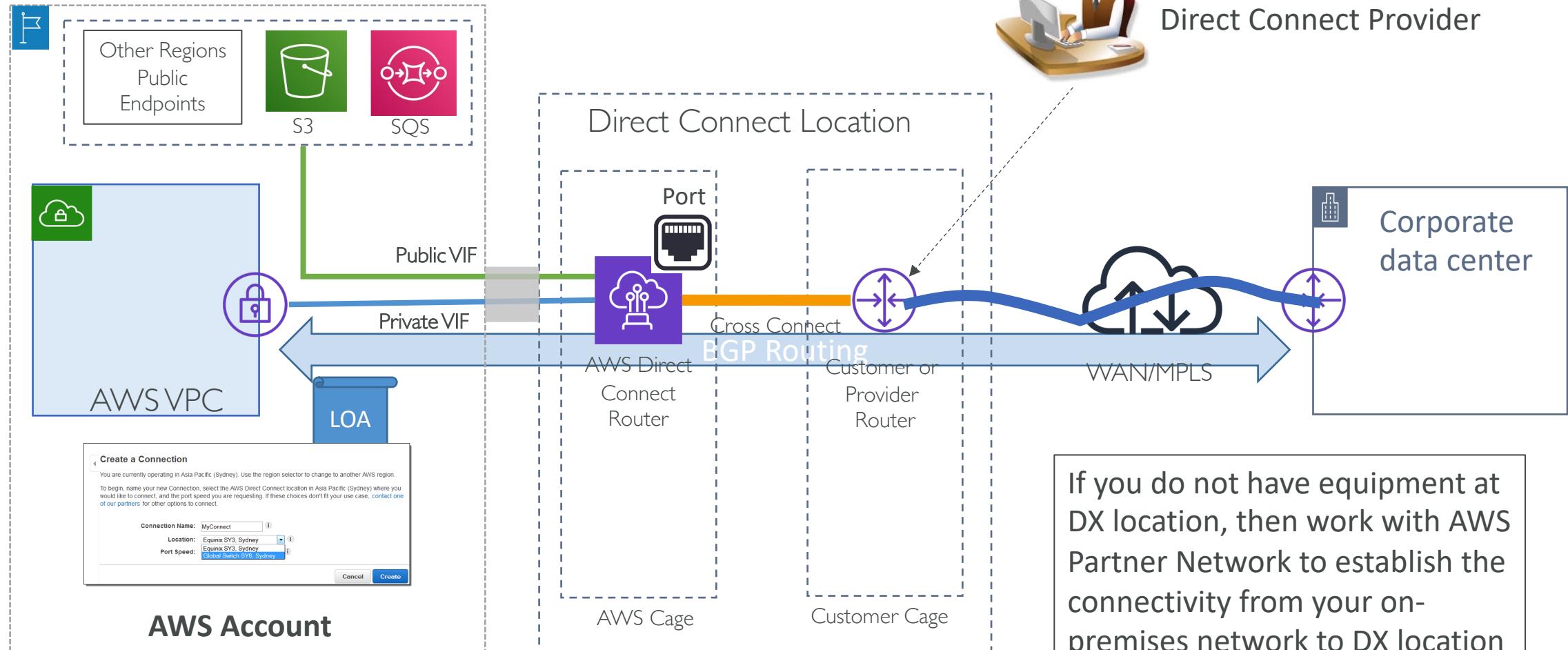
Direct Connect Provider



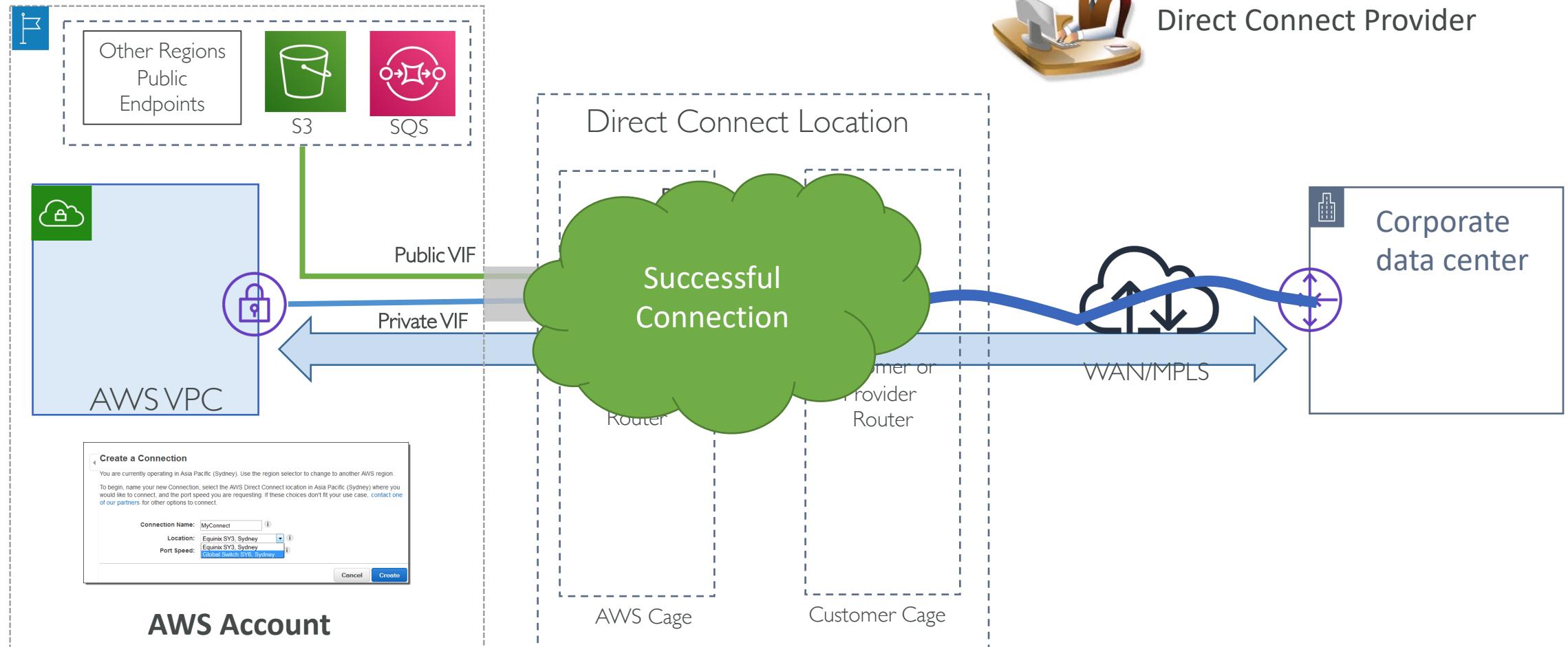
## How to request a connection

| Location                       | How to request a connection  |
|--------------------------------|--|
| GPX, Mumbai                    | Contact GPX at <a href="mailto:nkankane@gpxglobal.net">nkankane@gpxglobal.net</a> .  |
| NetMagic DC2, Bangalore        | Contact NetMagic Sales and Marketing toll-free at 18001033130 or at <a href="mailto:marketing@netmagsolutions.com">marketing@netmagsolutions.com</a> . |
| Sify Rabale, Mumbai            | Contact Sify at <a href="mailto:aws.directconnect@sifycorp.com">aws.directconnect@sifycorp.com</a> .   |
| STT Delhi DC2, Delhi           | Contact STT at <a href="mailto:enquiry.AWSIDX@sttelemediagdc.in">enquiry.AWSIDX@sttelemediagdc.in</a> .  |
| STT GDC Pvt. Ltd. VSB, Chennai | Contact STT at <a href="mailto:enquiry.AWSIDX@sttelemediagdc.in">enquiry.AWSIDX@sttelemediagdc.in</a> .  |
| STT Hyderabad DC1, Hyderabad   | Contact STT at <a href="mailto:enquiry.AWSIDX@sttelemediagdc.in">enquiry.AWSIDX@sttelemediagdc.in</a> .  |

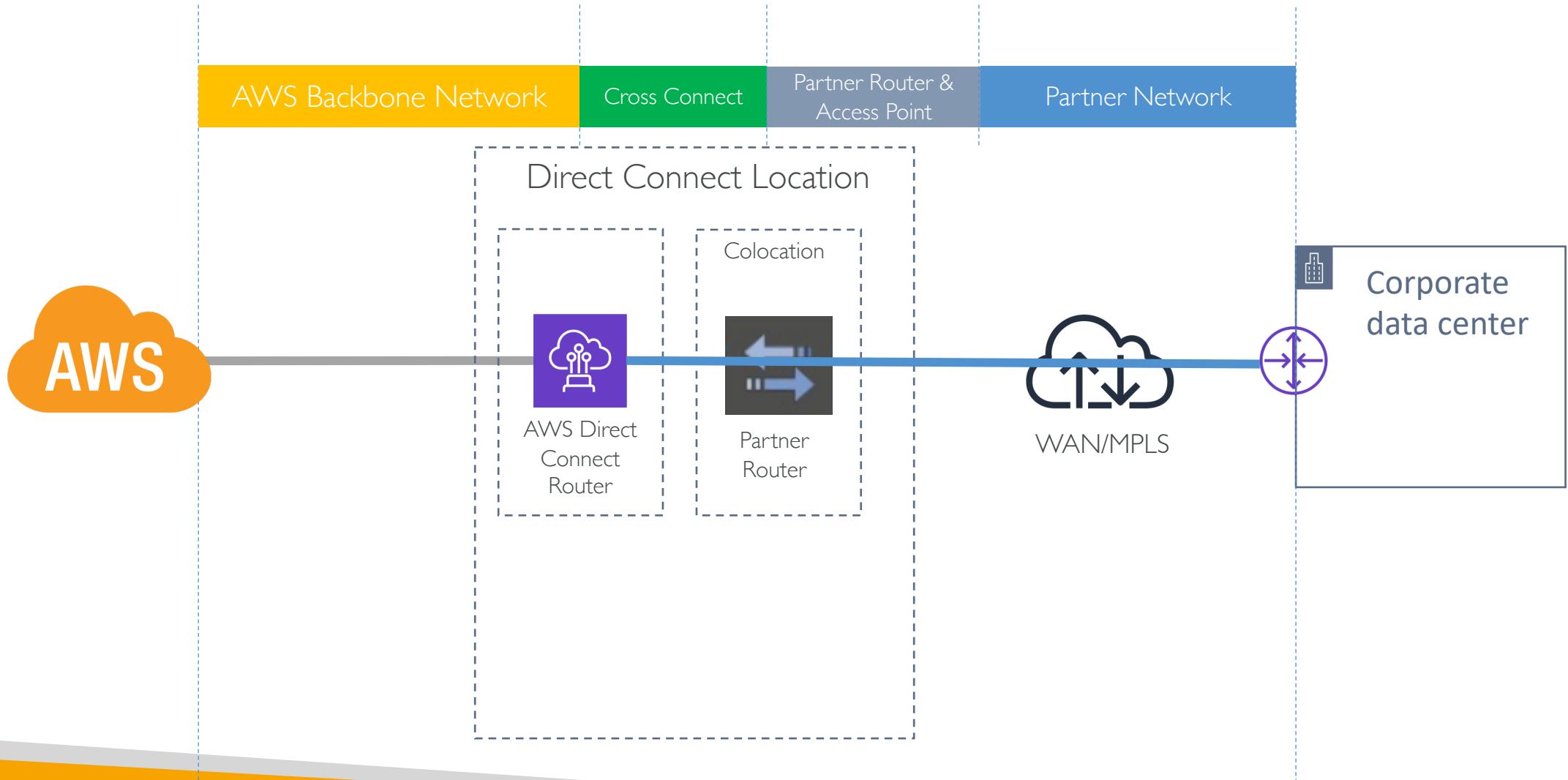
# Steps to setup DX Dedicated connection



# Steps to setup DX Dedicated connection



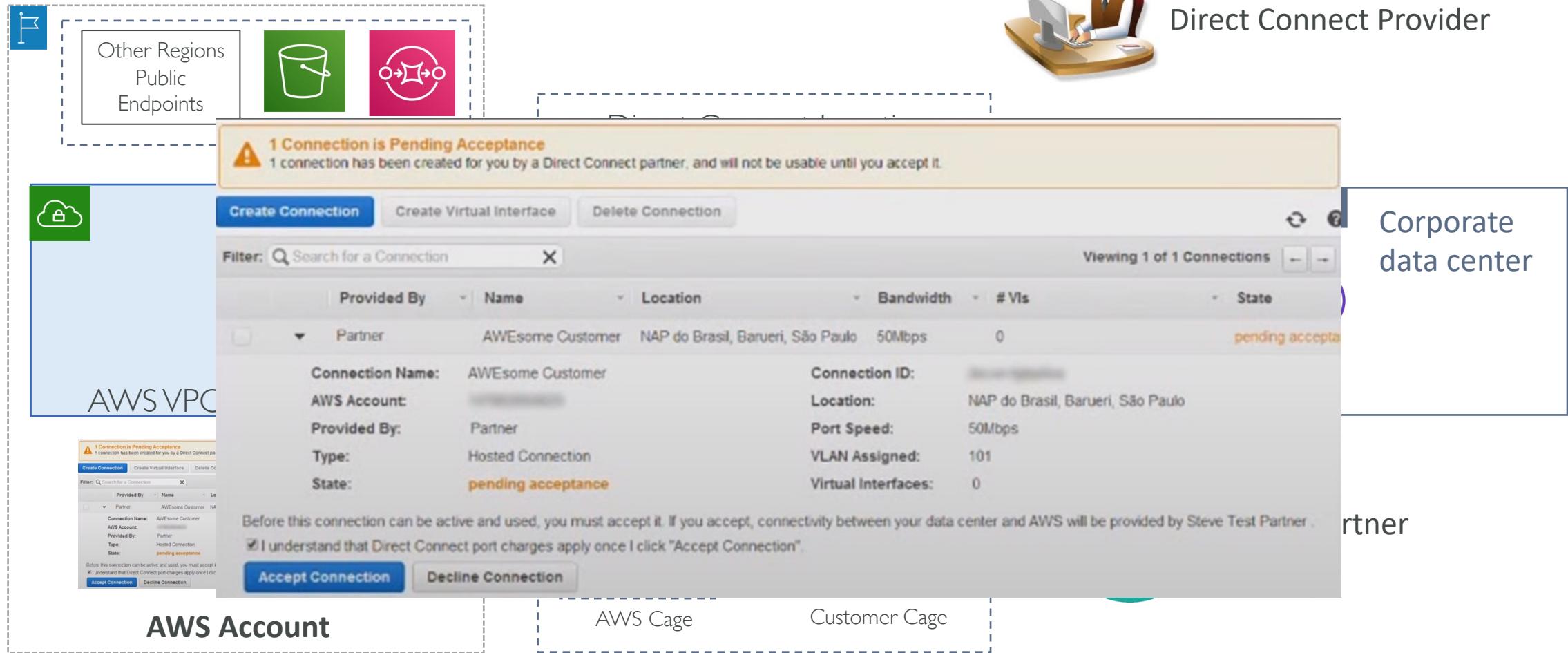
# Hosted Connection



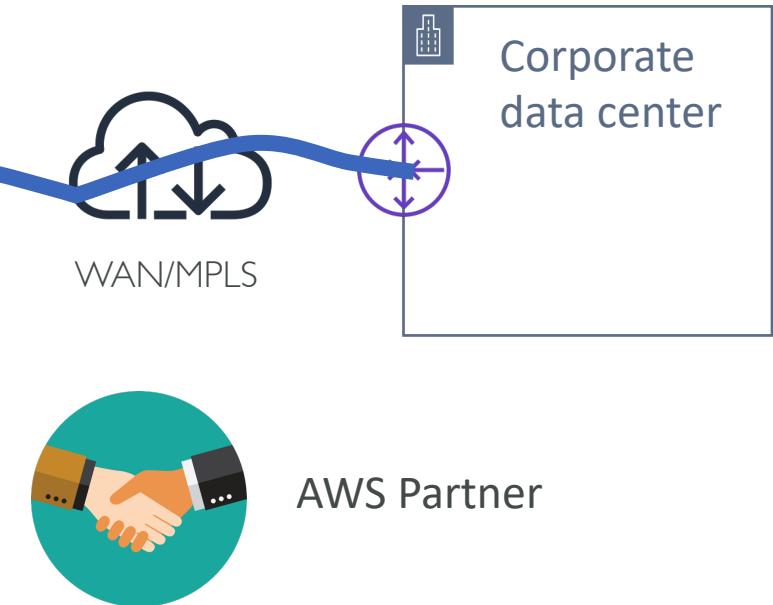
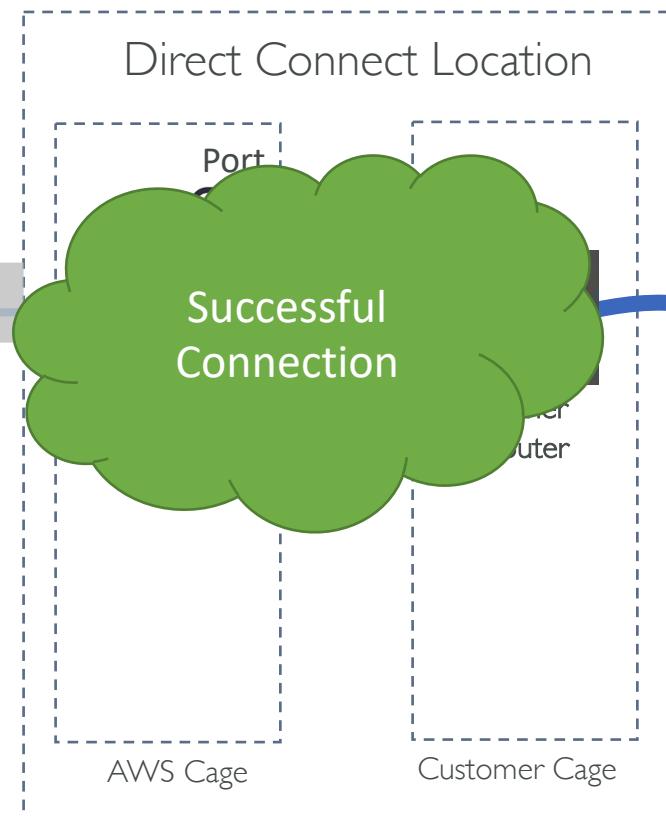
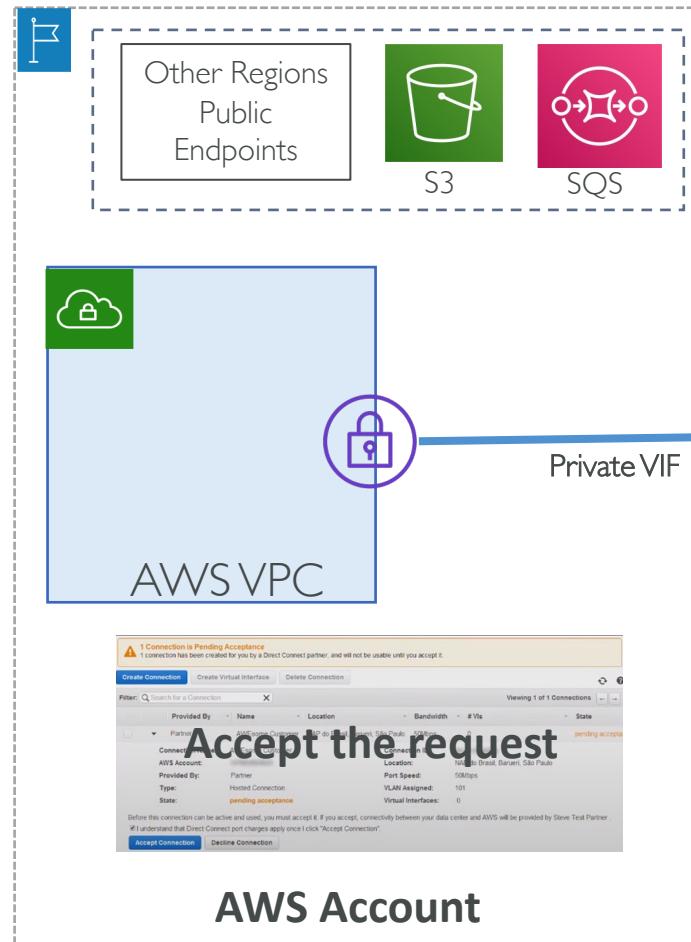
# Hosted connection – Sequence of events

- To order Hosted connection, you don't need to get LOA. You can directly contact a **Direct Connect Partner** to order the connection
- You provide your 12-digit AWS account number to the partner
- Partner will setup the hosted connection and this connection will be available in your account (in the given region) to accept it
- Once you accept the connection, it enables the billing for associated port hours and data transfer charges

# Steps to setup DX Hosted connection



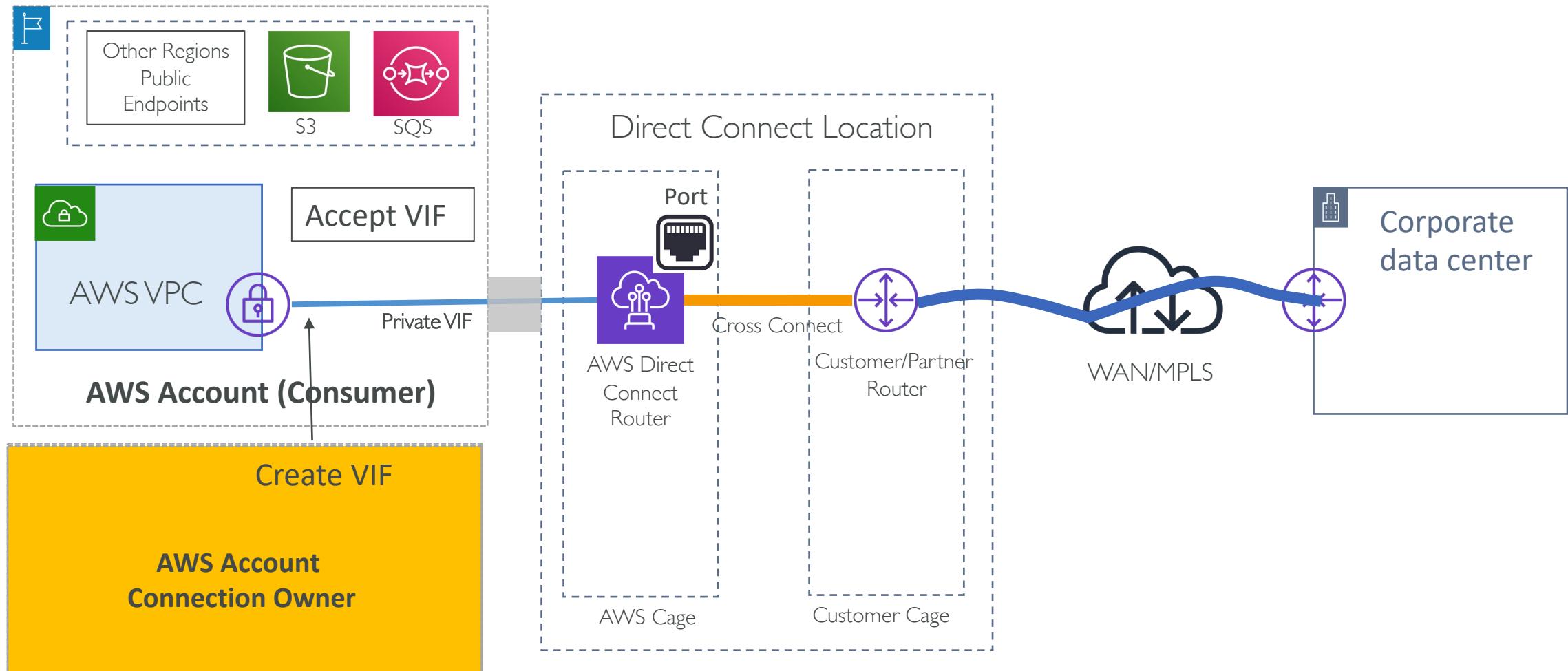
# Steps to setup DX Hosted connection



# Hosted VIF

- Don't get confused with Hosted Connection
- When creating a VIF you may choose "Another AWS account"
- You are still the owner of the Direct connect Connection however the VIF is created in another AWS account who must accept it
- For Private VIF, the other account should also associate it with VGW
- A connection of less than 1 Gbps supports only one virtual interface.
- Typically used in a scenario where centralized network team manages the DX connection and provisions VIFs for business accounts

# Steps to create Hosted VIF



# DX Connection creation steps walkthrough

# Demo

The screenshot shows the AWS Direct Connect service interface. The top navigation bar includes the AWS logo, a 'Services' dropdown, a search bar with placeholder text 'Search for services, features, marketplace products, and docs' and a keyboard shortcut '[Alt+S]', and a user icon.

The left sidebar, titled 'Direct Connect', has a 'Connections' section selected, showing options like Virtual interfaces, LAGs, Direct Connect gateways, Virtual private gateways, and Transit gateways.

The main content area is titled 'Create connection'. It displays a brief introduction: 'AWS Direct Connect enables you to establish a dedicated network connection between your network and one of the AWS Direct Connect locations. Select the AWS Direct Connect location where you would like to connect, and the port speed you are requesting.' followed by a 'Learn more' link.

The first step of the wizard is 'Connection ordering type'. It offers two options: 'Classic' (selected) and 'Connection wizard'. The 'Classic' option describes creating connections one at a time, while the 'Connection wizard' option describes using resiliency recommendations for new setups.

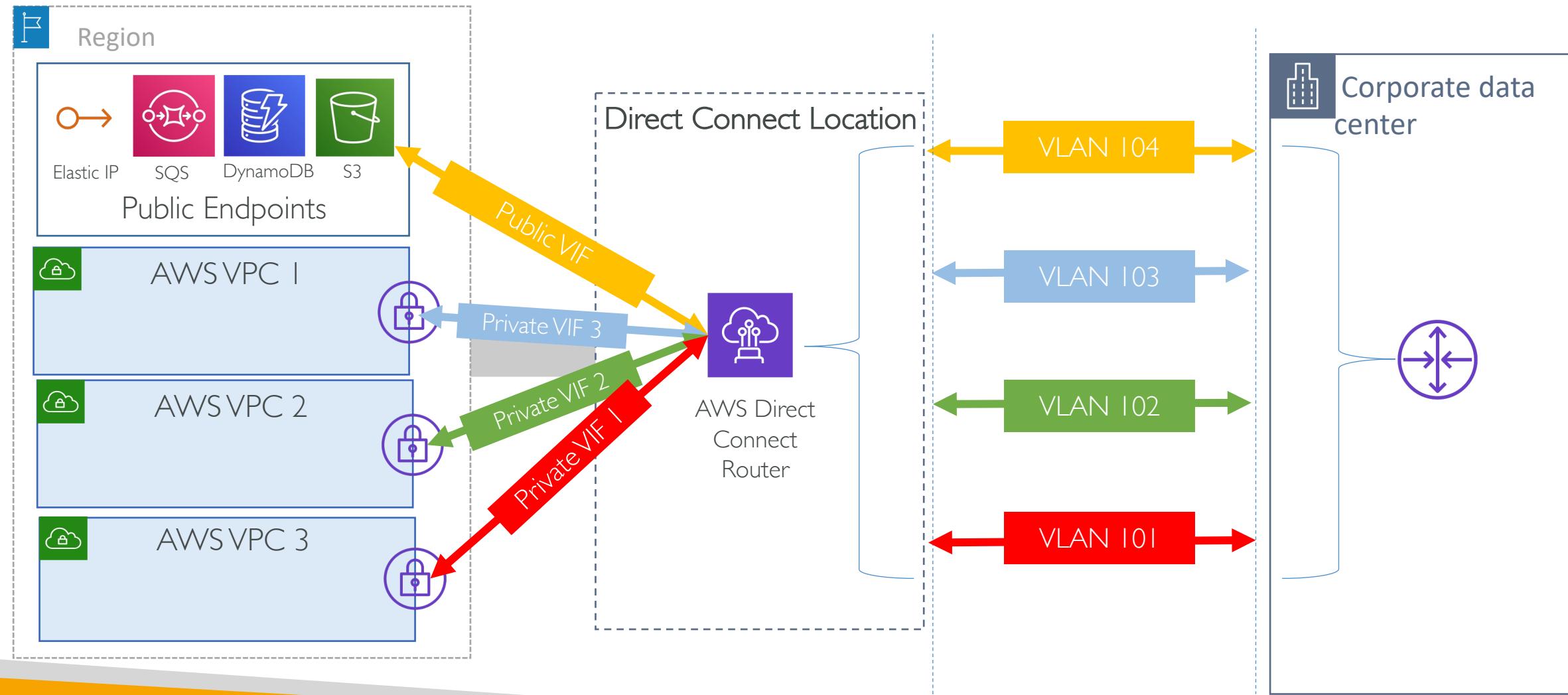
The second step is 'Connection settings'. It requires entering a 'Name' (e.g., 'Test') and selecting a 'Location' (e.g., 'CoreSite NY1, New York, NY').

# DX Virtual Interfaces

# Virtual Interfaces – VIF (Logical connectivity)

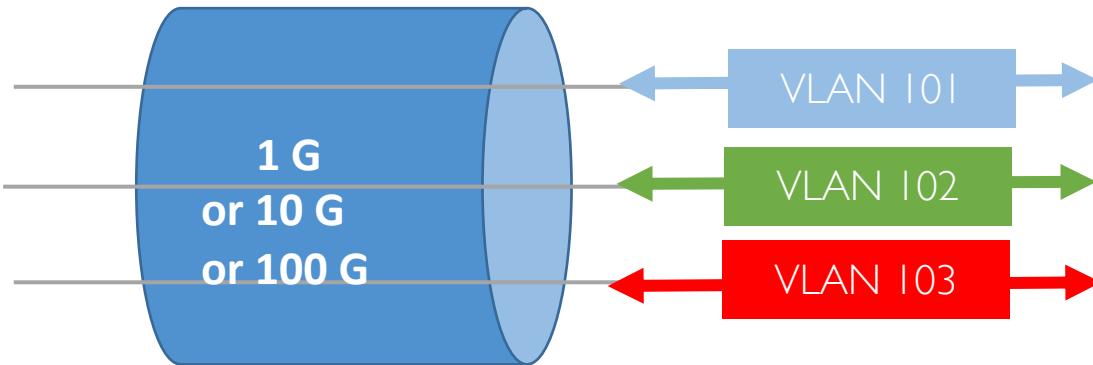
- In order to use the DX connection you must provision the Virtual Interfaces
- A VIF is a configuration consisting primarily of an 802.1Q VLAN
- There are 3 types of the VIFs
  - Public VIF - Enables the connectivity to all AWS public IP addresses
  - Private VIF - Enables the connectivity to resources within VPC using private IPs
  - Transit VIF – Enables access to Transit Gateways associated with Direct Connect gateways

# Public and Private VIFs



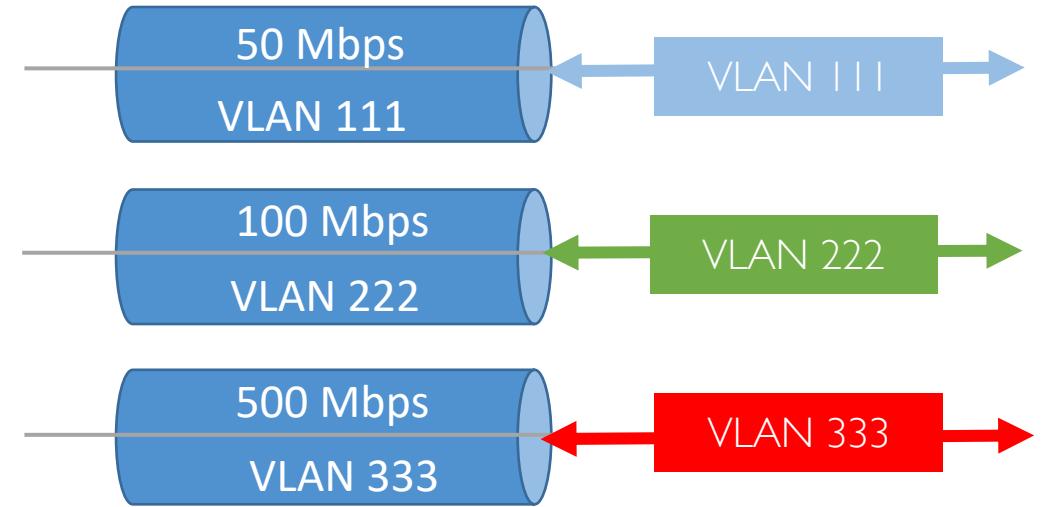
# VIFs for Dedicated vs Hosted Connection

Dedicated Connection



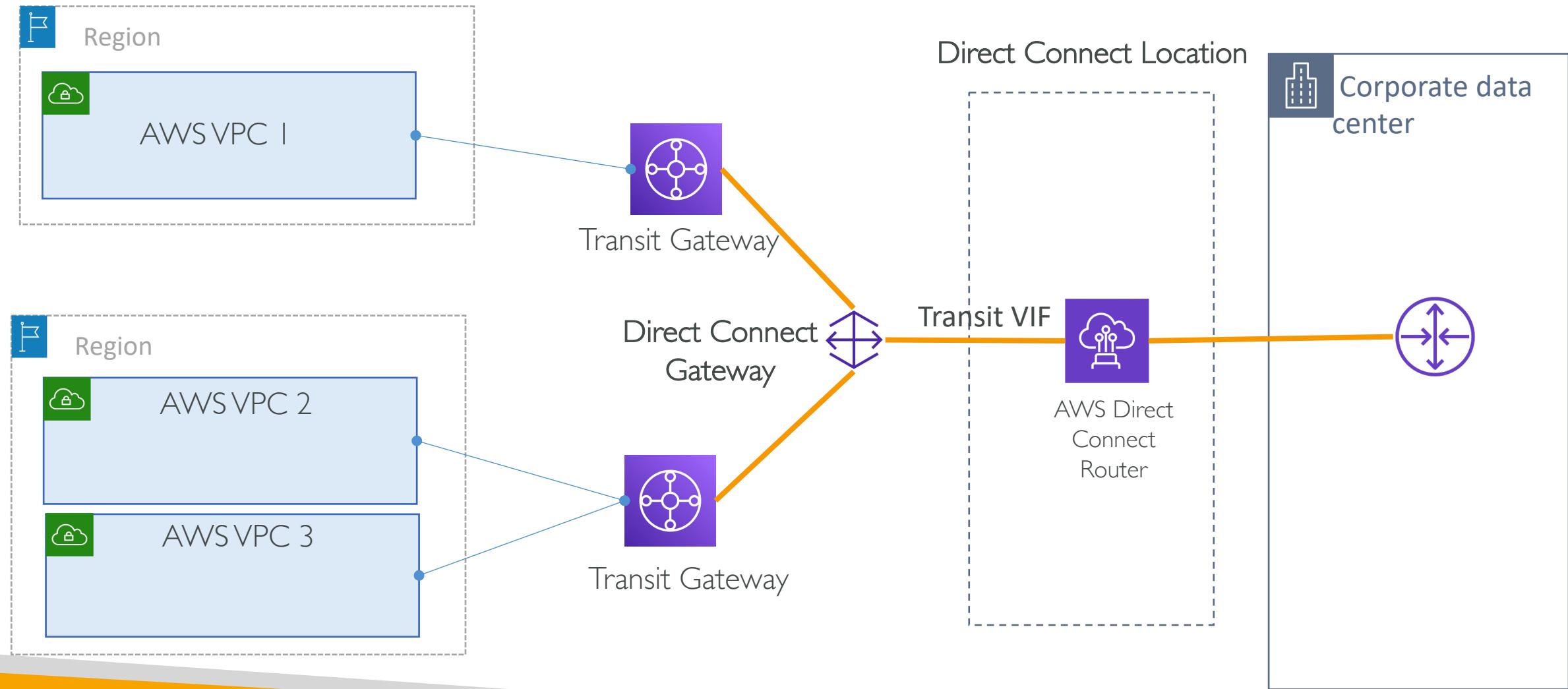
One DX connection x Multiple VIFs

Hosted Connection



Multile DX connections for multiple VIFs

# Transit VIF



# VIF Parameters

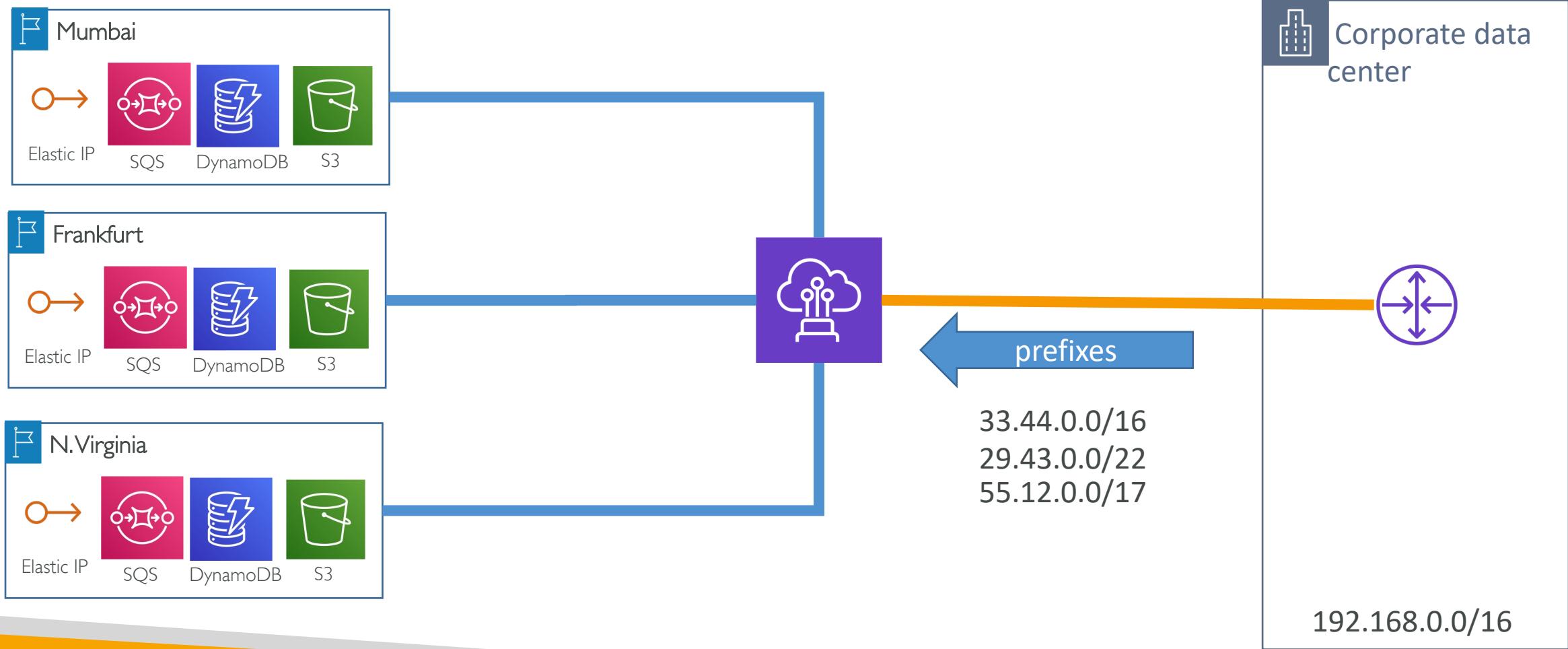
# VIF parameters

- Connection: AWS DX connection or a LAG
- VIF Type: Public or Private or Transit
- VIF Name: Anything
- VIF Owner: Your AWS account or other AWS account (hosted VIF)
- Gateway Type (Private VIF only)
  - Virtual Private Gateway
  - Direct Connect Gateway
- VLAN
  - Not duplicated on same DX connection (1-4094)
  - For hosted connection VLAN ID is already configured by the partner
- BGP address Family – IPv4 or IPv6
- BGP Peer IP Addresses
  - Public VIF (IPv4) - Public IPs (/30) allocated by you for the BGP session. Request to AWS if you don't have it.
  - Private VIF (IPv4) - Specify private IPs in the range 169.254.0.0/16. By default AWS provides address space.
  - IPv6: Amazon automatically allocates you a /125 IPv6 CIDR. You cannot specify your own peer IPv6 addresses.

# VIF parameters

- BGP ASN
  - Public or Private BGP ASN
  - Public ASN must be owned by the customer
  - Private ASN must be between 1 to 2147483647 range
- BGP MD5 authentication key. If not provided, AWS generates authentication key
- Prefixes to be advertised (Public VIF only)
  - Public IPv4 routes or IPv6 routes to advertise over BGP

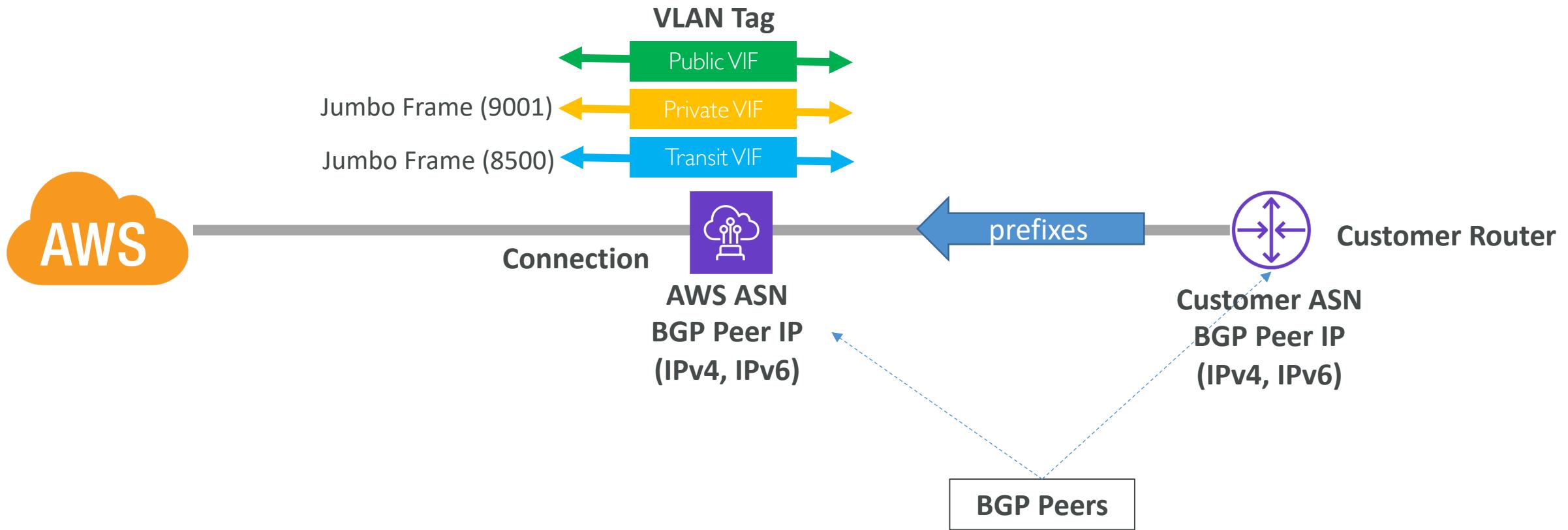
# Advertising prefixes over a public VIF



# VIF parameters

- BGP ASN
  - Public or Private BGP ASN
  - Public ASN must be owned by the customer
  - Private ASN must be between 1 to 2147483647 range
- BGP MD5 authentication key. If not provided, AWS generates authentication key
- Prefixes to be advertised (Public VIF only)
  - Public IPv4 routes or IPv6 routes to advertise over BGP
- Jumbo Frames (Private and Transit VIF only)
  - Private VIF: 9001 MTU supported for propagated routes only (Default is 1500 MTU)
  - Transit VIF: 8500 MTU supported

# VIF parameters

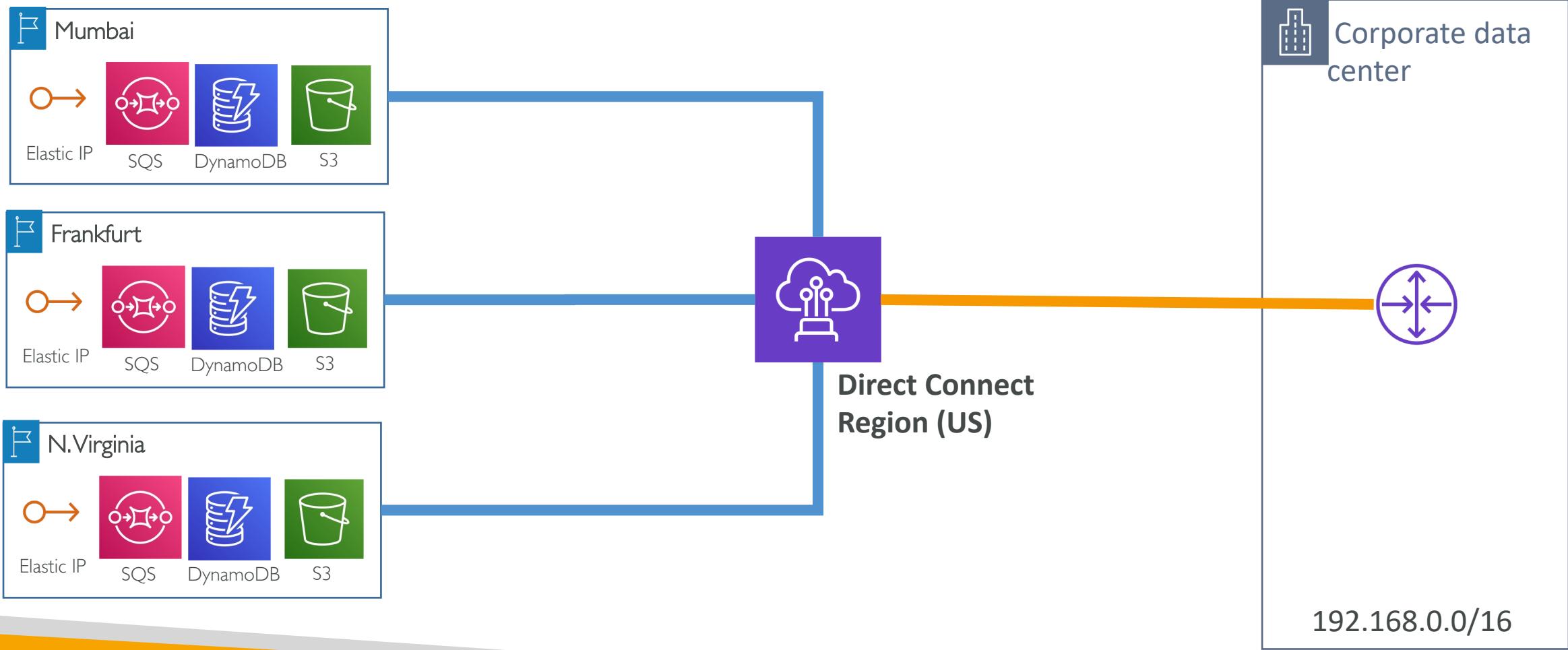


# Public VIF

# Public VIF

- Enables your network to connect to all AWS Public IPs **globally**

# Access to all global regions via Public VIF



# Public VIF

- Enables your network to connect to all AWS Public IPs **globally**
- You can access services outside VPC e.g S3, SQS, DynamoDB and other public endpoints like AWS managed VPN (VGW) Public IPs
- To create a public VIF with IPv4 addresses, you need to provide both the AWS router public IPs and your side of the router public IPs with /30 CIDR
- In case you don't have your own Public IPs for peering, raise a support case to get it from AWS owned IP ranges (AWS provides /31 range)
- You must also specify the IPv4 address prefixes you want to advertise
- AWS verifies with Internet registries that these IP prefixes are owned by you and you are authorized to advertise those prefixes

# Public VIF

- AWS advertises all Amazon prefixes over a BGP session. These includes prefixes for AWS services like EC2, S3 and Amazon.com
- No access to non-amazon prefixes
- Refer ip-ranges.json for current list of Amazon prefixes
- At customer router the firewall can be used to restrict access to and from specific Amazon prefixes
- From customer router to AWS maximum 1000 route prefixes can be advertised per Border Gateway Protocol (BGP) session

# Public VIF creation walkthrough

The screenshot shows the AWS Direct Connect console with the 'Virtual interfaces' section selected. The main content area is titled 'Create virtual interface'. It explains that you can create a private or public virtual interface. The 'Type' section contains three options: 'Private' (disabled), 'Public' (selected and highlighted with a blue border), and 'Transit' (disabled). Below this, the 'Public virtual interface settings' section shows a 'Virtual interface name' field containing 'PublicVIF'.

aws Services ▾

Search for services, features, marketplace products, and docs [Alt+S]

Direct Connect > Virtual interfaces > Create

## Create virtual interface

You can create a private virtual interface to connect to your VPC. Or, you can create a public virtual interface to connect to AWS services that aren't in a VPC, such as Amazon S3 and Glacier. For private virtual interfaces, you need one private virtual interface for each VPC to connect to from the AWS Direct Connect connection, or you can use a AWS Direct Connect gateway. [Learn more](#)

### Virtual interface type

Type

Private  
A private virtual interface should be used to access an Amazon VPC using private IP addresses.

Public  
A public virtual interface can access all AWS public services using public IP addresses.

Transit  
A transit virtual interface is a VLAN that transports traffic from a Direct Connect gateway to one or more transit gateways.

### Public virtual interface settings

Virtual interface name  
A name to help you identify the new virtual interface.

PublicVIF

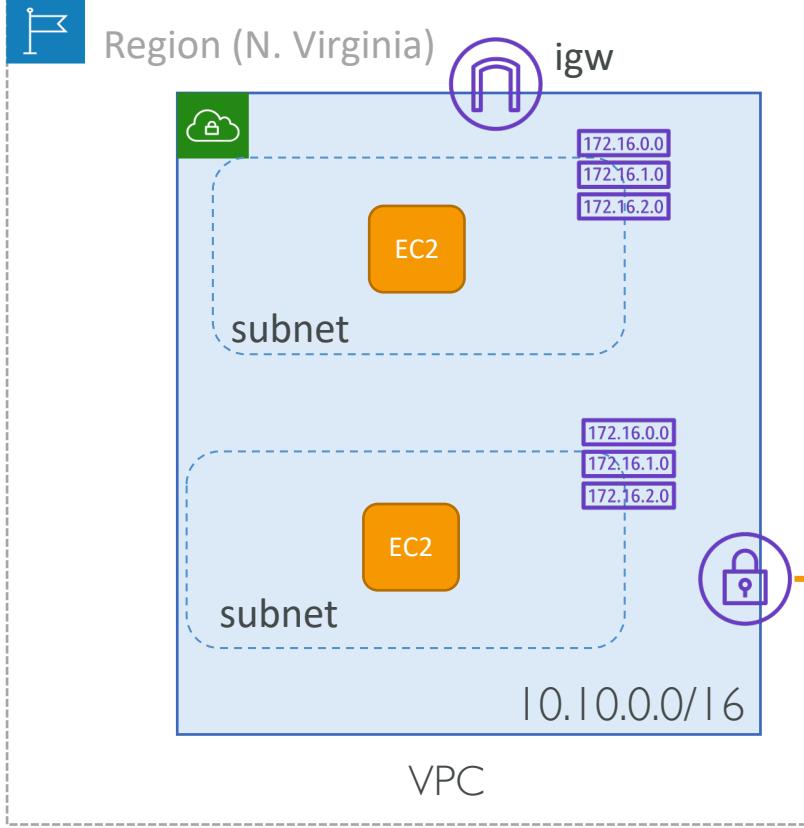
# Remember

- Public VIF enables your network to connect to all AWS Public IPs **globally**
- To create a public VIF with IPv4 addresses, you need to provide both the AWS router public IPs and your side of the router public IPs with /30 CIDR
- From customer router to AWS maximum 1000 route prefixes can be advertised per Border Gateway Protocol (BGP) session

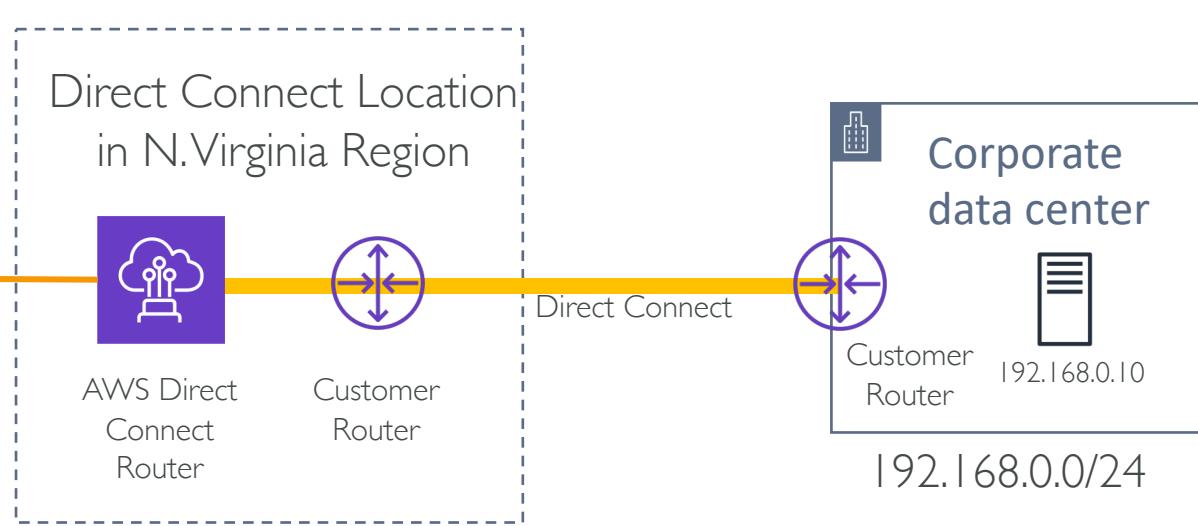
# Private VIF

# Private VIF

- Enables your network to connect to resources inside VPC using Private IPs for resources like EC2, RDS, Redshift over Private IPs
- You must attach your VPC to VGW and associate VGW with Private VIF
- **Private VIF and VGW must be in the same AWS Region**
- On BGP session, customer router will receive all the prefixes of VPC
- You can announce a maximum of **100** prefixes to AWS. These routes can be automatically be propagated into subnet route tables
- In order to advertise more than 100 prefixes, you should summarize the prefixes into larger range to reduce number of prefixes
- The propagated routes will take precedence over default route to internet via IGW



| Destination    | Target   | Route Type |
|----------------|----------|------------|
| 10.10.0.0/16   | Local    | Static     |
| 0.0.0.0/0      | igw-xxxx | Static     |
| 192.168.0.0/24 | vgw-xxxx | Propagated |
| 192.168.1.0/24 | vgw-xxxx | Propagated |
| 192.168.2.0/24 | vgw-xxxx | Propagated |

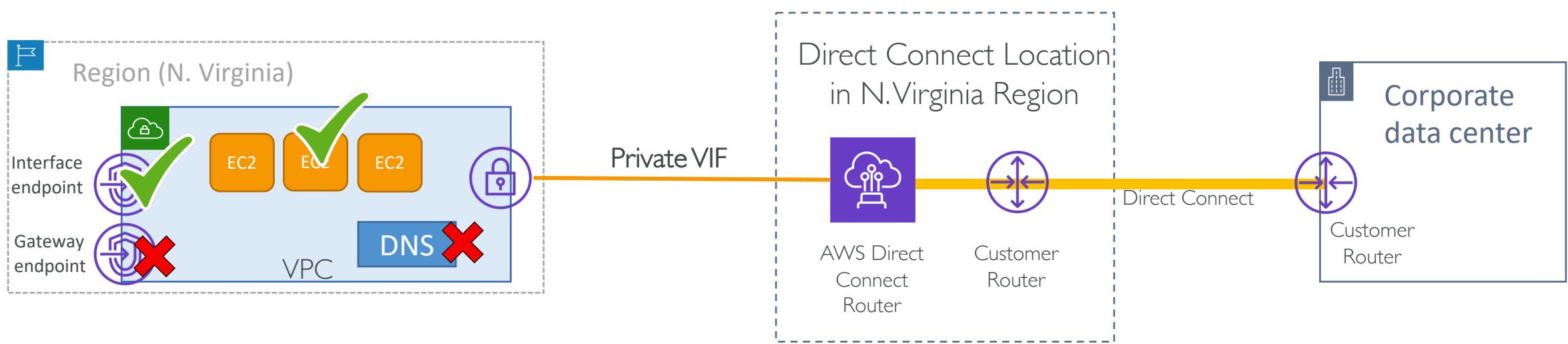


# Private VIF

- Supports MTU of 1500 (default) and 9001 for propagated routes

# Private VIF - What you can't access inside VPC?

- Does not provide access to VPC DNS resolver at Base + 2
- Does not provide access to VPC Gateway endpoints



# Private VIF creation walkthrough

The screenshot shows the AWS Direct Connect console with the 'Virtual interfaces' section selected. The main content area is titled 'Create virtual interface' and provides a brief overview of what a virtual interface is and its types. It highlights that private virtual interfaces are used for connecting to a VPC using private IP addresses. Three options are listed: 'Private' (selected), 'Public', and 'Transit'. Below this, a 'Private virtual interface settings' section allows naming the interface 'MyPrivateVIF'. A note specifies that names must be between 1 and 100 characters, containing only lowercase letters, numbers, and hyphens.

aws Services ▾

Search for services, features, marketplace products, and docs [Alt+S]

Direct Connect > Virtual interfaces > Create

## Create virtual interface

You can create a private virtual interface to connect to your VPC. Or, you can create a public virtual interface to connect to AWS services that aren't in a VPC, such as Amazon S3 and Glacier. For private virtual interfaces, you need one private virtual interface for each VPC to connect to from the AWS Direct Connect connection, or you can use a AWS Direct Connect gateway. [Learn more](#)

### Virtual interface type

Type

**Private**  
A private virtual interface should be used to access an Amazon VPC using private IP addresses.

**Public**  
A public virtual interface can access all AWS public services using public IP addresses.

**Transit**  
A transit virtual interface is a VLAN that transports traffic from a Direct Connect gateway to one or more transit gateways.

### Private virtual interface settings

Virtual interface name  
A name to help you identify the new virtual interface.  
MyPrivateVIF

Name must contain no more than 100 characters. Valid characters are a-z, 0-9, and – (hyphen)

# Remember

- Private VIF enables your network to connect to resources inside VPC using Private IPs for resources like EC2, RDS, Redshift **over Private IPs**
- You can announce a maximum of 100 prefixes to AWS

# Transit VIF

# Transit VIF

- Enables the connection between Direct Connect and Transit Gateway
- Transit VIF is connected to Direct Connect Gateway and DirectConnect Gateway connects to Transit Gateway
- Support MTU of 1500 and 8500 (Jumbo Frames)
- You can attach multiple DX Gateway to single Transit Gateway
- The ASN for DX Gateway and Transit Gateway must be different

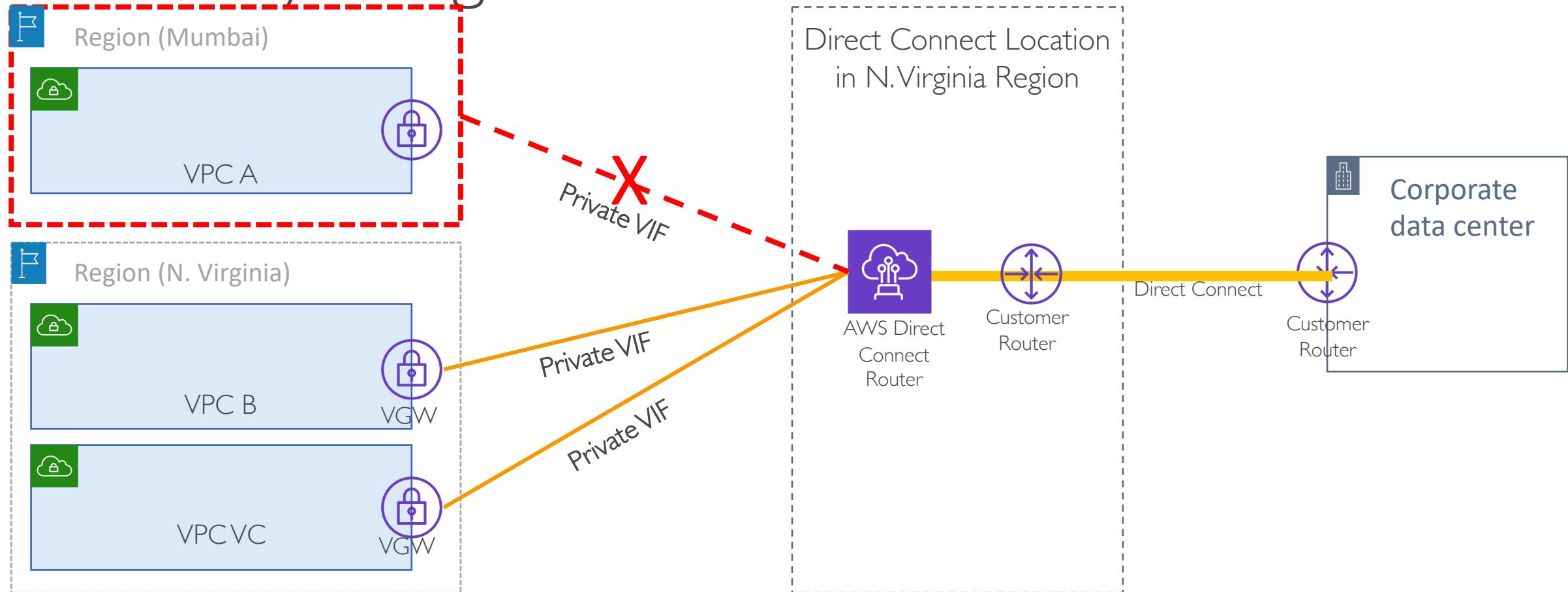
# Direct Connect Gateway and Virtual Private Gateway (VGW)

# Why Direct Connect Gateway?

- Problem: I want to access multiple VPCs using a single Private VIF
- Solution: AWS Direct Connect Gateway

A single direct connect gateway can be connected to multiple VPCs in the **same or different** AWS regions and in the **same or different** AWS accounts using Virtual Private Gateway (VGW)

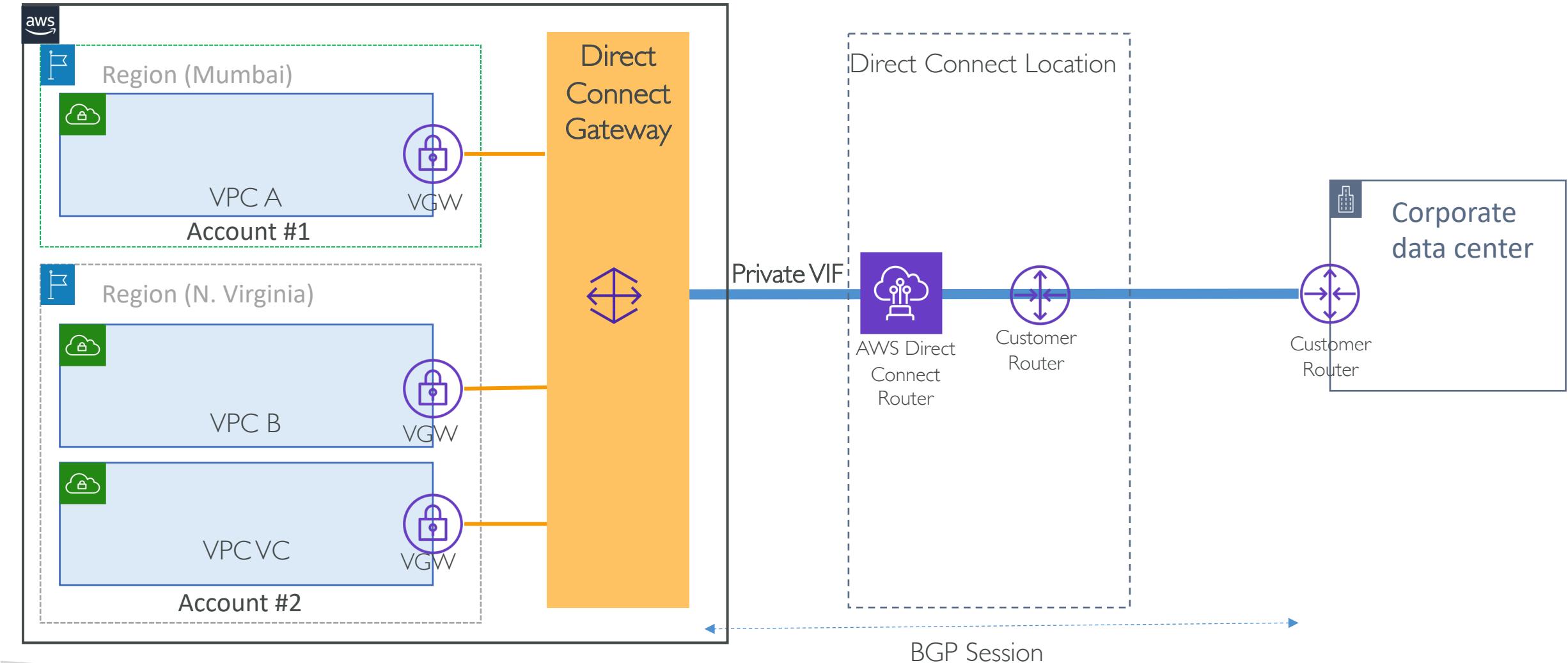
# Connecting VPCs outside of home region directly using Private VIF



# Direct Connect Gateway

- Global network device – Accessible in all regions
- Direct Connect integrates via a private VIF

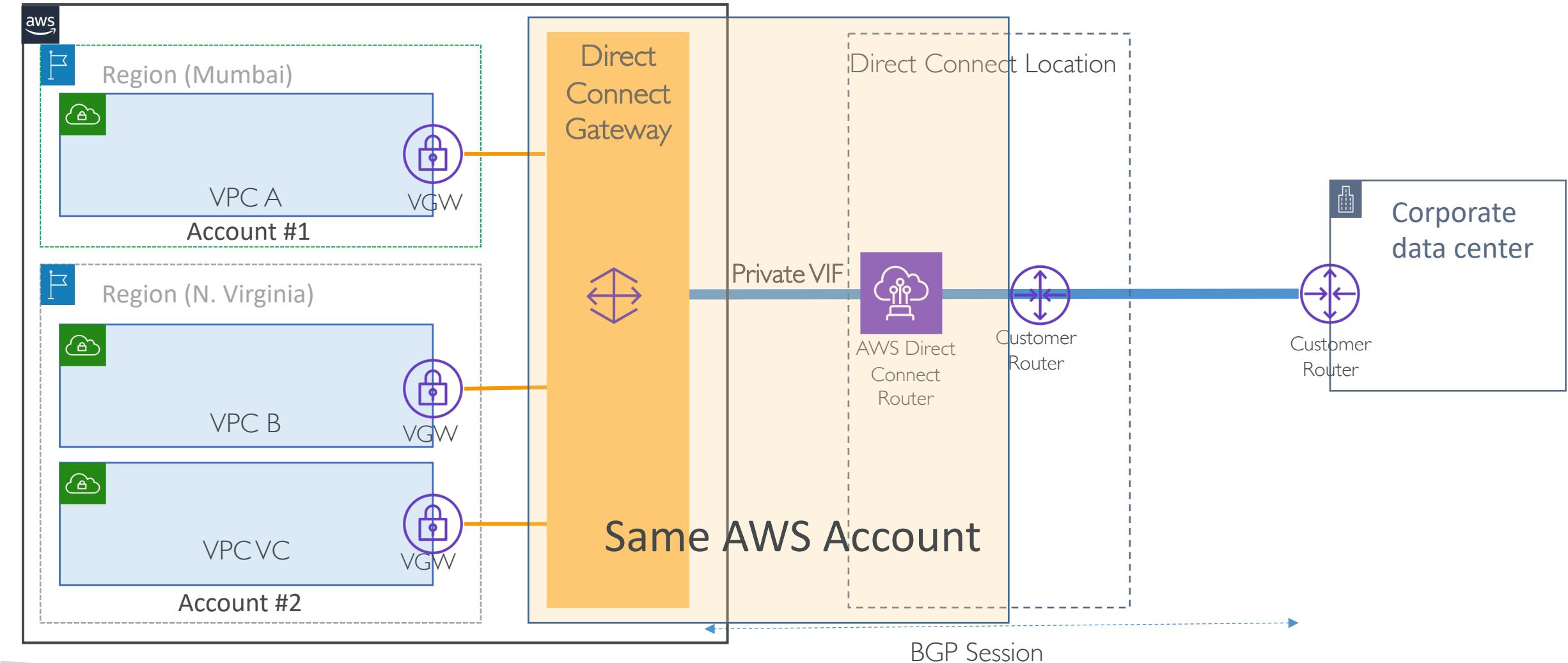
# Direct Connect Gateway



# Direct Connect Gateway

- Global network device – Accessible in all regions
- Direct Connect integrates via a private VIF
- DX Gateway is used for VPC <-> On-premise connectivity and can not be used for Public endpoint connectivity
- Reduces the number of Border Gateway Protocol sessions between your on-premises network and AWS deployments
- The Private VIF and DirectConnect gateway must be owned by same AWS account however VPCs (VGWs) can be from same or different AWS accounts

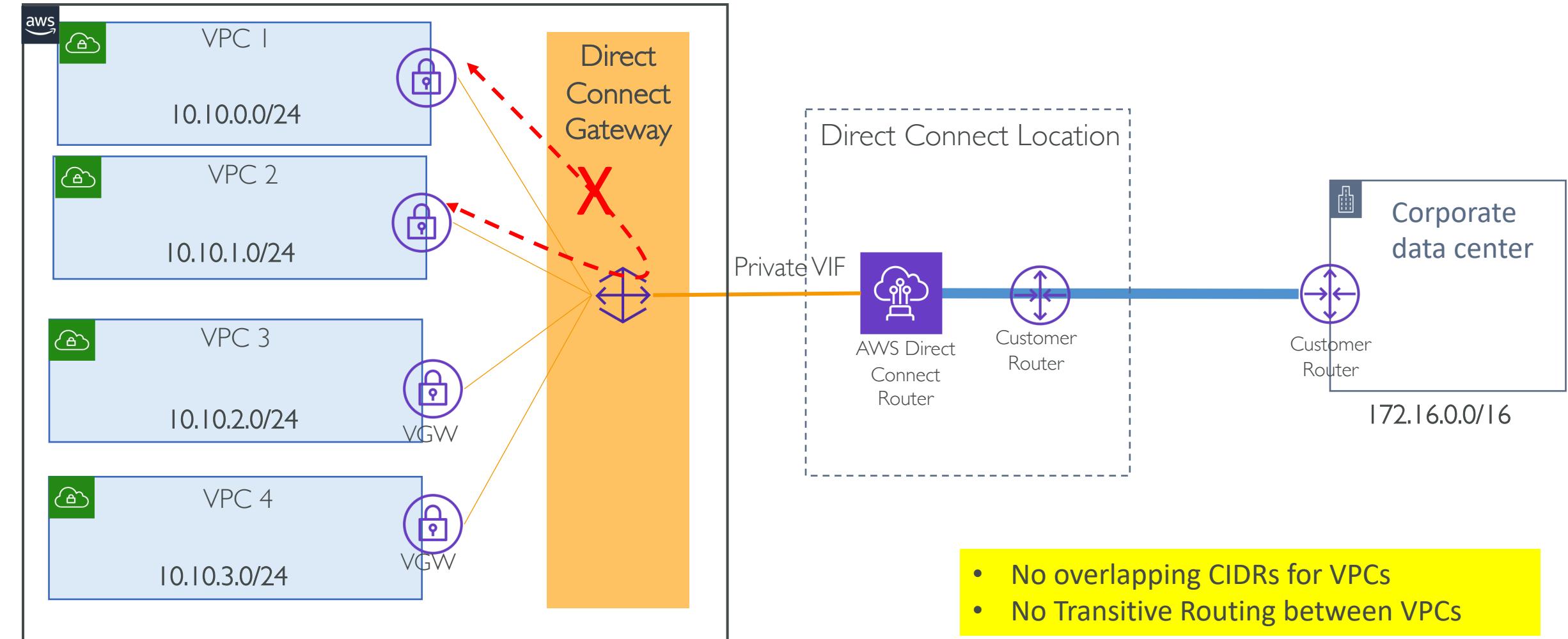
# Direct Connect Gateway



# Direct Connect Gateway

- Overlapping CIDRs for VPCs are not allowed
- VPC to VPC communication is **not** allowed via DX Gateway

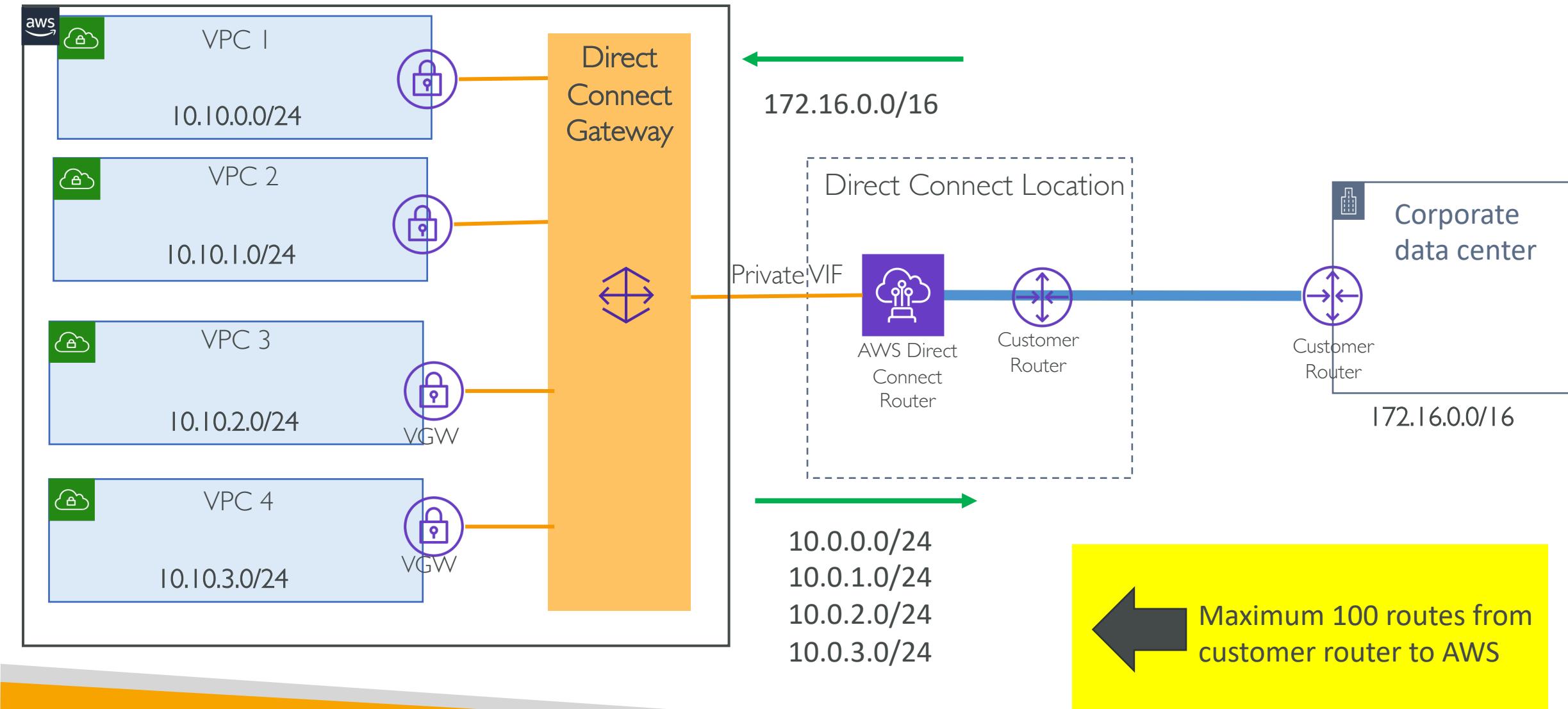
# Direct Connect Gateway - Routing



# Direct Connect Gateway

- Maximum 100 routes can be advertised from on-premises to AWS (similar to Private VIF limit)

# Direct Connect Gateway - Routes



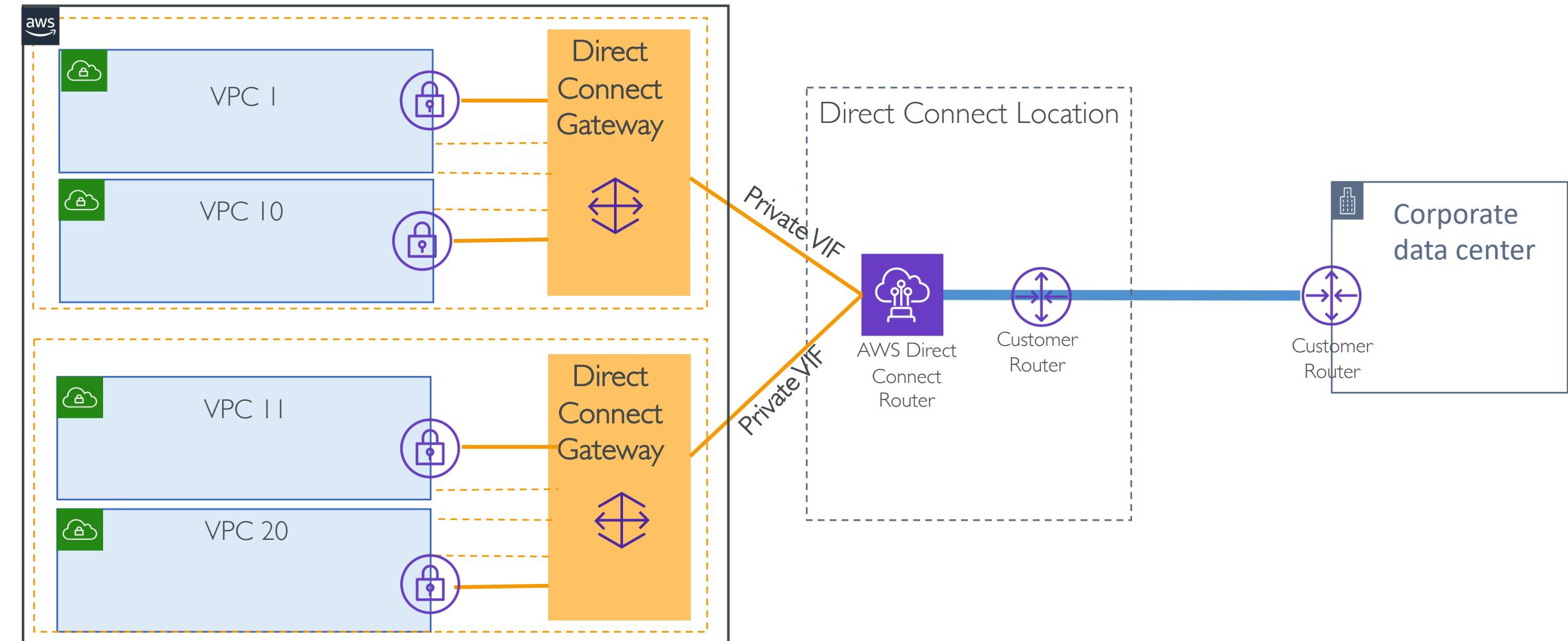
# Direct Connect Gateway – # VPC limits

- 1 Dedicated DX connection supports up to 50 VIFs
- 1 VIF can connect to single Direct Connect Gateway
- 1 Direct Connect Gateway can connect to 10 VGWs (VPCs)
- So with single DX connection we can connect  $50 \times 1 \times 10 = 500$  VPCs

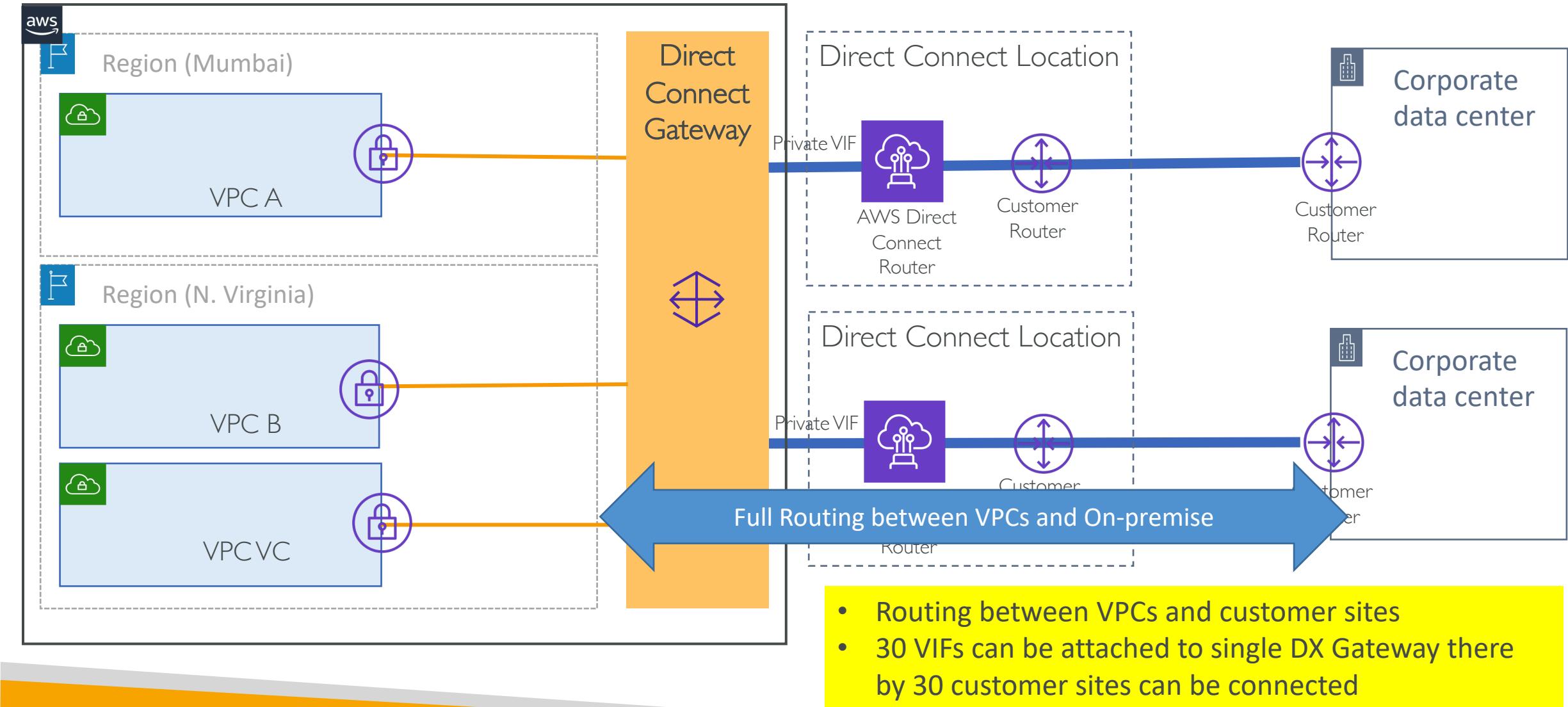
# Creating DX Gateway Walkthrough

# DX Gateway architectures

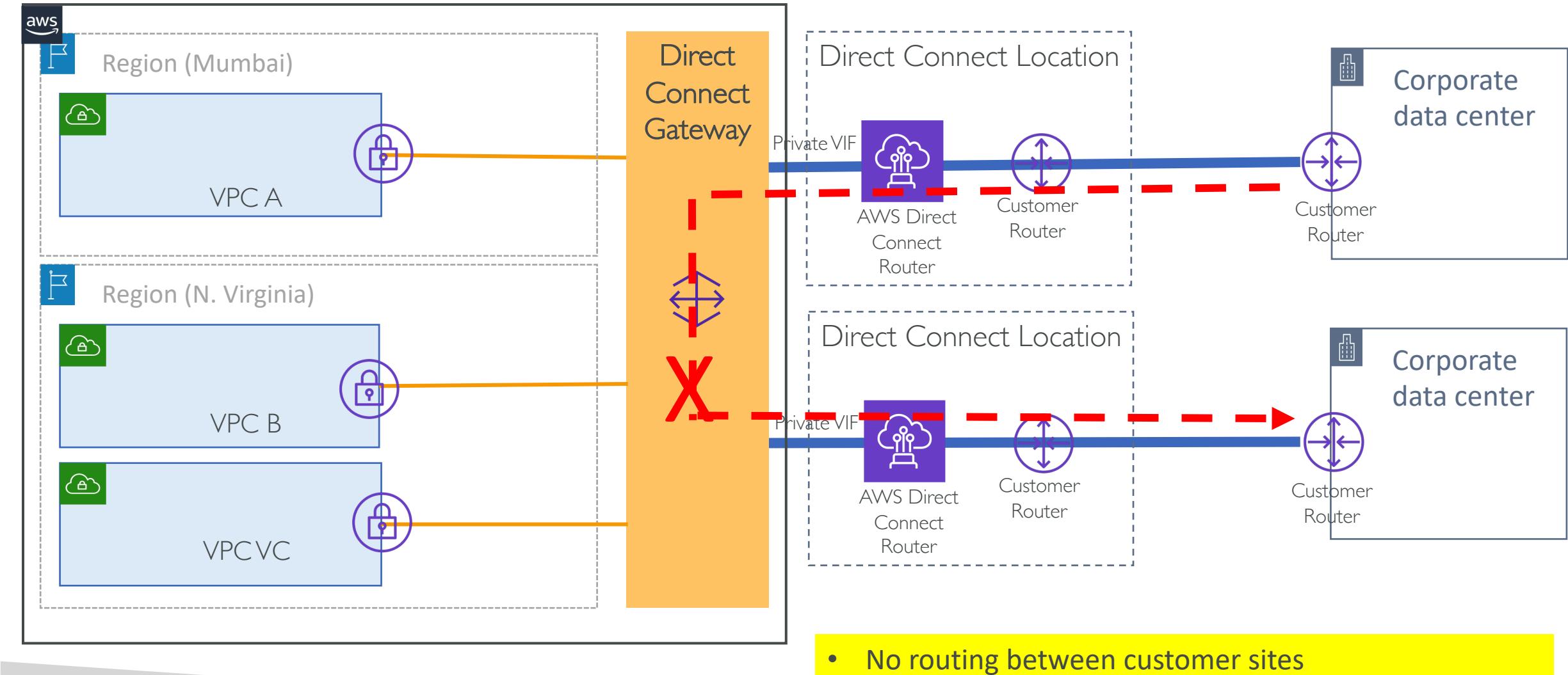
# Multiple Direct Connect Gateways



# DX Gateway and multiple customer sites



# DX Gateway and multiple customer sites



# Direct Connect Gateway + VGW summary

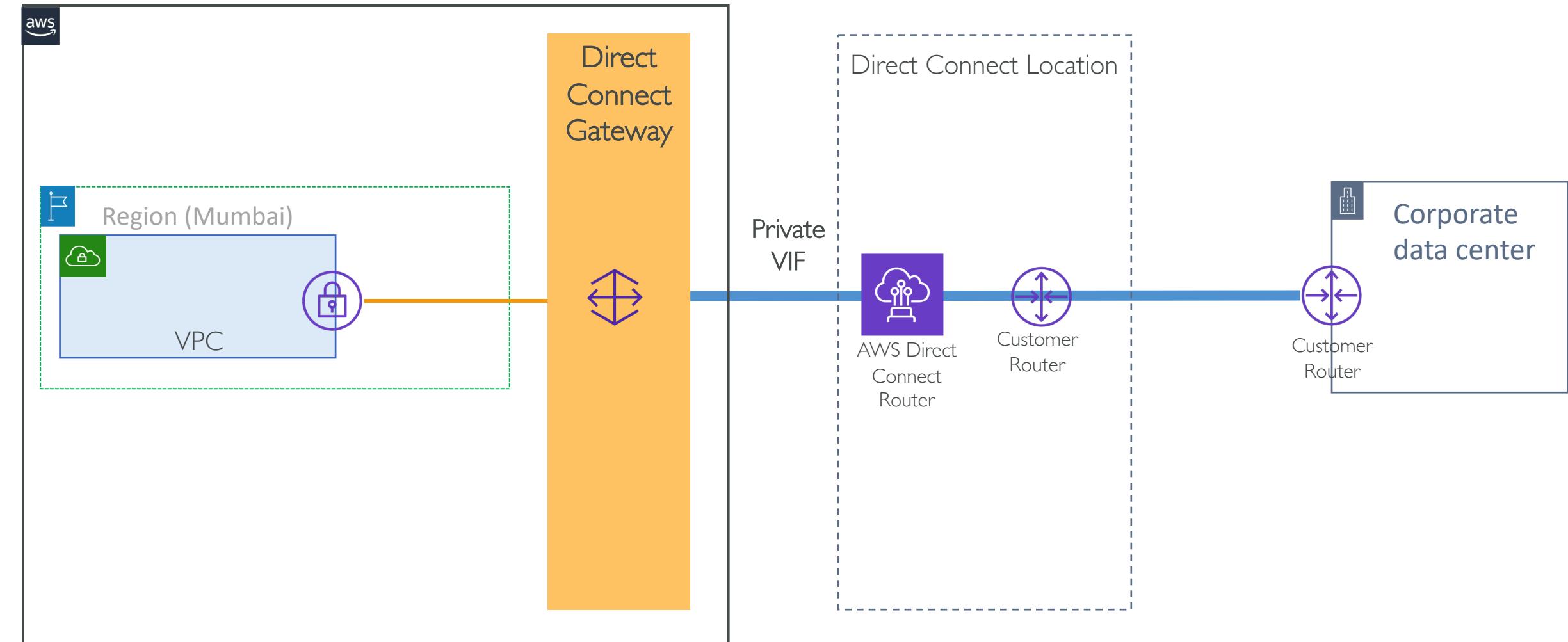
- Globally accessible
- Enables private connectivity between on-premises network and AWS VPCs
- Can connect to up to 10VGWs (VPCs) across AWS regions and across AWS accounts
- VPC CIDRs should not be overlapping
- VPC-to-VPC communication is not allowed via the DX gateway.
- DX Gateway and Private VIF should be created in the same AWS account

# Direct Connect Gateway + VGW summary

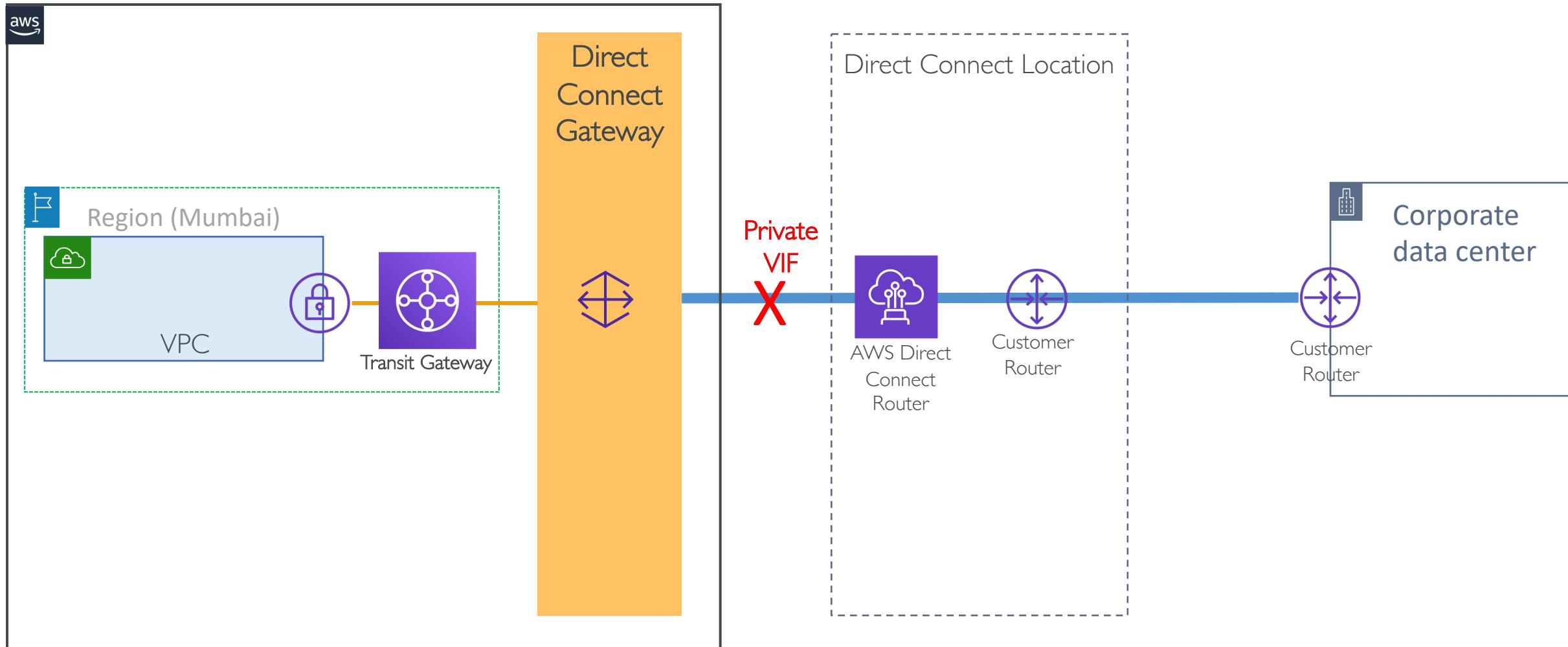
- There are **no charges** for using a Direct Connect gateway.
- You will pay applicable egress data charges based on the source remote AWS Region and port hour charges

# AWS DX Gateway and Transit Gateway (TGW)

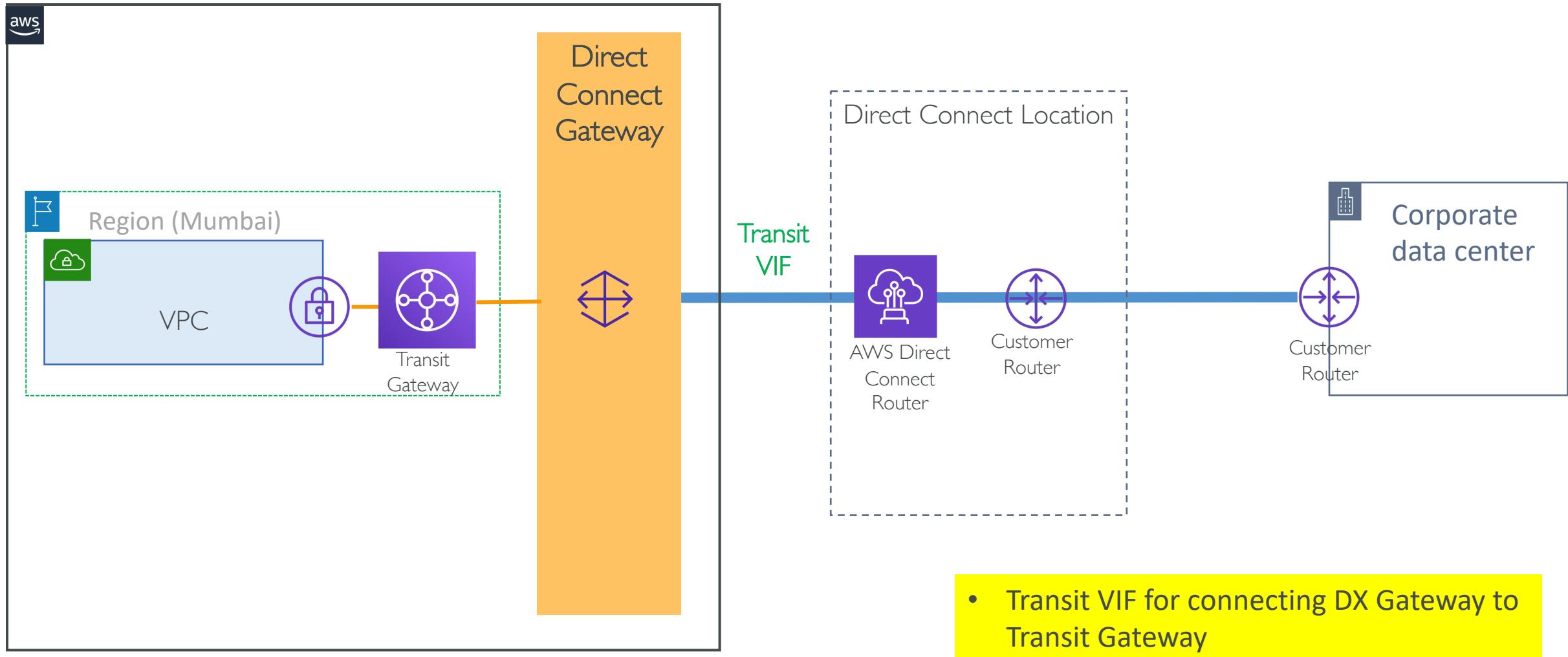
# DX Gateway without TGW



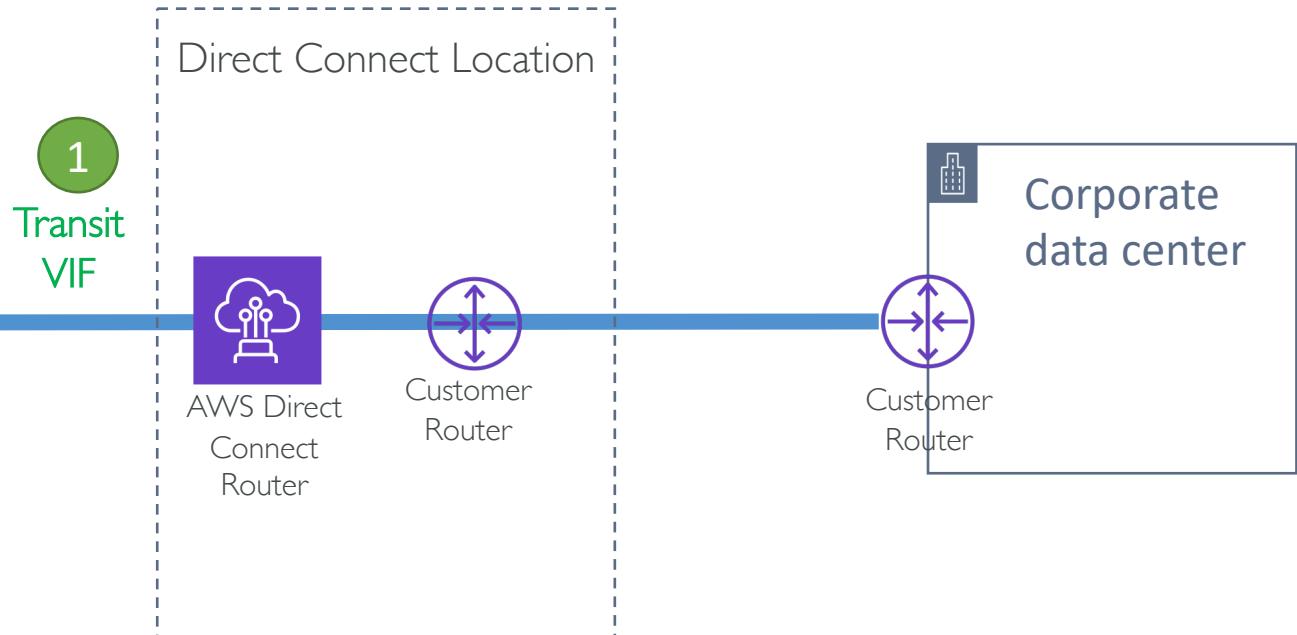
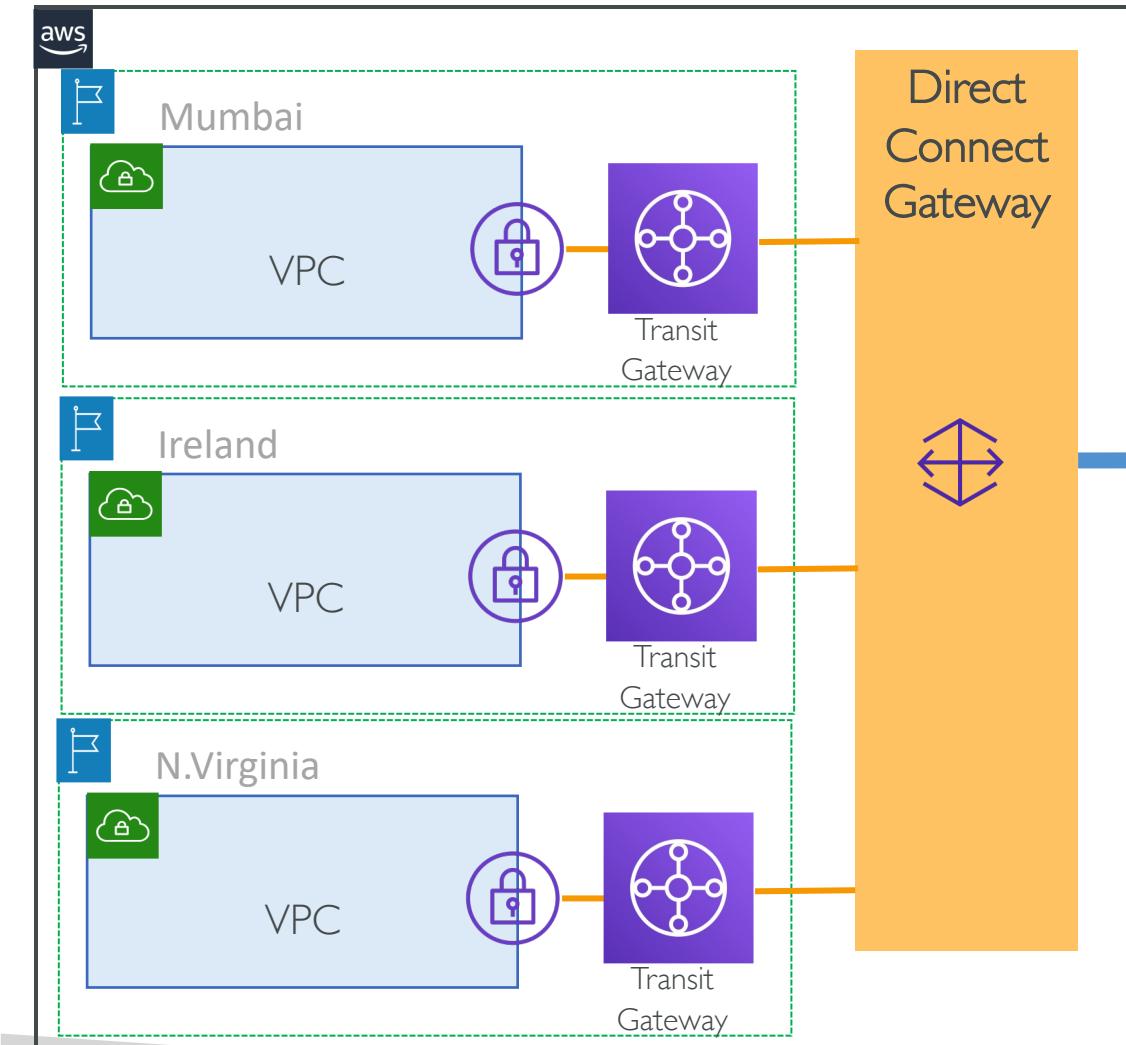
# DX Gateway without TGW



# DX Gateway with TGW

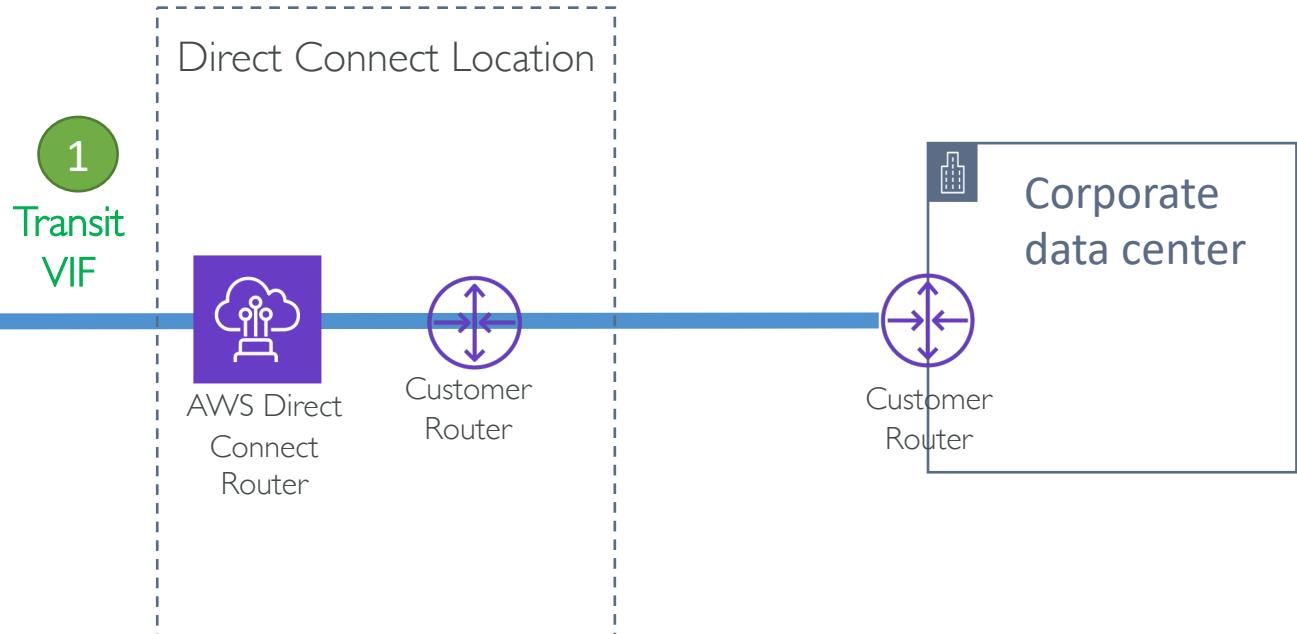
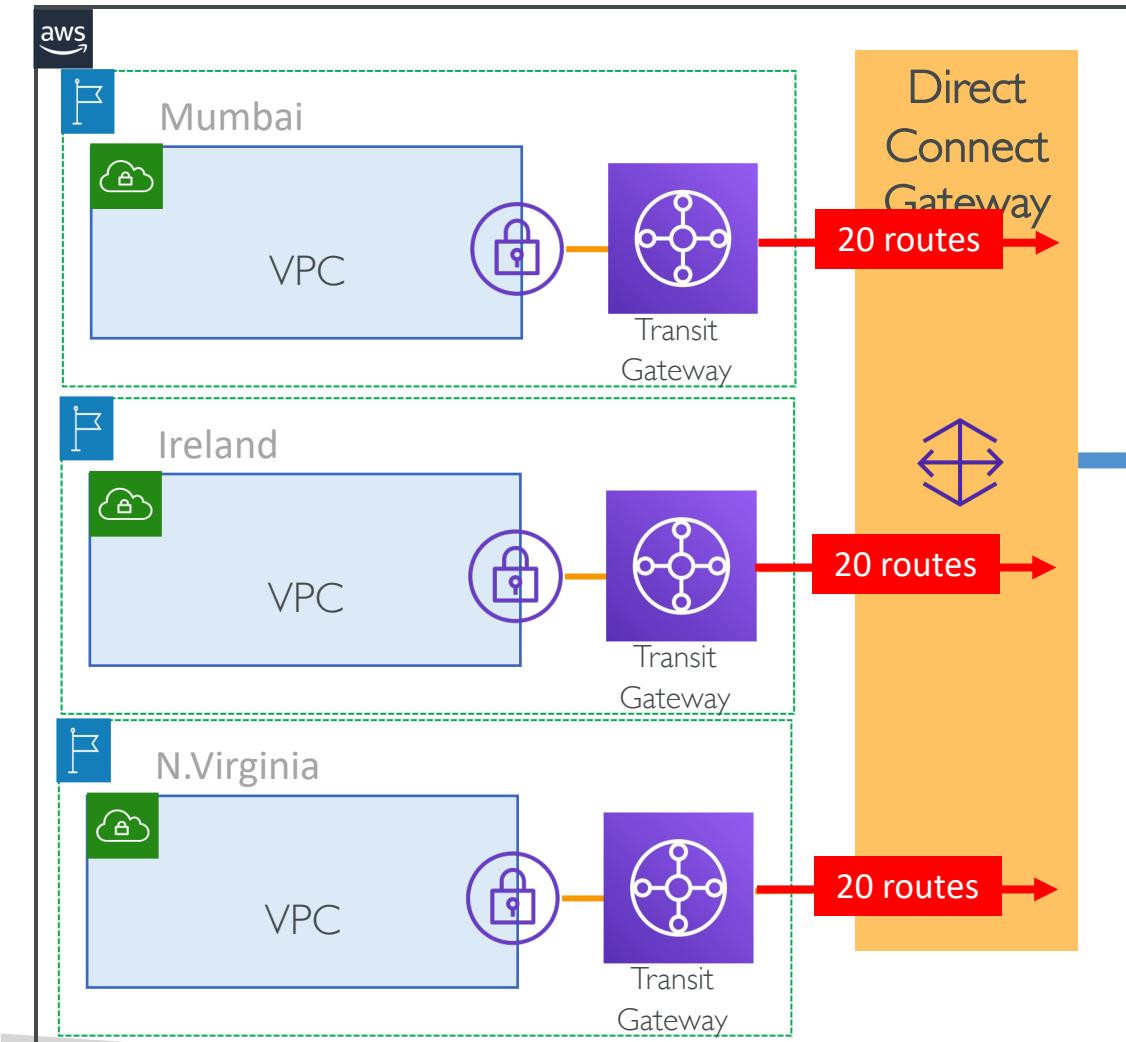


# DX Gateway with multiple TGWs



- 1 Transit VIF per DX connection
- Can connect maximum 3 TGWs to single DX Gateway

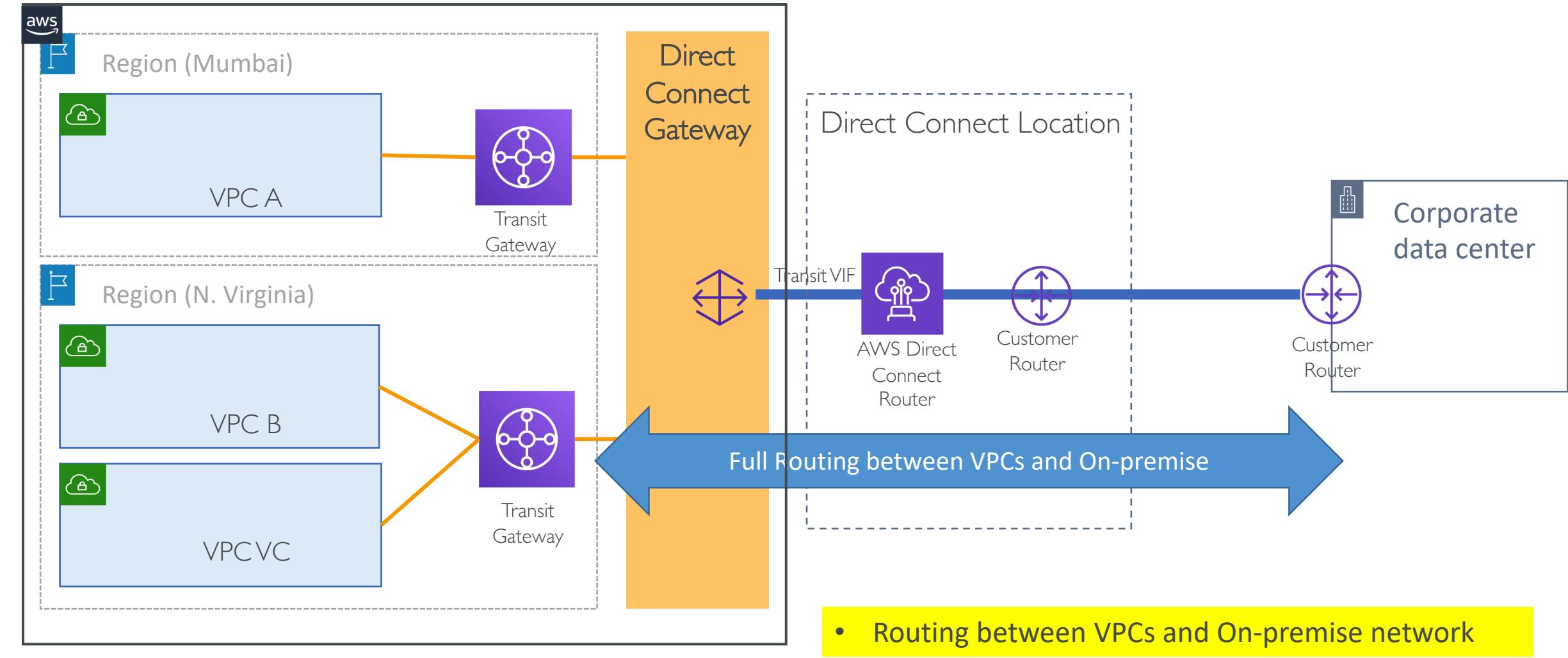
# DX Gateway with multiple TGWs



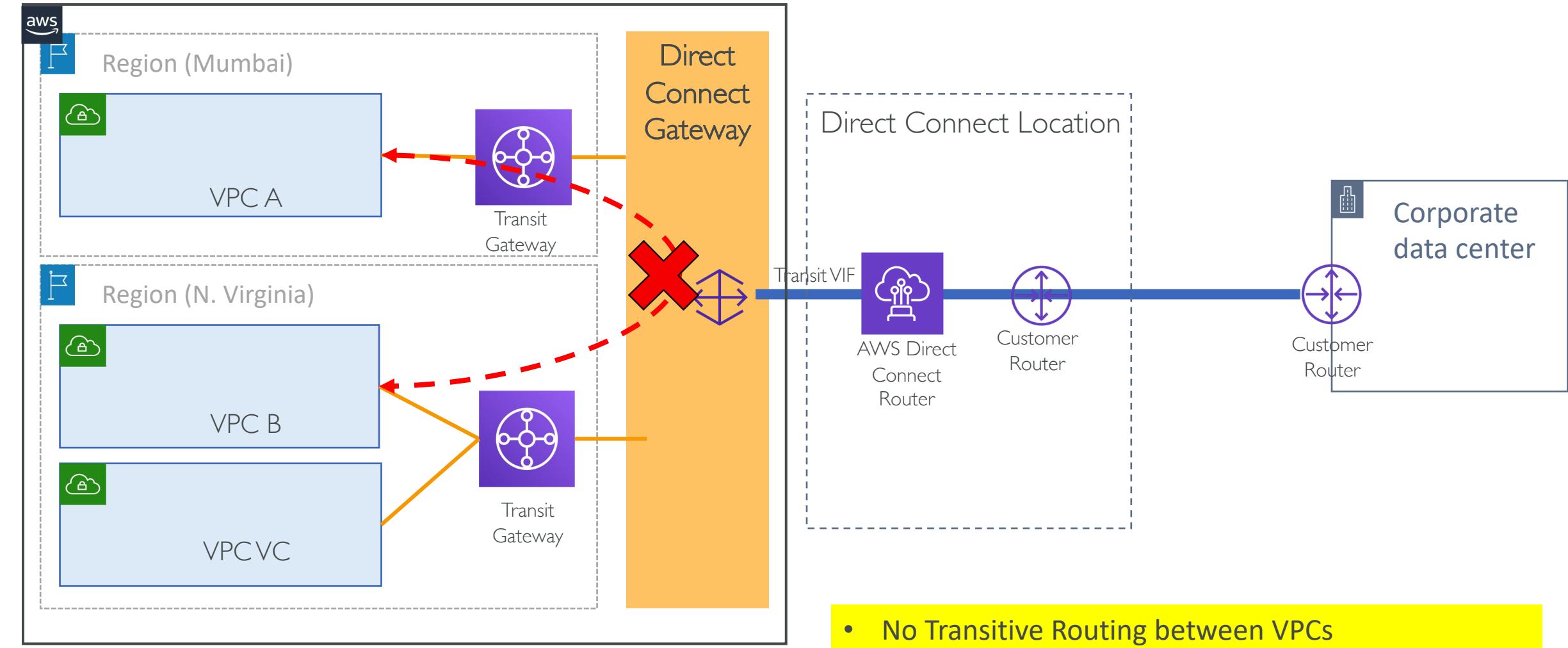
- 1 Transit VIF per DX connection
- Can connect maximum 3 TGWs to single DX Gateway
- 20 Routes per TGW can be advertised

# Let's look at some architectural patterns

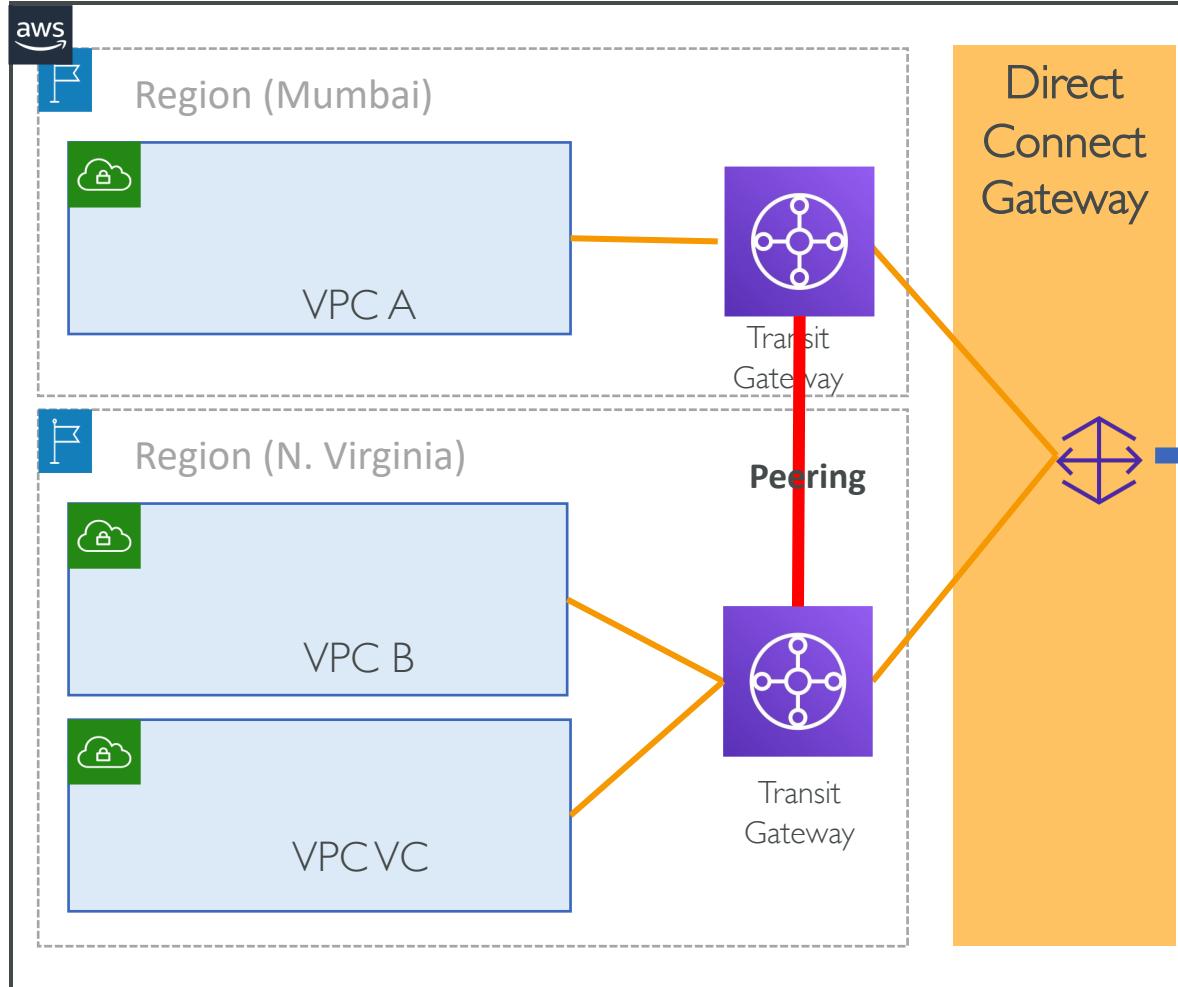
# Routing between VPCs and On-premises



# Routing between VPCs ?

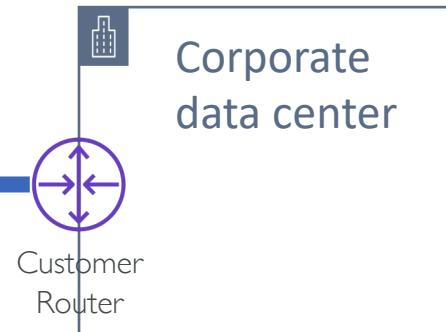


# Routing between VPCs



**With connectivity between VPCs**

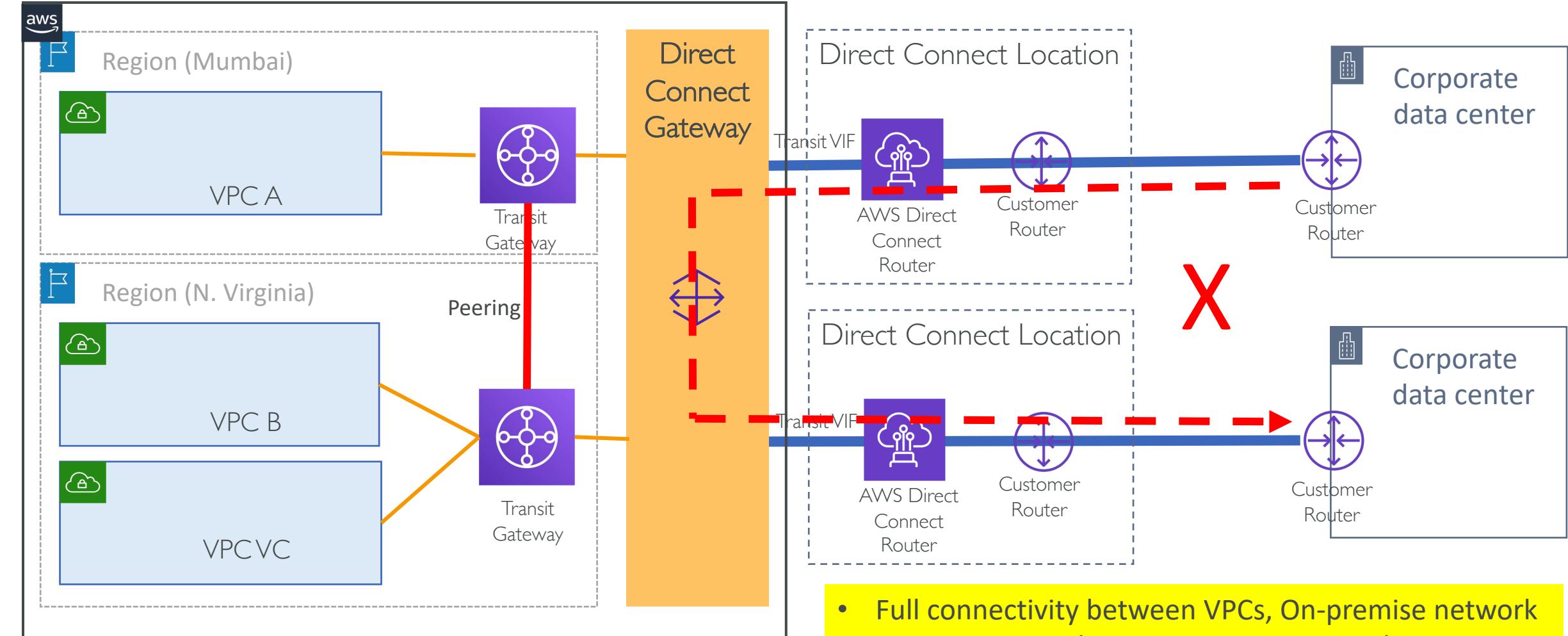
Direct Connect Location



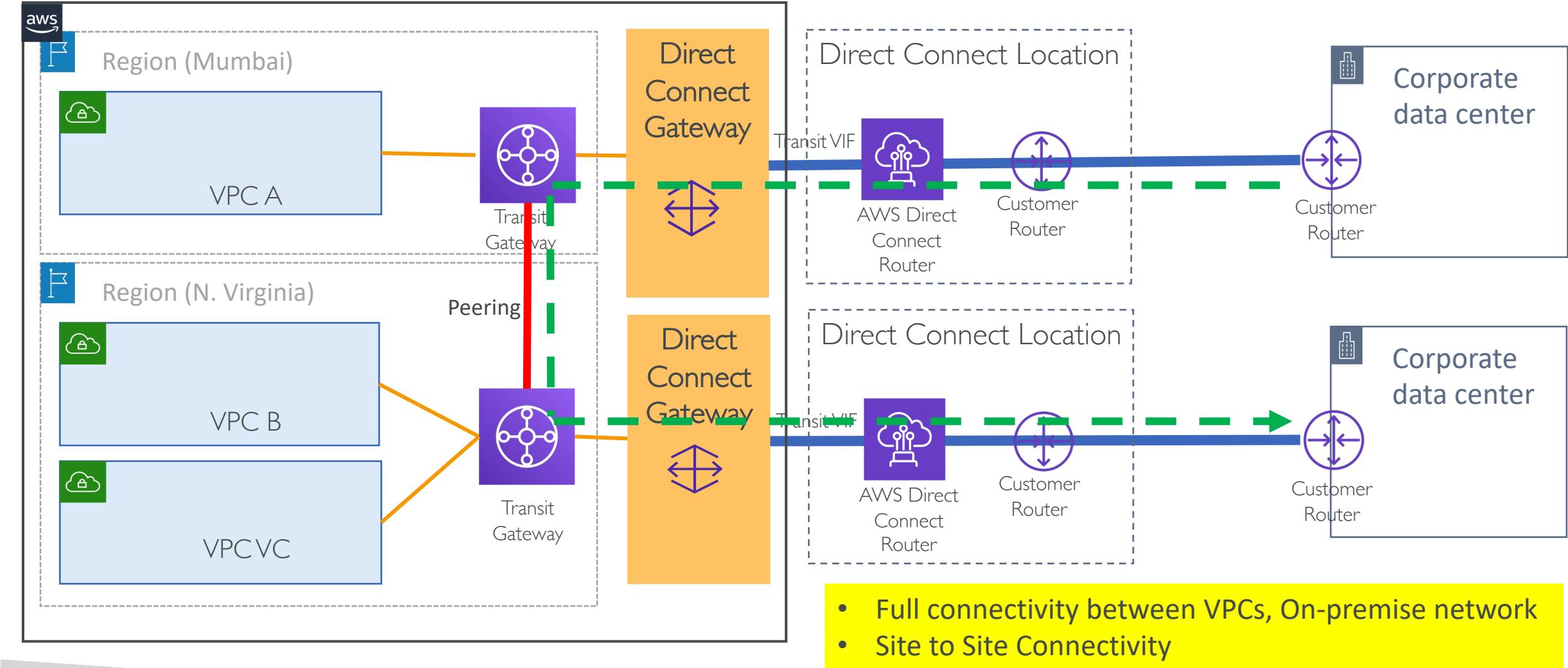
- Full routing between VPCs across regions and On-premise network.
- Routing between VPCs using Transit Gateway peering

# Routing between customer sites

# Multiple customer sites & routing



# Multiple customer sites & routing



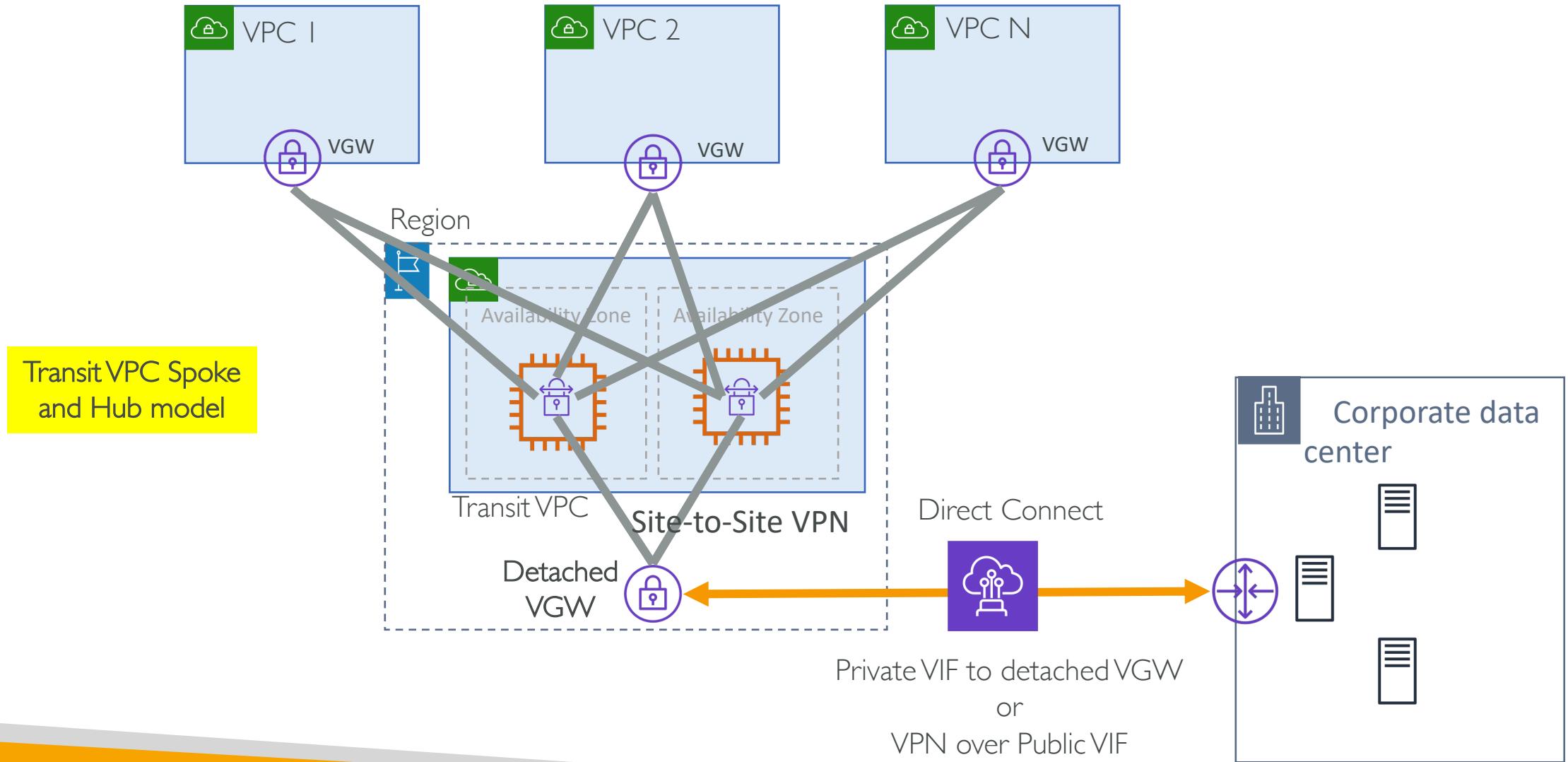


# DX Gateway with TGW - Summary

- 1 Transit VIF per Direct Connect connection. Hence total 50 VIFs + 1 Transit VIF per Direct Connect connection.
- TGW is associated with the Direct Connect Gateway
- 3 TGW's can be attached per Direct Connect Gateway
- TGW's can be peered across AWS Regions
- TGW supports transitive routing
- You cannot create a Transit VIF on a hosted connection with a capacity less than 1 Gbps

# DX and Transit VPC

# Direct Connect and Transit VPC



# Direct Connect and Transit VPC

- Direct connect can be used with Transit VPC to enable Hub and Spoke model
- Enables an IDS / IPS solution (running on EC2)
- VGW is used in detached mode which acts as a Hub
- Detached VGW is connected to one or more customer network using Public or Private VIFs

# Direct Connect Routing Policies and BGP Communities

# Routing Policies and BGP communities

- Routing policies influence the routing decision when traffic flows through Direct Connect connections.
- Inbound routing Policies - from on-premises to AWS 
- Outbound routing Policies - from AWS to on-premises 
- BGP community tags can scope the routes advertised and also influence the preferred path
- BGP community tags can be applied for the routes advertised in both the directions between BGP peers (AWS <-> on-premises)
- Routing policies and BGP communities work differently for Public and Private VIFs

# Public VIF Routing Policies

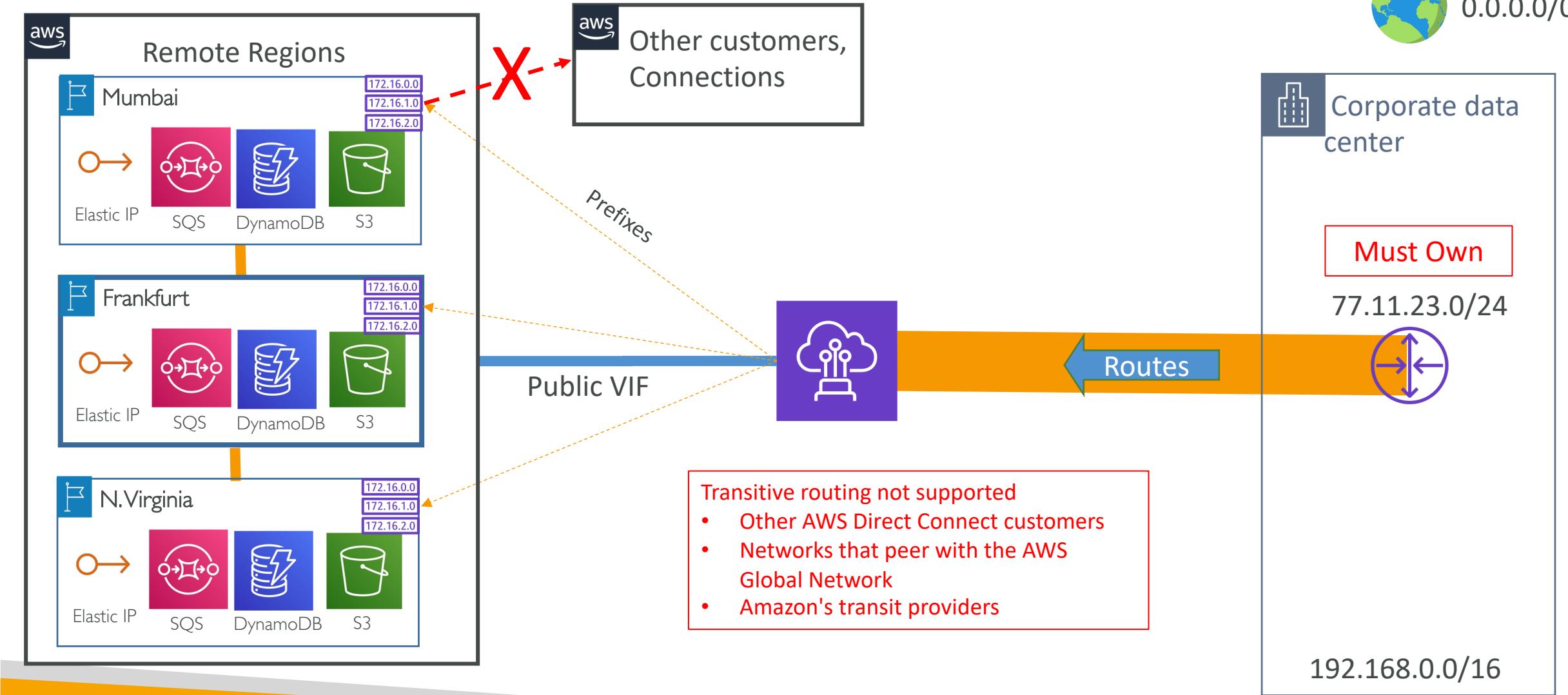
# Public VIF Routing policies - Inbound

- You must specify the public IPv4 prefixes or IPv6 prefixes to advertise over BGP .
- You must own the public prefixes (Regional Internet Registry).
- Traffic must be destined to Amazon public prefixes.
- Transitive routing between connections is not supported.
- AWS Direct Connect performs inbound packet filtering



Customer Router

# Public VIF Routing policies - Inbound



# Public VIF Routing policies - Outbound

- Longest Prefix Match and AS\_PATH can be used to influence the routing
- AWS Direct Connect advertises all local and remote AWS Region prefixes
- Advertises all public prefixes with NO\_EXPORT BGP community tag.
- Additionally AWS advertises 7224:8100 and 7224:8200 BGP community tags.
- The prefixes advertised by AWS Direct Connect must not be advertised beyond the network boundaries of your connection
- In case of multiple AWS DX connections, you can adjust the load-sharing of traffic by advertising prefixes with similar path attributes (ECMP)



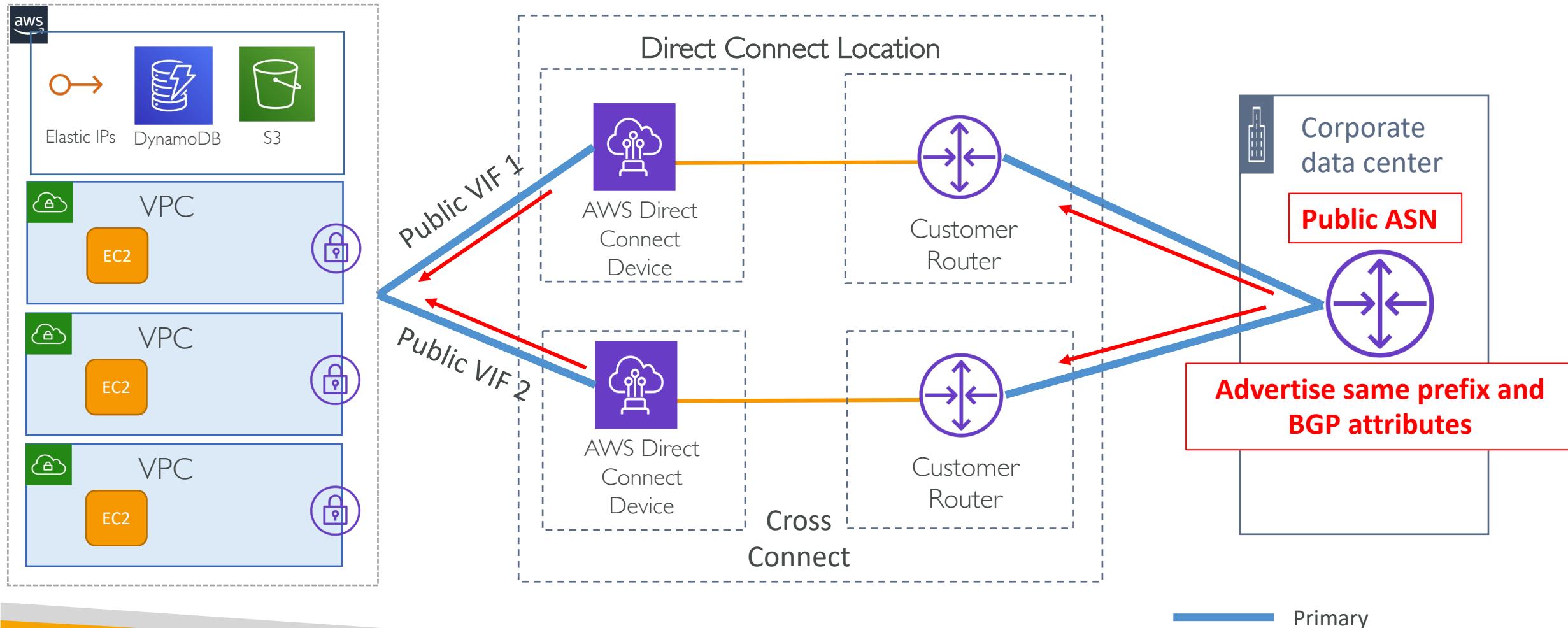
# Multiple DX connections traffic routing scenarios using Routing policies for Public VIF

# Active-Active connection using Public VIF

- If using a public ASN:
  - Customer gateway to advertise the same prefix with the same Border Gateway Protocol (BGP) attributes on both public virtual interfaces.
  - This configuration load balances traffic over both public virtual interfaces.
- If using a private ASN
  - Load balancing on a public virtual interface is not supported.

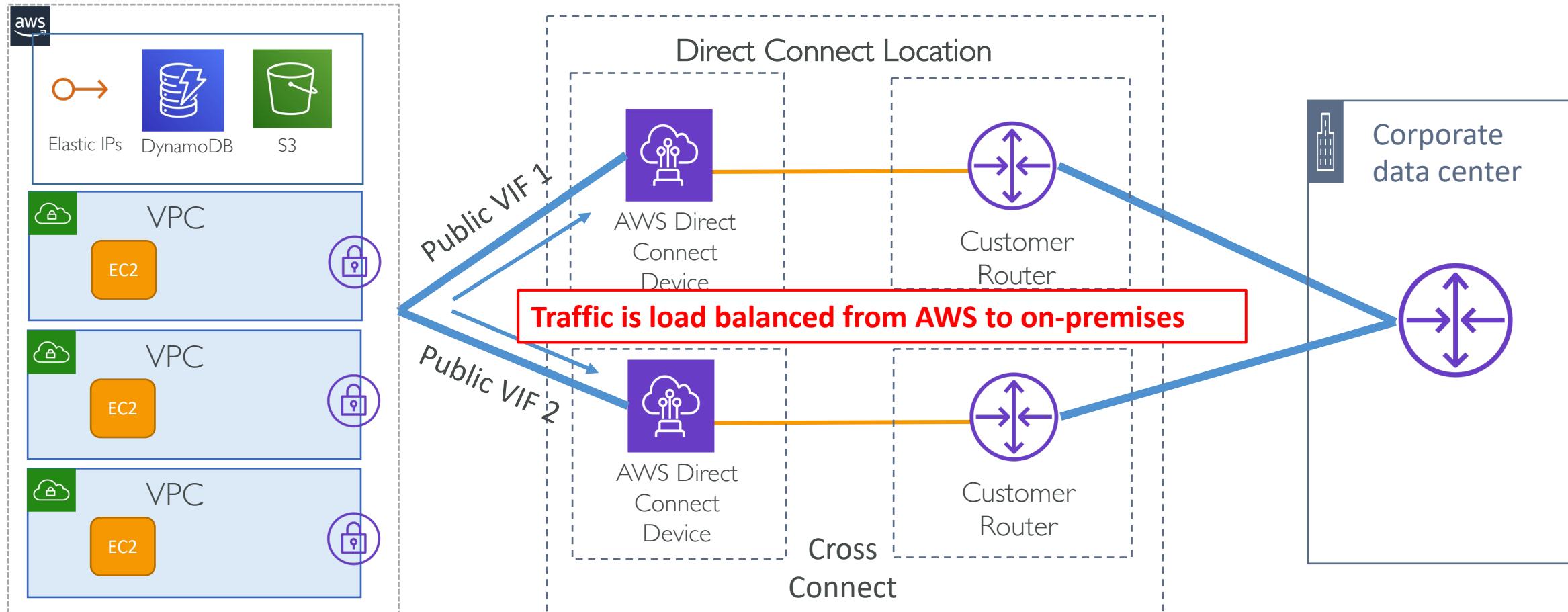
# Active-Active Connection using Public VIF

With **Public ASN**



# Active-Active Connection using Public VIF

With **Public ASN**

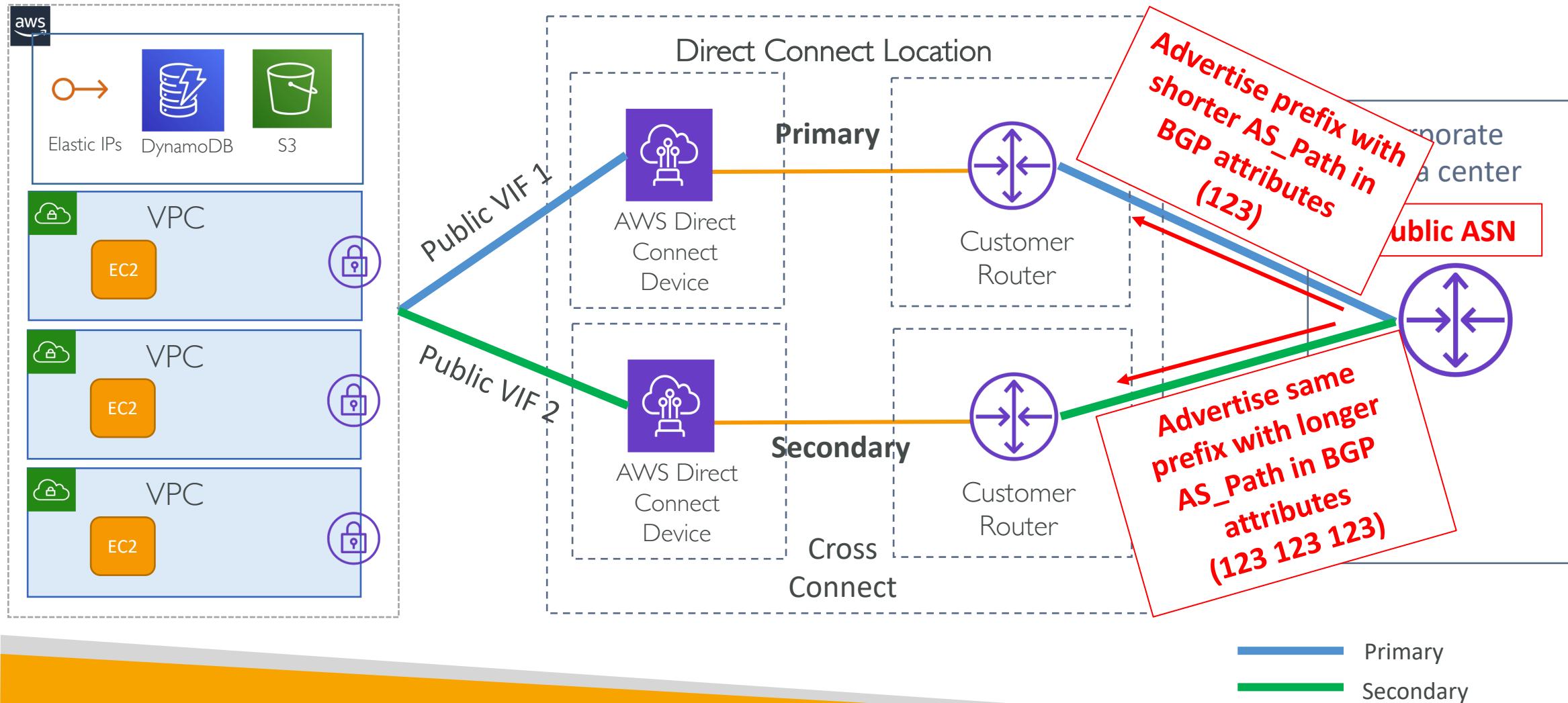


# Active-Passive Connection using Public VIF

- If using a public ASN:
  - Customer gateway to advertising the same prefix (public IP or network that you own) on both BGP sessions.
  - Start advertising the on-premises public prefixes with additional AS\_Path prepends in the BGP attributes for Secondary connection.
  - Increase the Local Preference (local-pref) to be sure that the on-premises router always chooses the correct path for sending traffic to AWS.
- If using a private ASN:
  - Use longer prefixes on primary connection.
  - Example: two prefixes (X.X.X.0/25 and X.X.X.128/25) on your primary connection and X.X.X.0/24 on secondary connection

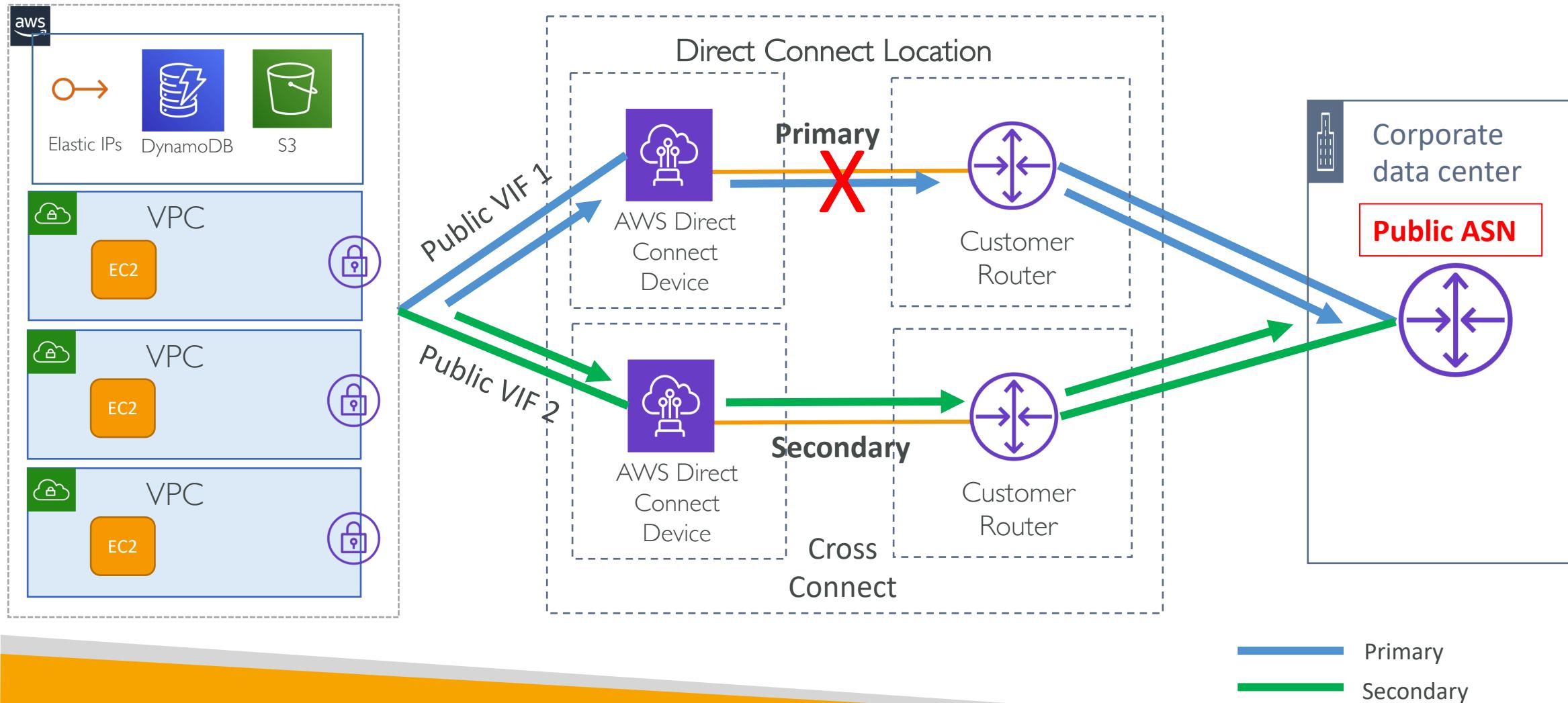
# Active-Passive Connection using Public VIF

With **Public ASN**



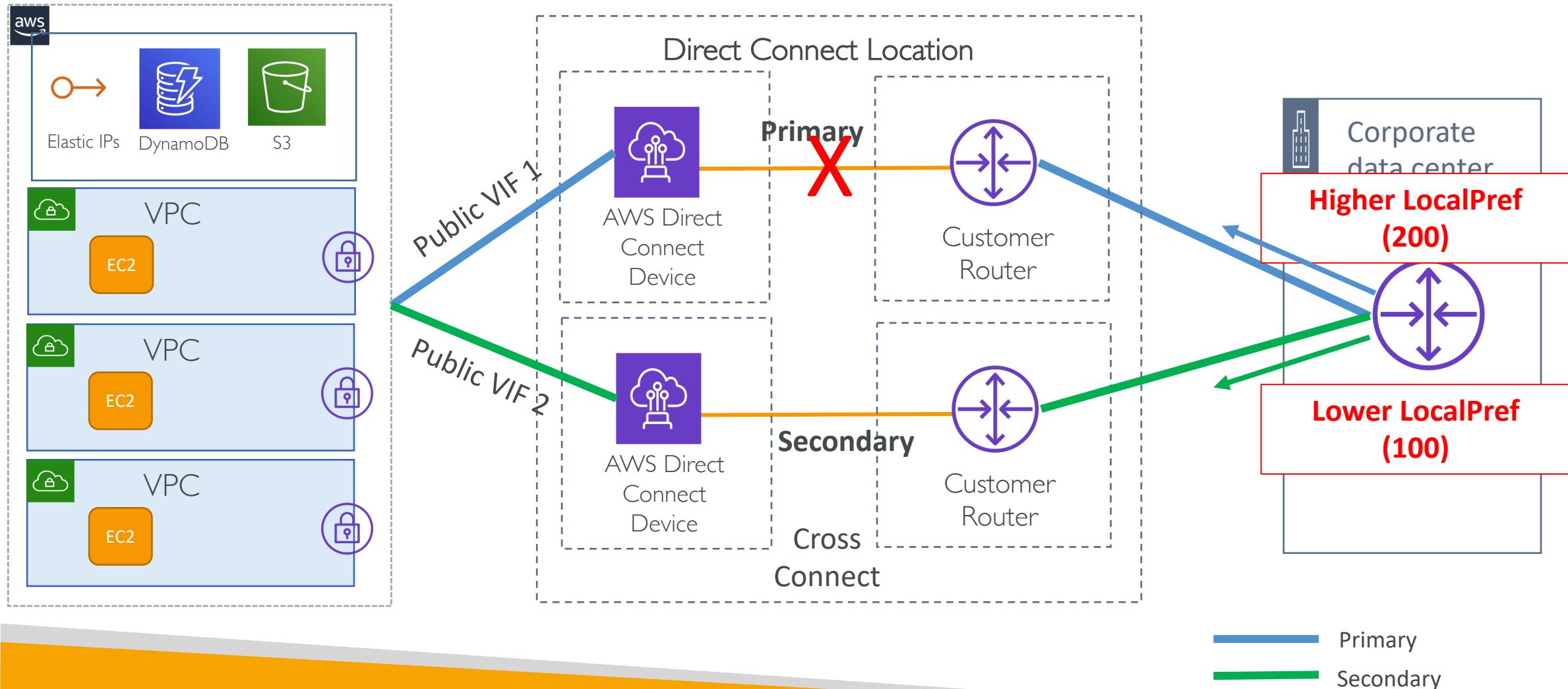
# Active-Passive Connection using Public VIF

With **Public ASN**



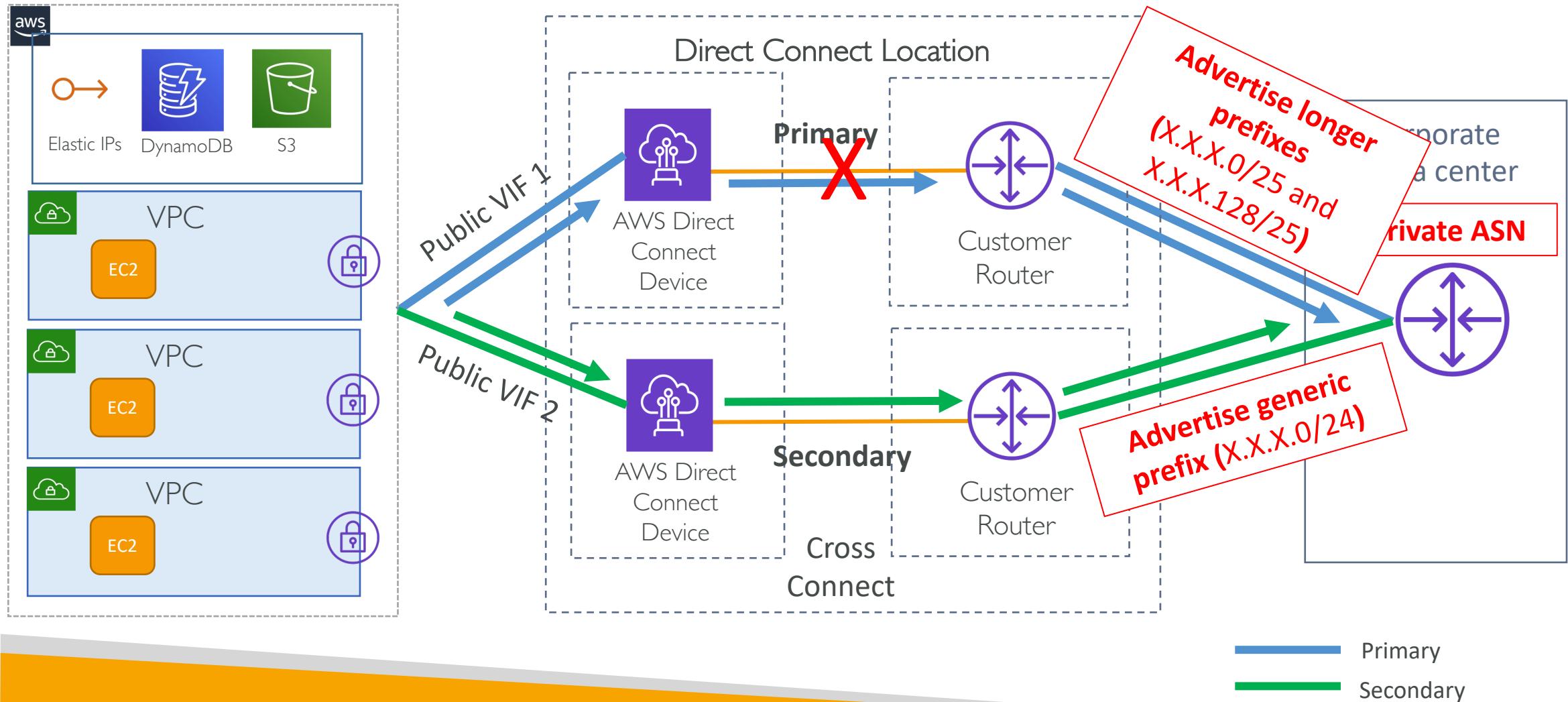
# Active-Passive Connection using Public VIF

With **Public ASN**



# Active-Passive Connection using Public VIF

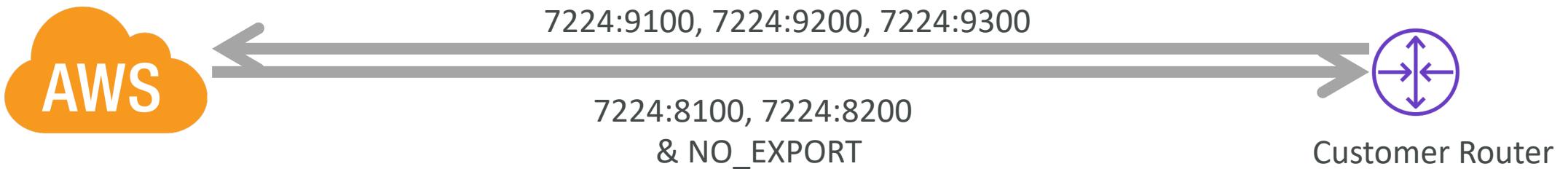
With **Private ASN**



# DX route advertisement scenarios for Public VIF using BGP communities

# Public VIF – BGP Communities

- BGP Community tags help control the scope for the advertisement of the prefixes (Regional or global)
- Supports scope BGP community tags
  - Inbound 7224:9100, 7224:9200, 7224:9300
  - Outbound 7224:8100, 7224:8200
- Supports NO\_EXPORT BGP community tag



# Public VIF BGP Communities - Inbound

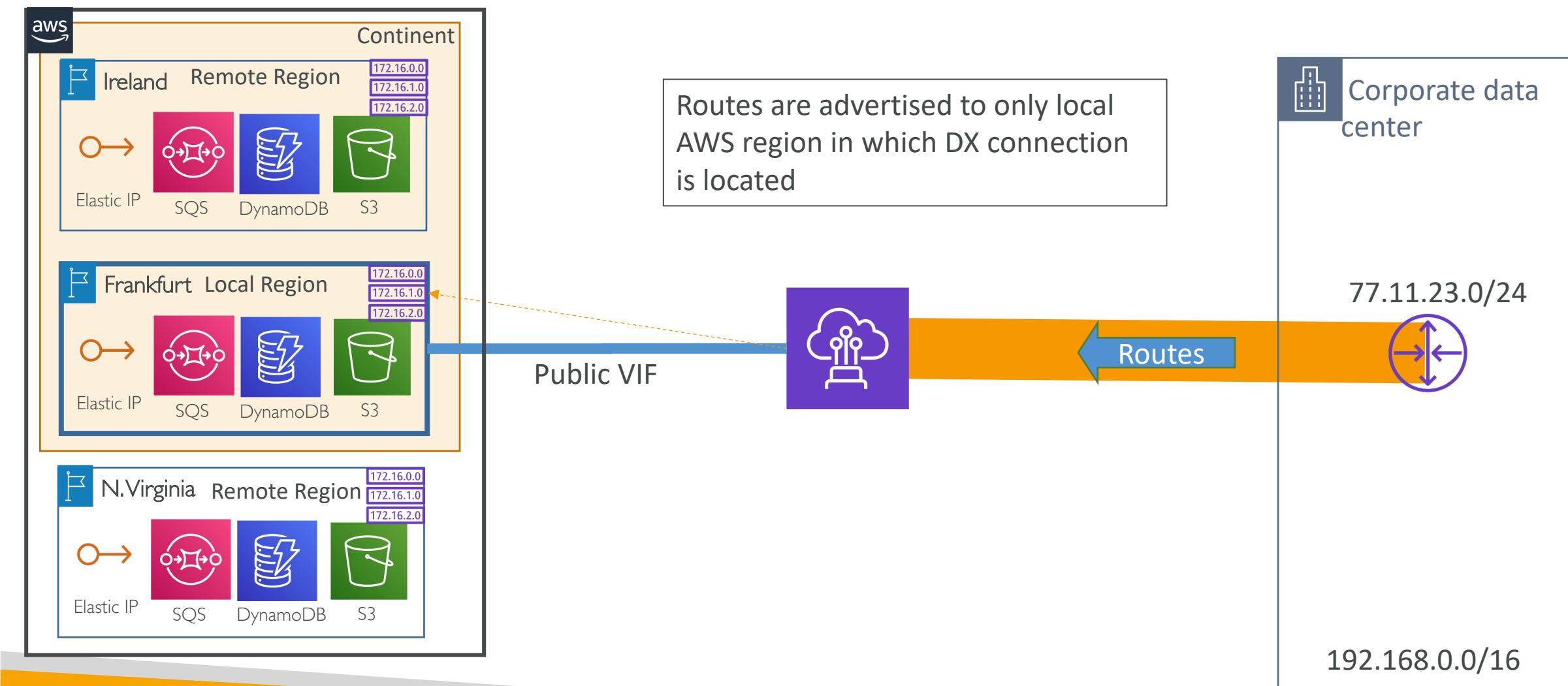
The prefixes received from customer on Public VIFs are advertised within AWS as per below BGP community tags

- 7224:9100—Local AWS Region
- 7224:9200—All AWS Regions for a continent
  - North America-wide
  - Asia Pacific
  - Europe, the Middle East and Africa
- 7224:9300—Global (all public AWS Regions)

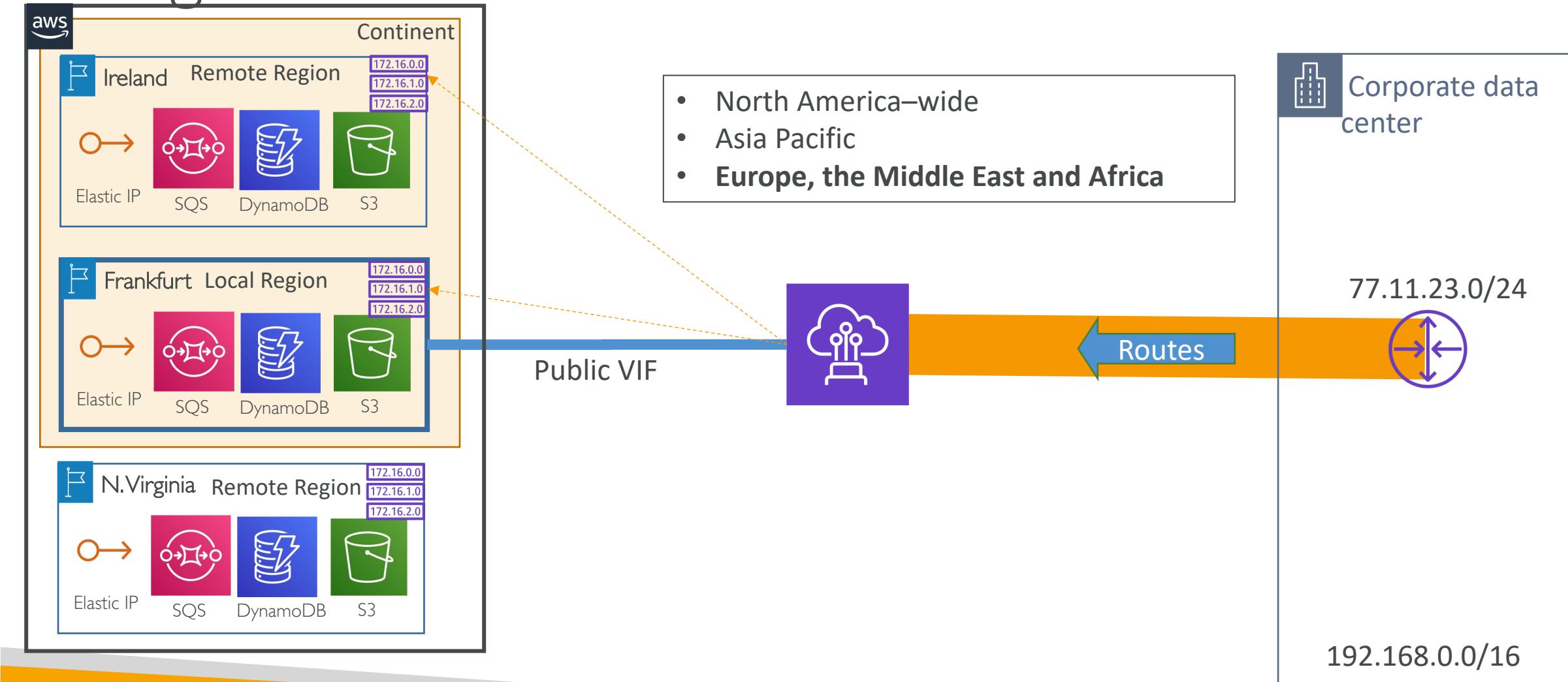


Default

# Public VIF BGP Communities Inbound – Local 7224:9100

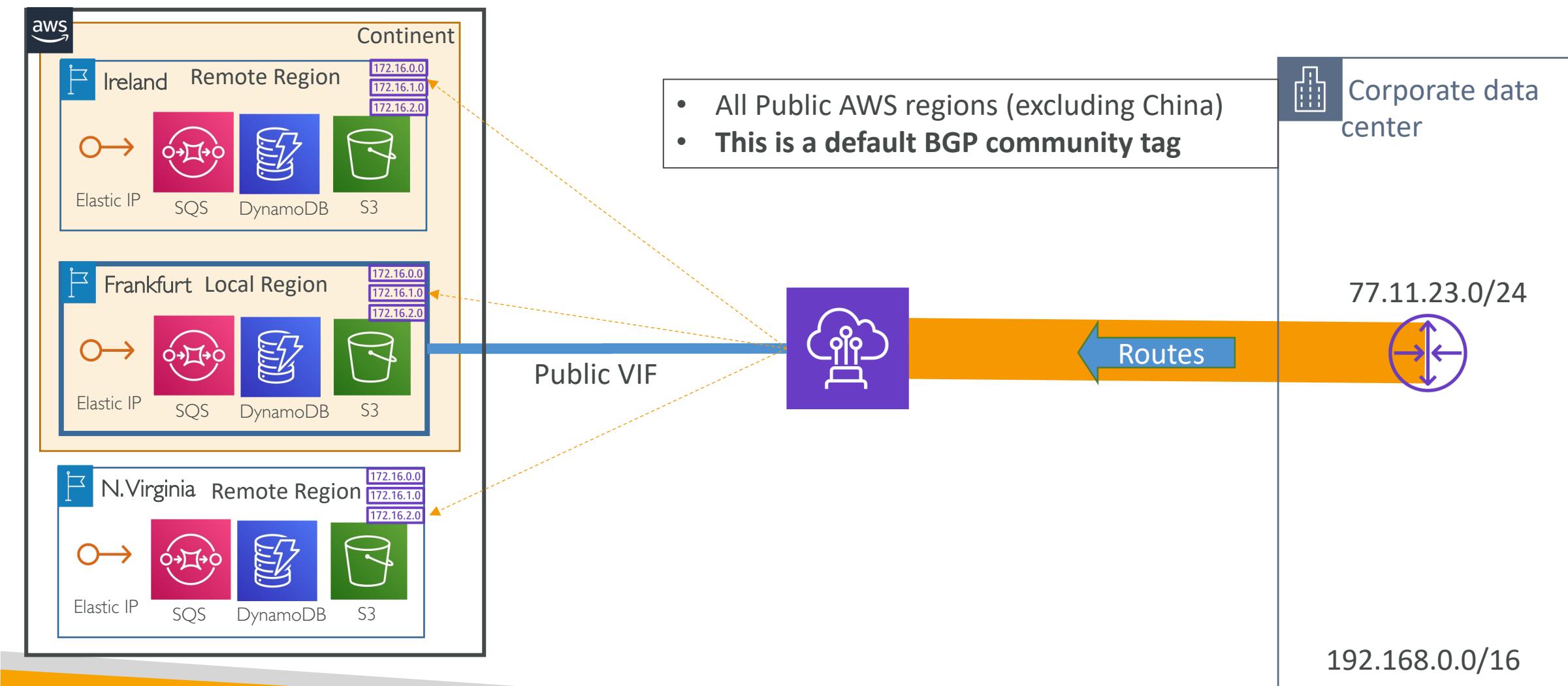


# Public VIF BGP Communities Inbound - All AWS regions for a continent 7224:9200



# Public VIF BGP Communities Inbound – Global

## 7224:9300

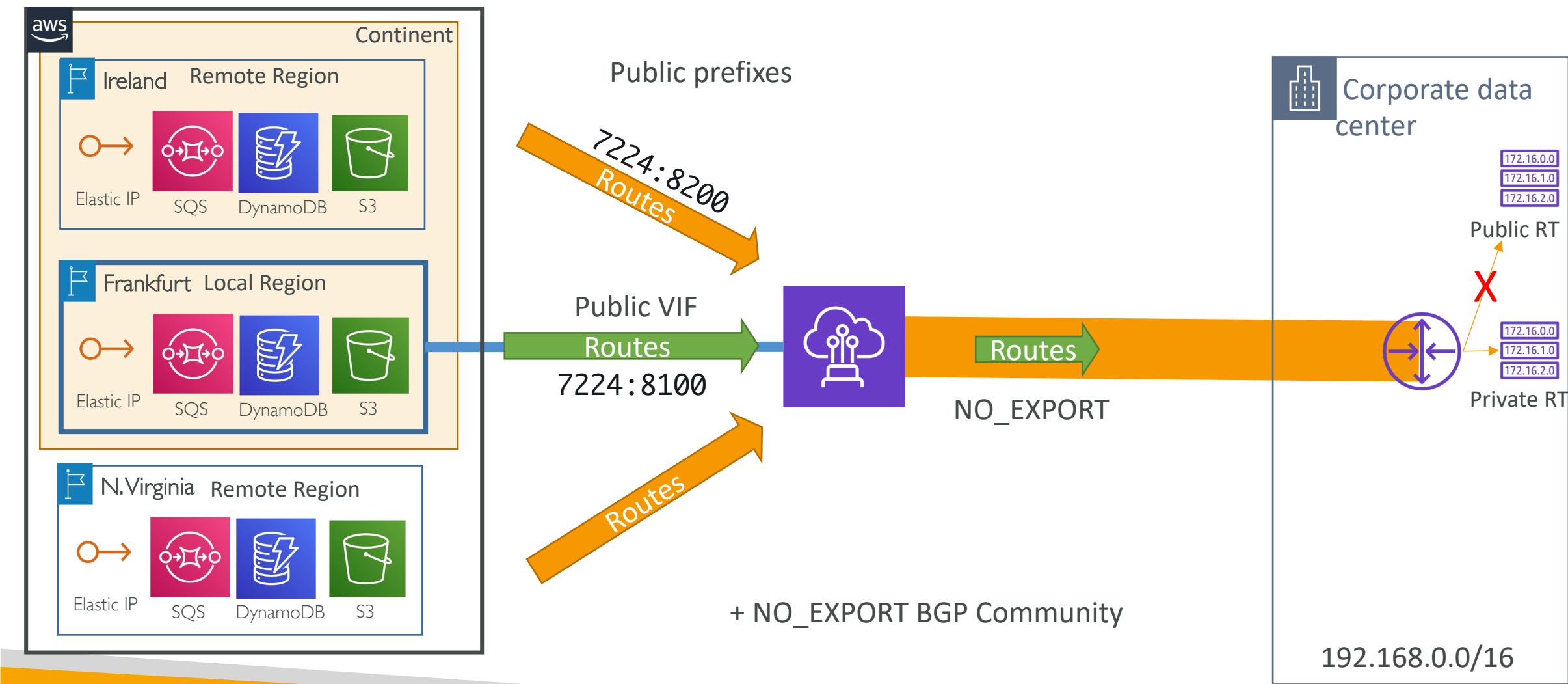


# Public VIF BGP Communities - Outbound

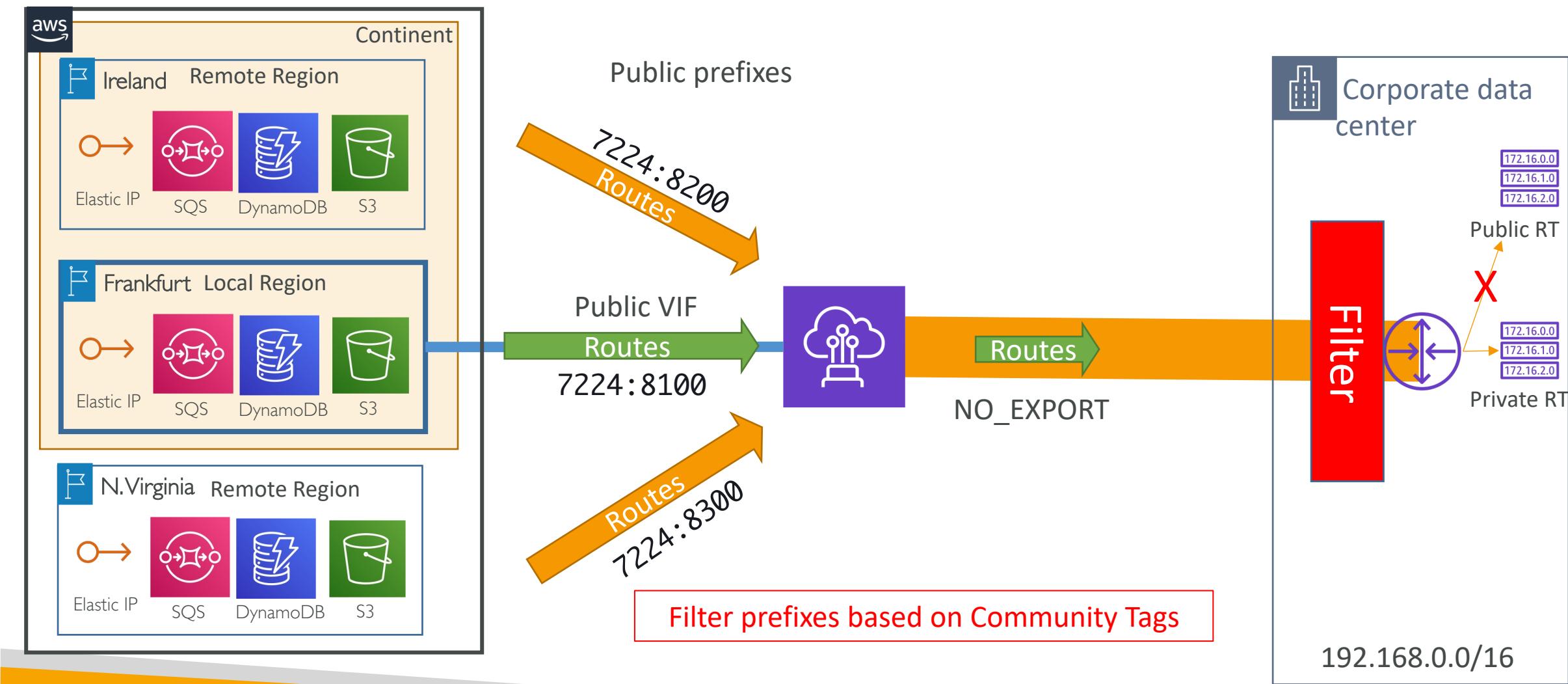
AWS applies following BGP community tags on the prefixes advertised based on the origin of the traffic at AWS end

- 7224:8100—Routes that originate from the same AWS Region in which the AWS Direct Connect point of presence is associated.
- 7224:8200—Routes that originate from the same continent with which the AWS Direct Connect point of presence is associated.
- No tag—Global (all public AWS Regions)
- Apart from above Tags, AWS Direct Connect also advertises NO\_EXPORT community tag for all routes
- Customer router can choose which prefixes to accept by filtering the routes based on BGP communities associates with the routes

# Public VIF BGP Communities Outbound



# Public VIF – Route filtering



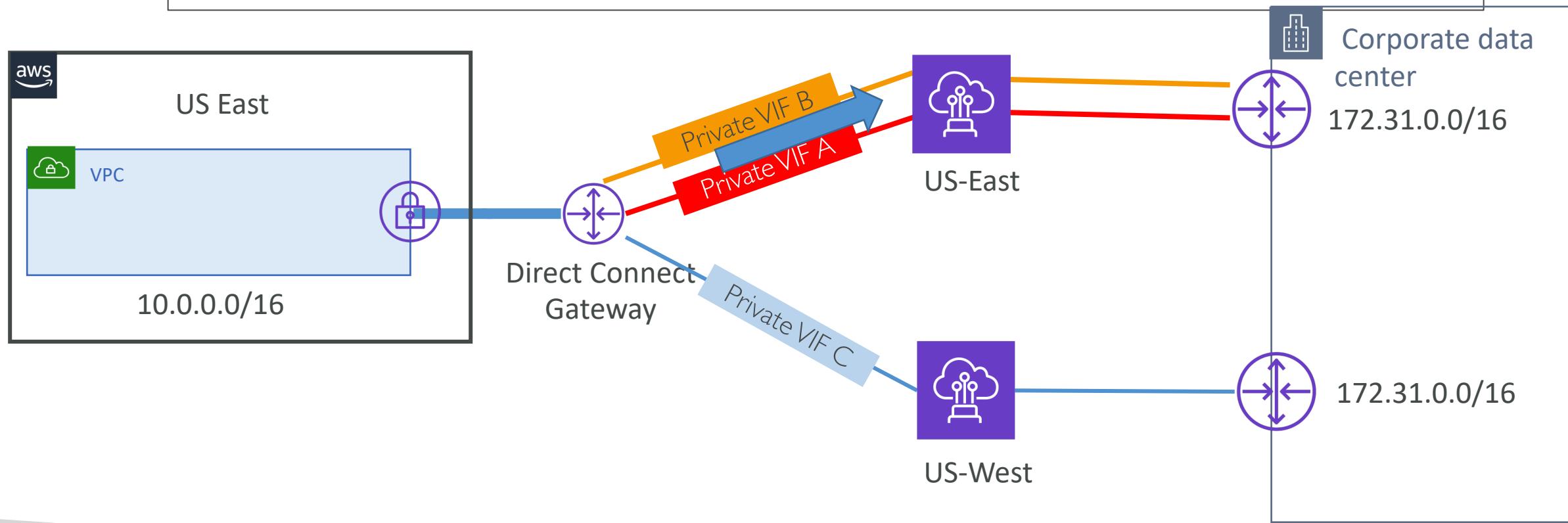
# Private VIF Routing Policies & BGP communities

# Private VIF Routing Policies

- AWS evaluates the longest prefix match first
- If prefix is same, AWS uses the distance from the local Region to the AWS Direct Connect location to determine the virtual (or transit) interface for routing.
- This behavior can be modified by assigning local preference BGP communities (7224:7300 > 7224:7200 > 7224:7100)
- In case of multiple virtual interfaces in a same local region, Set the AS\_PATH attribute to prioritize which interface AWS uses to route the traffic.

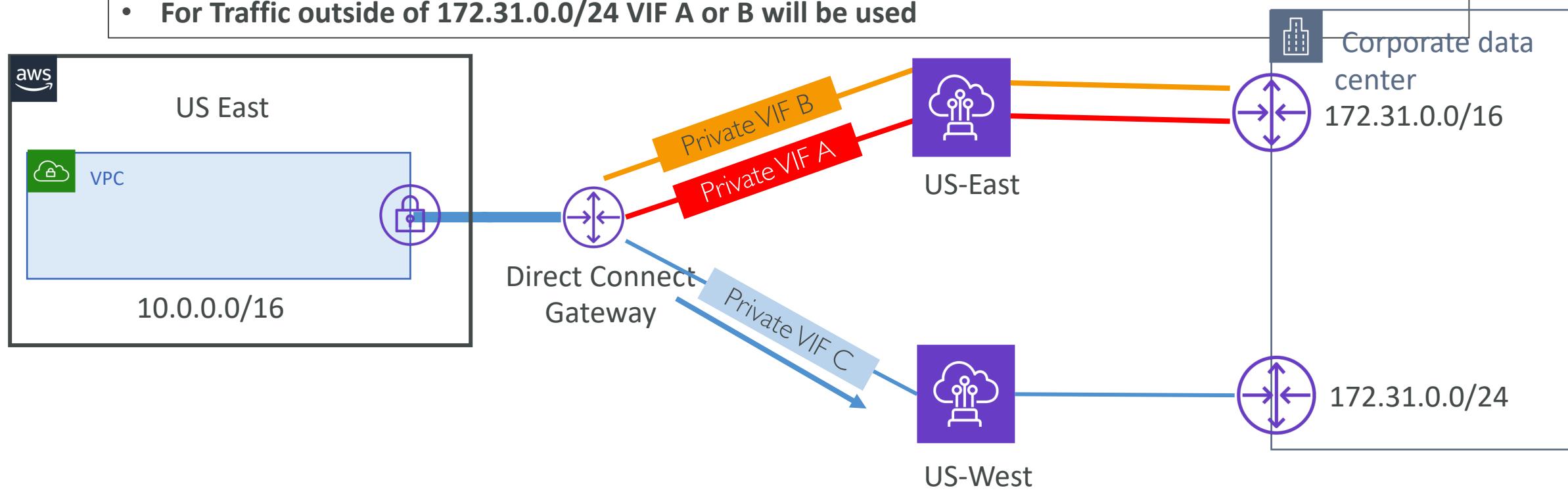
# Private VIF Routing policies – Home/Local region

- All VIFs advertises same Prefix – 172.31.0.0/16
- **US-East VIF A or B** will be used for the traffic originated or return traffic as distance is minimum
- The lower network cost takes the precedence



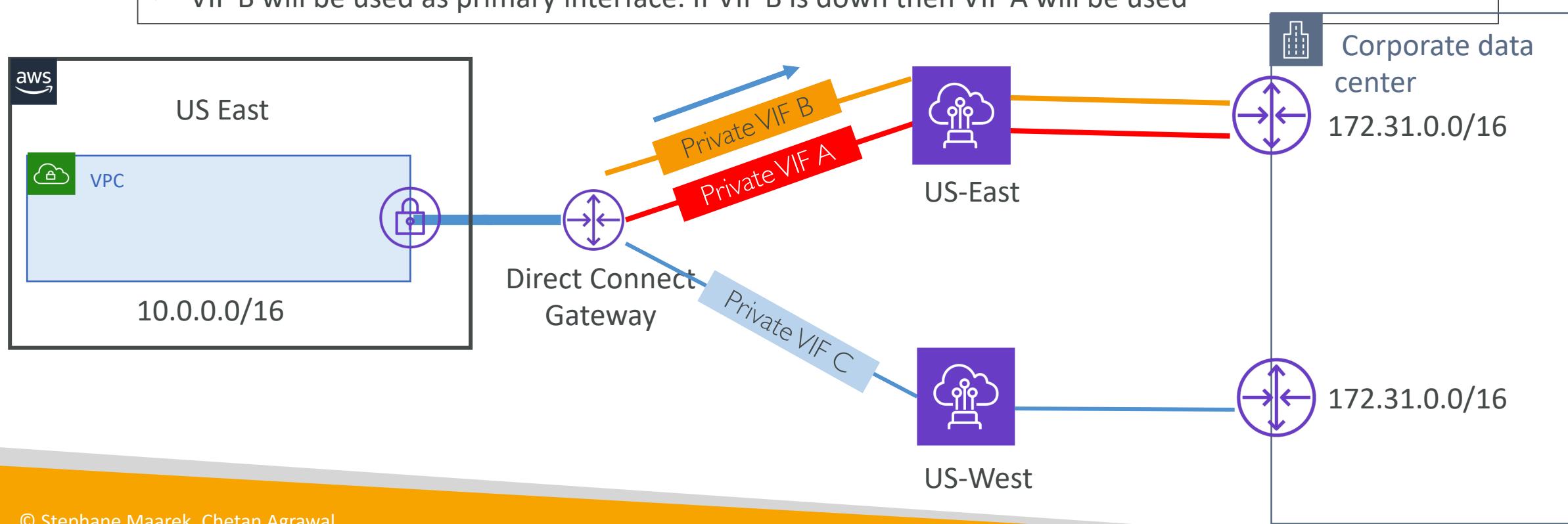
# Private VIF Routing policies – Longest Prefix

- VIFs A & B advertises same Prefix – 172.31.0.0/16
- VIF C advertises 172.31.0.0/24
- **VIF C will be used for the traffic originated or return traffic for destination 172.31.0.0/24**
- For Traffic outside of 172.31.0.0/24 VIF A or B will be used



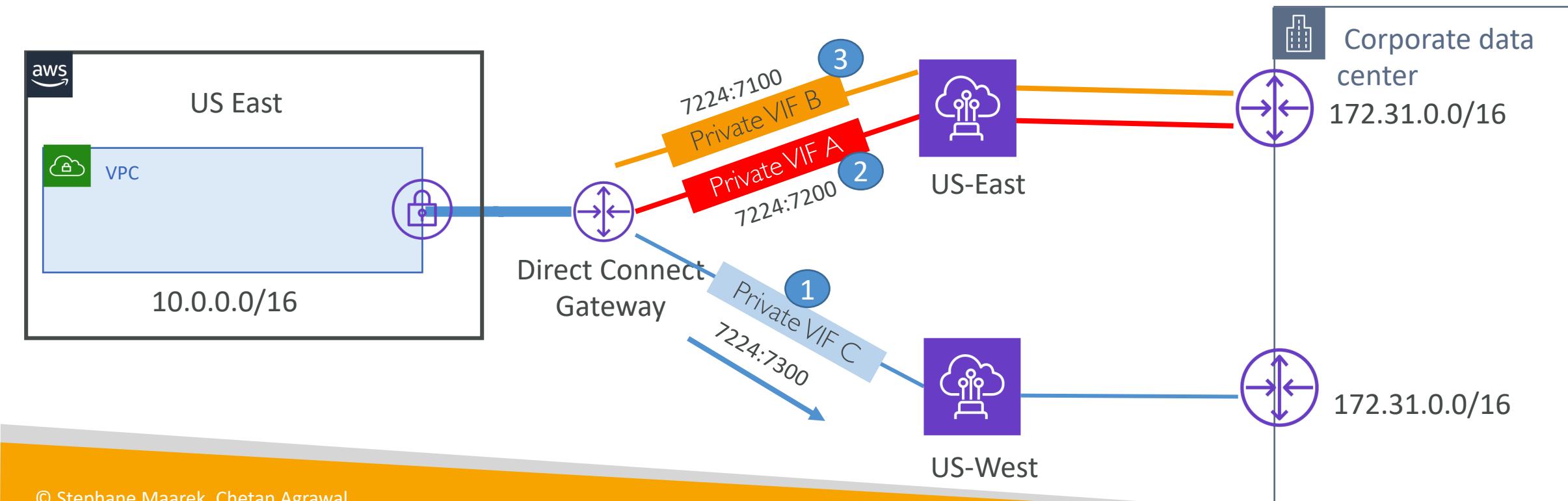
# Private VIF Routing policies – AS\_PATH

- All VIFs advertises same Prefix – 172.31.0.0/16
- **AS\_Path is in effect if the Direct Connect location is same**
- VIF A AS\_PATH attribute of 65001, 65001, 65001 VIF B: AS\_PATH attribute of 65001, 65001 and VIF C: AS\_PATH attribute of 65001
- Because US-East network cost is lower than US-West DX location – the VIF A or VIF B will be used
- VIF B will be used as primary interface. If VIF B is down then VIF A will be used



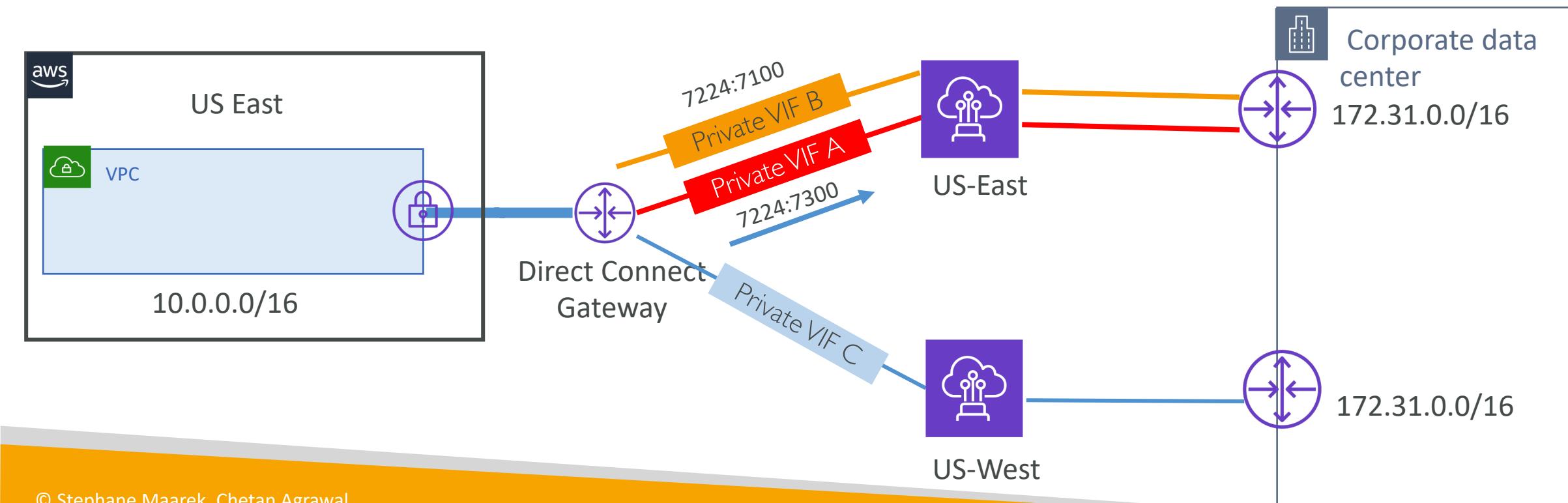
# Private VIF Routing policies – Local Preference BGP Community

- All VIFs advertises same Prefix – 172.31.0.0/16
- You can set Local preference BGP community tags.
- 7224:7100—Low preference, 7224:7200—Medium preference, 7224:7300—High preference
- Local preference community tags are evaluated before AS\_PATH



# Private VIF Routing policies – Local Preference BGP communities for Active/Passive traffic

- All VIFs advertises same Prefix – 172.31.0.0/16
- **VIF B: 7224:7100 (low preference), VIF A – 7224:7300 (high preference)**
- VIF A will be used as primary interface. If VIF A is down then VIF B will be used



# Routing precedence out of the VPC

+ Direct Connect

1. Local route to the VPC (no override with more specific routes)
2. Longest prefix match first
3. Static route table entries preferred over a dynamic/propagated routes
4. Dynamic routes
  1. Direct Connect BGP routes (over VPN BGP routes)
    1. Shorted AS\_PATH
    2. If equivalent, then load balance the traffic
  2. VPN Static routes (as defined on VPN connection)
  3. BGP routes from VPN
    1. Shorted AS\_PATH

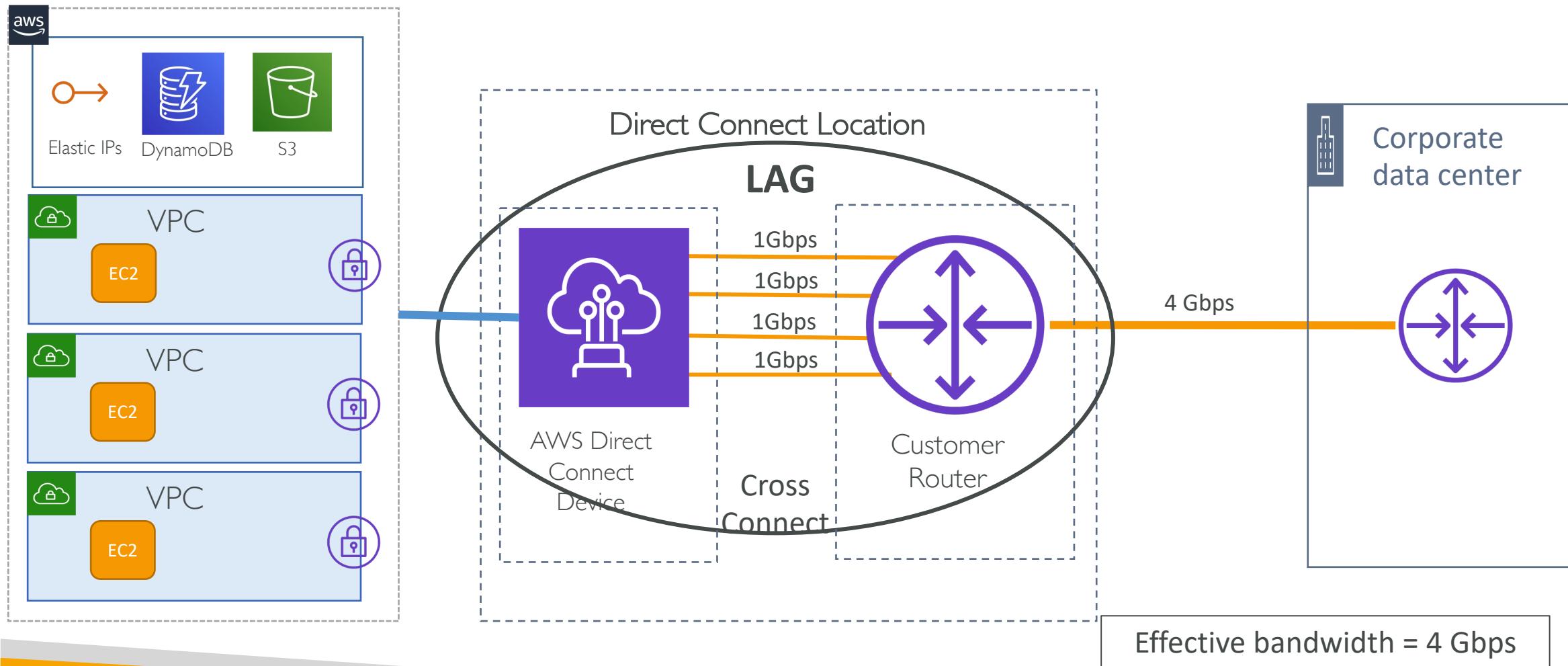
# Link Aggregation Group (LAG)

# Direct Connect Link Aggregation Groups (LAG)

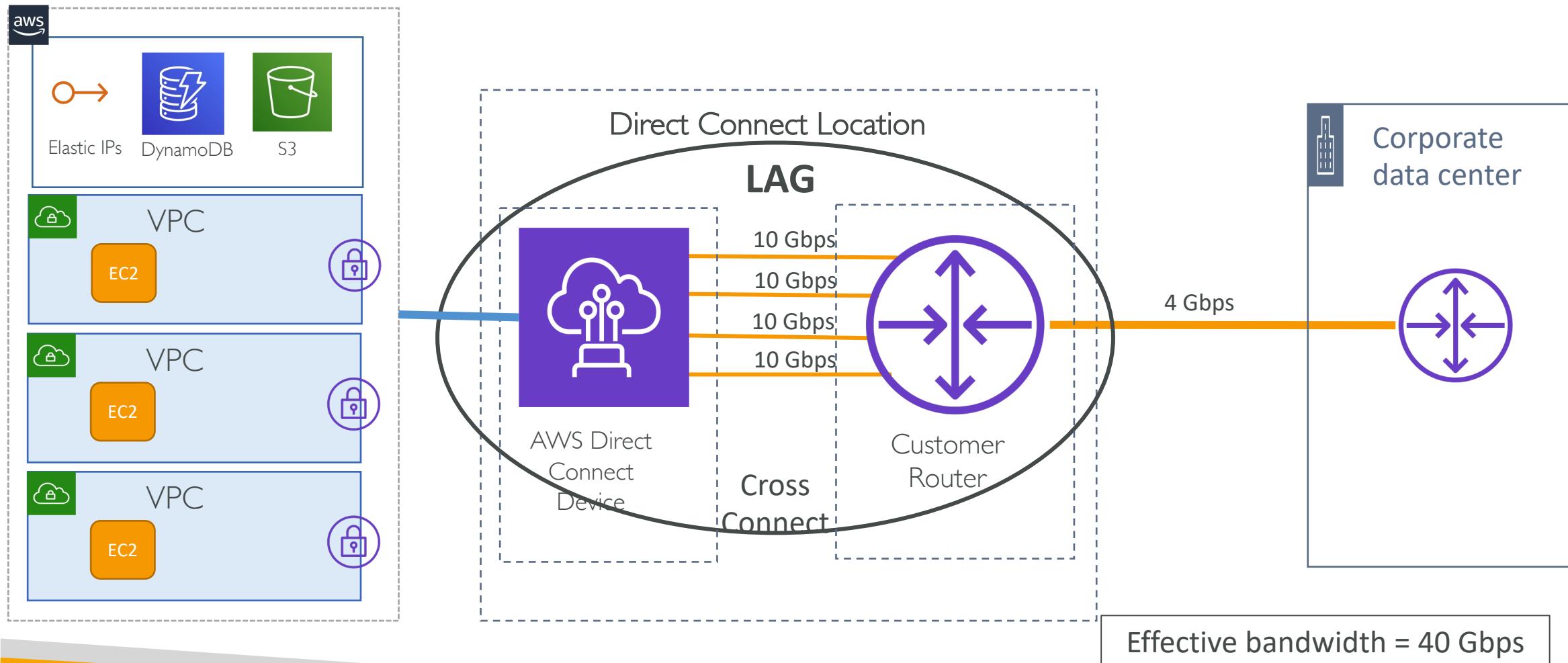
- Get **increased speed** and **failover** by summing up multiple Direct Connect connections into a **single logical connection**
- Uses Link Aggregation Control Protocol (LACP)
- All connections must be dedicated connections and have a port speed of 1 Gbps, 10 Gbps, or 100 Gbps.
- Can aggregate up to 4 (active active mode)
- Can use existing connection or add new connections to the LAG
- All connections in the LAG must have the same bandwidth
- All connections in the LAG must terminate at the same AWS Direct Connect Endpoint

<https://docs.aws.amazon.com/directconnect/latest/UserGuide/lags.html>

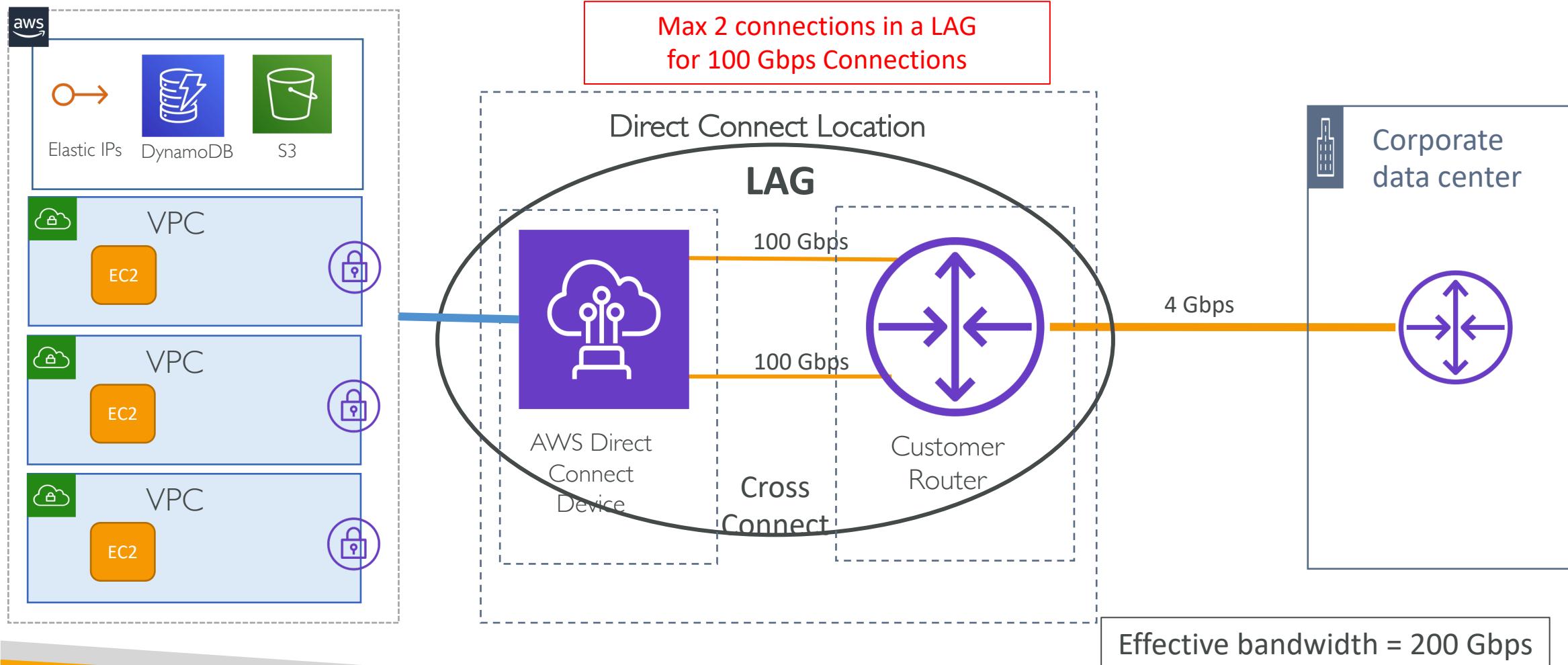
# LAG with 1 Gbps dedicated connections



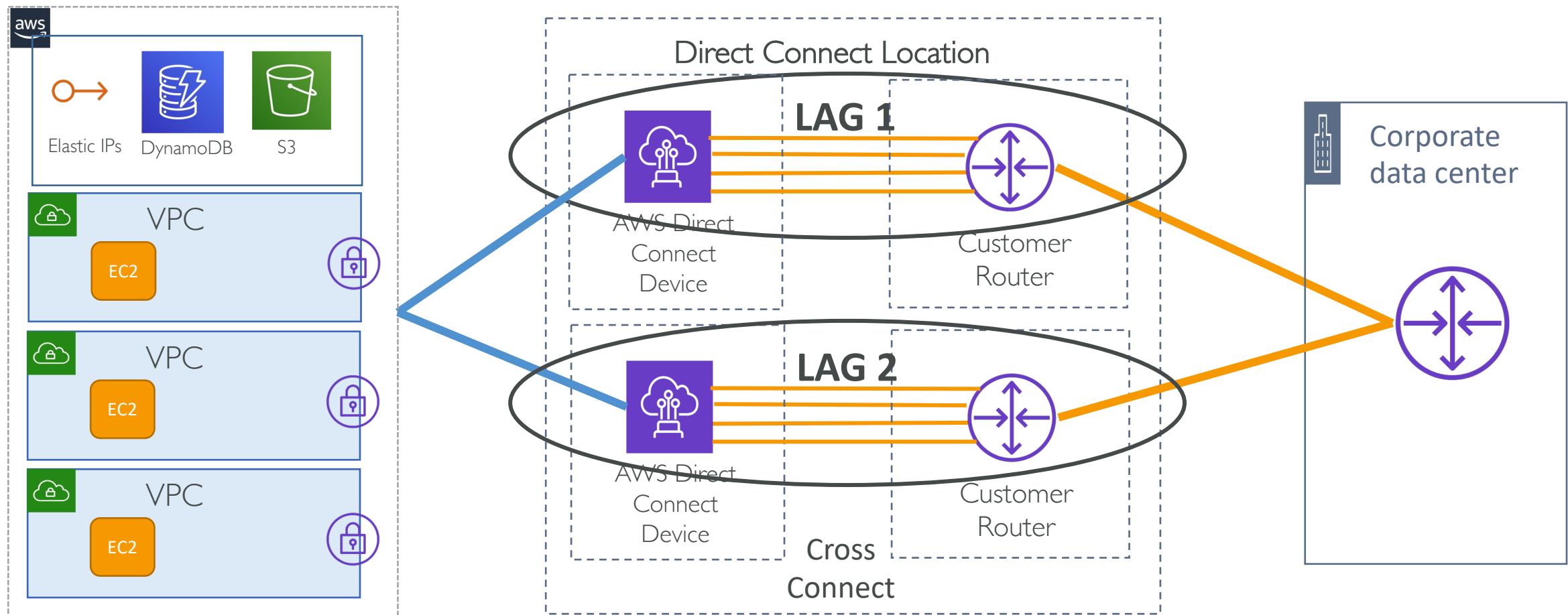
# LAG with 10 Gbps dedicated connections



# LAG with 100 Gbps dedicated connections



# LAG High Availability

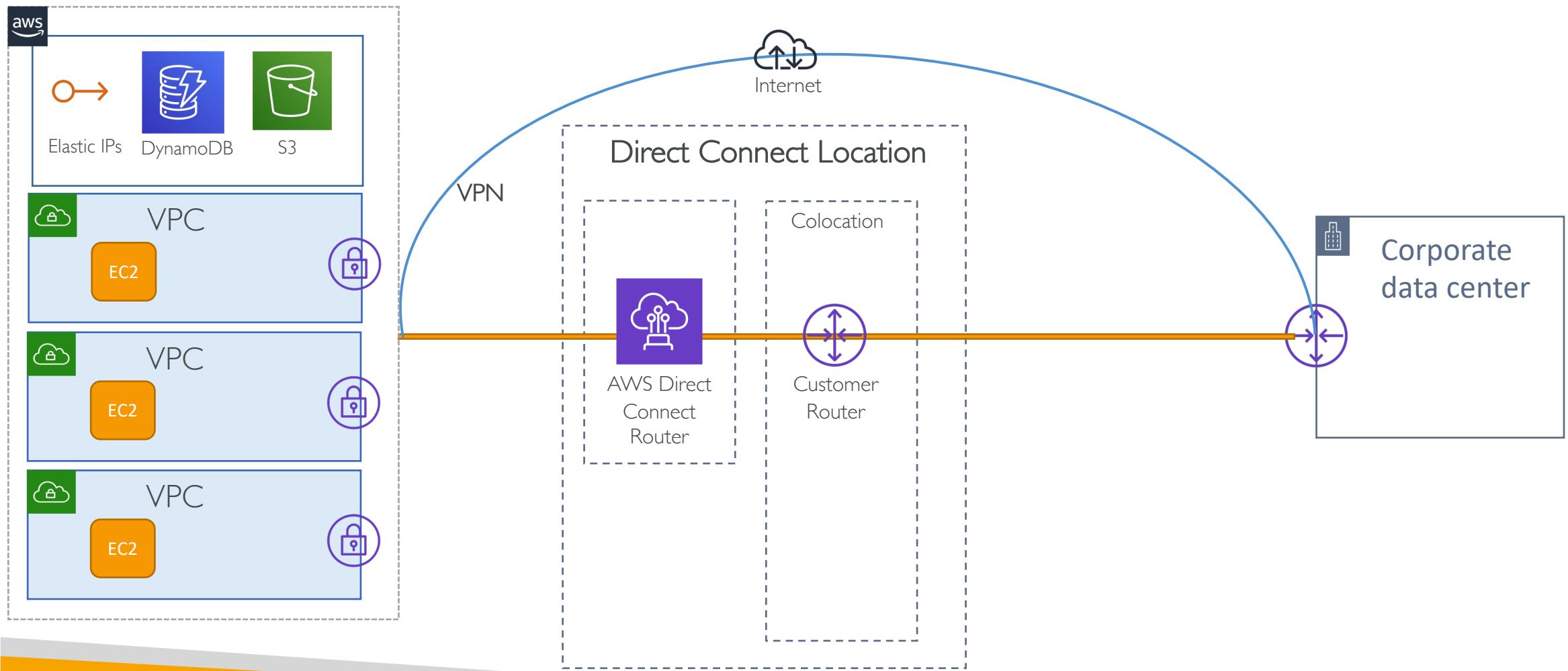


# LAG operational status

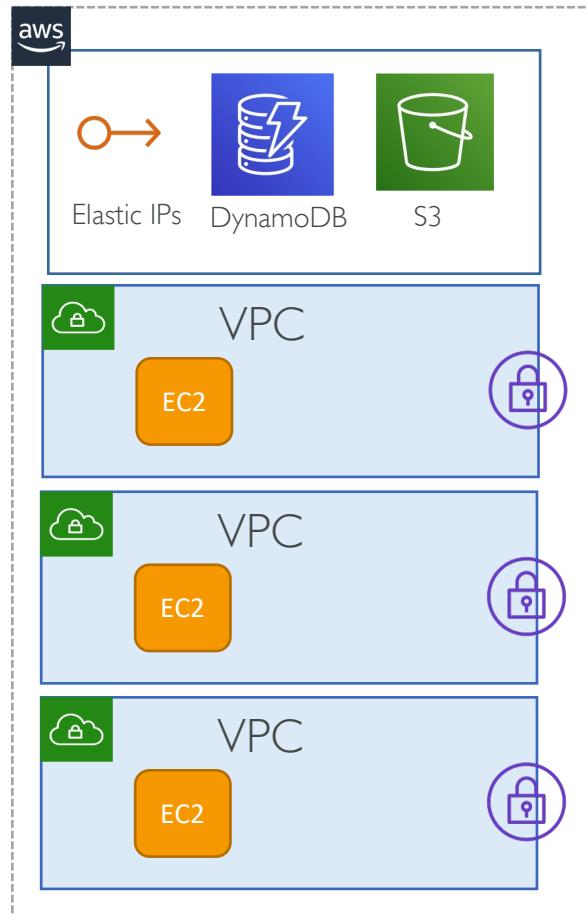
- All LAG connections operate in Active/Active mode
- LAG supports attribute to define minimum number of operational connections for the LAG to function (Default value = 0)
- If there are say 4 connections in the LAG and operational connection attribute value = 1 then LAG will be **operational** even if 3 connections are down
- If there are say 4 connections in the LAG and operational connection attribute value = 3 then LAG will be **non-operational** if 2 or more connections are down
- This attribute prevents over-utilization of active LAG connections in case of failures in other connections.

# Resilient DX Connections

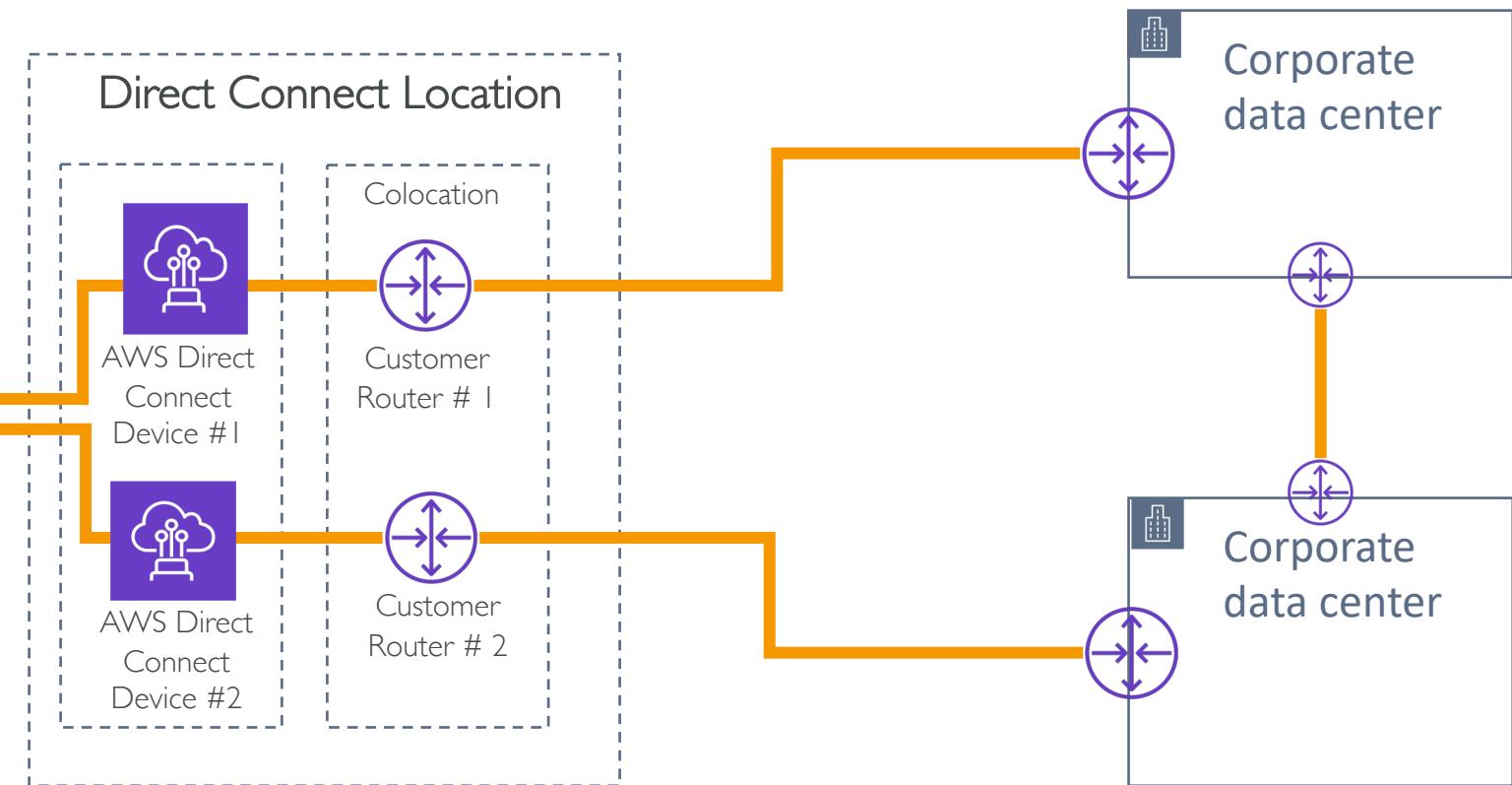
# Single DX Connection:VPN as a backup



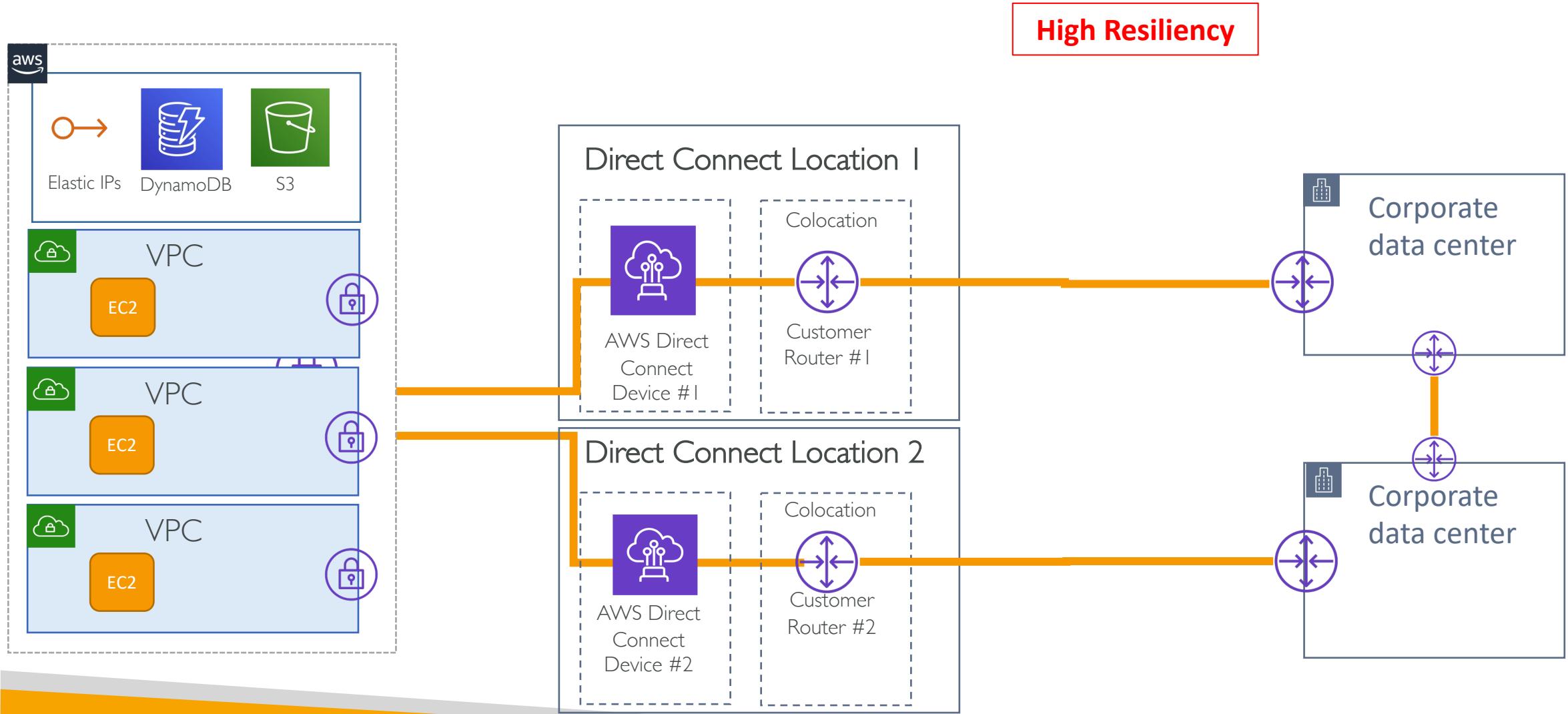
# Dual DX Connections – Dual devices



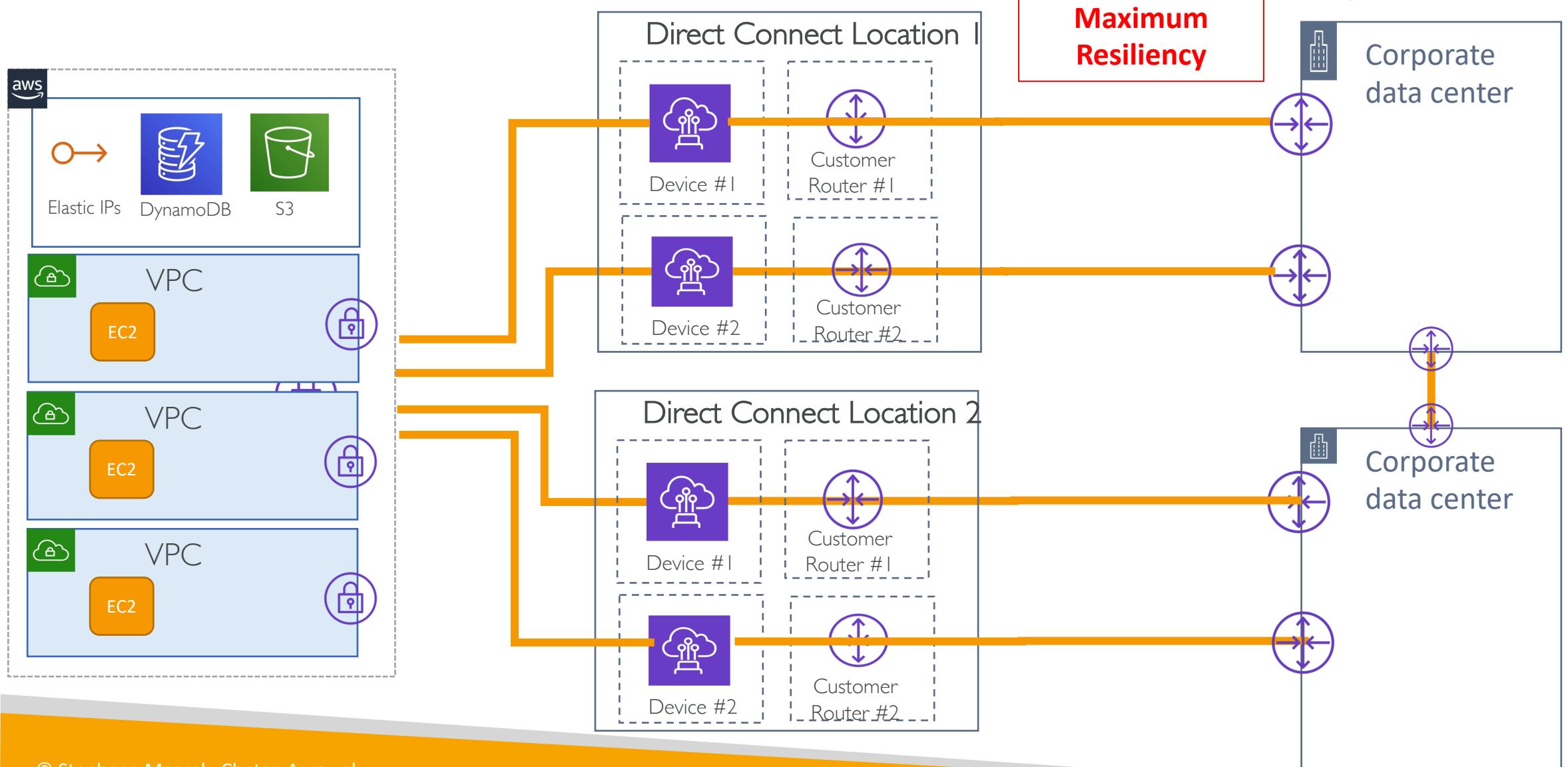
Make sure that both VIFs terminate on different AWS devices. For this check the AWS device IDs by opening the [Direct Connect console](#), and then choose Connections.



# Dual DX Connections – Dual locations



# Dual locations with DX connection backup



# Using DX console for Resiliency

- Use AWS DirectConnect console to choose between Classic or Connection Wizard
- Classic allows you to create a single connection
- Connection Wizard allows to choose from three types of resiliency options
  - Maximum Resiliency
  - High Resiliency
  - Development and Test

**Connection ordering type**

Connection ordering type

Classic  
Create connections one at a time. Best for augmenting an existing setup.

Connection wizard  
Create connections using our resiliency recommendations. Recommended for new setups.

**Resiliency level**

Maximum Resiliency  
Maximum Resiliency for Critical Workloads

High Resiliency  
High Resiliency for Critical Workloads

Development and Test  
Non Critical Workloads or Development Workloads

**Maximum Resiliency**

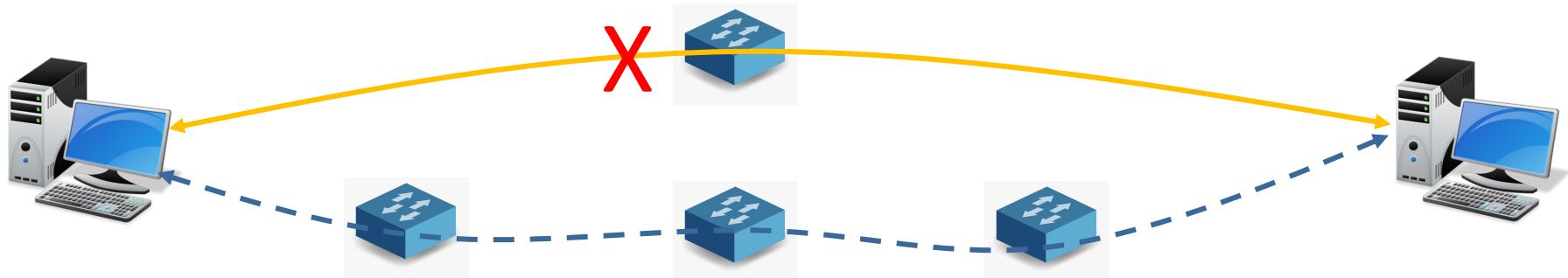
You can achieve maximum resiliency for critical workloads by using separate connections that terminate on separate devices in more than one location (as shown in the figure). This topology provides resiliency against device, connectivity, and complete location failures.

```
graph LR; subgraph AWS_Region [AWS Region]; direction TB; A((Cloud icon)); end; A --> B1[AWS Direct Connect Location - 1]; A --> B2[AWS Direct Connect Location - 2]; B1 --> C1[Customer data center]; B1 --> C2[Customer data center]; B2 --> C3[Customer data center]; B2 --> C4[Customer data center];
```

# DirectConnect fast failover using BFD

# BFD – Bidirectional Forwarding Detection

- It's a simple Hello Network Protocol
- Lowers the network failure detection time between the neighboring peers
- BFD control packets are transmitted and received periodically between the BFD peers (Asynchronous mode)
- Neighbors can use dynamic routing protocol or static routes (BGP/OSPF)
- Provides the detection time less than 1 sec



# Using BFD with Direct Connect

- While using multiple DX connections or using VPN backup, its important to have fast failover to redundant connection
- By default, BGP waits for three keep-alives to fail at a hold-down time of **90 seconds**.
- AWS enables BFD by default on DX connections. However BFD should be configured on customer side of the router
- AWS sets the BFD liveness detection interval to 300 ms and BFD liveness detection count to 3 which results in **failover under 1 second**
- Cisco Router example:  
*bfd interval 300 min\_rx 300 multiplier 3*

# DirectConnect Security

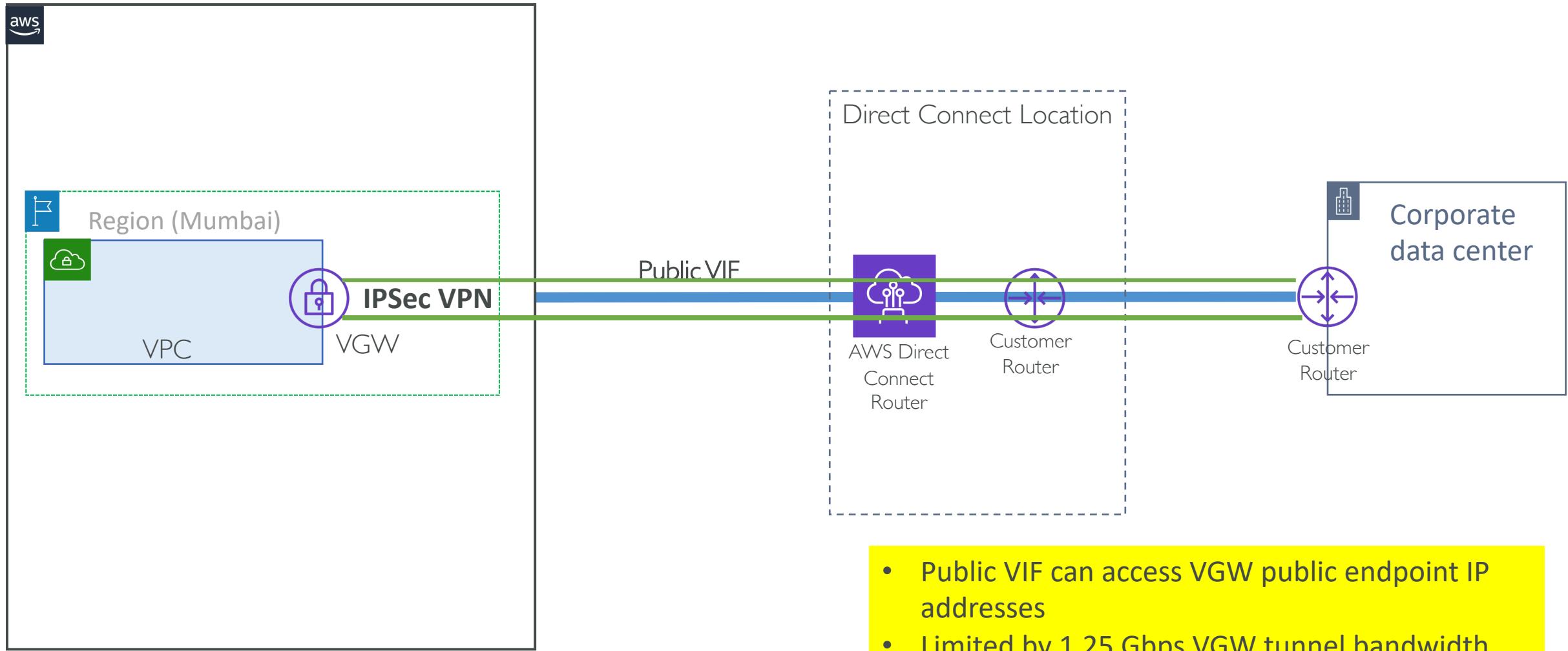
# DX network traffic security

- Layer3 VPN over a Direct Connect connection
- Layer2 encryption using MACSec

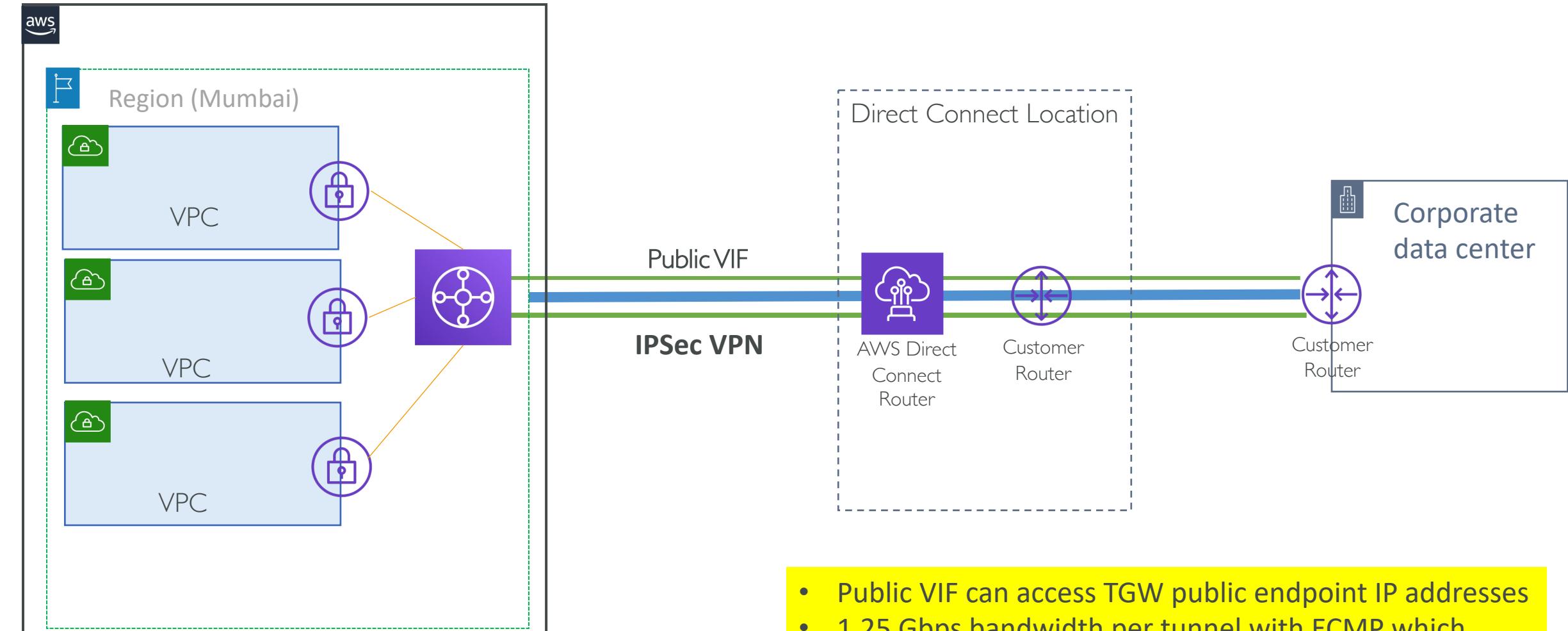
# Encrypting DX traffic

- DirectConnect traffic is not encrypted
- Ideal way to have the traffic encrypted is to have layer 4 (TLS) encryption between hosts communicating over Direct Connect
- If its still required to have Layer 3 encryption, set up a VPN over Direct Connect connection using Public VIF
- AWS publishes Public IPs which also includes the Public IPs of AWS managed VPN (VGW) and Transit Gateways (TGW)
- Set up IPSec tunnels using Public IPs of VGW or TGW on AWS end and customer router Public IP on the other end

# VPN over a DX using VGW (Layer3)



# VPN over a DX using TGW (Layer3)



- Public VIF can access TGW public endpoint IP addresses
- 1.25 Gbps bandwidth per tunnel with ECMP which allows bandwidth aggregation

# MAC Security (MACSec) – Layer2

- Mac Security (MACSec) is an IEEE 802.1 Layer 2 standard that provides data confidentiality, data integrity, and data origin authenticity
- MACSec provides Layer2 security for Dedicated connection.
- MACSec is available for certain Direct Connect partners only. Currently its not supported by all Direct connect partners.
- Make sure that you have a device on your end of the connection that supports MACSec
- When you configure your connection, you can specify one of the following values: MACSec – should\_encrypt, must\_encrypt, no\_encrypt

[Read more about MACSec](#)

# DirectConnct MTU & Jumbo frames

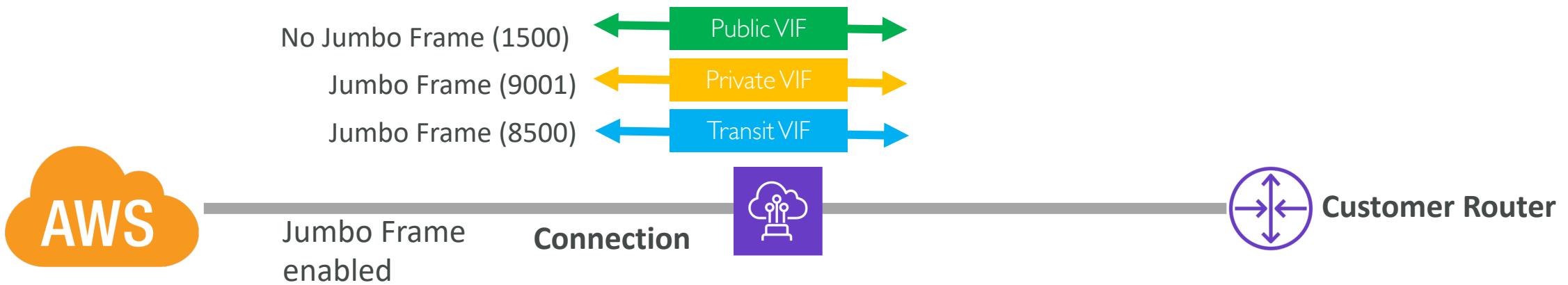
# MTU and Jumbo Frames

- MTU Recap: The size in bytes of the largest permissible packet that can be passed over the connection.
- Until Oct 2018, Direct connect used to support MTU of 1500 bytes
- Now Direct connect supports MTU up to 9001 bytes i.e. Jumbo Frames



# MTU and Jumbo Frames

- Public VIFs don't support Jumbo Frames
- Transit VIFs support MTU 1500 or 8500
- Private VIFs support MTU 9001
- For supporting jumbo frames the Direct connect connection or a LAG must be jumbo frame capable



# MTU and Jumbo Frames

- Jumbo frames apply only to propagated routes from DX (traffic routed through the static routes is sent using 1500 MTU)

| Destination    | Target   | Route Type |
|----------------|----------|------------|
| 10.10.0.0/16   | Local    | Static     |
| 0.0.0.0/0      | igw-xxxx | Static     |
| 192.168.0.0/24 | vgw-xxxx | Propagated |
| 192.168.1.0/24 | vgw-xxxx | Propagated |
| 192.168.2.0/24 | vgw-xxxx | Propagated |

172.16.0.0  
172.16.1.0  
172.16.2.0

# DirectConnect Monitoring

# Monitoring

- CloudWatch monitors DX connection and Virtual interfaces



# DX Connection CW Metrics

|                           |   |
|---------------------------|---|
| ConnectionState           | 1 indicates the connection is up and 0 indicates down               |
| ConnectionBpsEgress       | Bitrate for outbound data from the AWS side                         |
| ConnectionBpsIngress      | Bitrate for inbound data into the AWS                               |
| ConnectionPpsEgress       | Packet rate for outbound data from the AWS side                     |
| ConnectionPpsIngress      | Packet rate for inbound data into the AWS                           |
| ConnectionErrorCount      | The total error count for all types on the AWS device               |
| ConnectionLightLevelTx    | Health of the fiber connection for outbound (egress) from AWS       |
| ConnectionLightLevelRx    | Health of the fiber connection for inbound (ingress) traffic to AWS |
| ConnectionEncryptionState | 1 indicates the encryption is up and 0 indicates down (MACSec)      |

# DXVirtual Interface CW Metrics

`VirtualInterfaceBpsEgress`

Bitrate for outbound data from the AWS side of the virtual interface

`VirtualInterfaceBpsIngress`

Bitrate for inbound data to the AWS side of the virtual interface

`VirtualInterfacePpsEgress`

Packet rate for outbound data from the AWS side of the virtual interface

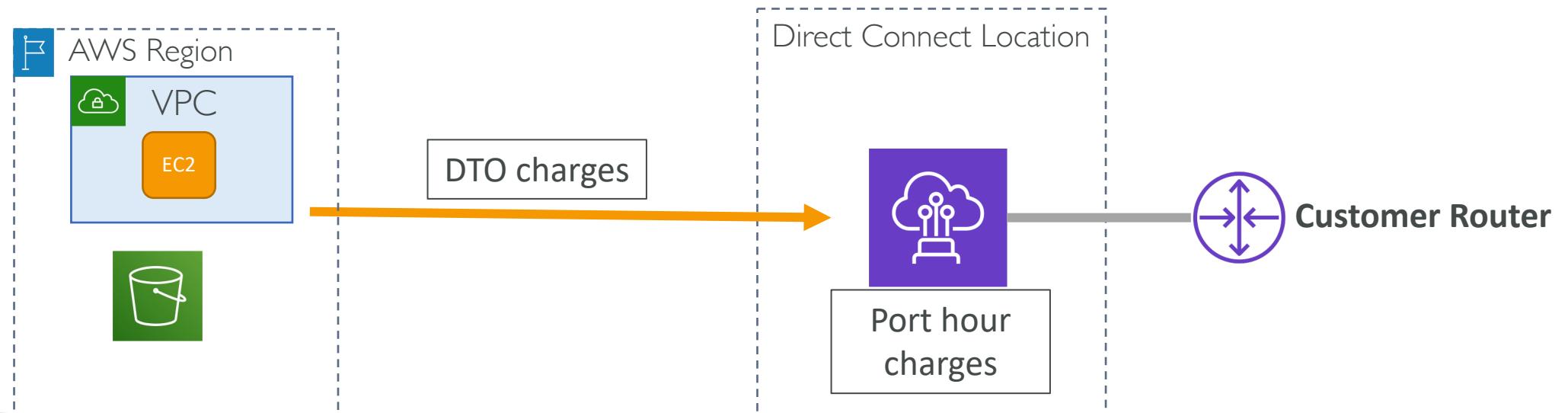
`VirtualInterfacePpsIngress`

Packet rate for inbound data to the AWS side of the virtual interface

# Direct Connect Billing

# Direct Connect Billing

- Port hour charges as per the DX connection type & capacity
  - Dedicated Connection
  - Hosted Connection
- Data transfer out charges – per GB depending on the DX location and source AWS region



# Port hour charges for Dedicated Connection

- Consistent across all DX locations (except Japan)

| Capacity | Port-Hour rate (All AWS Direct Connect locations except in Japan) | Port-hour rate in Japan |
|----------|---|-------------------------|
| 1G       | \$0.30/hour   | \$0.285/hour            |
| 10G      | \$2.25/hour   | \$2.142/hour            |
| 100G     | \$22.50/hour  | \$22.50/hour            |

# Port hour charges for Hosted Connection

| Capacity | Port-Hour rate (All AWS Direct Connect locations except in Japan) | Port-hour rate in Japan |
|----------|---|-------------------------|
| 50M      | \$0.03/hour   | \$0.029/hour            |
| 100M     | \$0.06/hour   | \$0.057/hour            |
| 200M     | \$0.08/hour   | \$0.076/hour            |
| 300M     | \$0.12/hour   | \$0.114/hour            |
| 400M     | \$0.16/hour   | \$0.152/hour            |
| 500M     | \$0.20/hour   | \$0.190/hour            |
| 1G*      | \$0.33/hour   | \$0.314/hour            |
| 2G*      | \$0.66/hour   | \$0.627/hour            |
| 5G*      | \$1.65/hour   | \$1.568/hour            |
| 10G*     | \$2.48/hour   | \$2.361/hour            |

# Data Transfer charges

- Data Transfer IN - \$0 per GB (free)
- Data Transfer OUT – Depends on DX location and Source AWS Region

| Data transfer from AWS region                           |   |                     |  |   |  |                             |                                    |                             |                             |                          |
|---|---|---------------------|--|---|--|-----------------------------|------------------------------------|-----------------------------|-----------------------------|--------------------------|
| AWS Direct Connect location in:                         | US East (Ohio),<br>US East<br>(Virginia), US<br>West (Northern<br>California), US<br>West (Oregon),<br>AWS GovCloud<br>(US-East), AWS<br>GovCloud (US-<br>West) | Canada<br>(Central) | From EU<br>(Frankfurt), EU<br>(Stockholm), EU<br>(Ireland), EU<br>(London), EU<br>(Paris), EU<br>(Milan) | Asia Pacific<br>(Tokyo), Asia<br>Pacific<br>(Osaka) | From Asia<br>Pacific (Seoul),<br>Asia Pacific<br>(Singapore),<br>Asia Pacific<br>(Hong Kong) | Asia<br>Pacific<br>(Mumbai) | South<br>America<br>(Sao<br>Paulo) | Asia<br>Pacific<br>(Sydney) | Middle<br>East<br>(Bahrain) | Africa<br>(Cape<br>Town) |
|   | United States   | \$0.0200            | \$0.0200   | \$0.0282  | \$0.0900   | \$0.0900                    | \$0.0850                           | \$0.1500                    | \$0.1300                    | \$0.1100                 |
| Canada  | \$0.0200  | \$0.0200            | \$0.0300   | \$0.0900  | \$0.0900   | \$0.0850                    | \$0.1500                           | \$0.1300                    | \$0.1100                    | \$0.1100                 |
| Europe  | \$0.0200  | \$0.0300            | \$0.0200   | \$0.0600  | \$0.0900   | \$0.0850                    | \$0.1107                           | \$0.1300                    | \$0.1000                    | \$0.1100                 |
| Japan   | \$0.0491  | \$0.0500            | \$0.0600   | \$0.0410  | \$0.0420   | \$0.1132                    | \$0.1700                           | \$0.1132                    | \$0.1500                    | \$0.1700                 |
| Hong Kong SAR, Malaysia, S.Korea,<br>Singapore & Taiwan | \$0.0491  | \$0.0500            | \$0.0600   | \$0.0410  | \$0.0410   | \$0.1000                    | \$0.1700                           | \$0.1107                    | \$0.1500                    | \$0.1600                 |
| India   | \$0.0600  | \$0.0600            | \$0.0625   | \$0.1132  | \$0.1107   | \$0.0450                    | \$0.1800                           | \$0.1400                    | \$0.1030                    | \$0.1400                 |

# Examples of DX port & DTO charges

Data Transfer OUT

Source Region:  
N. Virginia

Data Transfer IN

No Charge

Dest. DX location:  
Switch SUPERNAP  
Las Vegas

\$0.0200 / GB Out



S3



DynamoDB

Direct Connect Location

DX Port charges

**1G = \$0.30 /hour**

**10G = \$2.25 /hour**

**100G= \$22.5 /hour**

**\*All locations except Japan**



Dedicated Connections

# Examples of DX port & DTO charges

Data Transfer OUT

Source Region:  
N. Virginia

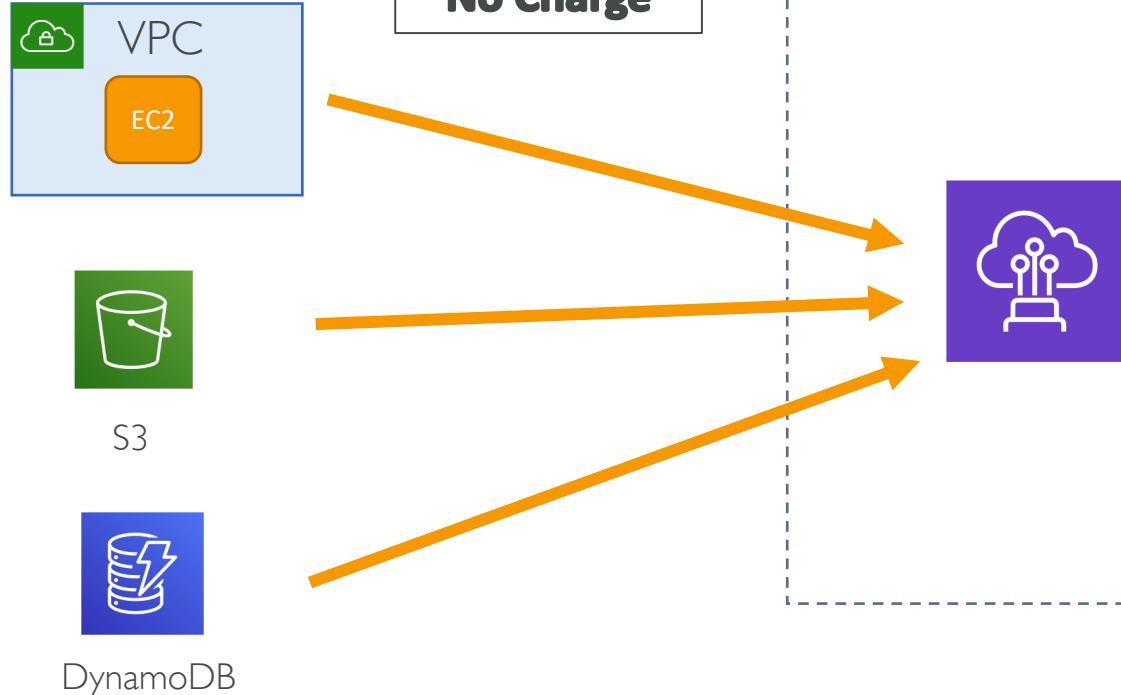
Dest. DX location:  
GPX, Mumbai,  
India

\$0.0600 / GB Out

Data Transfer IN

No Charge

Direct Connect Location



DX Port charges

**1G = \$0.30 /hour**  
**10G = \$2.25 /hour**  
**100G= \$22.5 /hour**  
**\*All locations except Japan**

Dedicated Connections

# Who pays for the DX charges?

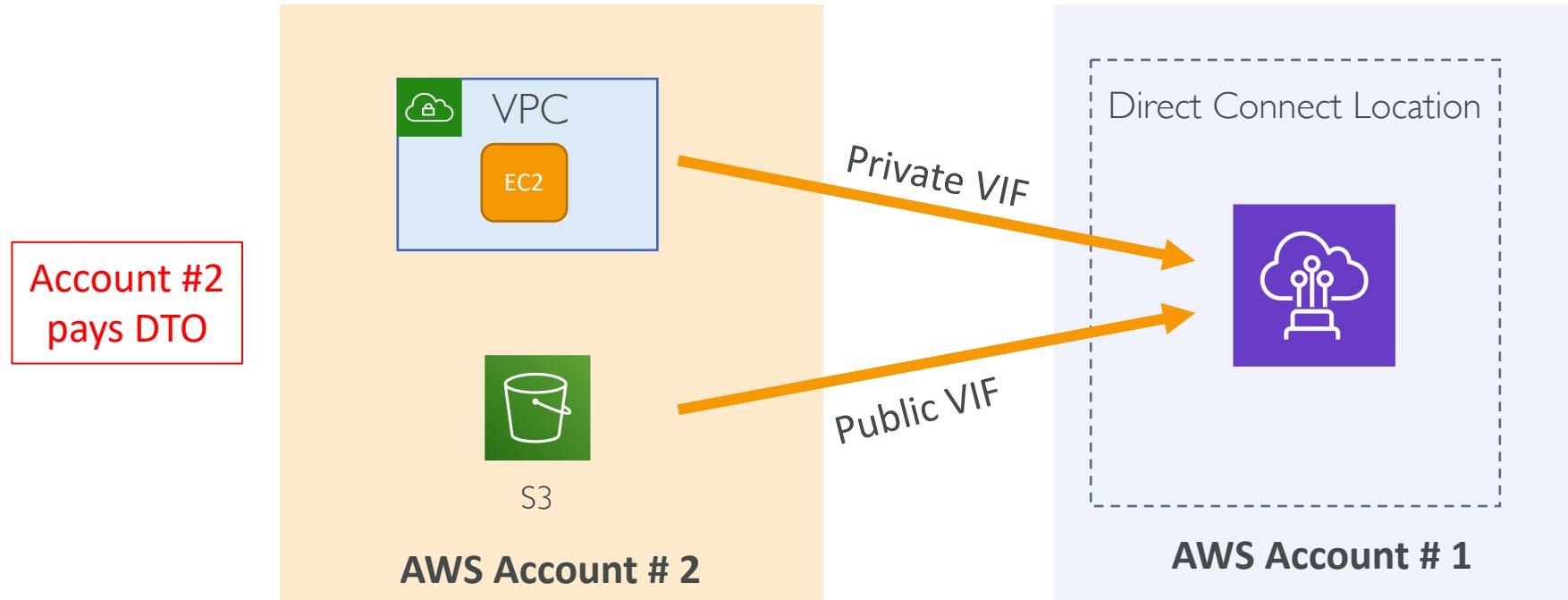
# Port hour charges

- The account that owns the Direct Connect connection i.e the account which requested the connection
- For Dedicated connection Port-hour charges are applied from the moment the connection is available. After ordering the DX connection, even if you don't proceed with connection setup process then it's billed post 90 days of connection requested
- For Hosted connections port-hours are billed once you have accepted the Hosted Connection

Billing for port-hours stops when the dedicated connection or hosted connection is deleted from your AWS account. Being in a “down” state does not cause the charges to stop.

# Data Transfer Out charges

Data Transfer out charges are usually allocated to the account that owns the resource sending the traffic



- For Public/Private/Transit VIFs the resource owner (EC2, S3, VPC etc) pays for DTO
- For S3 you can enable billing option of “Requester Pays”

# DTO while using Transit Gateway

- In case the traffic is sent through an AWS Transit Gateway:
  - Data Transfer out is allocated to the owner of the last resource to send traffic before traffic hits the Direct Connect VIF
  - If the owner of a Direct Connect connection and the owner of the resource sending traffic are in different AWS Organizations, Data Transfer out costs are allocated to the owner of the resource sending traffic, and charged the internet Data Transfer rate for the specific service they are using (not the Direct Connect DTO rate)

# Summary of the DX Billing

- Port hour charges & Data Transfer Out (DTO) charges
- Port hour charges depend on the DX capacity
- The AWS account which created the DX connection pays for the Port hour charges
- Data Transfer out charges are usually allocated to the account that owns the resource sending the traffic (with exception of TGW)
- In case of multi account setup where owner of Connection and AWS resources are in different AWS organizations, the DTO is charged to Resource owner account based on standard AWS service DTO rate and not the DX DTO rate

# Knowledge check

- Who pays for the charges for Hosted VIF?
- Answer:
  - Port hour charges will be billed to AWS account owning the connection whether it's a dedicated connection or a hosted connection
  - Data transfer out charges will be billed to the AWS account which contains the resource sending the traffic over the Hosted VIF

Ask 2 questions: Who owns the connection and Who sends the traffic, Simple?

# Troubleshooting DX issues

# Troubleshooting with DirectConnect

When no physical connectivity

When VIF is down

When BGP session is down

When not able to reach destination

1. Cross Connect is complete
2. Ports are correct
3. Routers are powered ON
4. Tx/Rx optical signals are receiving (CW)
5. Contact colocation provider & get report for Tx/Rx signals
6. CW Metrics for Physical error count
7. Contact AWS support

Layer 1 Issues

1. IP addresses are correct
2. VLAN IDs are correct
3. Router MAC address in ARP table
4. Try clearing ARP table
5. VLAN trunking enabled at intermediate devices
6. Contact AWS Support

Layer 2 Issues

1. Both Ends BGP ASN
2. Peer IPs are correct
3. MD5 Auth key – no spaces or extra chars
4. <= 100 prefixes for Private VIF
5. < = 1000 prefixes for Public VIF
6. Firewall not blocking TCP 179 port
7. BGP logs
8. Contact AWS Support

Layer 3/4 Issues

1. Advertising routes for on-premises prefixes
2. For Public VIF, it should be publicly routable prefixes
3. Security group and NACL
4. VPC Route table

BGP/Routing issues

# Troubleshooting with DirectConnect

- [Troubleshooting layer 1 \(physical\) issues](#)
- [Troubleshooting layer 2 \(data link\) issues](#)
- [Troubleshooting layer 3/4 \(Network/Transport\) issues](#)
- [Troubleshooting routing issues](#)
- <https://aws.amazon.com/premiumsupport/knowledge-center/troubleshoot-bgp-dx/>

# Direct Connect Summary

# Summary

- Understand various parties involved in DX e.g DX location provider, Direct connect partner, Network provider for connectivity back to your data center
- Understand various segments involved in DX connection e.g AWS device/router, Customer router, Cross connect, Virtual Interfaces
- Dedicated Connection and Hosted connection
- Public VIF, Private VIF and Transit VIF
- Transit Gateway integration with DirectConnect Gateway
- Routing Policies and BGP communities for Public and Private VIF

# Summary

- Link Aggregation Group (LAG)
- Bidirectional Forwarding Detection (BFD)
- DX HA architectures
- DX encryption – VPN over DX, MacSec
- DX MTU and Jumbo frames
- DX billing - Port hour and Data transfer Out charges
- DX Troubleshooting

# Exam Essentials

- AWS Direct connect provides 1 or 10 or 100 Gbps dedicated connection or sub-1 Gbps Hosted connections
- BGP protocol is used to exchange routes
- 802.1Q VLANs tags are used to separate virtual interfaces on the same connection
- Public VIF provides global connectivity to public AWS resources
- Private VIF provides connectivity to resources inside VPC
- Hosted connection allows a single VIF per connection
- Parameters required for creating DX connection
- Parameters required for creating Public/Private/Transit VIFs
- BGP path selection/failover parameters (AS\_PATH, Routes etc)

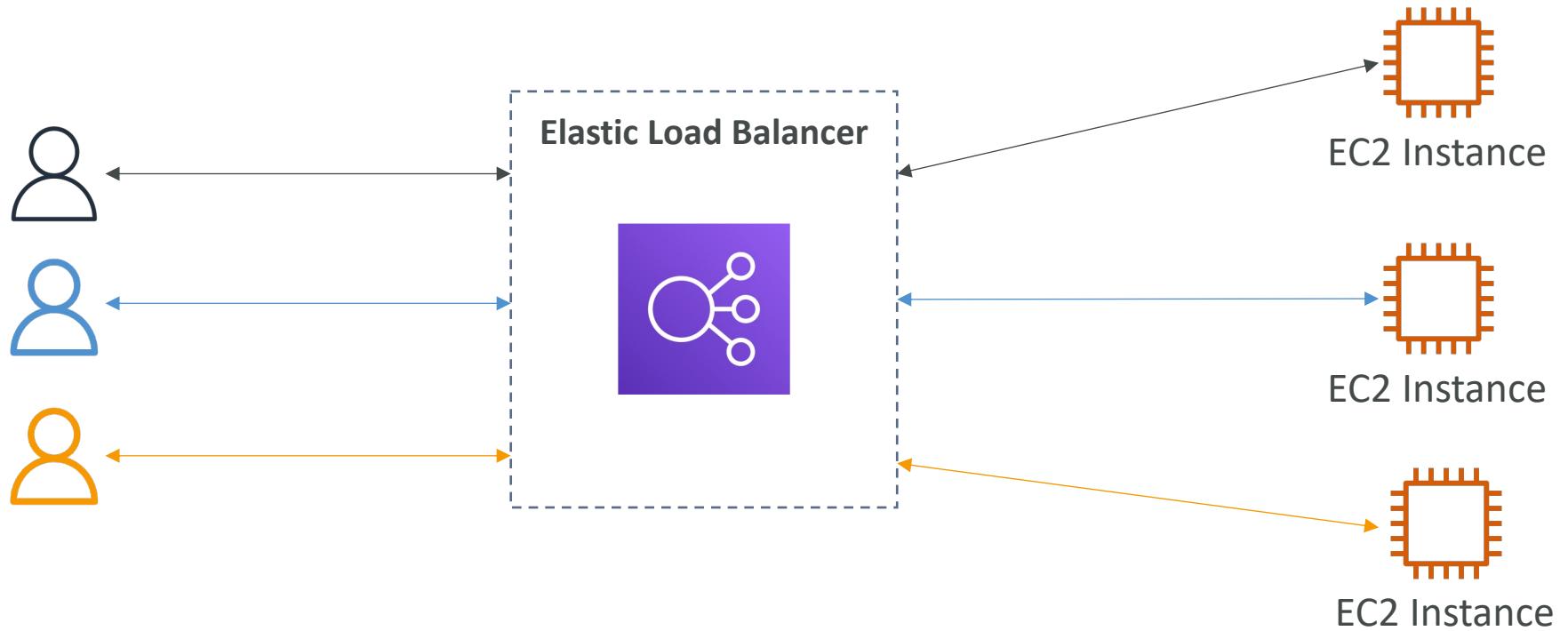
# Exam Essentials

- VIF Routing policies and BGP Communities
- Link Aggregation Group (LAG) for higher aggregated DX bandwidth
- BFD can be used to increase the speed at which connection failures are detected and cause failover to alternative routes (1 sec)
- DX HA options – Dual device, Dual Location, VPN backup etc
- Public VIF includes Hardware VPN IP addresses. Can use for VPN over a DX connection
- 100 routes can be published for Private VIFs and 1000 for Public VIF from customer end to AWS

# AWS Elastic Load Balancer

# What is load balancing?

- Load Balances are servers that forward traffic to multiple servers (e.g., EC2 instances) downstream



# Why use a load balancer?

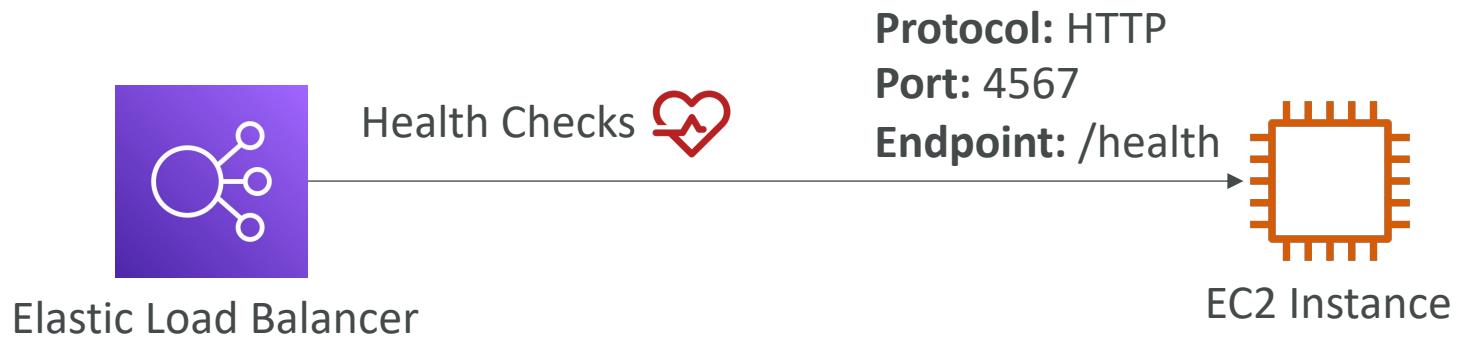
- Spread load across multiple downstream instances
- Expose a single point of access (DNS) to your application
- Seamlessly handle failures of downstream instances
- Do regular health checks to your instances
- Provide SSL termination (HTTPS) for your websites
- Enforce stickiness with cookies
- High availability across zones
- Separate public traffic from private traffic

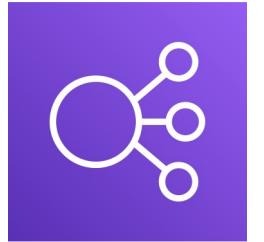
# Why use an Elastic Load Balancer?

- An Elastic Load Balancer is a **managed load balancer**
  - AWS guarantees that it will be working
  - AWS takes care of upgrades, maintenance, high availability
  - AWS provides only a few configuration knobs
- It costs less to setup your own load balancer but it will be a lot more effort on your end
- It is integrated with many AWS offerings / services
  - EC2, EC2 Auto Scaling Groups, Amazon ECS
  - AWS Certificate Manager (ACM), CloudWatch
  - Route 53, AWS WAF, AWS Global Accelerator

# Health Checks

- Health Checks are crucial for Load Balancers
- They enable the load balancer to know if instances it forwards traffic to are available to reply to requests
- The health check is done on a port and a route (/health is common)
- If the response is not 200 (OK), then the instance is unhealthy





# Types of load balancer on AWS

- AWS has **4 kinds of managed Load Balancers**
- **Classic Load Balancer** (v1 - old generation) – 2009 – CLB
  - HTTP, HTTPS, TCP, SSL (secure TCP)
- **Application Load Balancer** (v2 - new generation) – 2016 – ALB
  - HTTP, HTTPS, WebSocket
- **Network Load Balancer** (v2 - new generation) – 2017 – NLB
  - TCP, TLS (secure TCP), UDP
- **Gateway Load Balancer** – 2020 – GWLB
  - Operates at layer 3 (Network layer) – IP Protocol
- Overall, it is recommended to use the newer generation load balancers as they provide more features
- Some load balancers can be setup as **internal** (private) or **external** (public) ELBs

# Load Balancer Security Groups



## Load Balancer Security Group:

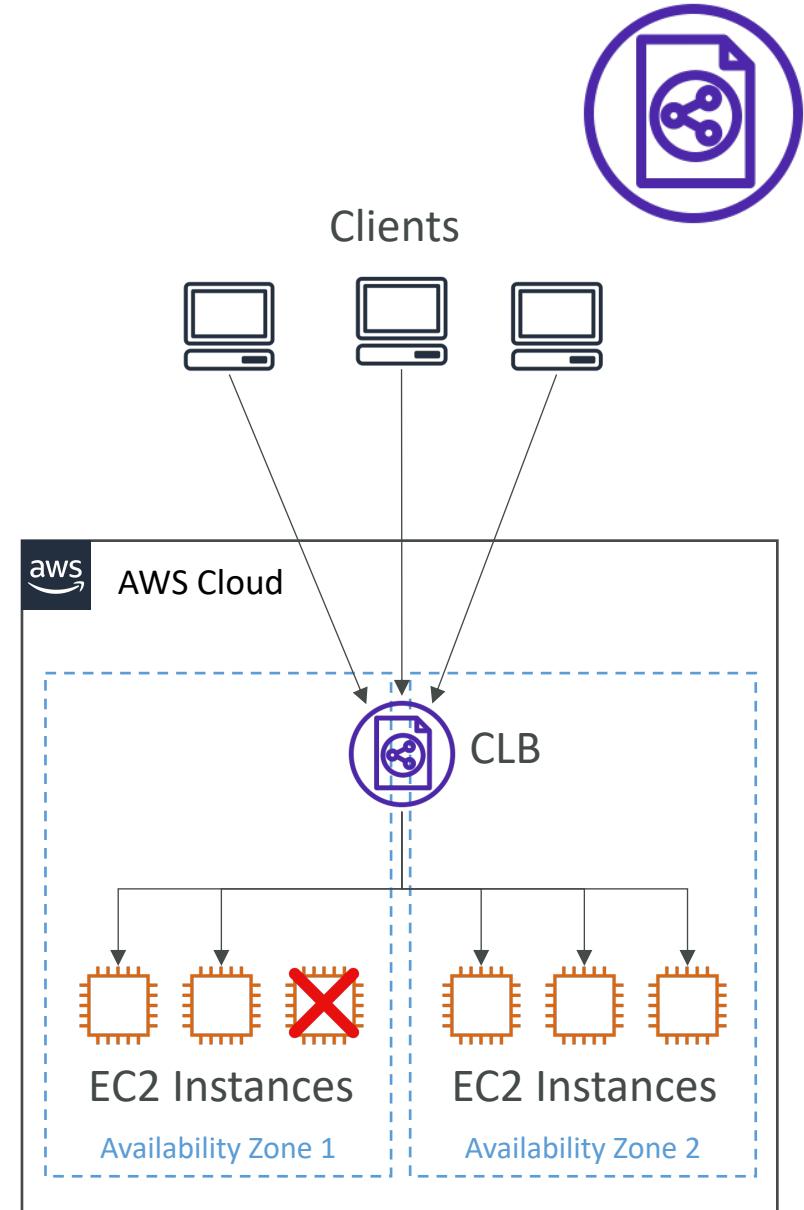
| Type  | Protocol | Port Range | Source    | Description           |
|-------|----------|------------|-----------|-----------------------|
| HTTP  | TCP      | 80         | 0.0.0.0/0 | Allow HTTP from an... |
| HTTPS | TCP      | 443        | 0.0.0.0/0 | Allow HTTPS from a... |

## Application Security Group: Allow traffic only from Load Balancer

| Type | Protocol | Port Range | Source                       | Description           |
|------|----------|------------|------------------------------|-----------------------|
| HTTP | TCP      | 80         | sg-054b5ff5ea02f2b6e (load-b | Allow Traffic only... |

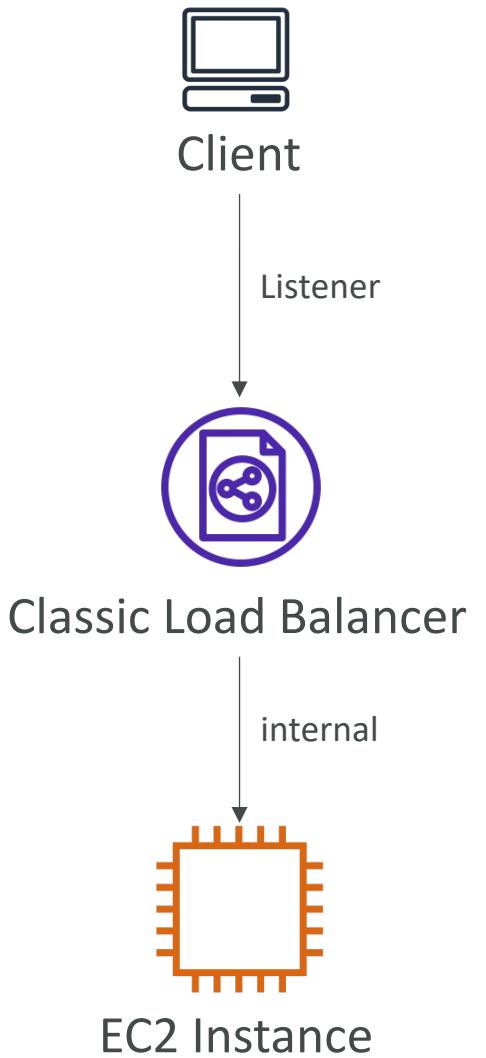
# Classic Load Balancer

- Operates at layer 4 and layer 7
- Supported protocols HTTP, HTTPS, TCP, and SSL/TLS
- EC2 instances registered directly with the CLB (no Target Groups)
- Health Checks can be HTTP, HTTPS, or TCP
- Supports EC2-Classic networks



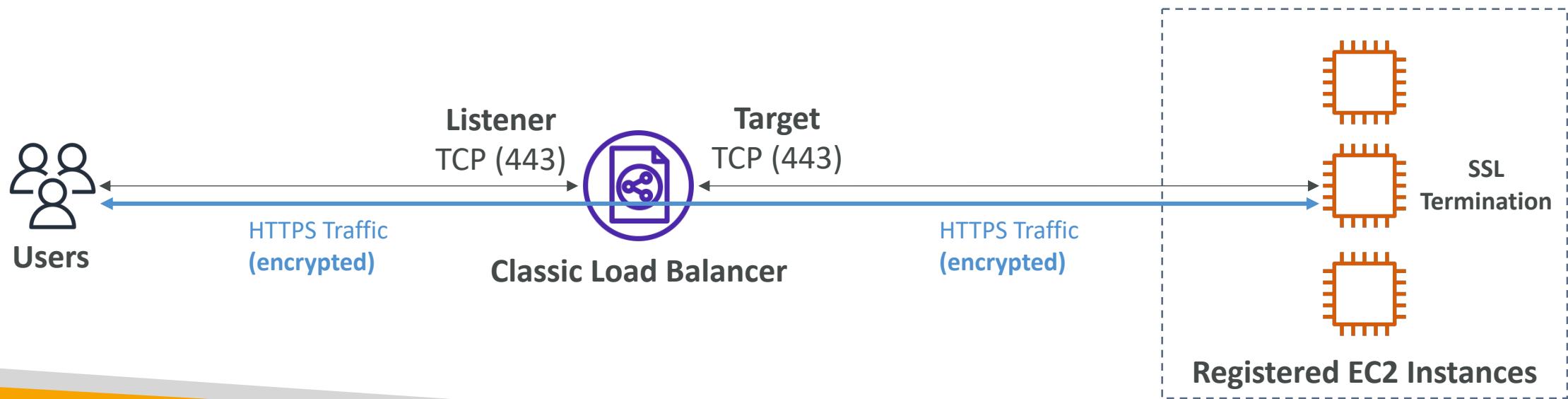
# Classic Load Balancer

| Listener  | Internal   |
|---|--|
| HTTP (L7)   | HTTP or<br>HTTPS (Must install certificate on EC2) |
| HTTPS (L7) – SSL Termination<br>(Must install certificate on CLB) | HTTP or<br>HTTPS (Must install certificate on EC2) |
| TCP (L4)  | TCP or<br>SSL (Must install certificate on EC2)    |
| SSL (L4) (Must install certificate on CLB)                        | TCP or<br>SSL (Must install certificate on EC2)    |

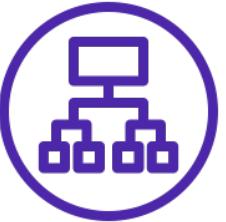


# Classic Load Balancer – SSL considerations

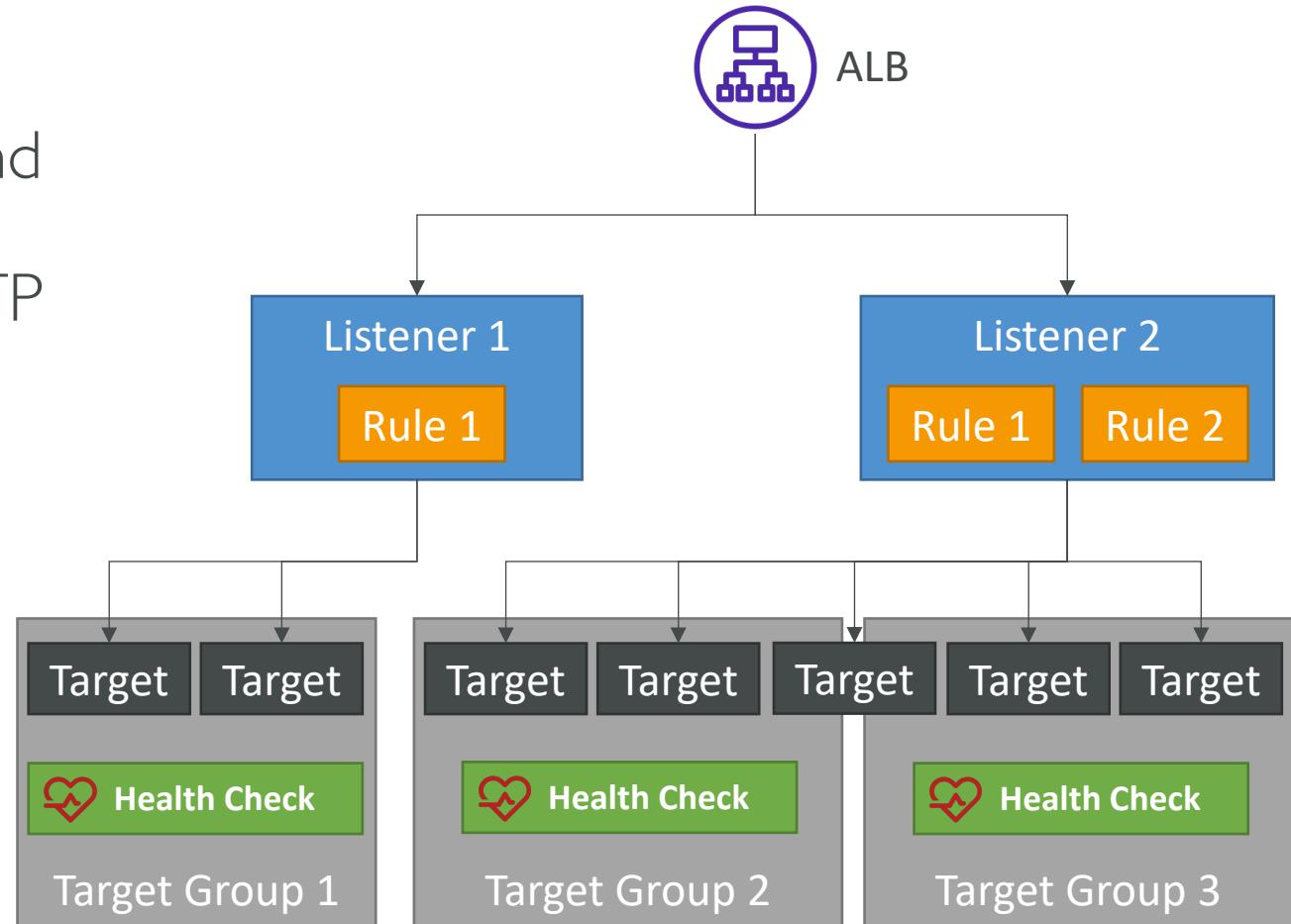
- Enabling HTTPS or SSL on EC2 instances is called “Backend Authentication” (between the CLB and the backend EC2 instances)
- TCP => TCP passes all the traffic to the EC2 instance (no termination):
  - The only way to do 2-way Mutual SSL Authentication



# Application Load Balancer



- Operates at Layer 7 (HTTP...)
- Supported protocols HTTP, HTTPS, WebSocket, HTTP/2 and gRPC
- Load balancing to multiple HTTP applications across machines (Target Groups)
- Load balancing to multiple applications/ports on the same server (e.g., containers)
- Support for returning custom HTTP responses
- Supports redirects (e.g., from HTTP to HTTPS)



# Application Load Balancer

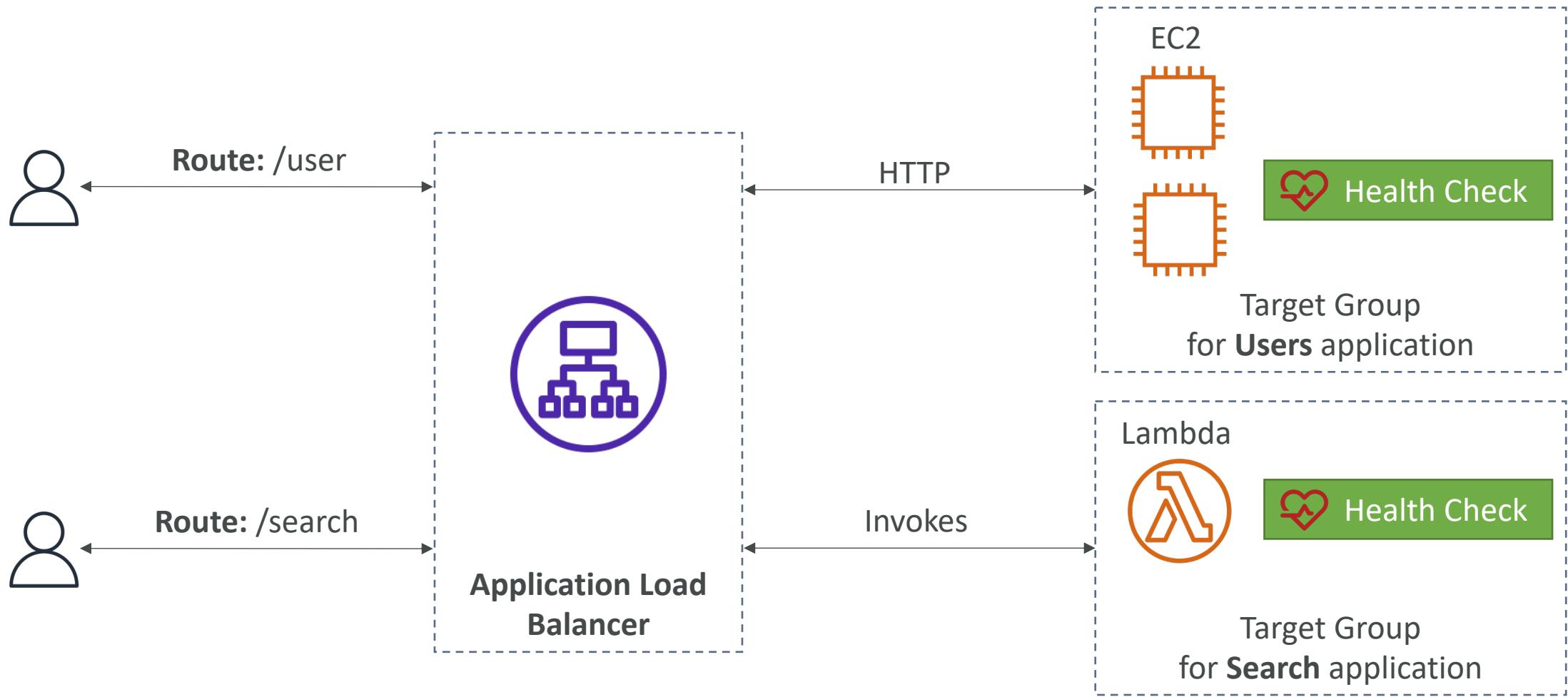
- Target Groups
  - EC2 Instances (can be managed by an ASG) – HTTP
  - ECS Tasks (managed by ECS itself) – HTTP
  - Lambda functions – HTTP request is translated into a JSON event
  - IP Addresses – must be private IP addresses (e.g., EC2 instances in peered VPC, on-premises servers accessed over AWS Direct Connect or VPN connection)
- Supports Weighted Target Groups
  - Example: multiple versions of your application, blue/green deployment
- Health Checks can be HTTP or HTTPS (WebSocket is not supported)
- Each subnet must have a min of /27 and 8 free IP addresses
- Across all subnets, a maximum of 100 IP addresses will be used per ALB

# Application Load Balancer

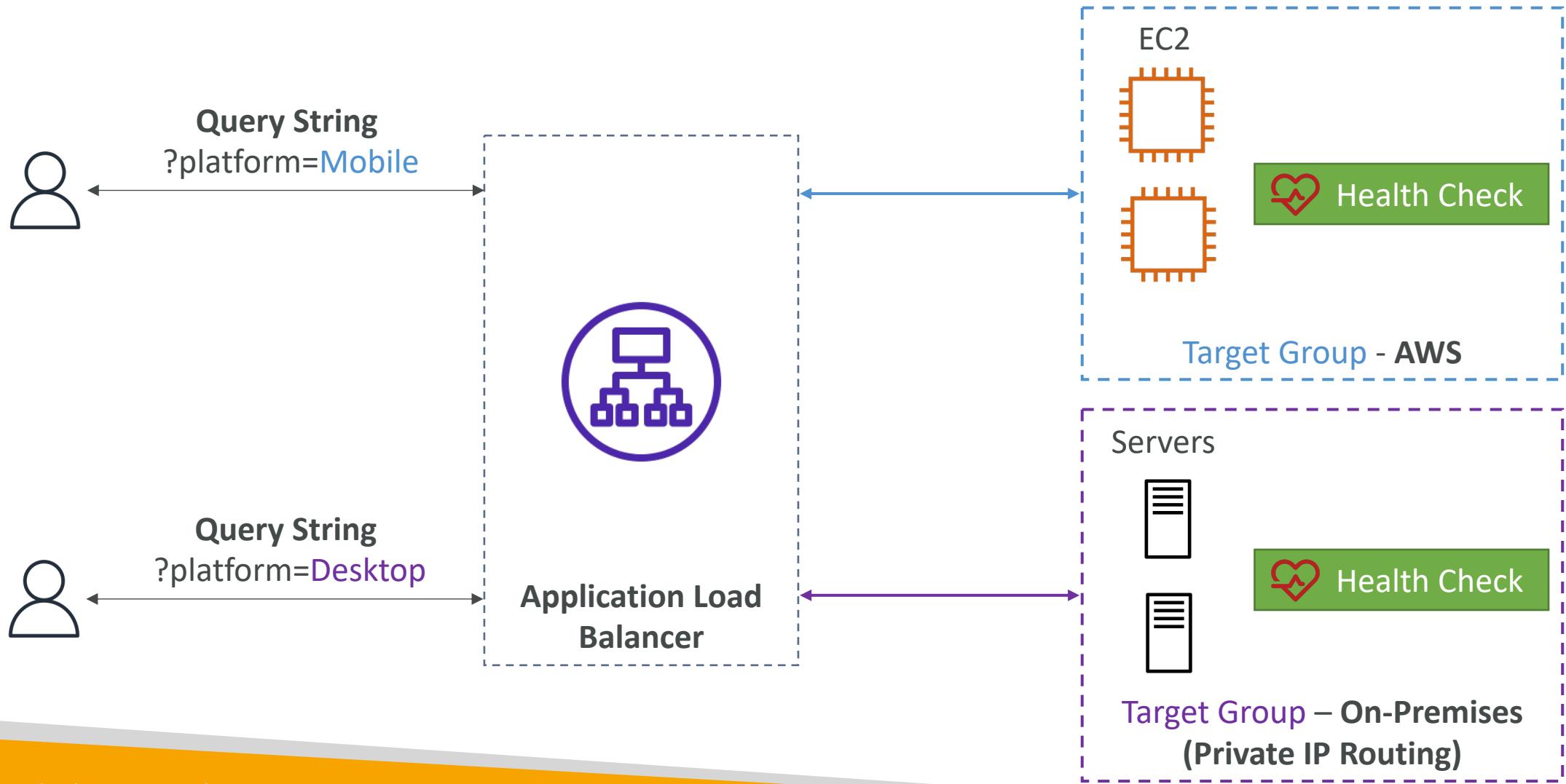
- Routing to different target groups
  - Routing based on URL Path (example.com/[users](#), example.com/[posts](#))
  - Routing based on Hostname
    - [one.example.com](#), [other.example.com](#)
    - [\\*.example.com](#), [example.com](#)
  - Routing based on Query String, HTTP Headers, Source IP Address (example.com/users?id=123&order=false)
- ALB are a great fit for micro services & container-based applications (e.g., Docker & Amazon ECS)
- Has a **port mapping feature** to redirect to a dynamic port in ECS
  - In comparison, we'd need multiple Classic Load Balancers per application

# Application Load Balancer

## Path-based Routing



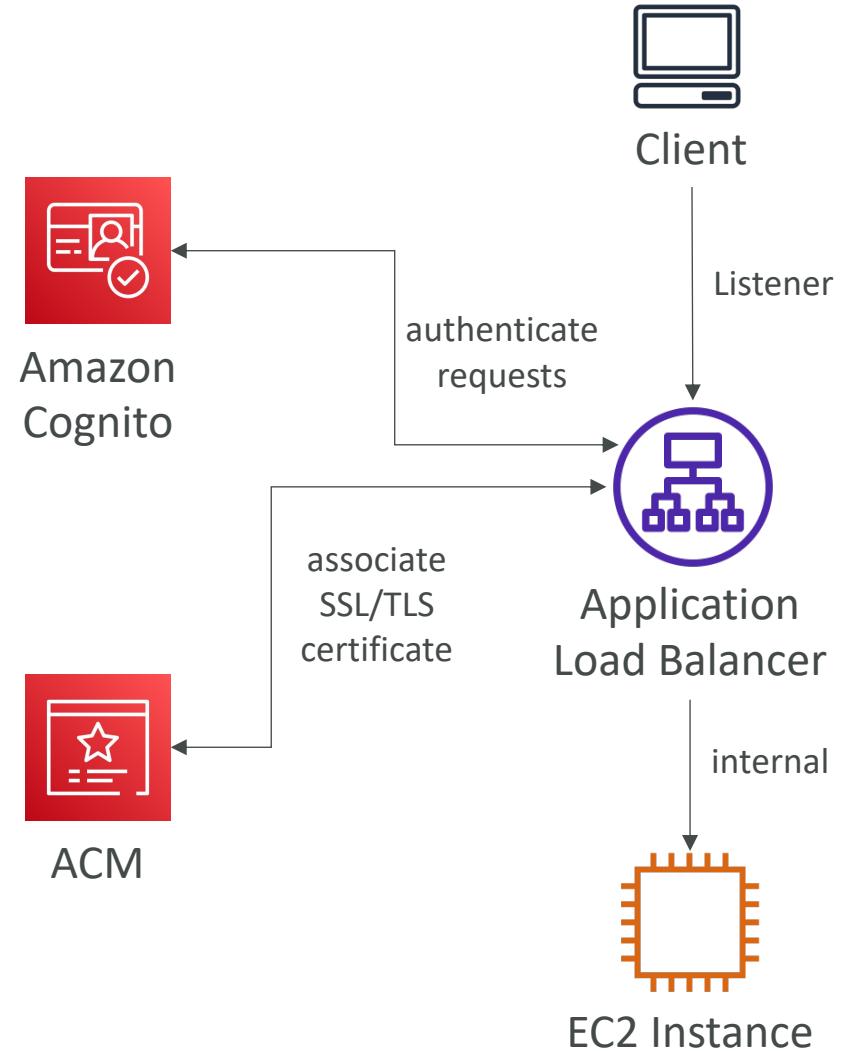
# Application Load Balancer Query String/Parameters Routing



# Application Load Balancer

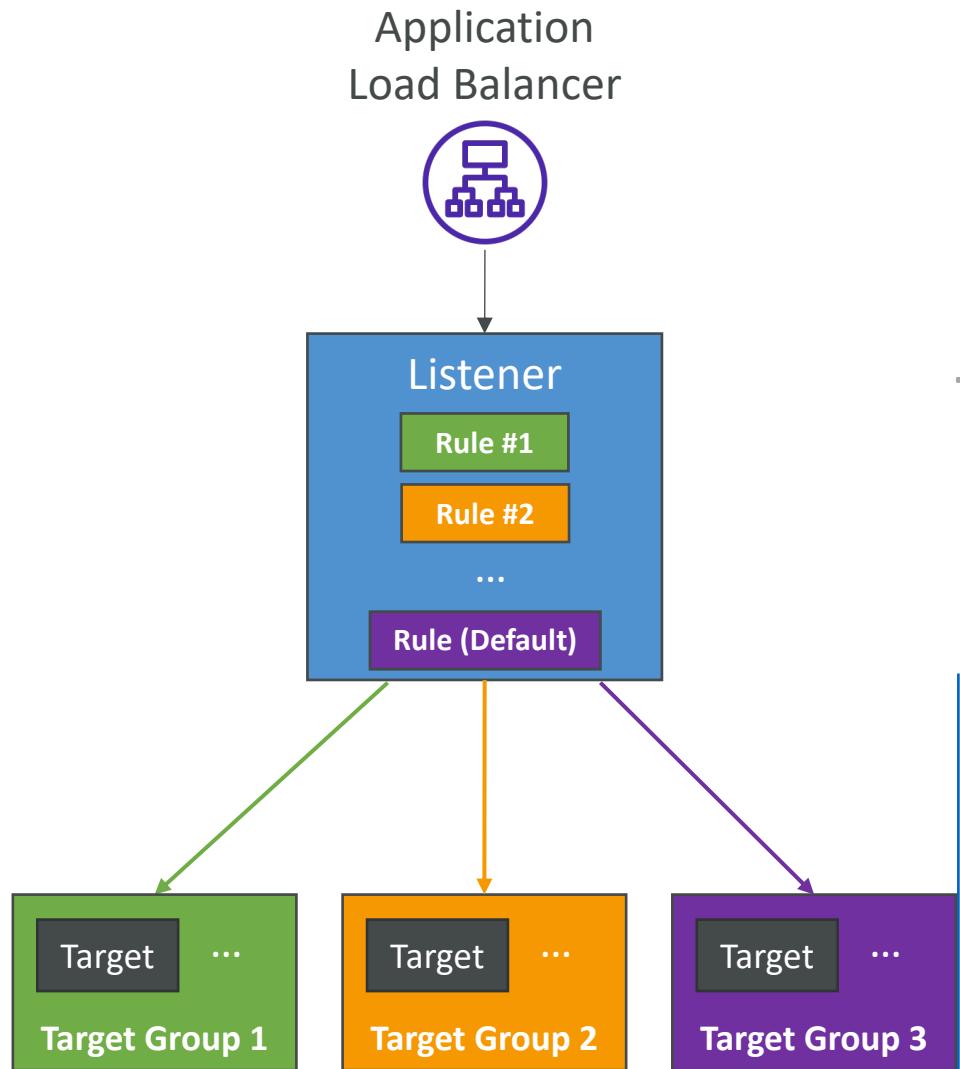
- Ability to **authenticate users** before routing requests to registered targets
  - Amazon Cognito User Pools and Identity Providers
  - Microsoft Active Directory, OIDC, SAML, LDAP, OpenID
  - Social Identity Providers such as Amazon, Facebook, Google
- TLS Certificates (multiple listeners & SNI)

| Listener                      | Internal   |
|-------------------------------|--|
| HTTP                          | HTTP or HTTPS (Must install certificate on target) |
| HTTPS – SSL Termination (ACM) | HTTP or HTTPS (Must install certificate on target) |



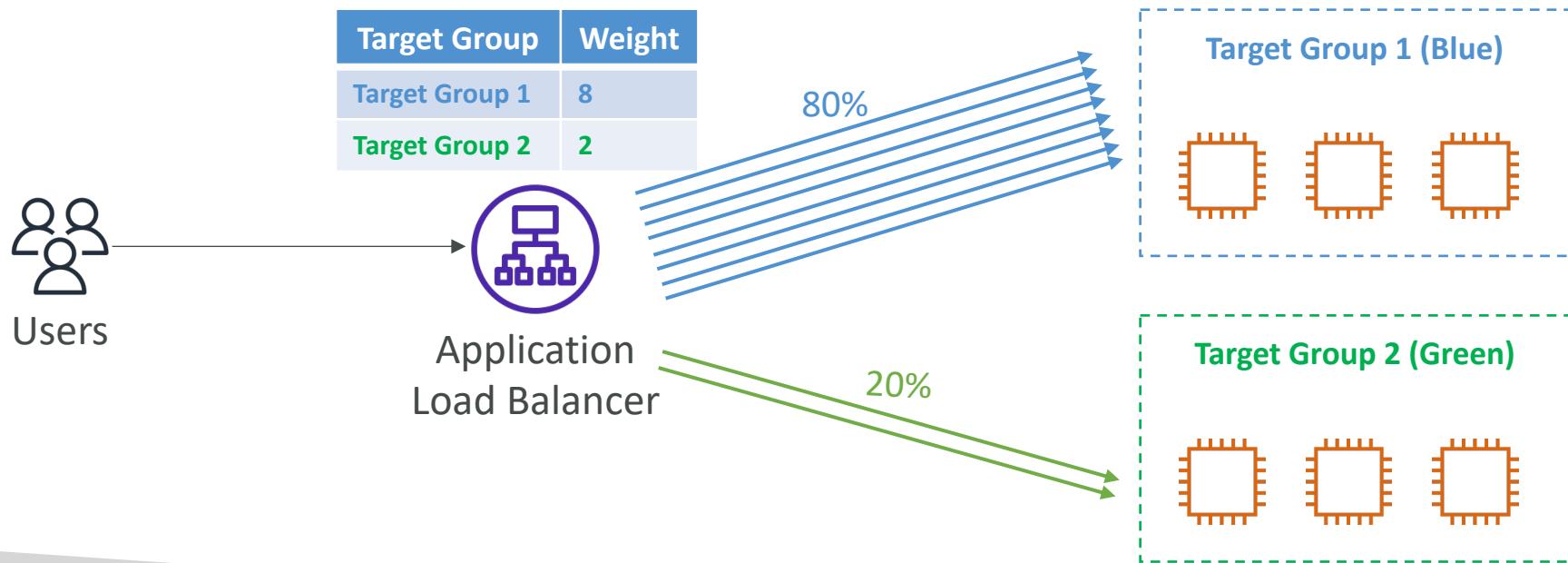
# ALB – Listener Rules

- Processed in order (last is Default Rule)
- Supported Actions (forward, redirect, fixed-response)
- Rule Conditions:
  - host-header
  - http-request-method
  - path-pattern
  - source-ip
  - http-header
  - query-string



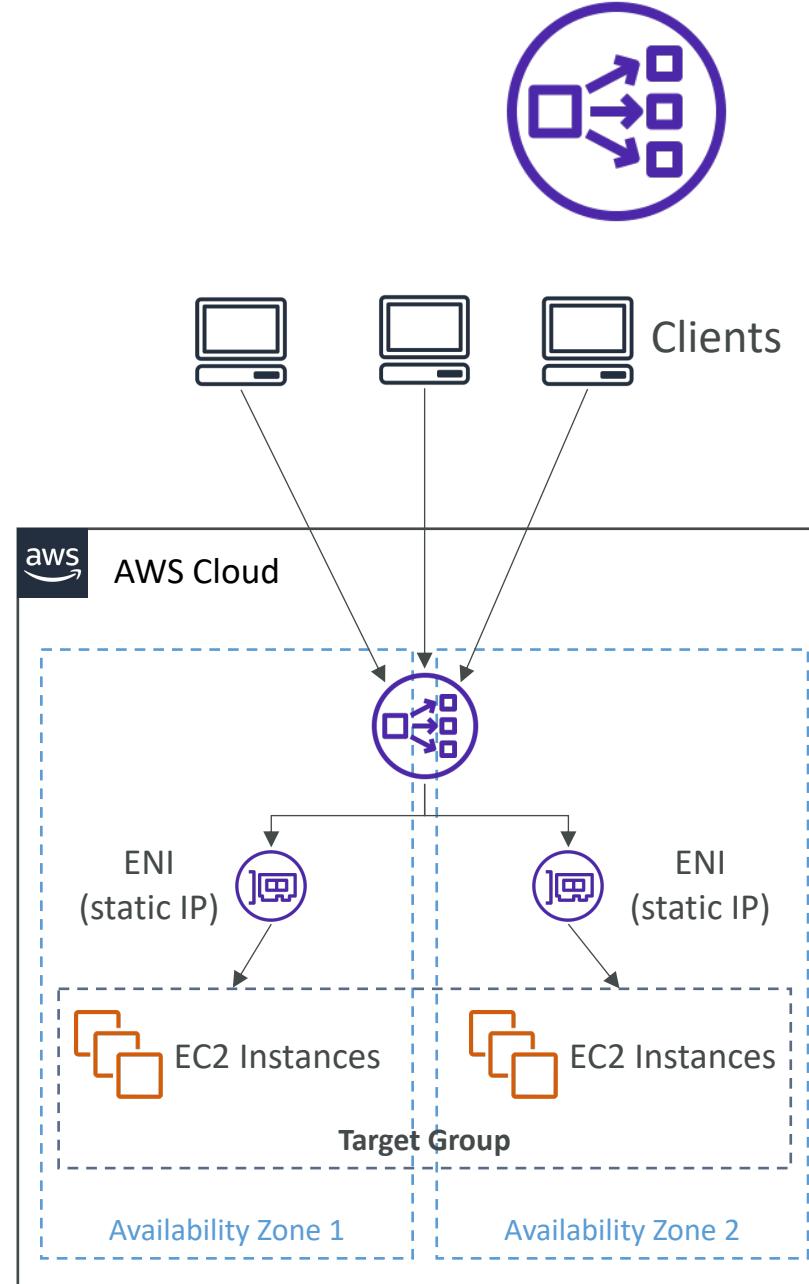
# Target Group Weighting

- Specify weight for each Target Group on a single Rule
- Example: multiple versions of your app, blue/green deployment
- Allows you to control the distribution of the traffic to your applications



# Network Load Balancer

- Operates at Layer 4
- Supported protocols TCP, UDP, and TLS
- Handle millions of requests per second
- NLB has one static IP per AZ, and supports assigning Elastic IP (Internet-facing NLB)  
(helpful for whitelisting specific IP addresses)
- Load balancing to multiple applications/ports on the same machine (e.g., containers, Amazon ECS)
- Less latency ~100 ms (vs. 400 ms for ALB)
- Supports WebSocket protocol
- Network Load Balancers are mostly used:
  - For extreme performance, TCP or UDP traffic
  - With AWS PrivateLink: privately expose an internal service

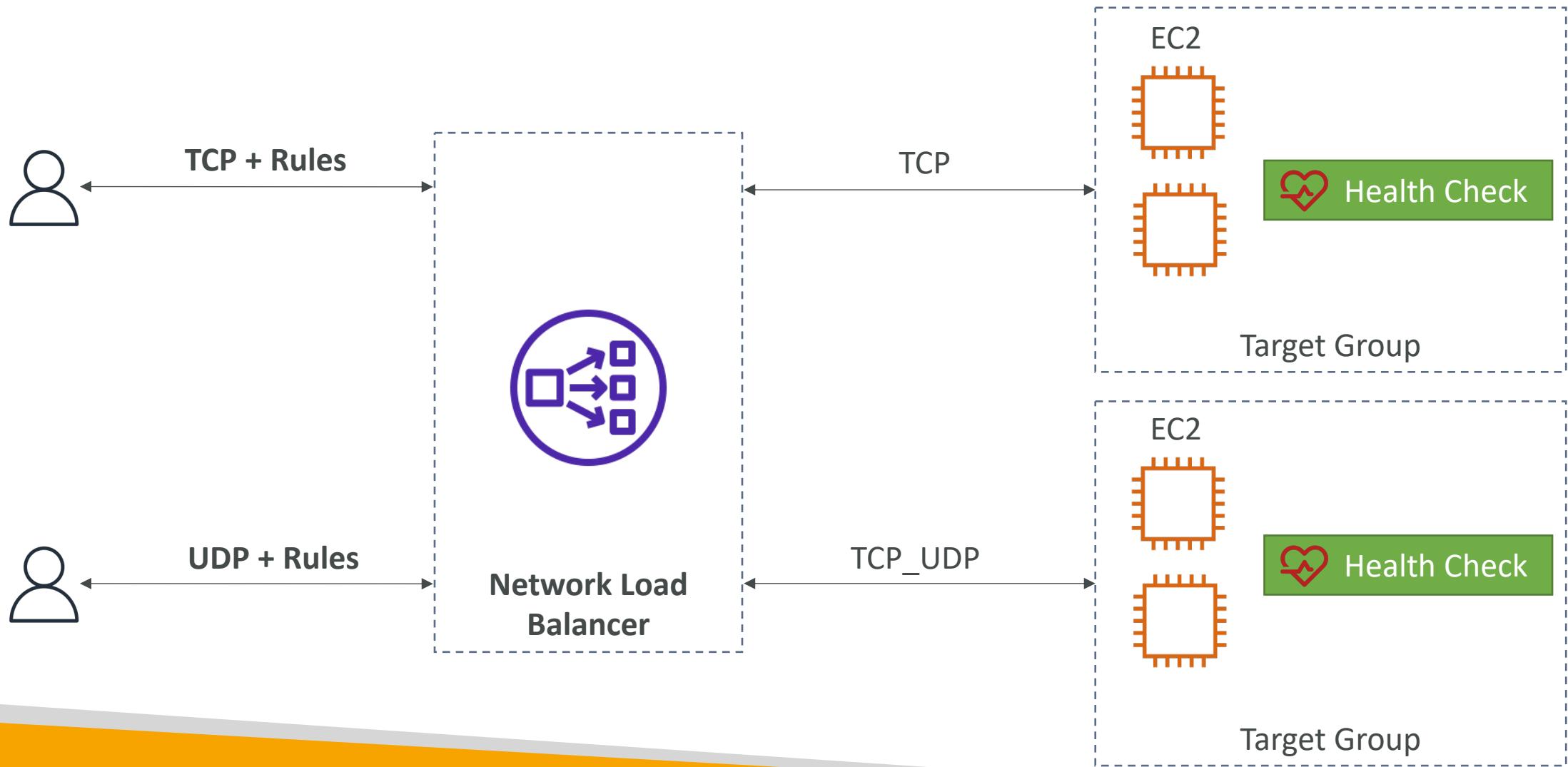


# Network Load Balancer

- Target Groups
  - EC2 Instances – can be managed by an ASG
  - ECS Tasks – (managed by ECS itself)
  - IP Addresses
    - must be private IP addresses, TCP listeners only (on-premises servers over AWS DX or VPN)
    - can be inter-region peered VPC
- You can't register EC2 instances by instance ID if these EC2 instances are in another VPC (even if peered with NLB VPC)
  - Register by IP address instead
- Health Checks
  - Supported protocols HTTP, HTTPS, and TCP
  - **Active Health Check** – periodically sends a request to registered targets
  - **Passive Health Check** – observe how targets respond to connections. Detect unhealthy targets before Active Health Checks (can't be disabled or configured)

# Network Load Balancer

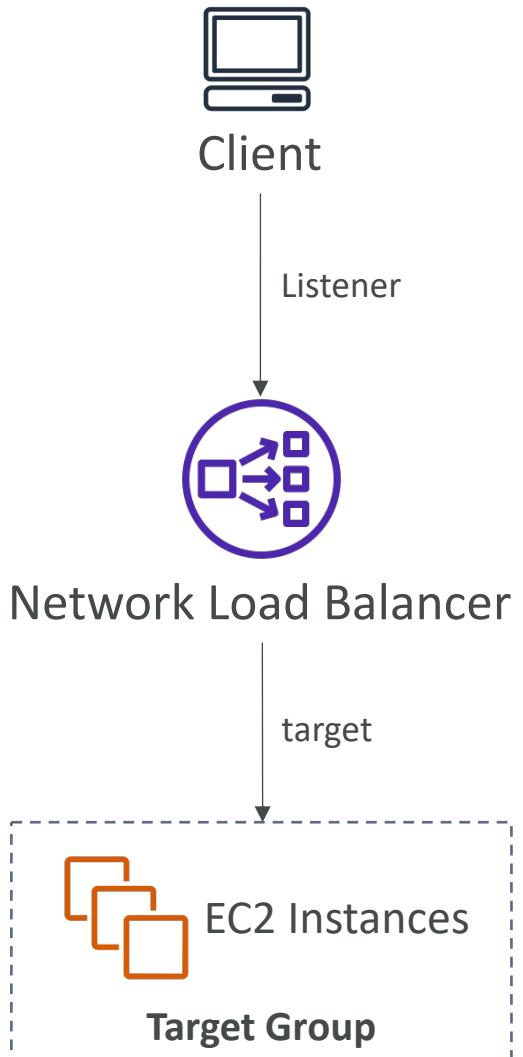
## TCP & UDP -based Traffic



# Network Load Balancer

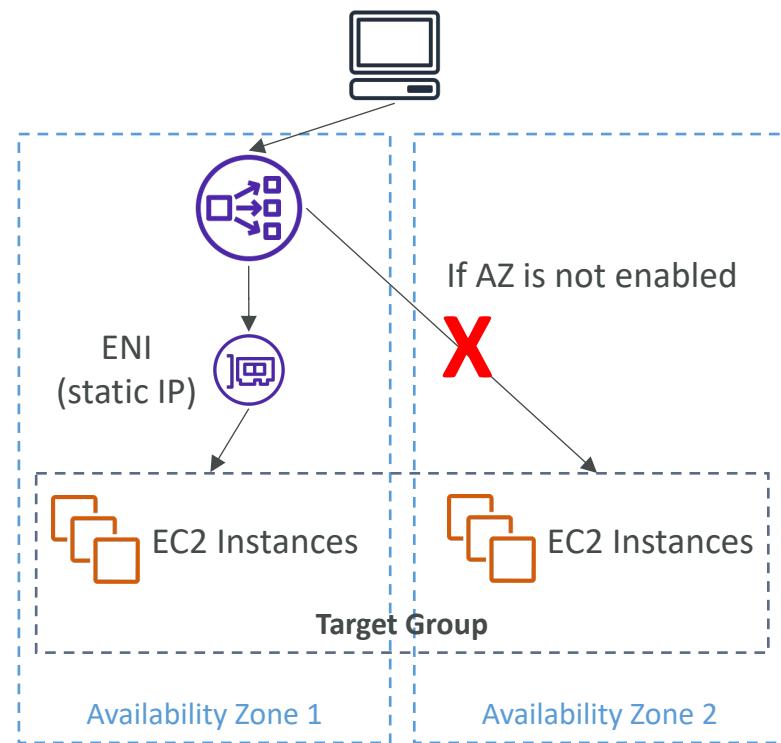
- Client IP Preservation: client IP is forwarded to targets
  - Targets by instance ID / ECS Tasks: **Enabled**
  - Targets by IP address TCP & TLS: **Disabled by default**
  - Targets by IP address UDP & TCP\_UDP: **Enabled by default**
- When **disabled**, use Proxy Protocol v2 (will add headers)

| Listener   | Target   |
|--|--|
| TCP  | TCP or TCP_UDP                                   |
| TLS – SSL Termination<br>(Must install certificate on NLB) | TCP or TLS (Must install certificate on targets) |
| UDP  | UDP or TCP_UDP                                   |
| TCP_UDP  | TCP_UDP  |

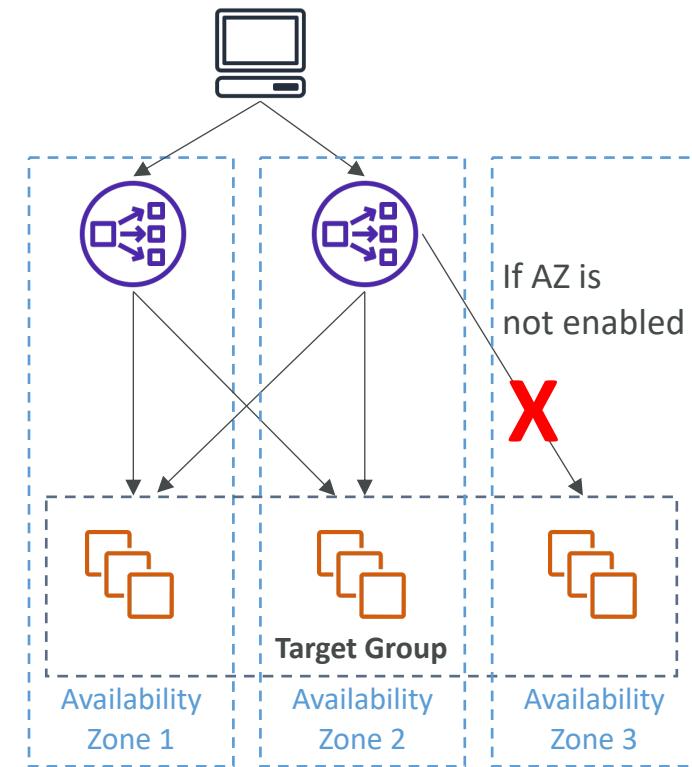


# Network Load Balancer – Availability Zones

- You must enable an AZ before traffic is sent to that AZ (can be added after NLB creation)
- You cannot remove an AZ after it is enabled

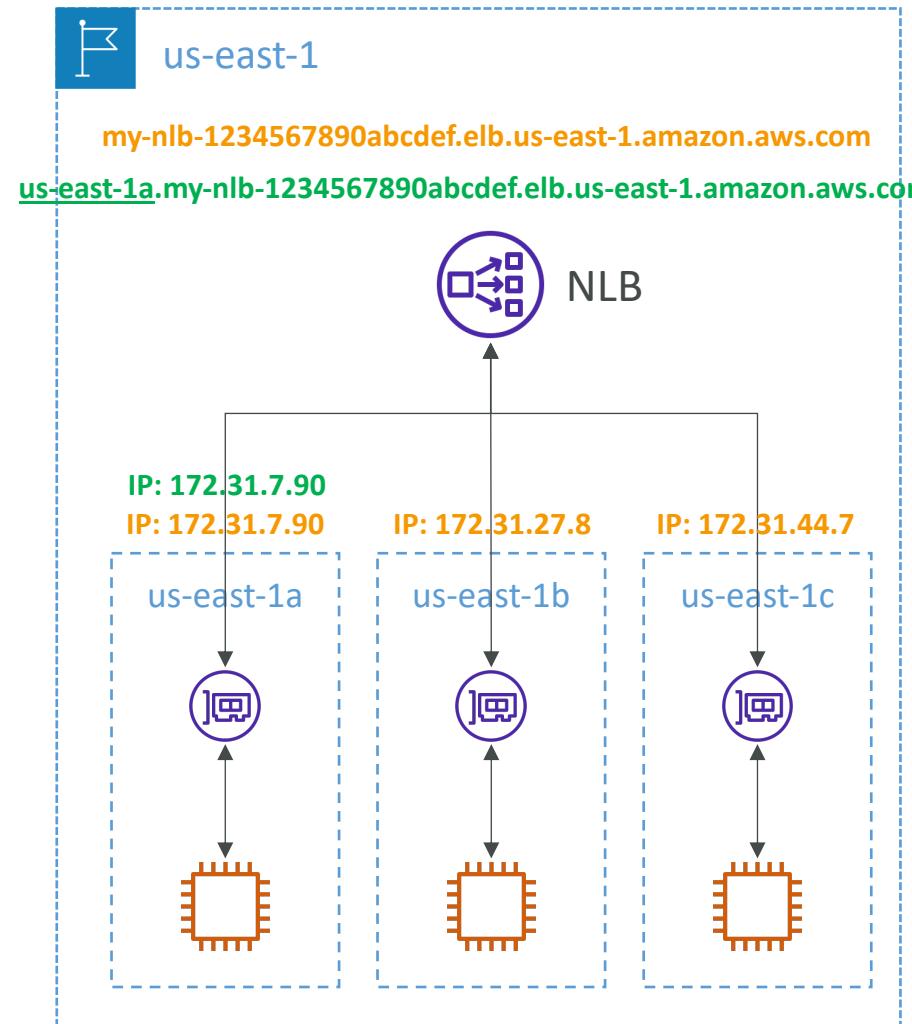


- Cross Zones Load Balancing only works for the availability zones that are enabled on the NLB



# Network Load Balancer – Zonal DNS Name

- Resolving **Regional NLB DNS** name returns the IP addresses for all NLB nodes in all enabled AZs
  - my-nlb-1234567890abcdef.elb.us-east-1.amazonaws.com
- **Zonal DNS Name**
  - NLB has DNS names for each of its nodes
  - Use to determine the IP address of each node
  - **us-east-1a.my-nlb-1234567890abcdef.elb.us-east-1.amazonaws.com**
  - Used to minimize latency and data transfer costs
  - You need to implement app specific logic

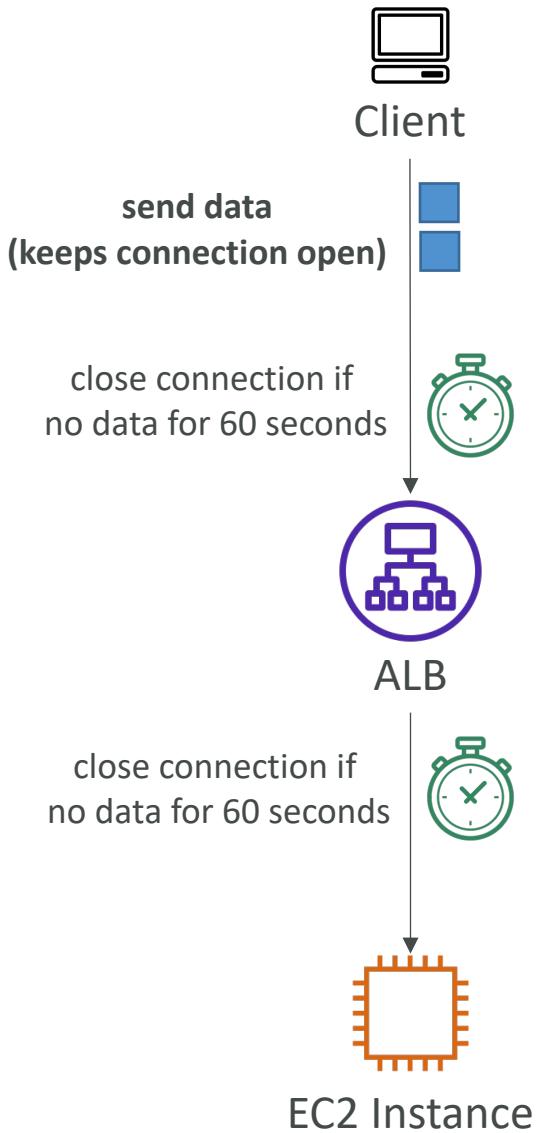


# Network Load Balancer – Good To Know

- You can't associate security groups to NLBs (SGs rules associated to targets are the ones leveraged for network security)
- You can't disable/remove an AZ after you create it
- You can't modify ENIs created for the NLB in each AZ (view only)
- You can't change EIPs and private IPv4 addresses attached to the ENIs after you create the load balancer
- Supports 55,000 simultaneous connections/minute per target
  - If exceeded, you'll receive **port allocation errors**
  - Solution: add more targets to the target group
- For internet-facing load balancers, the subnets that you specify must have **at least 8 available IP addresses (e.g. min /28)**. For internal load balancers, this is only required if you let AWS select a private IPv4 address from the subnet.

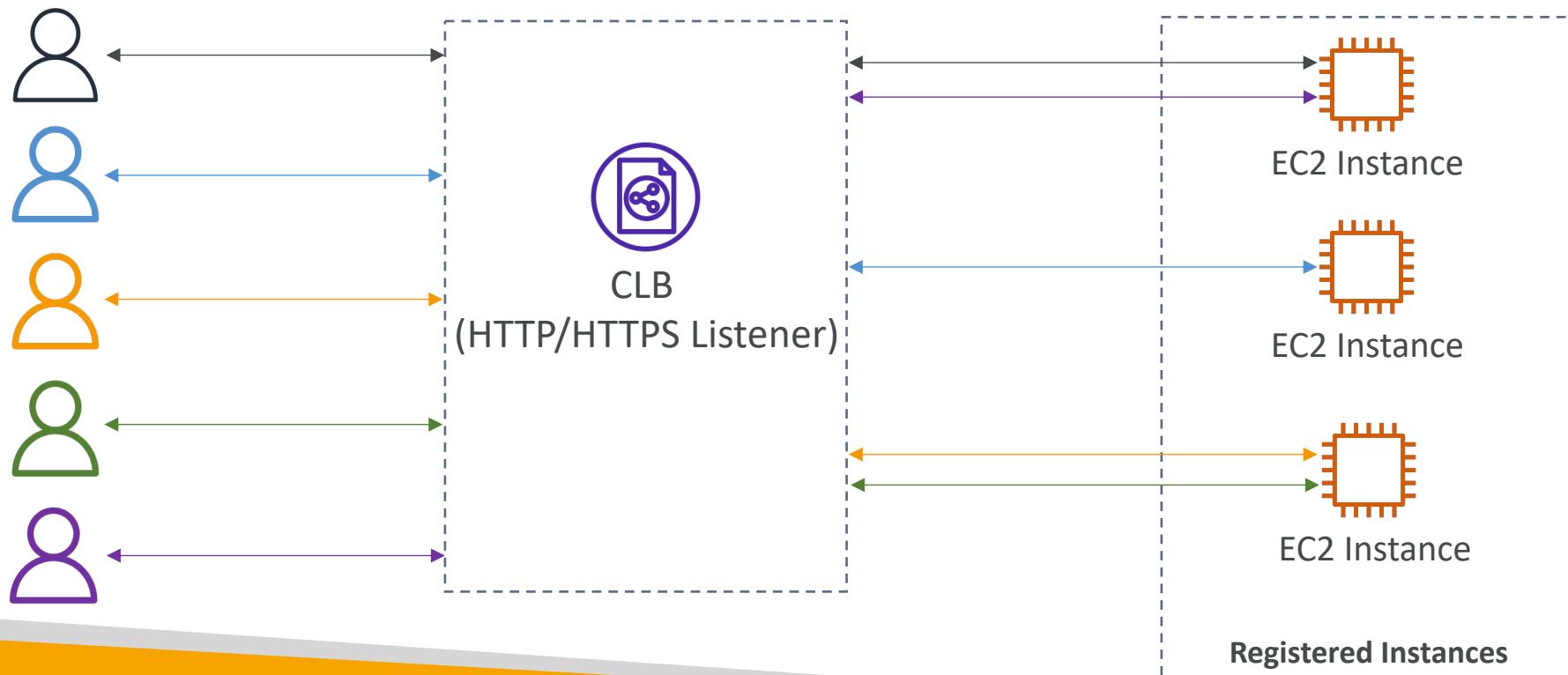
# Connection Idle Timeout

- Idle Timeout period for ELB's connections (client-ELB connection & ELB-target connection)
  - Connection closed if no data has been sent/received during that period
  - Opened if at least 1 byte is sent before that timeout period elapses
- Supported for CLB, ALB, and NLB
- Can be configured for CLB & ALB (default 60 seconds)
- Can't be configured for NLB (350 sec. for TCP, 120 sec. for UDP)
- Usage: avoid timeouts while uploading files
- Recommended to enable HTTP keep-alive in the web server settings for your EC2 instances, thus makes the ELB reuse the backend connections until the keep-alive timeout expires



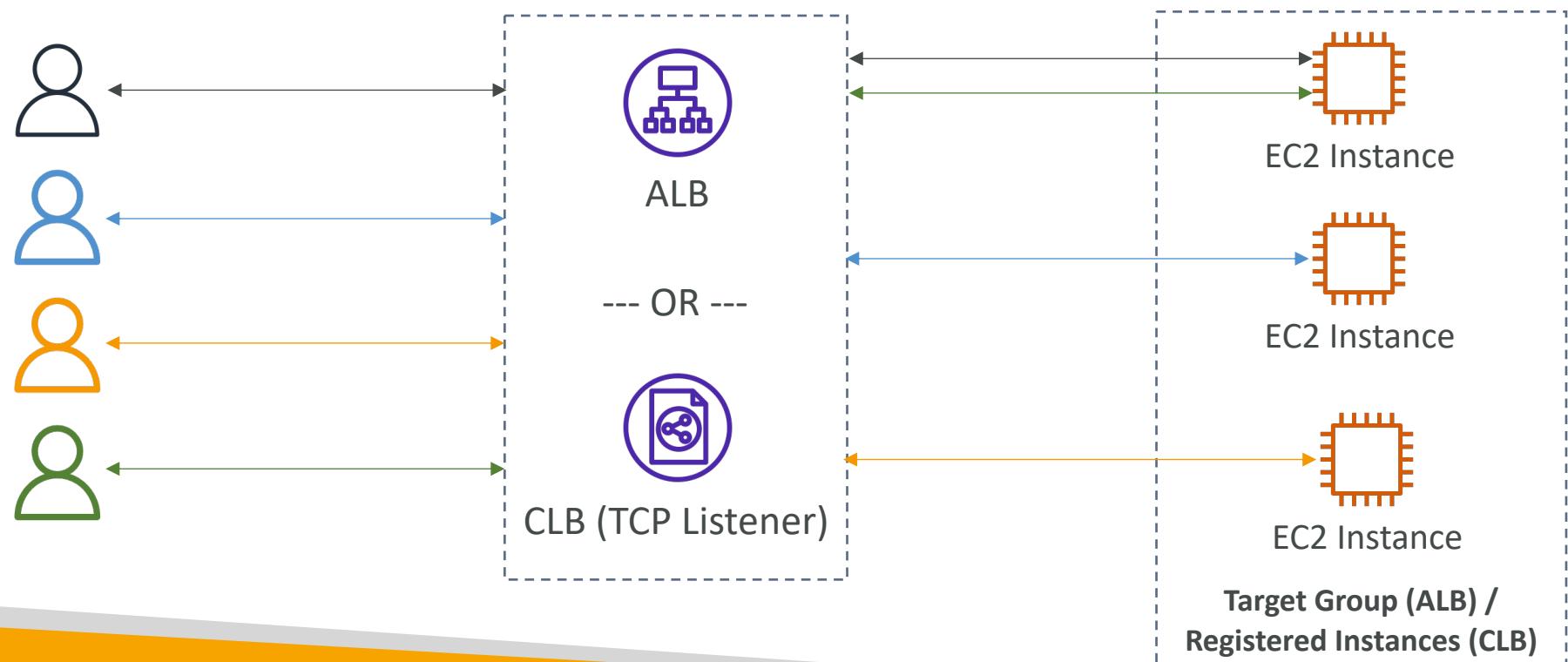
# Request Routing Algorithms – Least Outstanding Requests

- Chooses the next instance to receive the request by selecting the instance that has the lowest number of pending/unfinished requests
- Works with Application Load Balancer and Classic Load Balancer (HTTP/HTTPS)



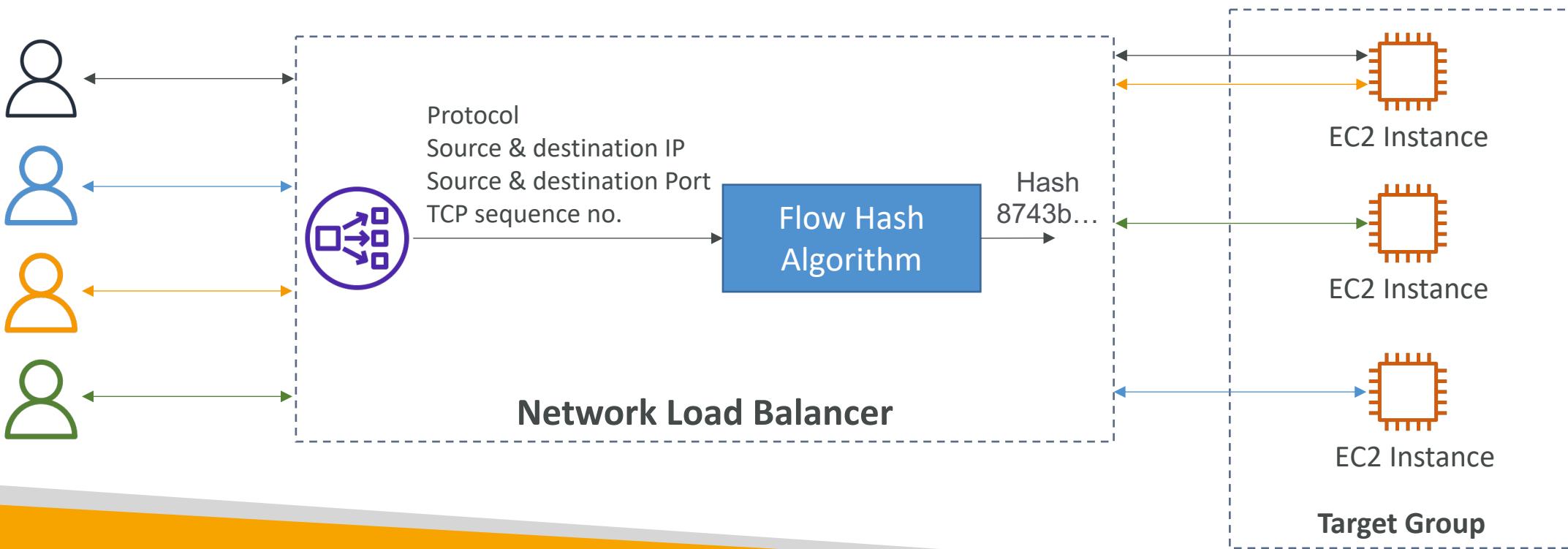
# Request Routing Algorithms – Round Robin

- Equally choose the targets from the target group
- Works with Application Load Balancer and Classic Load Balancer (TCP)



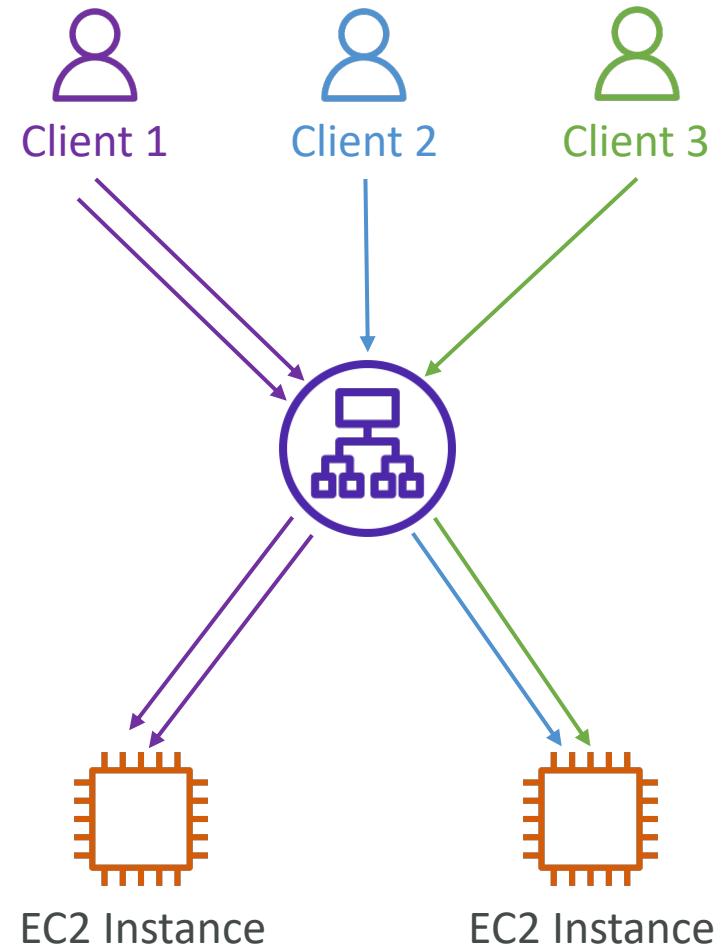
# Request Routing Algorithms – Flow Hash

- Selects a target based on the protocol, source/destination IP address, source/destination port, and TCP sequence number
- Each TCP/UDP connection is routed to a single target for the life of the connection
- Works with Network Load Balancer



# Sticky Sessions (Session Affinity)

- It is possible to implement stickiness so that the same client is always redirected to the same instance behind a load balancer
- This works for Classic Load Balancers & Application Load Balancers
- The “cookie” used for stickiness has an expiration date you control
- Use case: make sure the user doesn’t lose his session data
- Enabling stickiness may bring imbalance to the load over the backend EC2 instances



# Sticky Sessions – Cookie Names

- Application-based Cookies

- Custom cookie
  - Generated by the target
  - Can include any custom attributes required by the application
  - Cookie name must be specified individually for each target group
  - Don't use **AWSALB**, **AWSALBAPP**, or **AWSALBTG** (reserved for use by the ELB)
- Application cookie
  - Generated by the load balancer
  - Cookie name is **AWSALBAPP**
- Should be used if you need sticky sessions across all layers

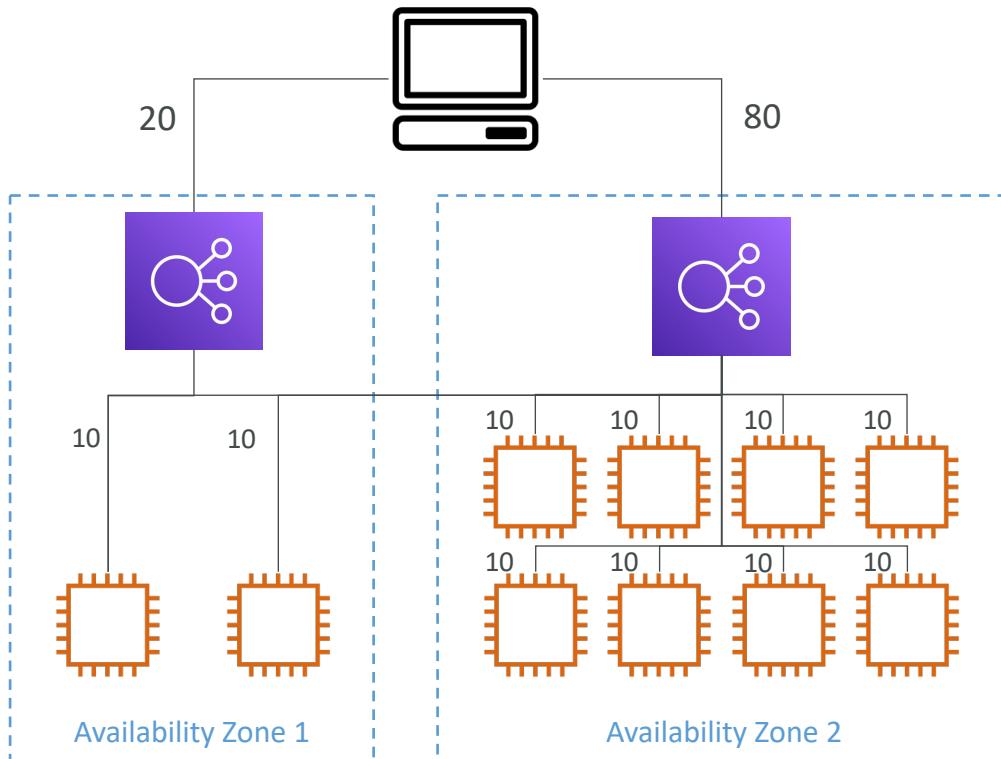
- Duration-based Cookies

- Cookie generated by the load balancer
- Cookie name is **AWSALB** for ALB, **AWSELB** for CLB

# Cross-Zone Load Balancing

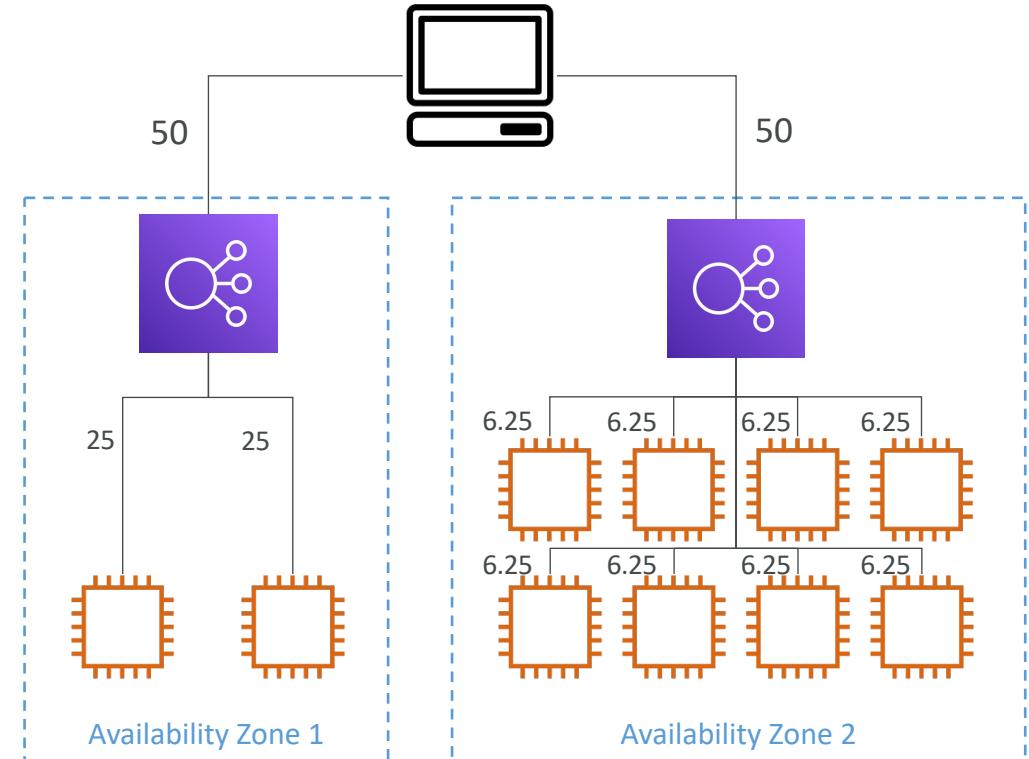
## With Cross Zone Load Balancing

Requests are distributed evenly across all registered instances in all AZ



## Without Cross Zone Load Balancing

Requests are distributed in the instances of the node of the Elastic Load Balancer



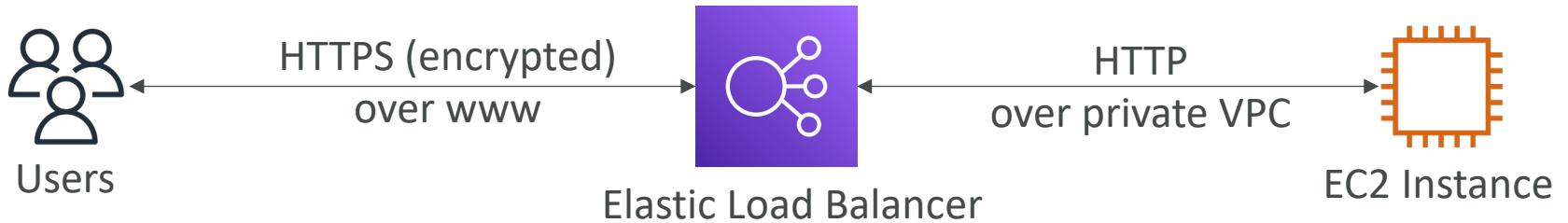
# Cross-Zone Load Balancing

- Application Load Balancer
  - Enabled by default (can be disabled at the Target Group level)
  - No charges for inter AZ data
- Network Load Balancer & Gateway Load Balancer
  - Disabled by default
  - You pay charges (\$) for inter AZ data if enabled
- Classic Load Balancer
  - Disabled by default
  - No charges for inter AZ data if enabled

# SSL/TLS - Basics

- An SSL Certificate allows traffic between your clients and your load balancer to be encrypted in transit (in-flight encryption)
- SSL refers to Secure Socket Layer, used to encrypt connections
- TLS refers to Transport Layer Security, which is a newer version
- Nowadays, **TLS certificates are mainly used**, but people still refer as SSL
- Public SSL certificates are issued by Certificate Authorities (CA)
- Comodo, Symantec, GoDaddy, GlobalSign, DigiCert, Let's Encrypt, ...
- SSL certificates have an expiration date (you set) and must be renewed

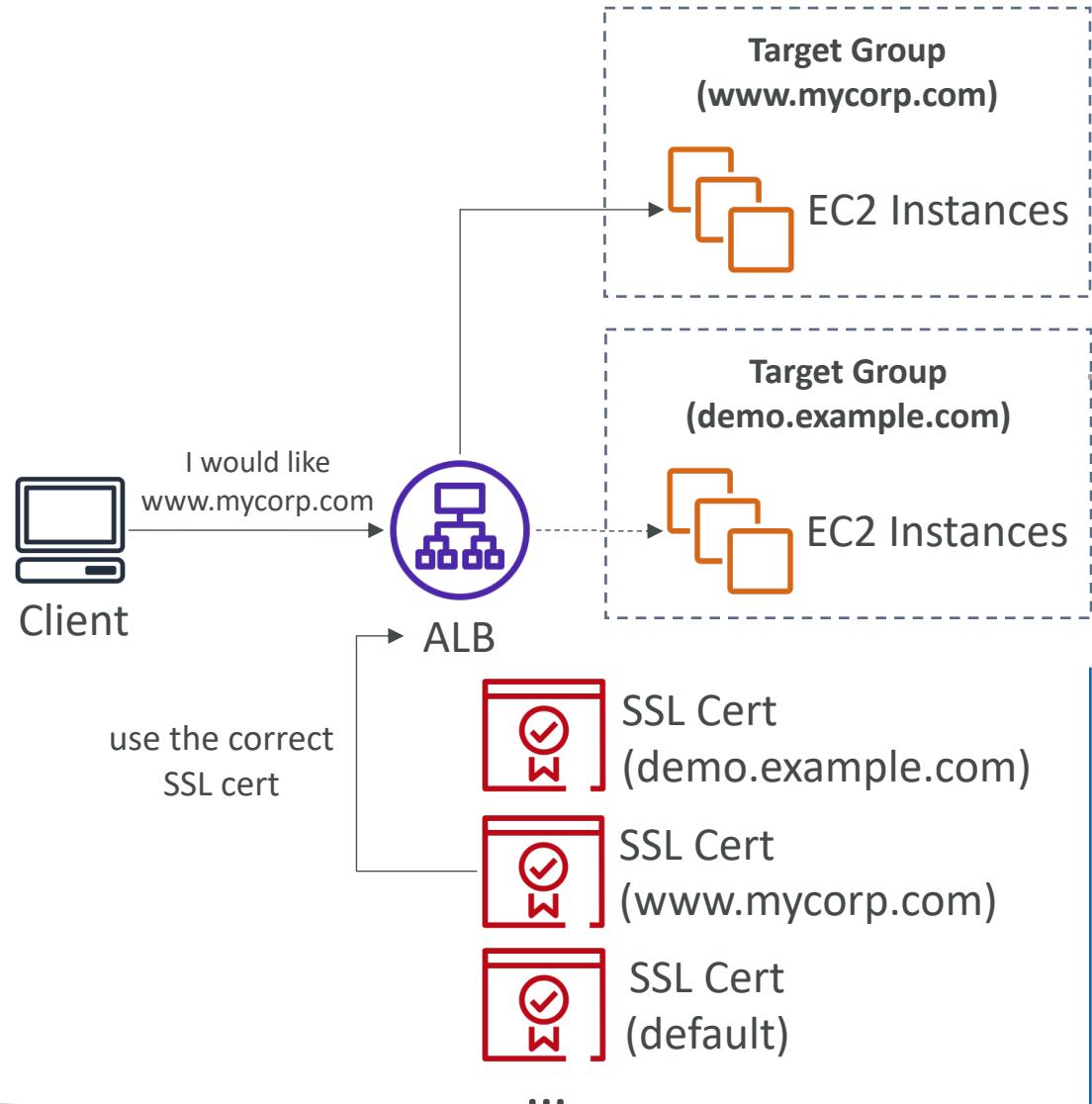
# Elastic Load Balancer – SSL Certificates



- The load balancer uses an X.509 certificate (SSL/TLS server certificate)
- You can manage certificates using ACM (AWS Certificate Manager)
- You can create/upload your own certificates alternatively
- HTTPS listener:
  - You must specify a default certificate
  - You can add an optional list of certs to support multiple domains
  - **Clients can use SNI (Server Name Indication) to specify the hostname they reach**
  - Ability to specify a Security Policy (for compliance, features, compatibility or security)

# SSL – Server Name Indication (SNI)

- SNI solves the problem of loading multiple SSL certificates onto one web server (to serve multiple websites)
- It's a “newer” protocol, and requires the client to indicate the hostname of the target hostname in the initial SSL handshake
- The server will then find the correct certificate, or return the default one
- Only works for ALB & NLB



# Elastic Load Balancers – SSL Certificates

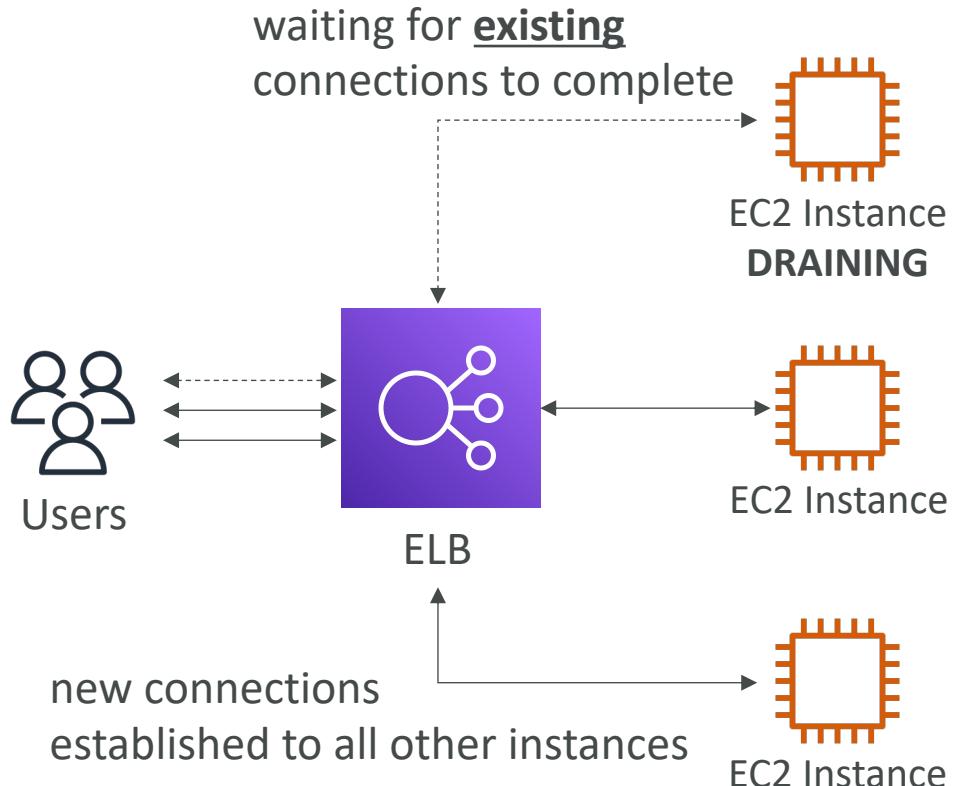
- **Classic Load Balancer**
  - Supports only one SSL certificate
  - The SSL certificate can have many Subject Alternate Name (SAN), but the SSL certificate must be changed anytime a SAN is added / edited / removed
  - Must use multiple CLB for multiple hostnames with multiple SSL certificates
  - Better to use ALB with Server Name Indication (SNI) if possible
- **Application Load Balancer**
  - Supports multiple listeners with multiple SSL certificates
  - Uses Server Name Indication (SNI) to make it work
- **Network Load Balancer**
  - Supports multiple listeners with multiple SSL certificates
  - Uses Server Name Indication (SNI) to make it work

# HTTPS/SSL Listener – Security Policy

- A combination of SSL protocols, SSL ciphers, and Server Order Preference option supported during SSL negotiations
- Predefined Security Policies (e.g., ELBSecurityPolicy-2016-08)
- For **ALB and NLB**
  - Frontend connections, you can use a predefined Security Policy
  - Backend connections, **ELBSecurityPolicy-2016-08** Security Policy is always used
- Use **ELBSecurityPolicy-TLS** policies
  - To meet compliance and security standards that require certain TLS protocol version
  - To support older versions of SSL/TLS (legacy clients)
- Use **ELBSecurityPolicy-FS** policies, if you require **Forward Secrecy**
  - Provides additional safeguards against the eavesdropping of encrypted data
  - Using a unique random session key

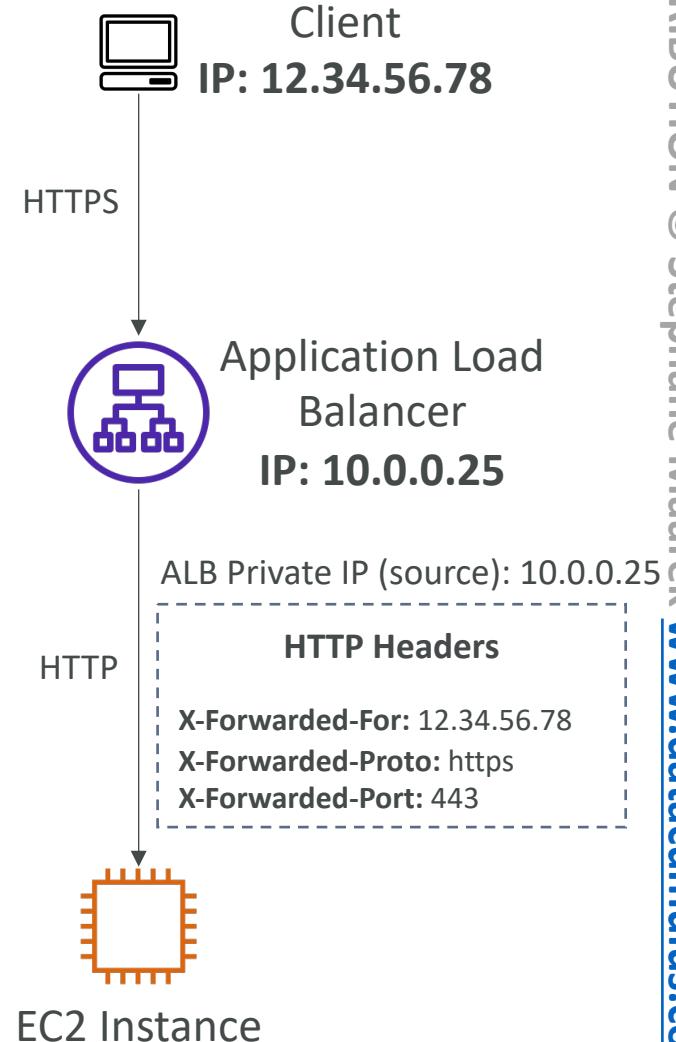
# Connection Draining

- Feature naming
  - Connection Draining – for CLB
  - Deregistration Delay – for ALB & NLB
- Time to complete “in-flight requests” while the instance is de-registering or unhealthy
- Stops sending new requests to the EC2 instance which is de-registering
- Between 1 to 3600 seconds (default: 300 seconds)
- Can be disabled (set value to 0)
- Set to a low value if your requests are short



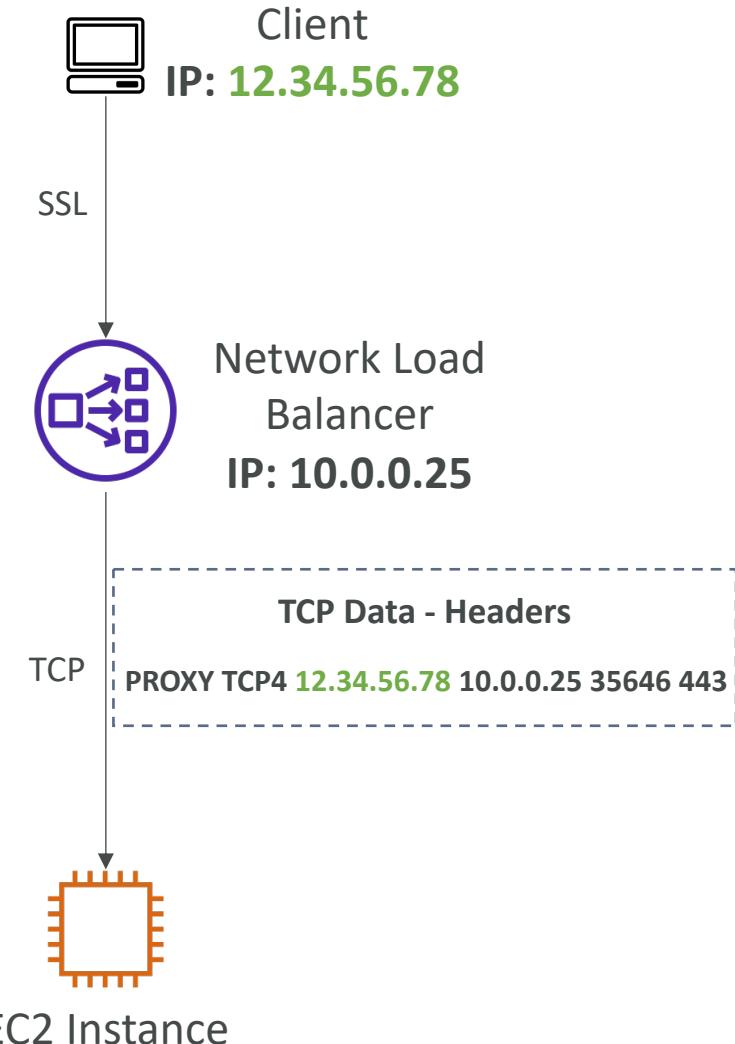
# X-Forwarded Headers (HTTP)

- Non-standard HTTP headers that have an **X-Forwarded** prefix
- Used by ELB to forward client information to the targets (e.g., client IP address)
- You can use to log client requests on your server
- Supported by Classic Load Balancer (HTTP/HTTPS) and Application Load Balancer
- **X-Forwarded-For**
  - Contains the IP address of the client
  - May contain comma-separated list of multiple IP addresses, such as proxies (left-most is the client IP address)
- **X-Forwarded-Proto** – the protocol used between client and the ELB (HTTP/HTTPS)
- **X-Forwarded-Port** – the destination port the client used to connect to the ELB



# Proxy Protocol

- An Internet protocol used to carry information from the source (requesting the connection) to the destination (where connection was requested)
- If the LB terminates the connection, the source IP address of the client cannot be preserved
- We use the Proxy Protocol to pass the source/destination IP address and port numbers
- The load balancer prepends a proxy protocol header to the TCP data
- Available for both Classic Load Balancer (TCP/SSL) and Network Load Balancer

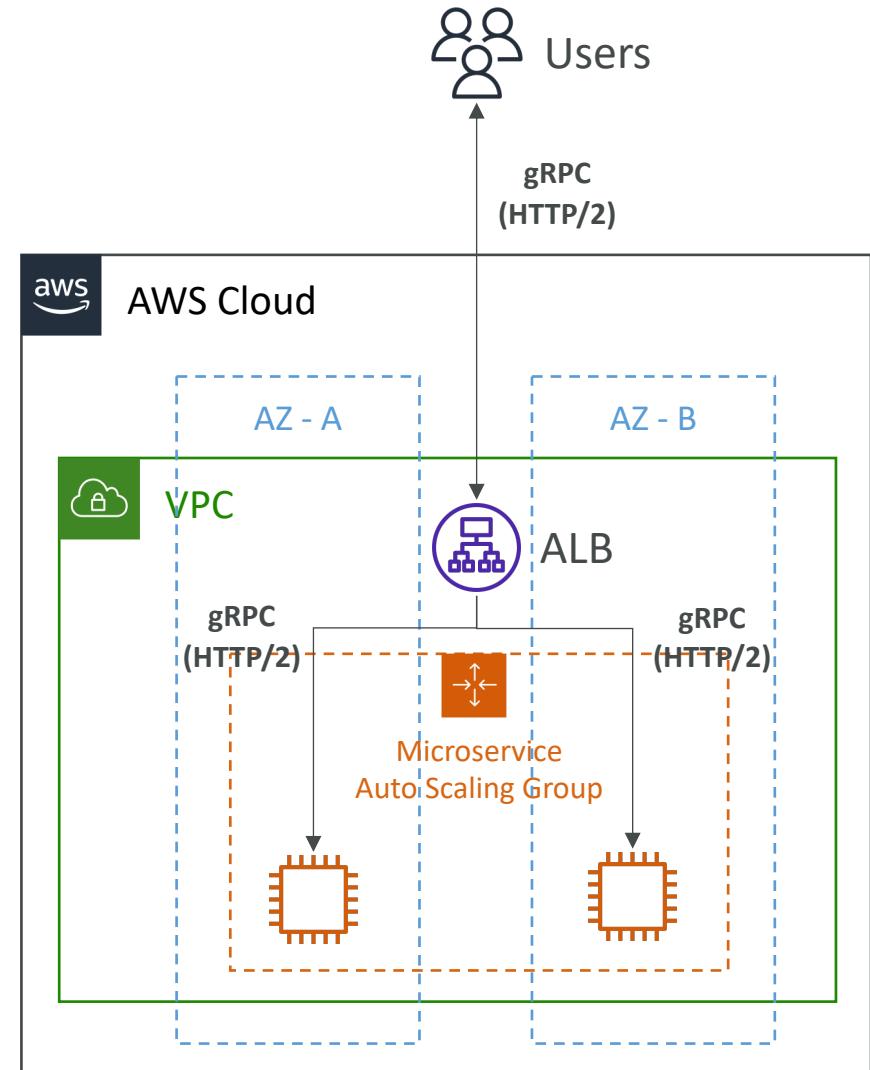


# Proxy Protocol

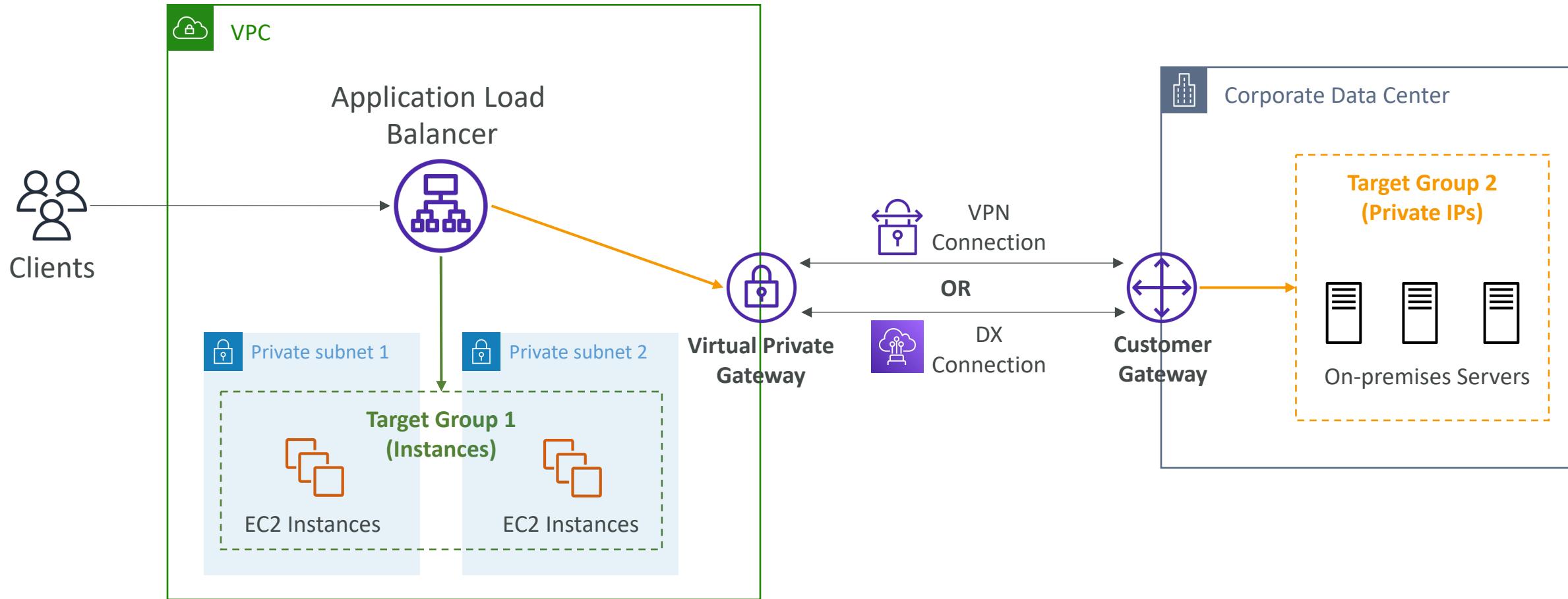
- CLB uses Proxy Protocol version 1 & NLB uses Proxy Protocol version 2
- For Network Load Balancer
  - Target with Instance ID and ECS Tasks: the source IP of the client is preserved
  - Target with IP Address:
    - TCP & TLS: the source IP of the client isn't preserved, enable Proxy Protocol
    - UDP & TCP\_UDP: the source IP of the client is preserved
- Load balancer should not be behind proxy server, otherwise backend may receive duplicate configuration information resulting in error
- Proxy protocol is not needed when using an Application Load Balancer, as ALB already inserts **HTTP X-Forwarded-For** headers

# Application Load Balancer & gRPC

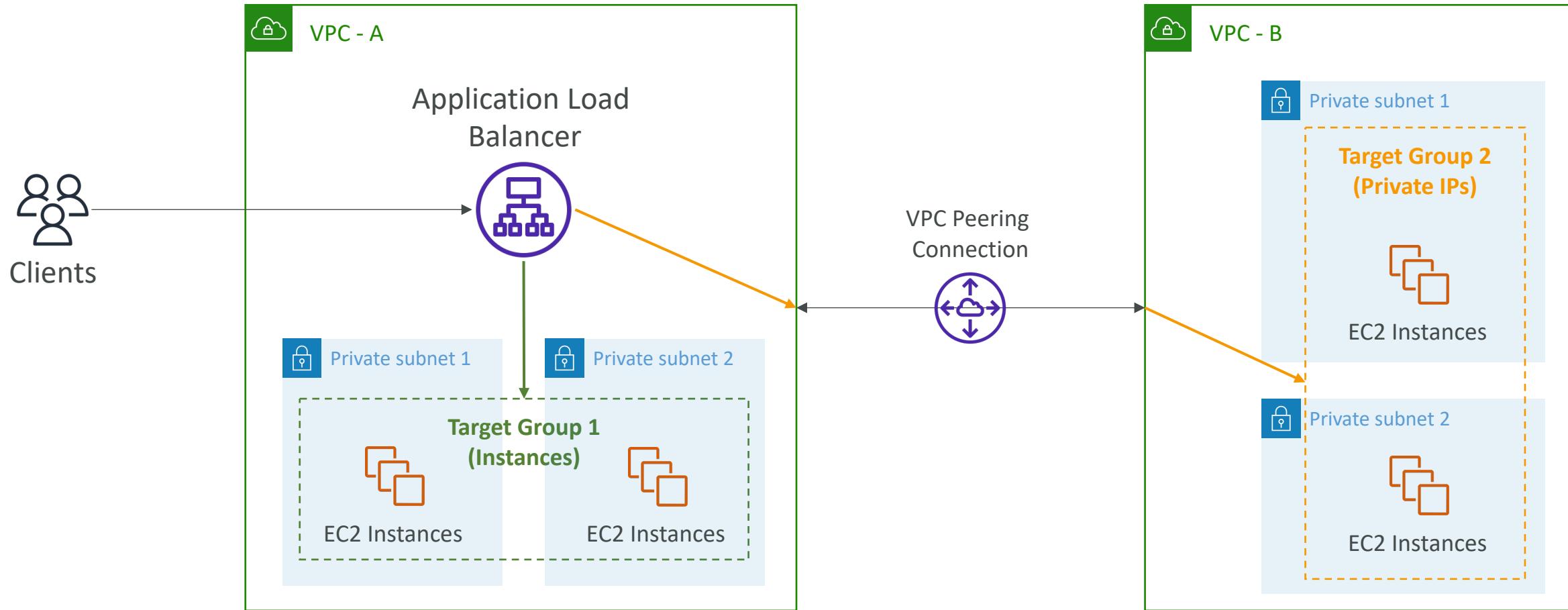
- gRPC is a popular choice for microservice integrations using HTTP/2
- The Application Load Balancer fully supports gRPC
  - Supports gRPC health checks from target group
  - Content based routing feature to route to the appropriate service
  - Supports all kind of gRPC communication (including bidirectional streaming)
  - Listener protocol is HTTPS
- Note: gRPC would work with NLB but you wouldn't have any "HTTP-specific" features



# Load Balancing across on-premises servers



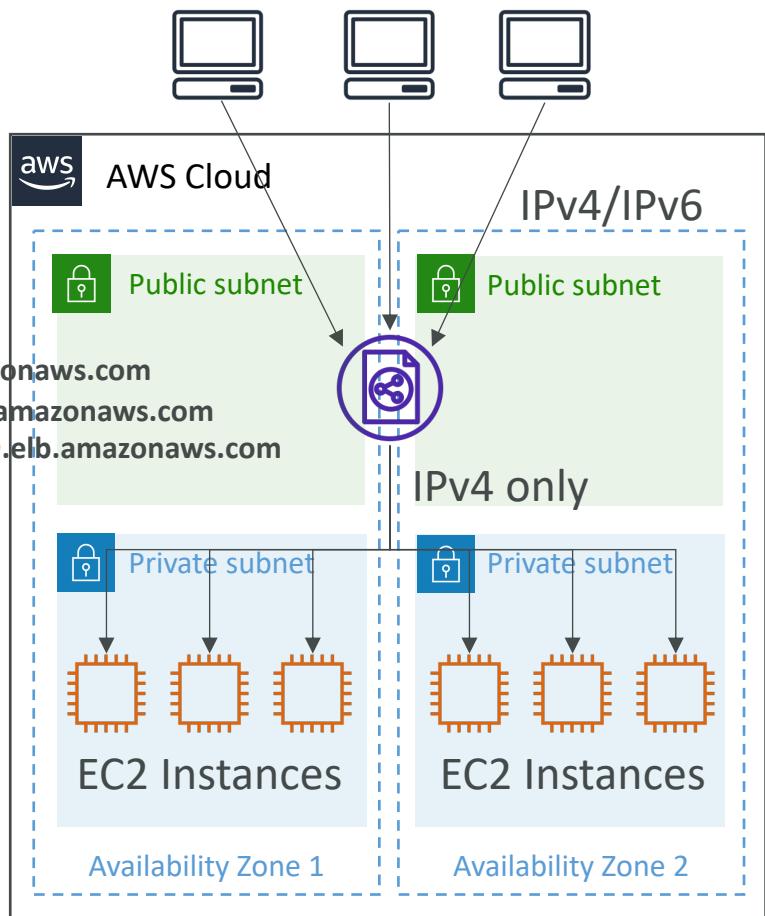
# Load Balancing across Peered VPCs



# Internet-facing vs. Internal Load Balancer

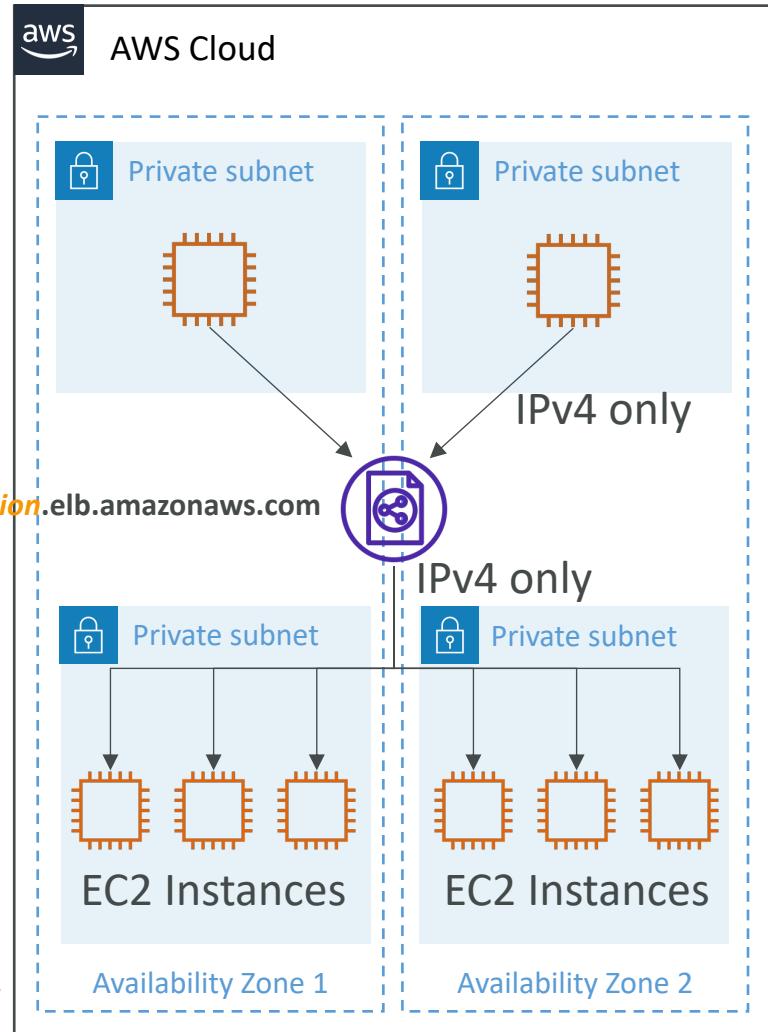
**Internet-facing ELB  
(IPv4 or Dualstack)**

*name-1234567890.region.elb.amazonaws.com*  
*ipv6.name-1234567890.region.elb.amazonaws.com*  
*dualstack.name-1234567890.region.elb.amazonaws.com*



**Internal ELB  
(IPv4)**

*internal-name-1234567890.region.elb.amazonaws.com*



**NOTE: ELB connects to backend instances using Private IP address only**

# Q1

Your company has deployed a bursty web application to AWS and would like to improve the user experience. It is important for only the web host to have the private key for Transport Layer Security (TLS), so the Classic Load Balancer has a listener on Transmission Control Protocol (TCP) port 443. What are some approaches that you can use to reduce latency and improve the scale-out process for the application?

1. Use an Application Load Balancer in front of the application, enabling better utilization of multiple target groups with different HTTP paths and hosts.
2. Configure enhanced networking on the Classic Load Balancer for lower latency load balancing.
3. Use Amazon Certificate Manager (ACM) to distribute new certificates to Amazon CloudFront to accomplish handling content at the edge.
4. Use a Network Load Balancer in front of your application to increase network performance.

## Q2

Why is referencing the Application Load Balancer or Classic Load Balancer by its DNS CNAME recommended?

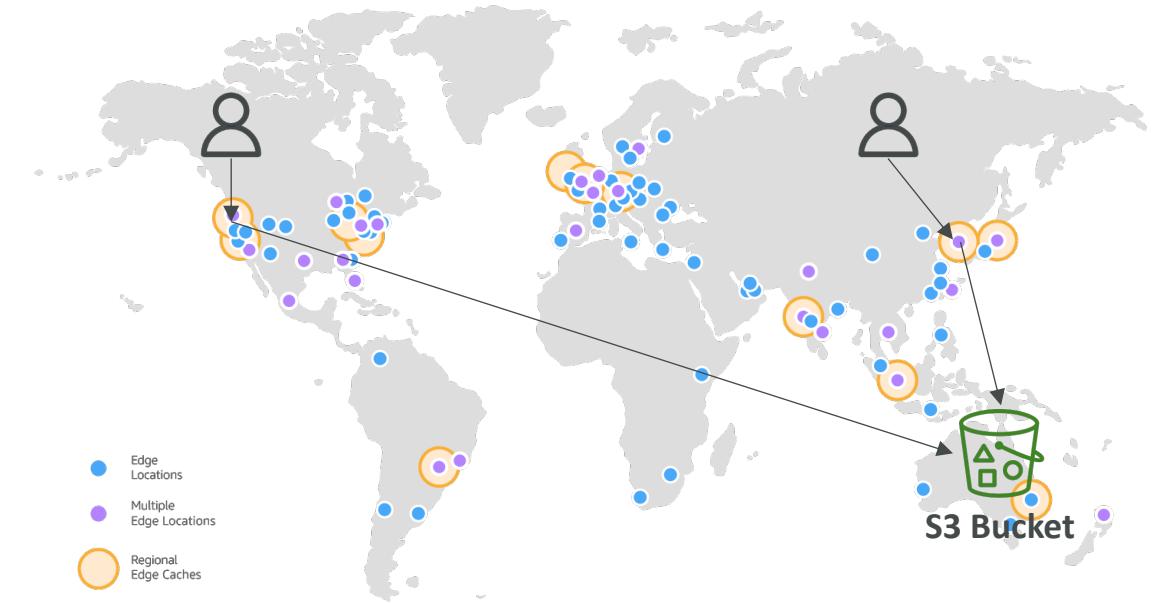
1. IP addresses may change as the load balancers scale.
2. DNS CNAMEs provide a lower latency than IP addresses.
3. You want to preserve the source IP of the client.
4. IP addresses are public and open to the Internet.

# AWS CloudFront – CDN Service

# AWS CloudFront



- Content Delivery Network (CDN)
- Improves read performance, content is cached at the edge
- 225+ Point of Presence globally (215+ Edge Locations & 13 Regional Edge Caches)
- Protect against Network and Application layer attacks (e.g., DDoS attacks)
- Integration with AWS Shield, AWS WAF, and Route 53
- Can expose external HTTPS and can talk to internal HTTPS backends
- Supports WebSocket protocol



Source: <https://aws.amazon.com/cloudfront/features/>

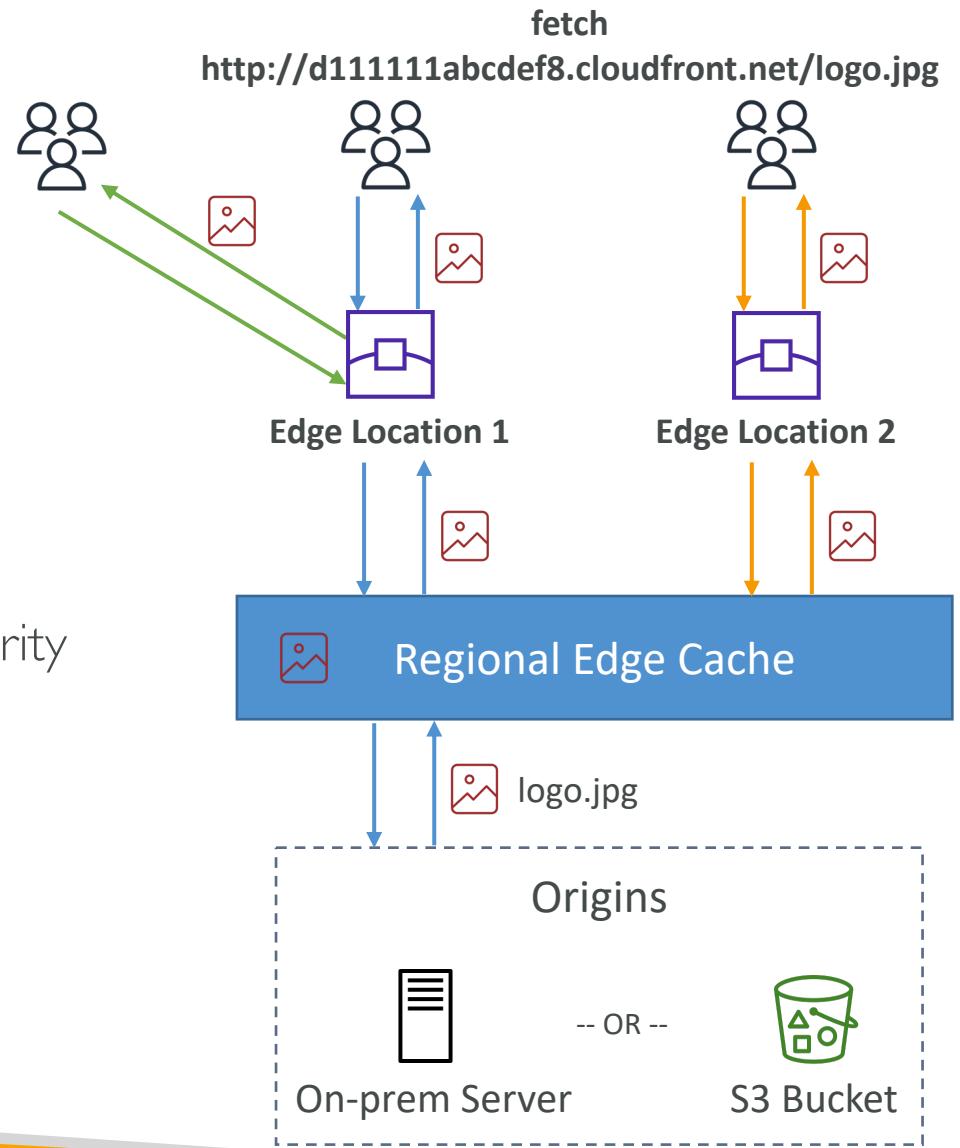
# Edge Locations & Regional Edge Caches

- **Edge Locations**

- Serve content quickly/directly to users
- Cache more popular content

- **Regional Edge Caches**

- Serve content to Edge Locations
- Cache less popular content that might suddenly find popularity
- Larger cache than Edge Location (objects remain longer)
- Improve performance, reduce the load on your origins
- Dynamic content doesn't pass through it (directly to origin)



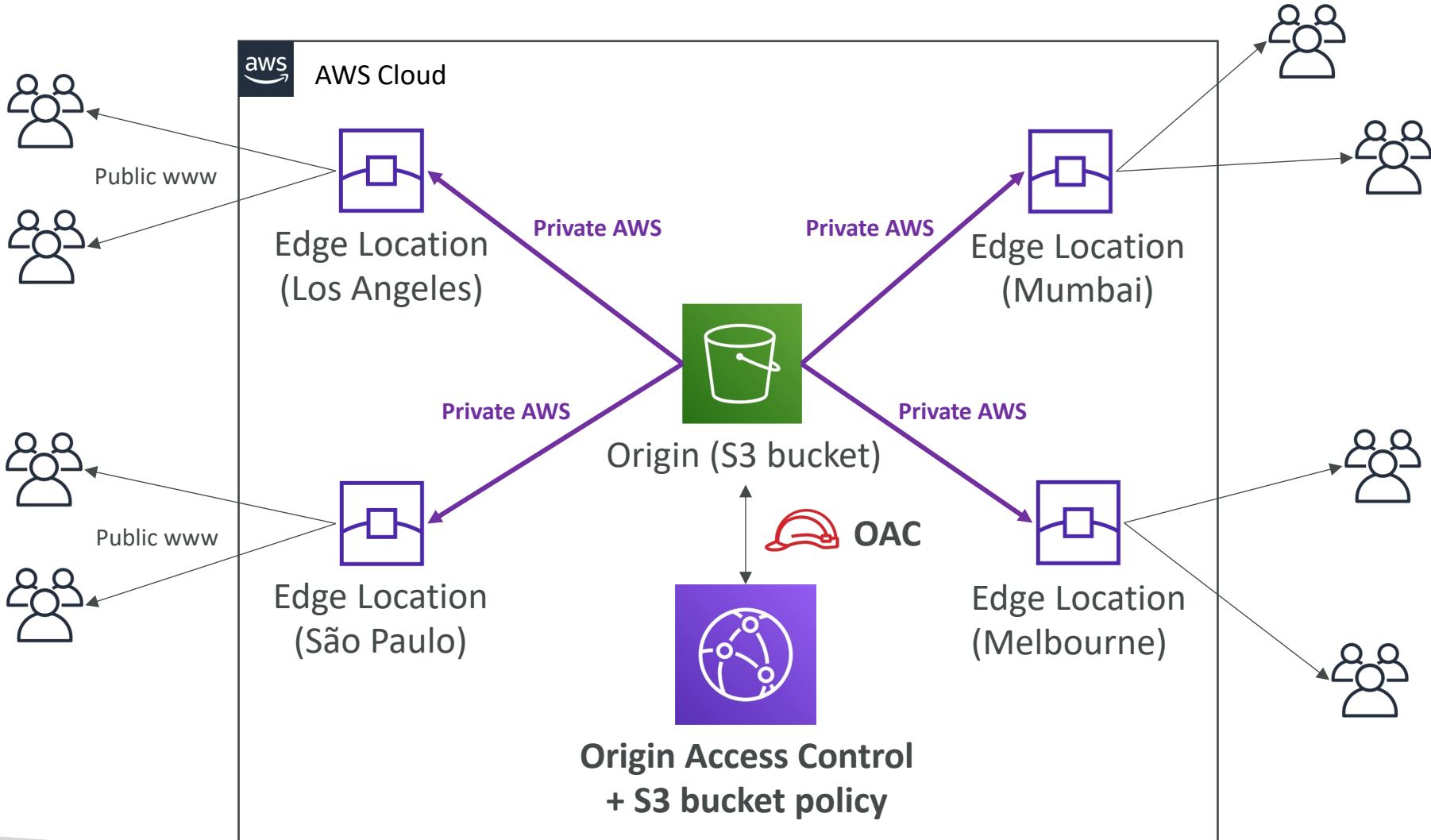
# CloudFront Components

- **Distribution**
  - Identified by a domain (e.g., d11111abcdef8.cloudfront.net)
  - You can use this distribution domain name to access the website
  - You can also use Route53 CNAME (non-root) or Alias (root & non-root) which points to distribution's domain name
- **Origin**
  - Where actual contents resides (S3 Bucket, ALB, HTTP Server, API Gateway, etc.)
- **Cache Behavior**
  - Cache configurations (e.g., Object Expiration, TTL, Cache invalidations)

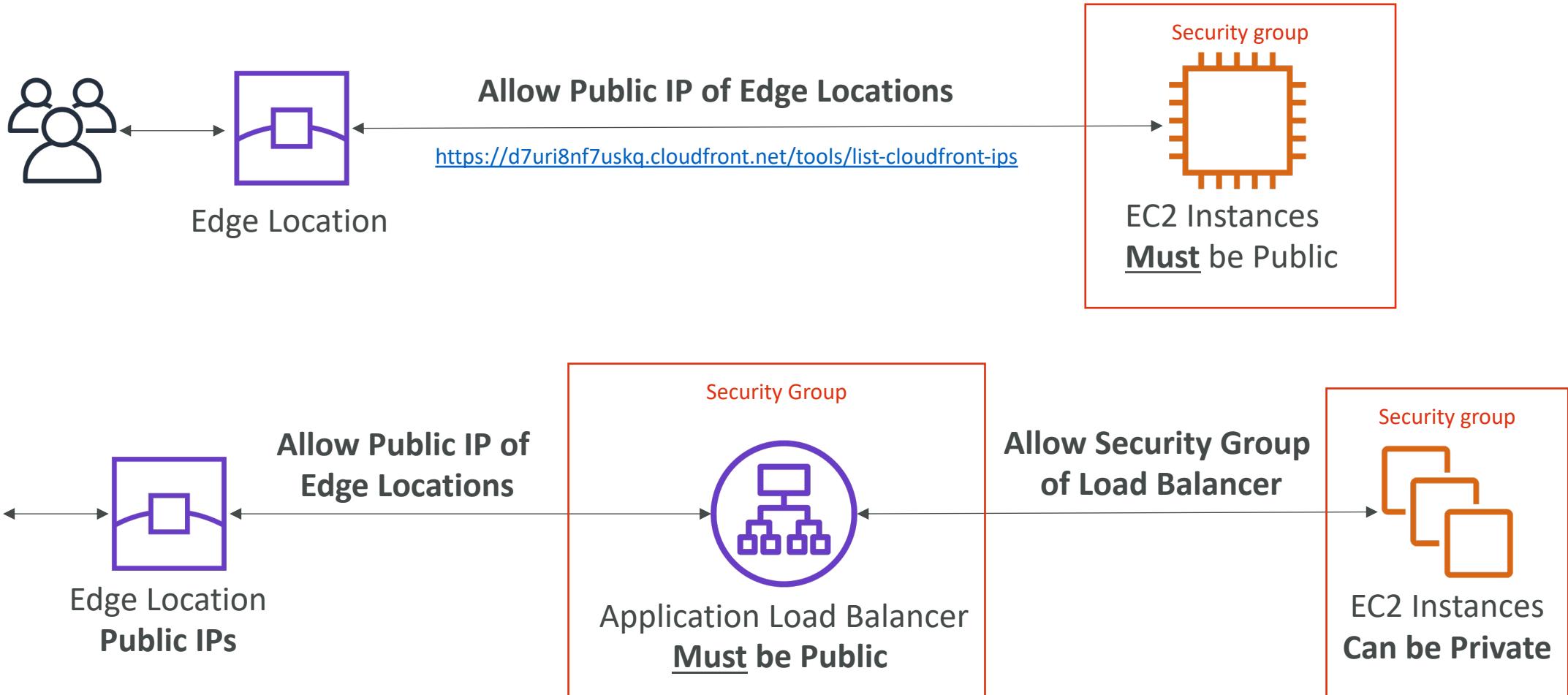
# CloudFront Origins

- S3 Bucket
  - For distributing files
  - Enhanced security with CloudFront Origin Access Control (OAC) – previously OAI
  - CloudFront can be used as an ingress (to upload files to S3)
- S3 Bucket configured as a website
  - First, enable Static Website hosting on the bucket
- MediaStore Container & MediaPackage Endpoint
  - To deliver Video On Demand (VOD) or live streaming video using AWS Media Services
- Custom Origin (HTTP)
  - EC2 instance
  - Elastic Load Balancer (CLB or ALB)
  - API Gateway (for more control... otherwise use API Gateway Edge)
  - Any HTTP backend you want

# CloudFront Origins – S3 as an Origin

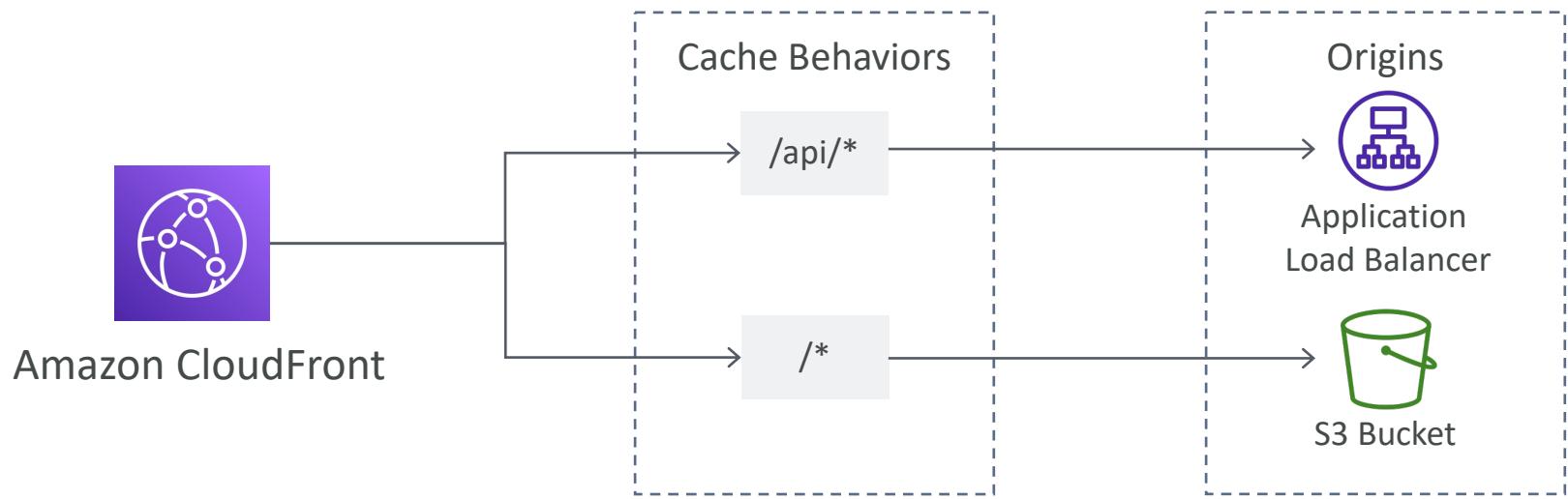


# CloudFront Origins – ALB or EC2 as an origin



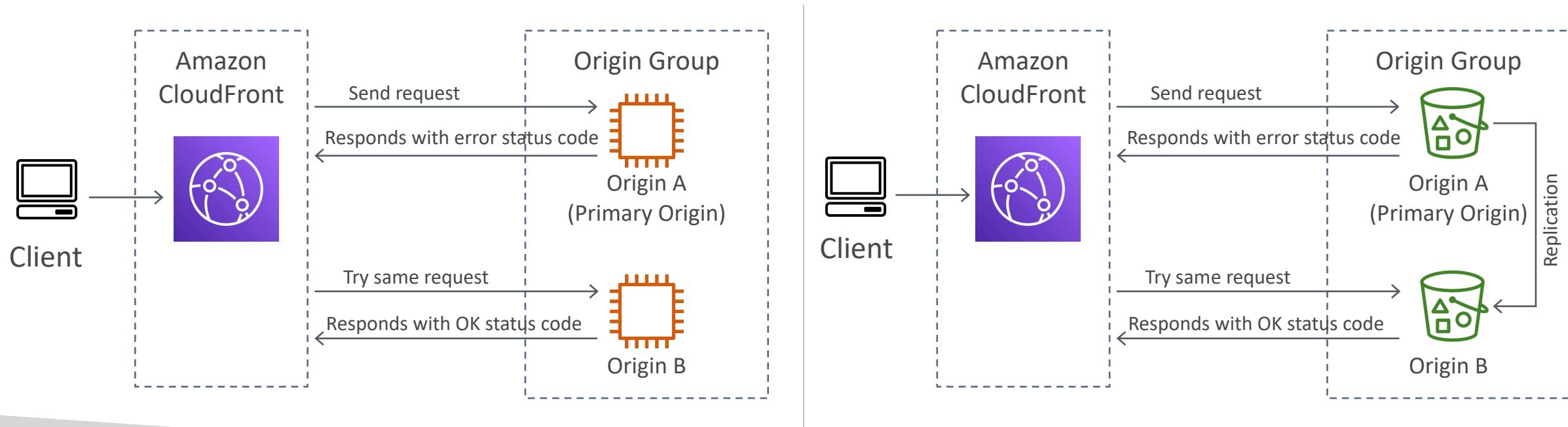
# CloudFront – Multiple Origin

- To route to different kind of origins based on the content type
- Based on path pattern:
  - /images/\*
  - /api/\*
  - /\*



# CloudFront – Origin Groups

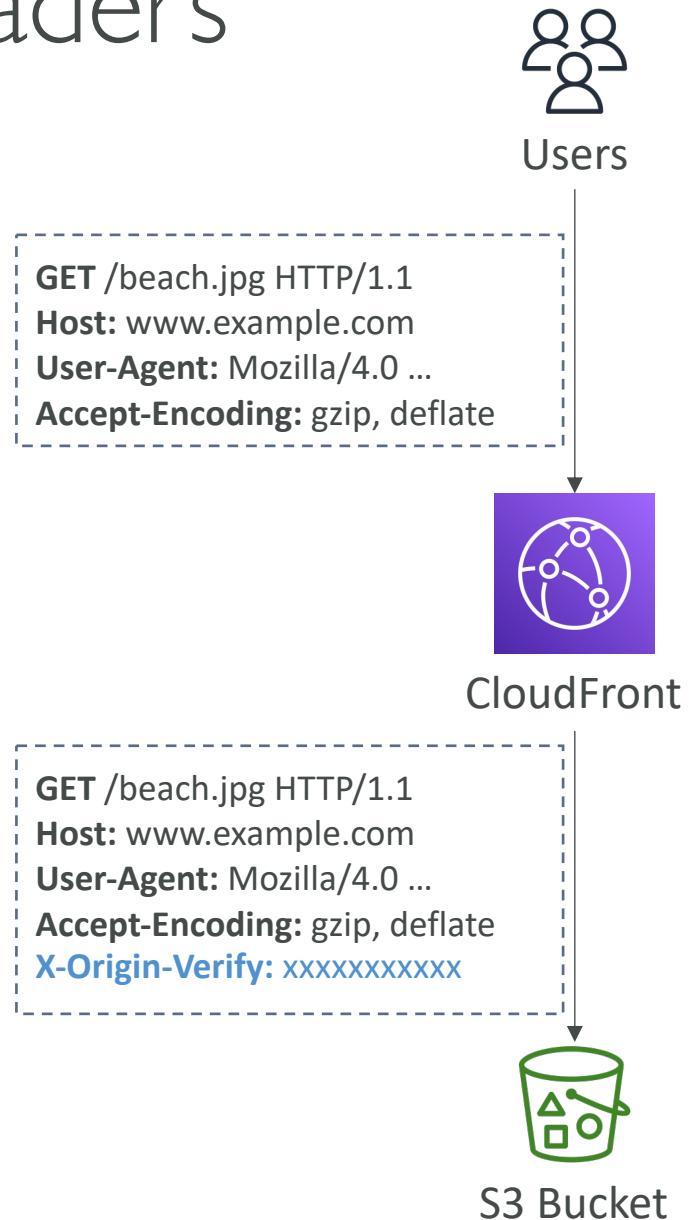
- To increase high-availability and do failover
- Origin Group: one primary and one secondary origin
- If the primary origin fails, the second one is used



S3 + CloudFront – Region-level High Availability

# CloudFront – Origin Custom Headers

- Add custom headers to requests CloudFront sends to your origin
- Can be customized for each origin
- Supports custom and S3 origins
- Use cases:
  - Identify which requests coming from CloudFront or a particular distribution
  - Control access to content (configure origin to respond to requests only when they include a custom header)



# CloudFront HTTP Headers

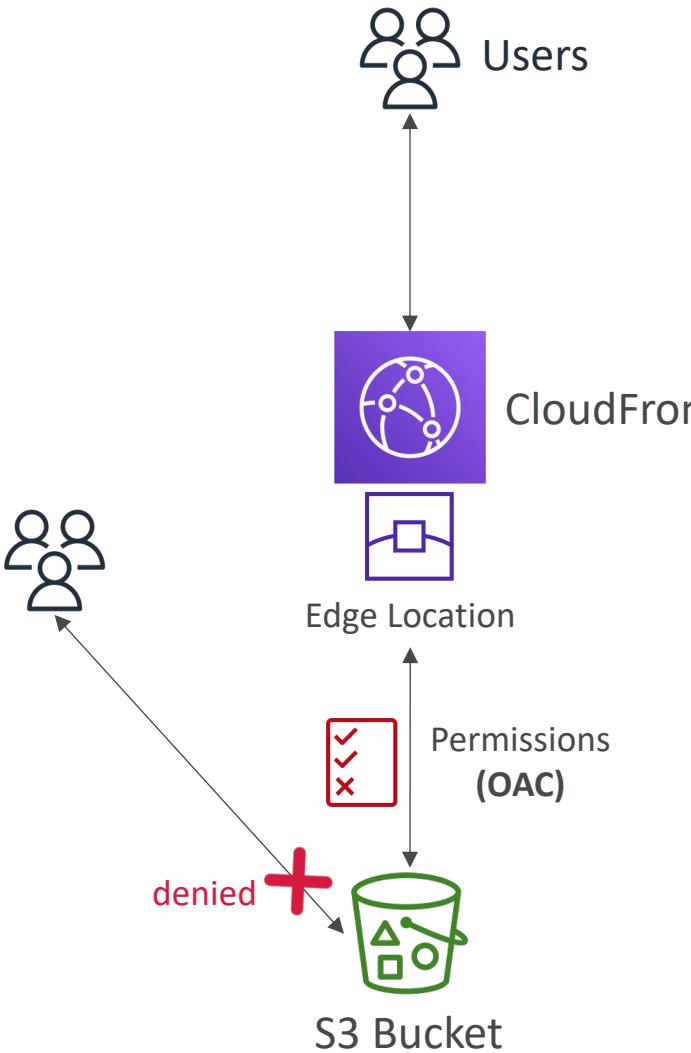
- Specific HTTP headers added based on the viewer request
- **Viewer's Device Type Headers** (based on User-Agent)
  - CloudFront-Is-(Android/Desktop/IOS/Mobile/SmartTV/Tablet)-Viewer
- **Viewer's Location Headers** (based on viewer's IP address)
  - CloudFront-Viewer-(City/Country/Latitude/Longitude, ...)
- **Viewer's Request Protocol & HTTP Version**
  - CloudFront-Forwarded-Proto & CloudFront-Viewer-Http-Version
- You can include them in the Cache Key (using **Legacy Cache Settings** or **Cache Policies**) or receive them at your origin (using **Origin Request Policies**)

# CloudFront – Restrict Access to S3 Buckets

- Prevent direct access to files in your S3 buckets (only access through CloudFront)
- First, create an **Origin Access Control** and associate it with your distribution (previously known as OAI)
- Second, edit your **S3 bucket policy** so that only OAC has permission

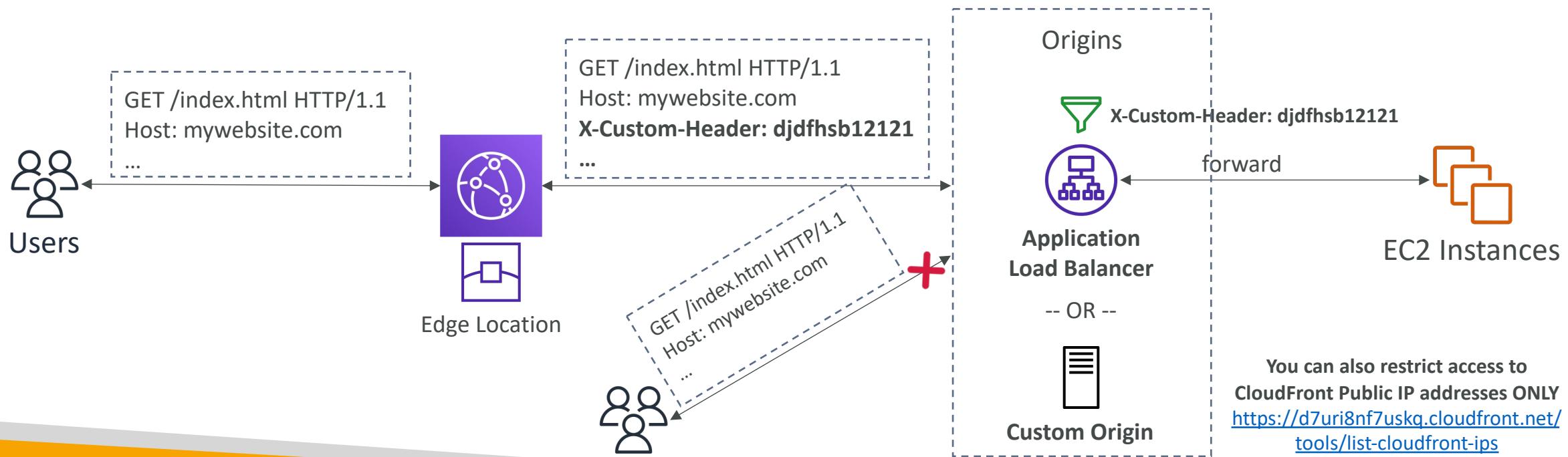
```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "service": "cloudfront.amazonaws.com"
      },
      "Action": "s3:GetObject",
      "Resource": "arn:aws:s3:::mybucket/*",
      "Condition": {
        "StringEquals": {
          "AWS:SourceArn": "arn:aws:cloudfront::ACCOUNT_ID:distribution/EDFDVBD6EXAMPLE"
        }
      }
    }
  ]
}
```

**Bucket Policy**

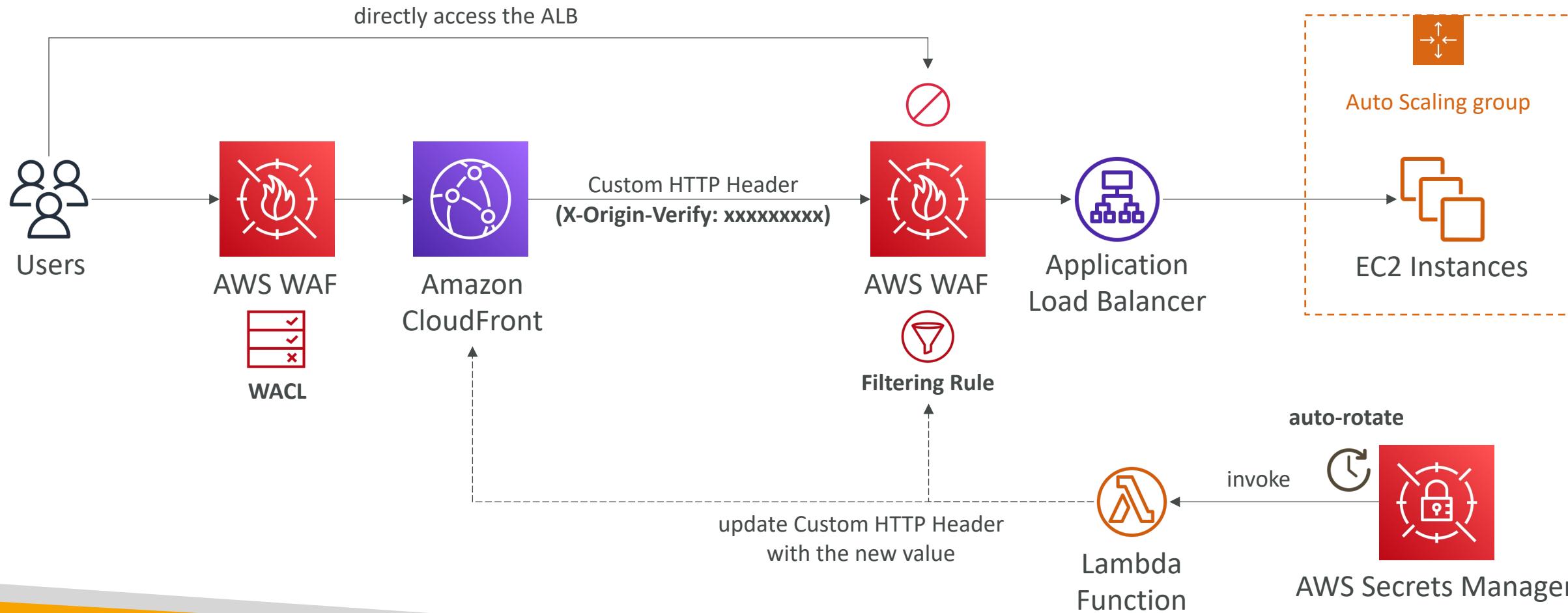


# CloudFront – Restrict Access to Application Load Balancers and Custom Origins

- Prevent direct access to your ALB or Custom Origins (only access through CloudFront)
- First, configure CloudFront to add a **Custom HTTP Header** to requests it sends to the ALB
- Second, configure the ALB to only forward requests that contain that Custom HTTP Header
- Keep the custom header name and value secret!

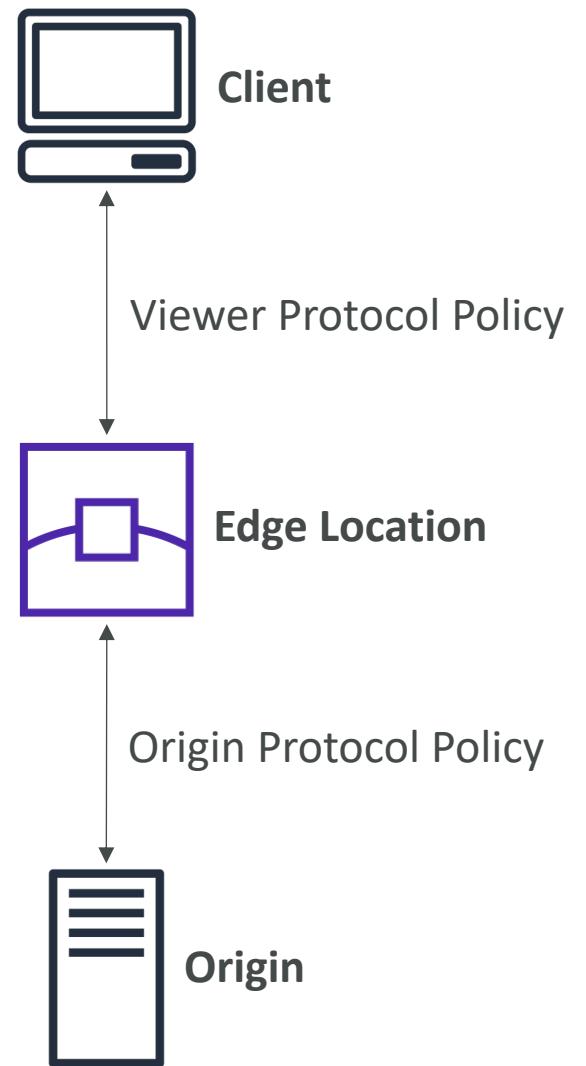


# Solution Architecture – Enhance CloudFront Origin Security with AWS WAF & AWS Secrets Manager



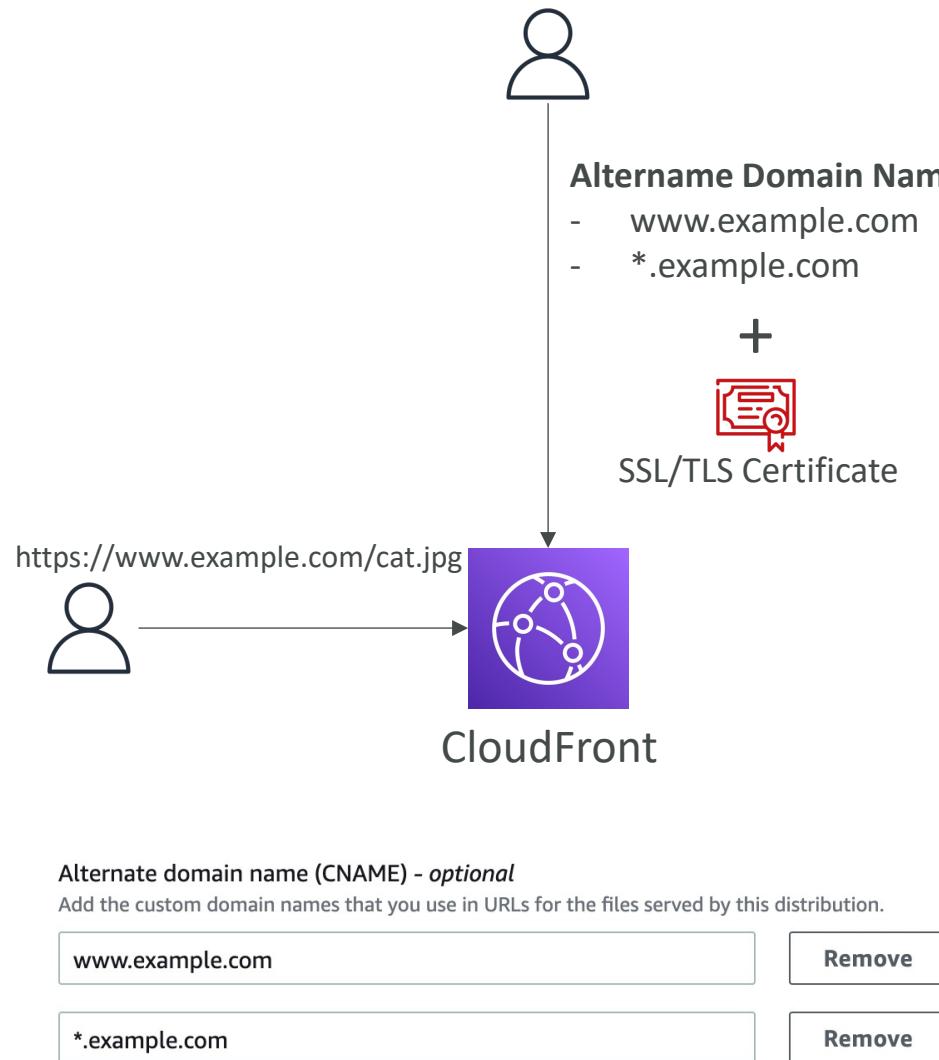
# CloudFront and HTTPS

- **Viewer Protocol Policy:**
  - HTTP and HTTPS
  - Redirect HTTP to HTTPS
  - HTTPS Only
- **Origin Protocol Policy (HTTP or S3):**
  - HTTP Only (default for S3 Static Website)
  - HTTPS Only
  - Or Match Viewer  
(HTTP => HTTP & HTTPS => HTTPS)
- **Note:**
  - S3 bucket “static websites” don’t support HTTPS
  - You must use a valid SSL/TLS certificate between CloudFront and your origin (can’t use self-signed certificates)



# Alternate Domain Names

- Use your own domain name instead of the domain assigned by CloudFront to your distribution
- Example:  
`http://d111111abcdef8.cloudfront.net/cat.jpg`  
 $\Rightarrow$  `http://www.example.com/cat.jpg`
- You **must** have a valid SSL/TLS Certificate from an authorized CA that covers:
  - Your domain name
  - All Alternate Domain Names you added to your distribution
- You can use wildcards in Alternate Domain Names (e.g., \*.example.com)



# CloudFront – SSL Certificates

- Default CloudFront Certificate (\*.cloudfront.net)
  - Used with default CloudFront domain names assigned to your distribution
    - `https://d111111abcdef8.cloudfront.net/cat.jpg`
- Custom SSL Certificate
  - When using your own domain name – Alternate Domain Names (e.g., `https://www.example.com`)
  - CloudFront can serve HTTPS requests using:
    - Server Name Indication (SNI) – Recommended
    - Dedicated IP Address in Each Edge Location (expensive)
  - You can use
    - Certificates provided by ACM
    - 3<sup>rd</sup> party certificates uploaded to ACM or IAM Certificate Store (manually rotate when expired)
  - Certificate must be created/imported in US East (N.Virginia) Region
- Ability to specify a Security Policy (min. SSL/TLS protocol & ciphers to use)

# End-to-End Encryption: CloudFront, ALB, EC2

CloudFront



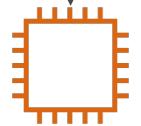
HTTPS

- CloudFront
  - Origin protocol policy: HTTPS Only
  - Install an SSL / TLS certificate on your custom origin
  - The origin certificate must include either the Origin domain field (configured in CloudFront) or the domain in the “Host” header if it’s forwarded to Origin
  - Does not work with self-signed certificate
- Application Load Balancer
  - Use a certificate provided by AWS Certificate Manager or imported into ACM
- EC2 Instance
  - ACM is not supported on EC2
  - Can use a third-party SSL certificate (any domain name)
  - Can use a self-signed certificate (ALB does not verify the certificate itself)

ALB



HTTPS

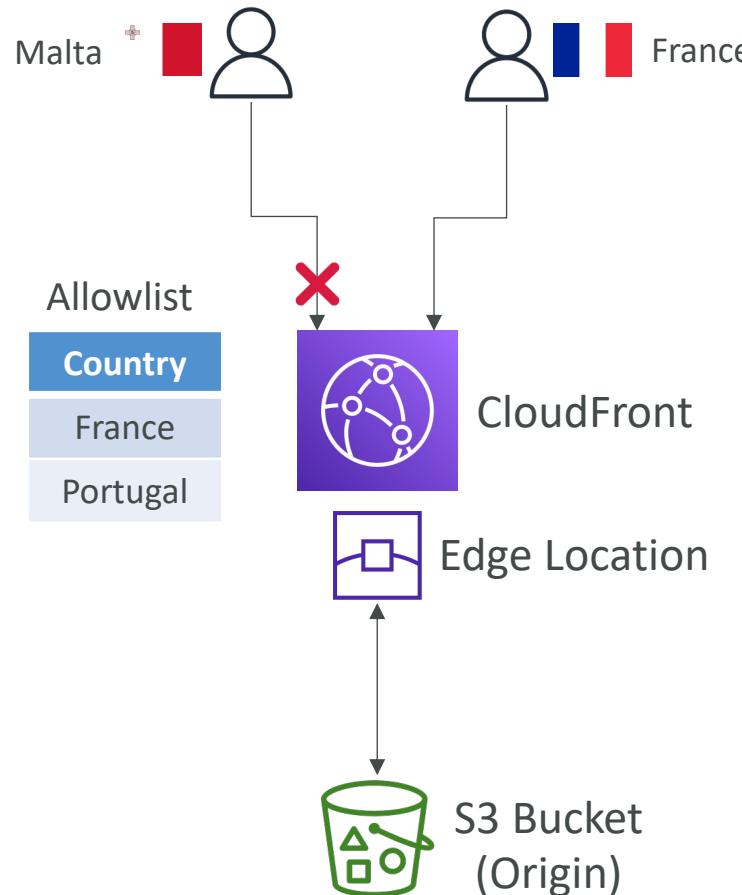


EC2 Instance

# CloudFront – Restrict Content Geographically

- Prevent users in specific locations from accessing your content/distribution
- CloudFront Geo Restriction
  - Restrict access at the country level (country determined using a 3<sup>rd</sup> party GeolP database)
  - Allow list – allow access only if is from one of the approved countries
  - Block list – prevent access if users in one of the countries on a blacklist of banned countries
  - Applied to an entire CloudFront distribution
- Use case: Copyright Laws to control access to content

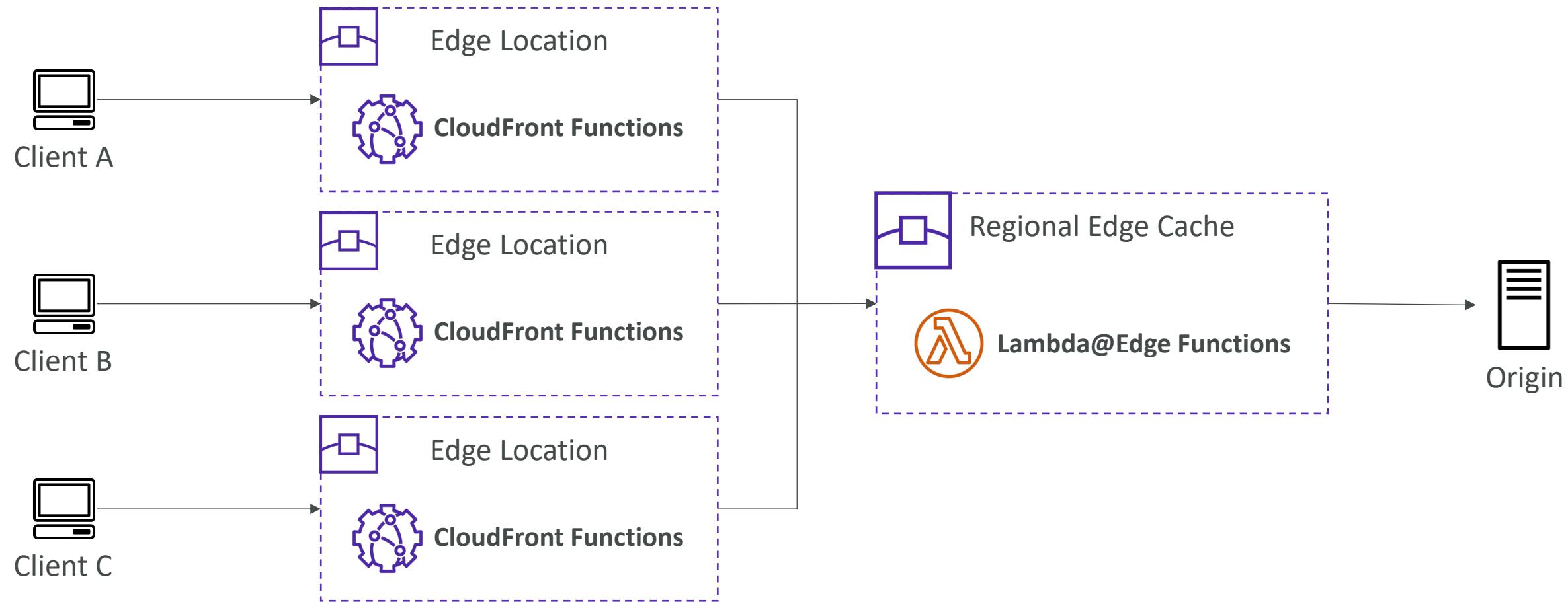
## CloudFront Geo Restriction



# CloudFront – Customization At The Edge

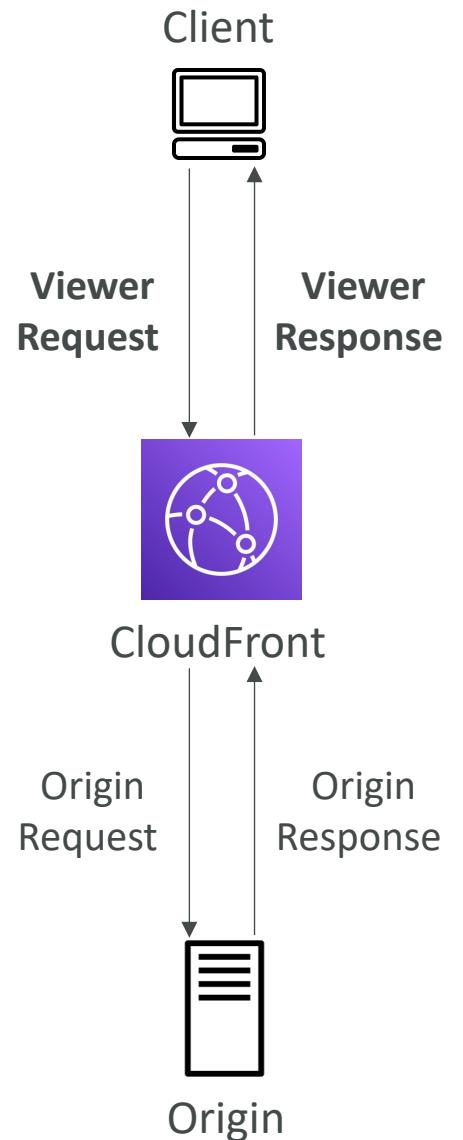
- Many modern applications execute some form of the logic at the edge
- **Edge Function:**
  - A code that you write and attach to CloudFront distributions
  - Runs close to your users to minimize latency
  - Doesn't have any cache, only to change requests/responses
  - CloudFront provides two types: **CloudFront Functions & Lambda@Edge**
- Use cases:
  - Manipulate HTTP requests and responses
  - Implement request filtering before reaching your application
  - User authentication and authorization
  - Generate HTTP responses at the edge
  - A/B Testing
  - Bot mitigation at the edge
- You don't have to manage any servers, deployed globally

# CloudFront Functions & Lambda@Edge



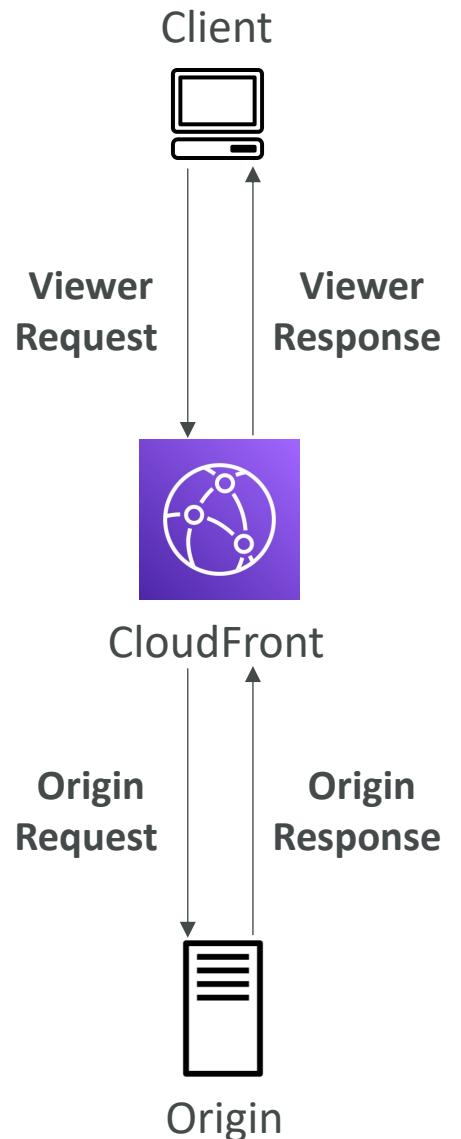
# CloudFront – CloudFront Functions

- Lightweight functions written in JavaScript
- For high-scale, latency-sensitive CDN customizations
- Sub-ms startup times, millions of requests/second
- Run at Edge Locations
- Process-based isolation
- Used to change Viewer requests and responses:
  - **Viewer Request:** after CloudFront receives a request from a viewer
  - **Viewer Response:** before CloudFront forwards the response to the viewer
- Native feature of CloudFront (manage code entirely within CloudFront)



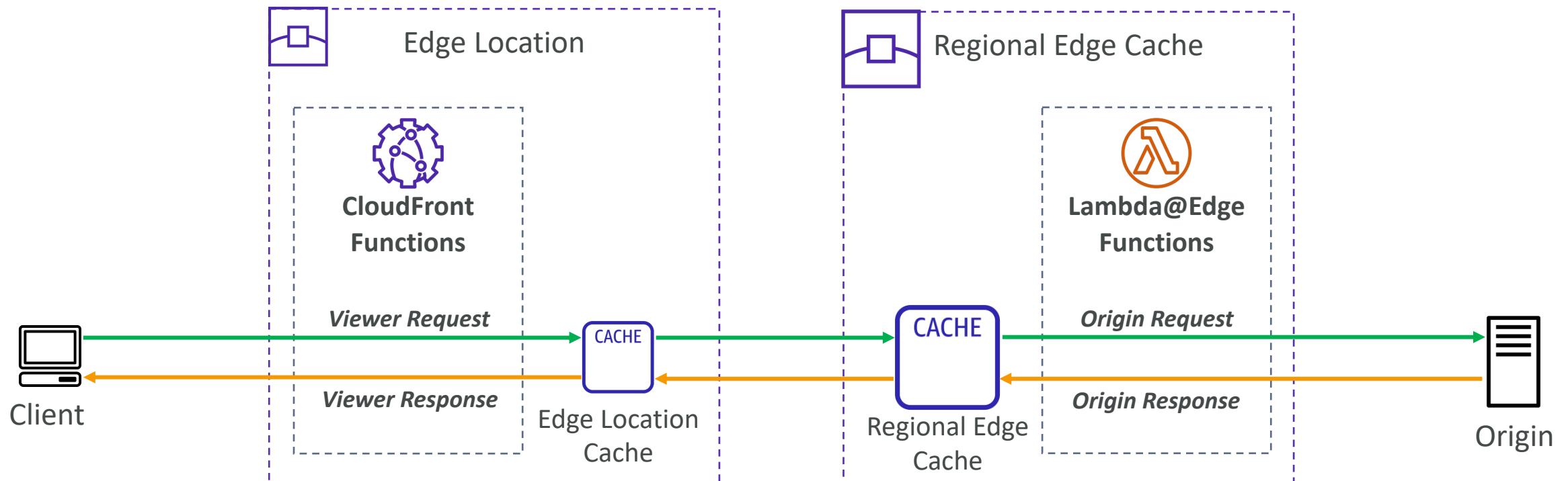
# CloudFront – Lambda@Edge

- Lambda functions written in NodeJS or Python
- Scales to 1000s of requests/second
- Runs at the nearest Regional Edge Cache
- VM-based isolation
- Used to change CloudFront requests and responses:
  - **Viewer Request** – after CloudFront receives a request from a viewer
  - **Origin Request** – before CloudFront forwards the request to the origin
  - **Origin Response** – after CloudFront receives the response from the origin
  - **Viewer Response** – before CloudFront forwards the response to the viewer
- Author your functions in one AWS Region (us-east-1), then CloudFront replicates to its locations



# CloudFront Functions with Lambda@Edge

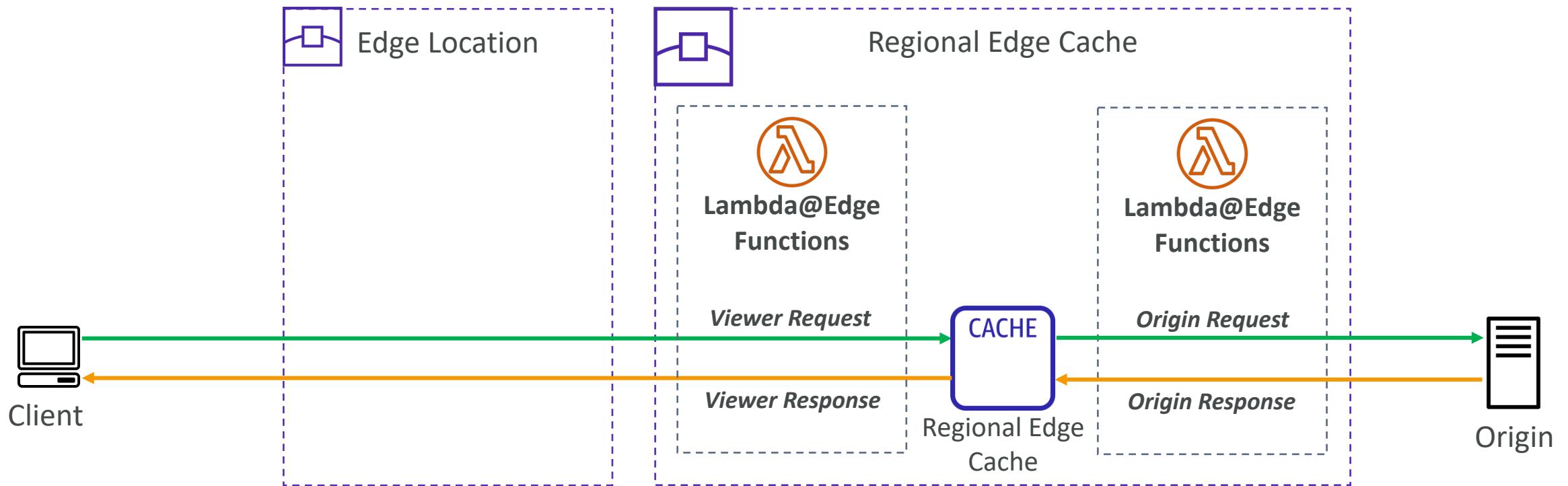
**CloudFront Functions and Lambda@Edge can be used together**



**NOTE: You can't combine CloudFront Functions and Lambda@Edge in viewer events (viewer request & viewer response)**

# Using Lambda@Edge Only

Use when you need some of the capabilities of Lambda@Edge that aren't available with CloudFront Functions (e.g., longer execution time, network access, ...)



# CloudFront Functions vs. Lambda@Edge

|                                    | CloudFront Functions                                  | Lambda@Edge  |
|------------------------------------|---|--|
| Runtime Support                    | JavaScript  | Node.js, Python  |
| Execution Location                 | Edge Locations  | Regional Edge Caches   |
| CloudFront Triggers                | - Viewer Request/Response                             | - Viewer Request/Response<br>- Origin Request/Response           |
| Isolation                          | Process-based   | VM-based   |
| Max. Execution Time                | < 1 ms  | - 5 seconds (viewers triggers)<br>- 30 seconds (origin triggers) |
| Max. Memory                        | 2 MB  | - 128 MB (viewer triggers)<br>- 10 GB (origin triggers)          |
| Total Package Size                 | 10 KB   | - 1 MB (viewer triggers)<br>- 50 MB (origin t                    |
| Network Access, File System Access | No  | Yes  |
| Access to the Request Body         | No  | Yes  |
| Pricing                            | Free tier available, 1/6 <sup>th</sup> price of @Edge | No free tier, charged per request & duration                     |

# CloudFront Functions vs. Lambda@Edge – Use Cases

## CloudFront Functions

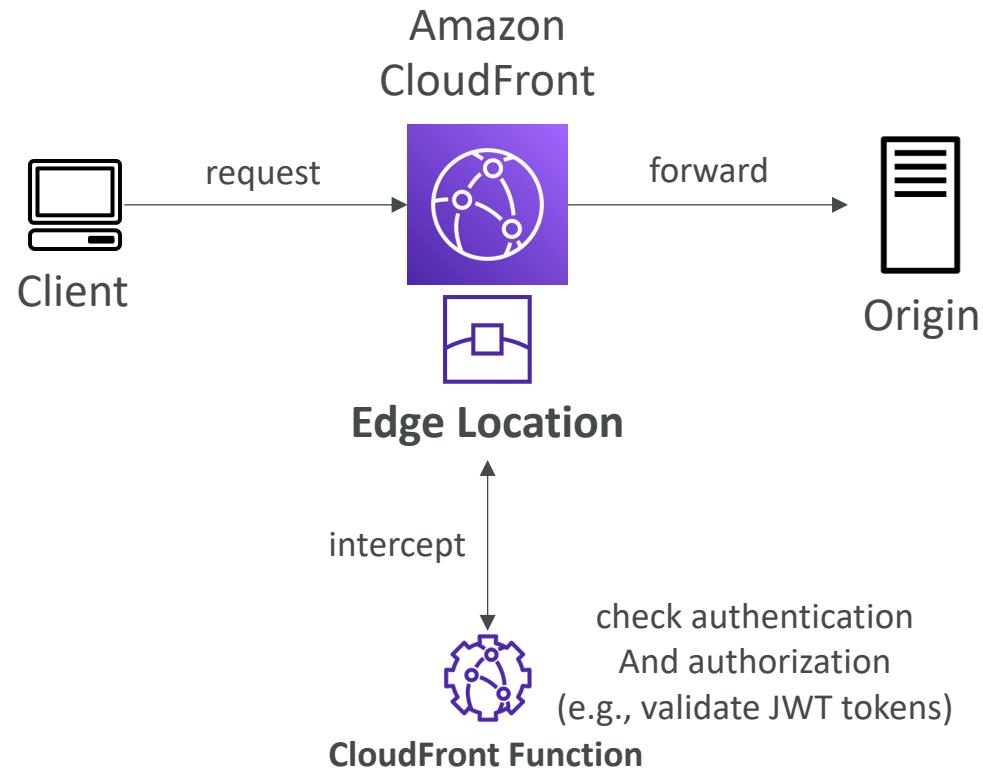
- Cache key normalization
  - Transform request attributes (headers, cookies, query strings, URL) to create an optimal Cache Key
- Header manipulation
  - Insert/modify/delete HTTP headers in the request or response
- URL rewrites or redirects
- Request authentication & authorization
  - Create and validate user-generated tokens (e.g., JWT) to allow/deny requests

## Lambda@Edge

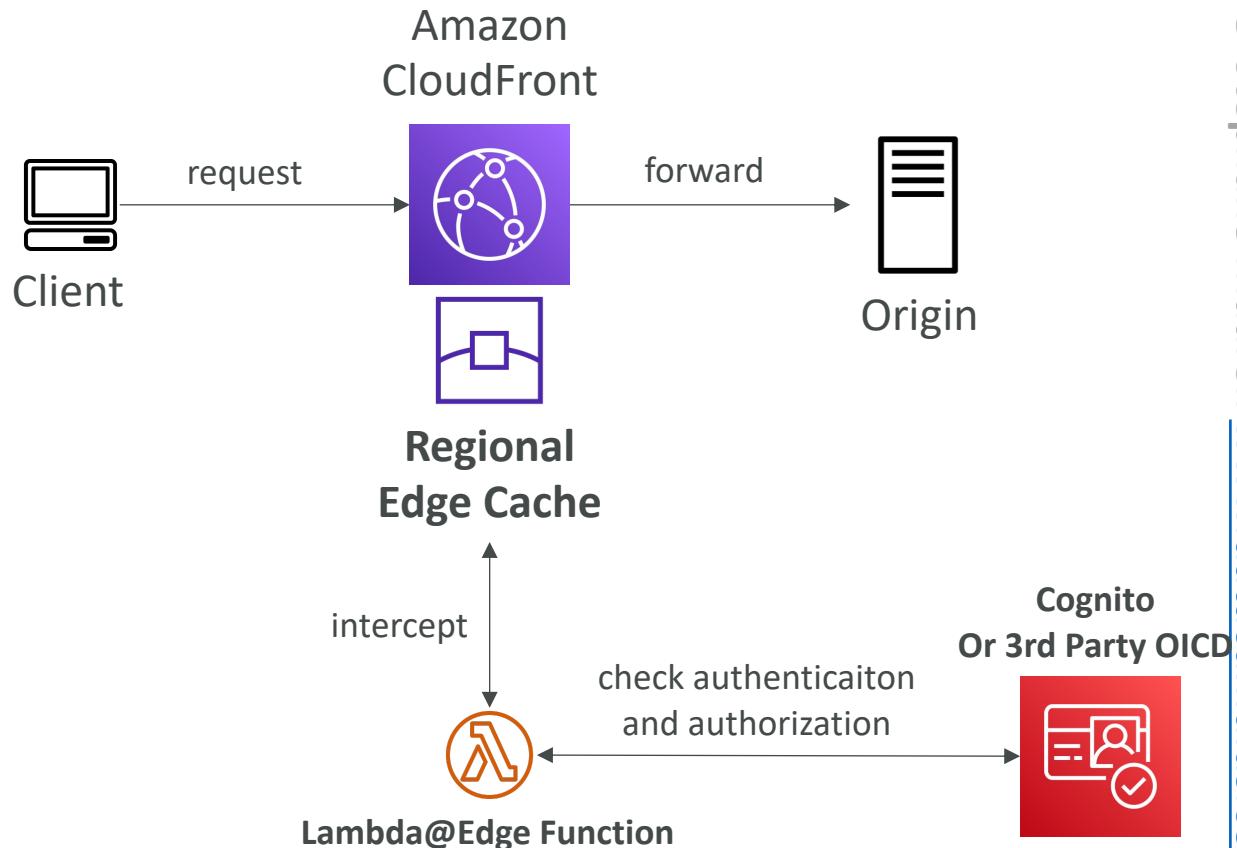
- Longer execution time (several ms)
- Adjustable CPU or memory
- Your code depends on a 3rd libraries (e.g., AWS SDK to access other AWS services)
- Network access to use external services for processing
- File system access or access to the body of HTTP requests

# CloudFront Functions vs. Lambda@Edge – Authentication and Authorization

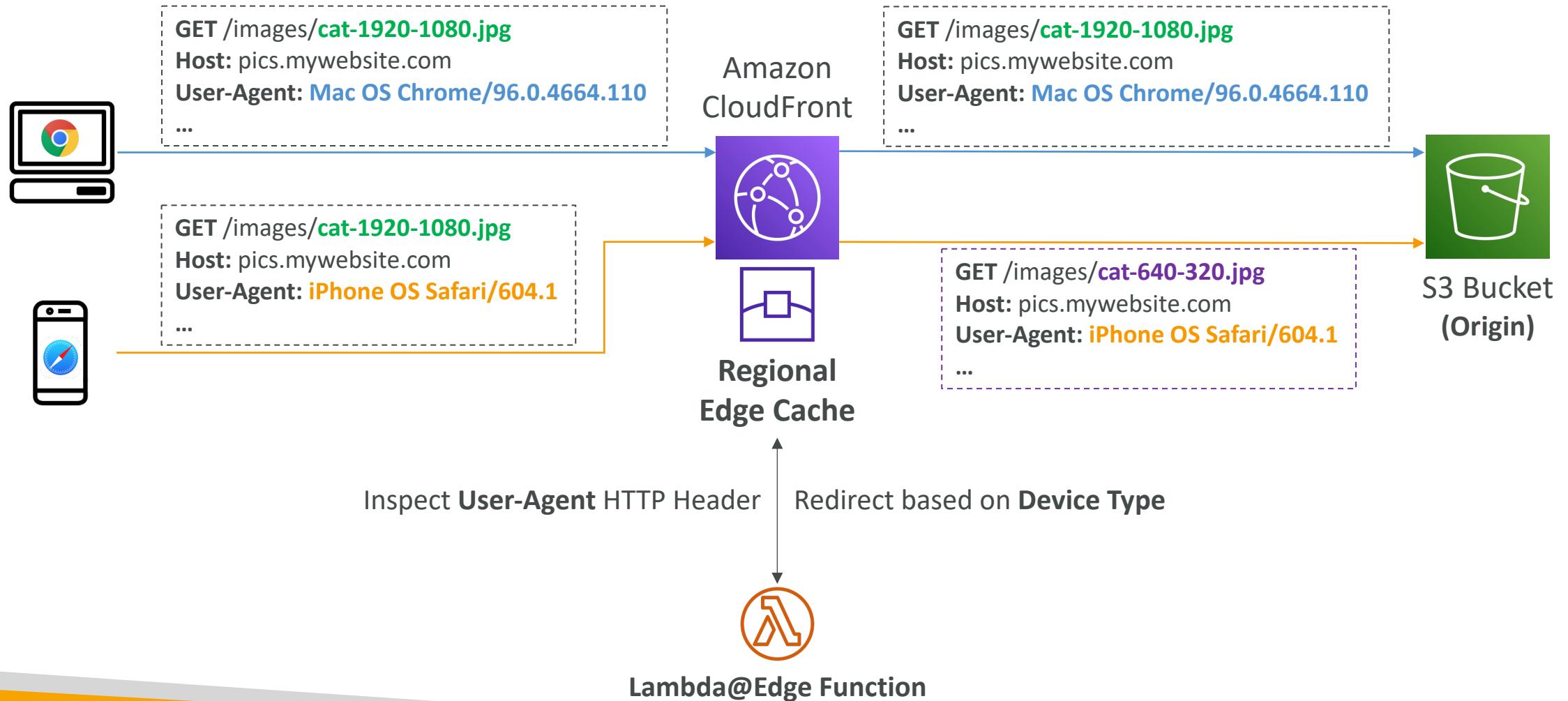
## CloudFront Functions



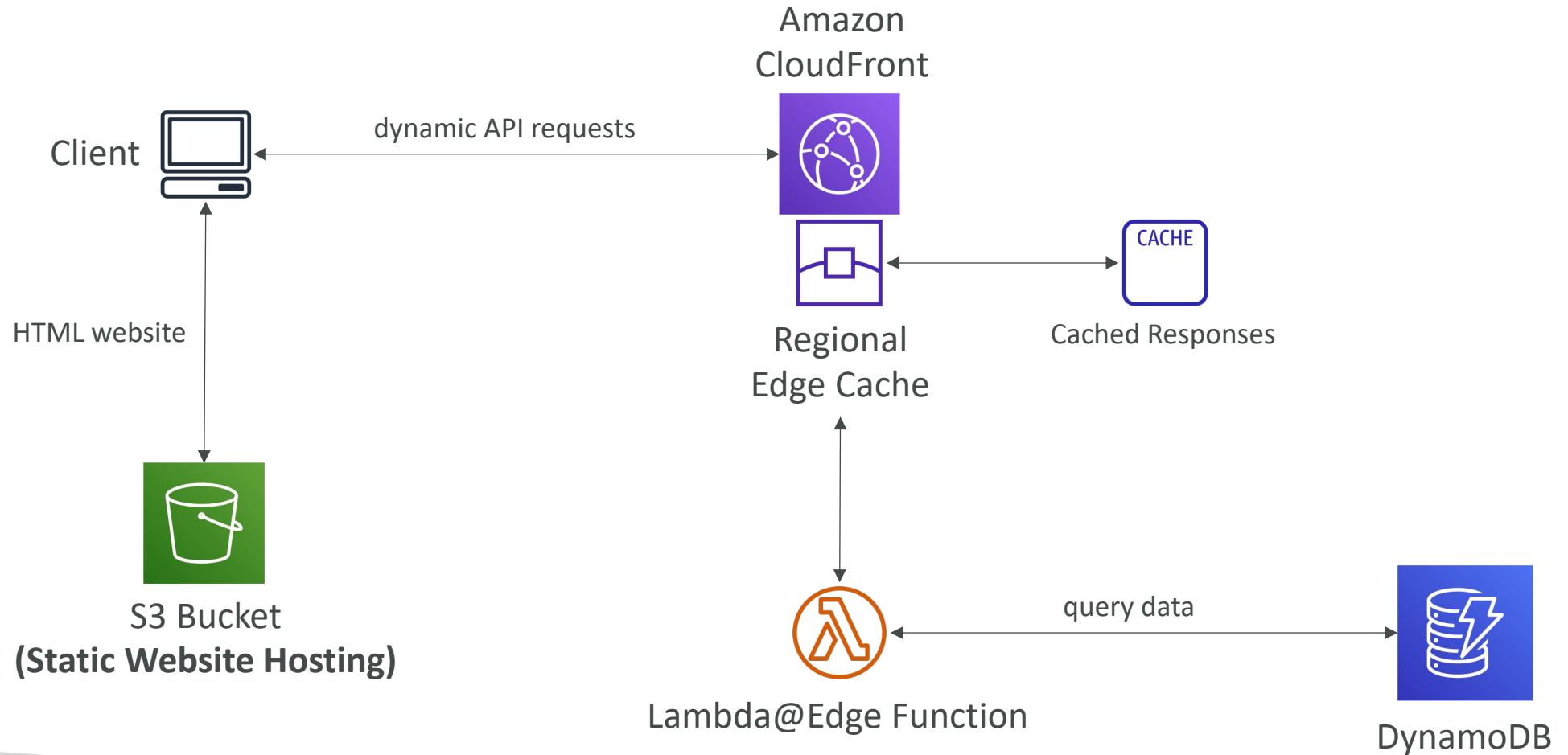
## Lambda@Edge



# Lambda@Edge: Loading content based on User-Agent

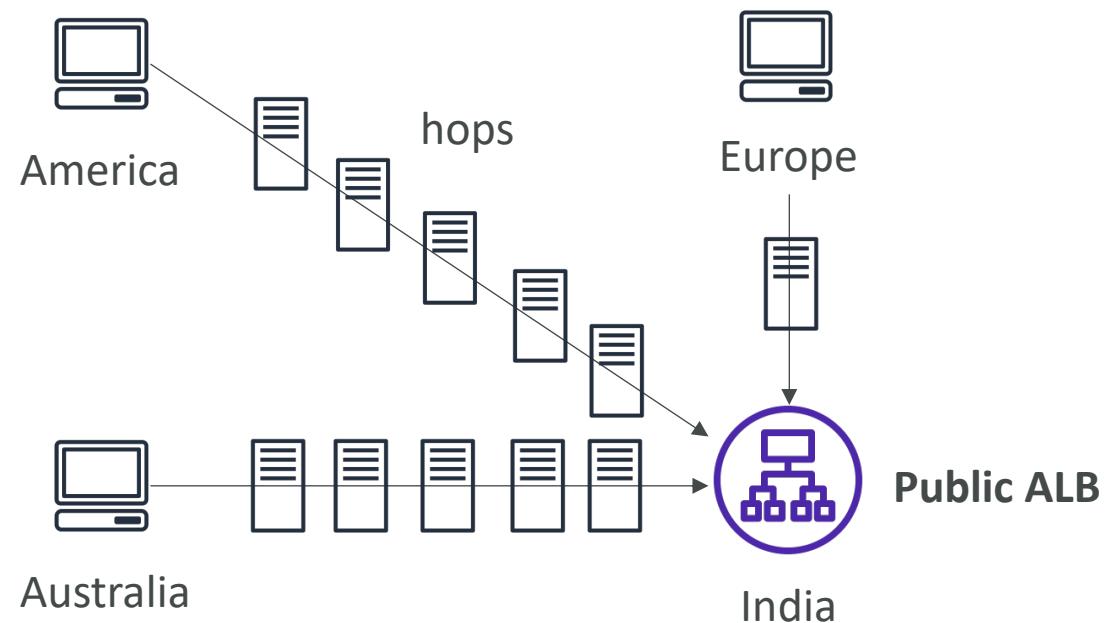


# Lambda@Edge – Global Application



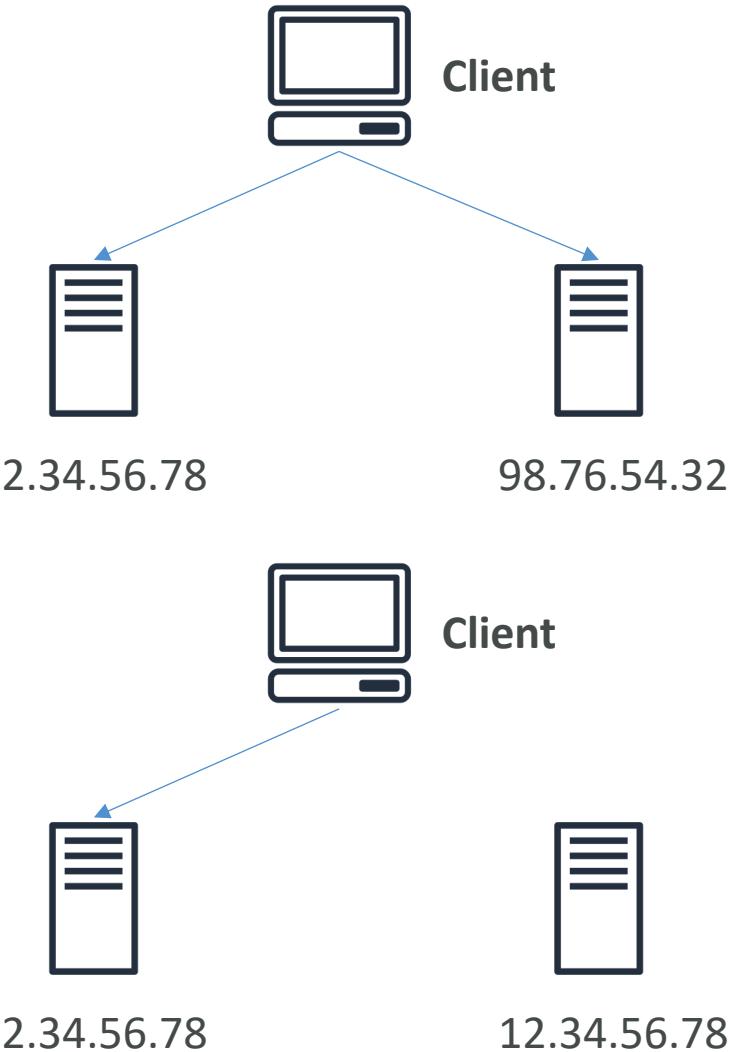
# Global users for our application

- You have deployed an application and have global users who want to access it directly
- They go over the public internet, which can add a lot of latency due to many hops
- We wish to go as fast as possible through AWS network to minimize latency



# Unicast IP vs Anycast IP

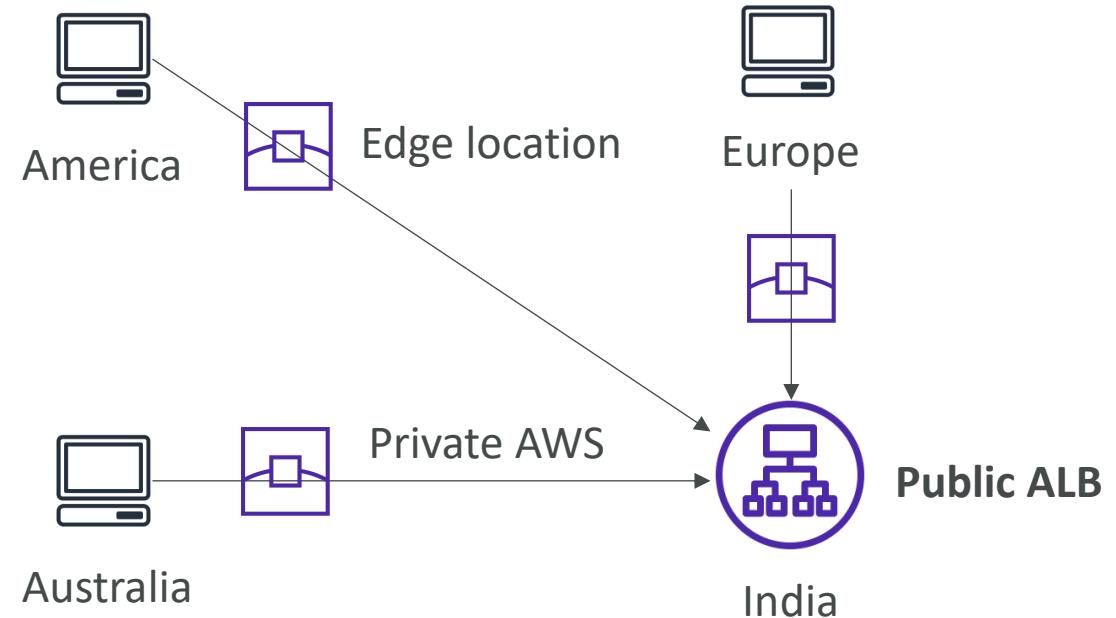
- **Unicast IP:** one server holds one IP address
- **Anycast IP:** all servers hold the same IP address and the client is routed to the nearest one



# AWS Global Accelerator



- Leverage the AWS internal network to route to your application
- 2 Anycast IP are created for your application
- The Anycast IP send traffic directly to Edge Locations
- The Edge locations send the traffic to your application



# AWS Global Accelerator

- Works with Elastic IP, EC2 instances, ALB, NLB, public or private
- Consistent Performance
  - Intelligent routing to lowest latency and fast regional failover
  - No issue with client cache (because the IP doesn't change)
  - Internal AWS network
- Health Checks
  - Global Accelerator performs a health check of your applications
  - Helps make your application global (failover less than 1 minute for unhealthy)
  - Great for disaster recovery (thanks to the health checks)
- Security
  - only 2 external IP need to be whitelisted
  - DDoS protection thanks to AWS Shield

# AWS Global Accelerator vs. CloudFront

- They both use the AWS global network and its edge locations around the world
- Both services integrate with AWS Shield for DDoS protection.
- **CloudFront**
  - Improves performance for both cacheable content (such as images and videos)
  - Dynamic content (such as API acceleration and dynamic site delivery)
  - Content is served at the edge
- **Global Accelerator**
  - Improves performance for a wide range of applications over TCP or UDP
  - Proxying packets at the edge to applications running in one or more AWS Regions.
  - Good fit for non-HTTP use cases, such as gaming (UDP), IoT (MQTT), or Voice over IP
  - Good for HTTP use cases that require static IP addresses
  - Good for HTTP use cases that required deterministic, fast regional failover

# Amazon Route 53

# What is DNS?

- Domain Name System which translates the human friendly hostnames into the machine IP addresses
- www.google.com => 172.217.18.36
- DNS is the backbone of the Internet
- DNS uses hierarchical naming structure

.com

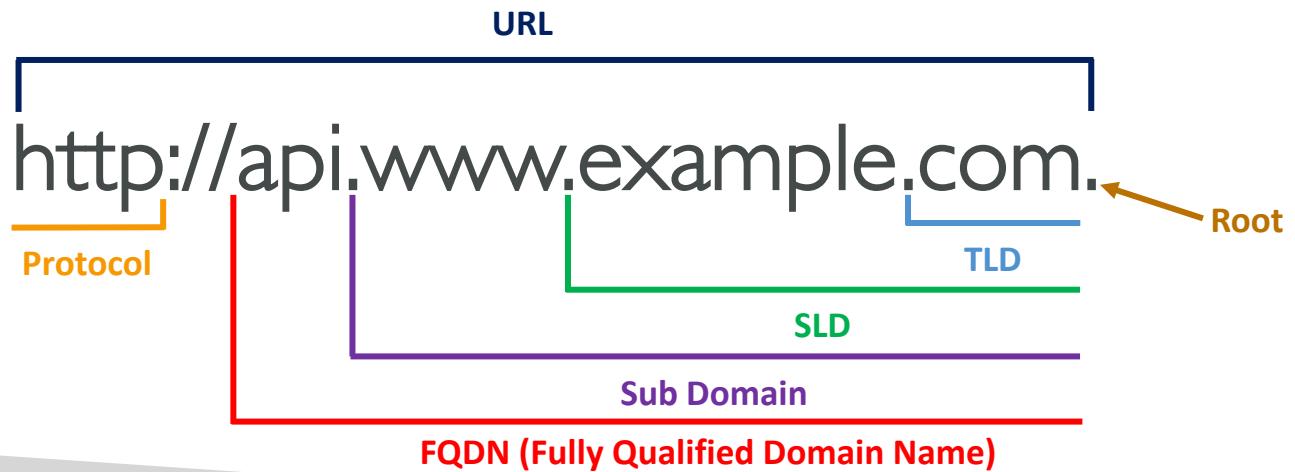
example.com

www.example.com

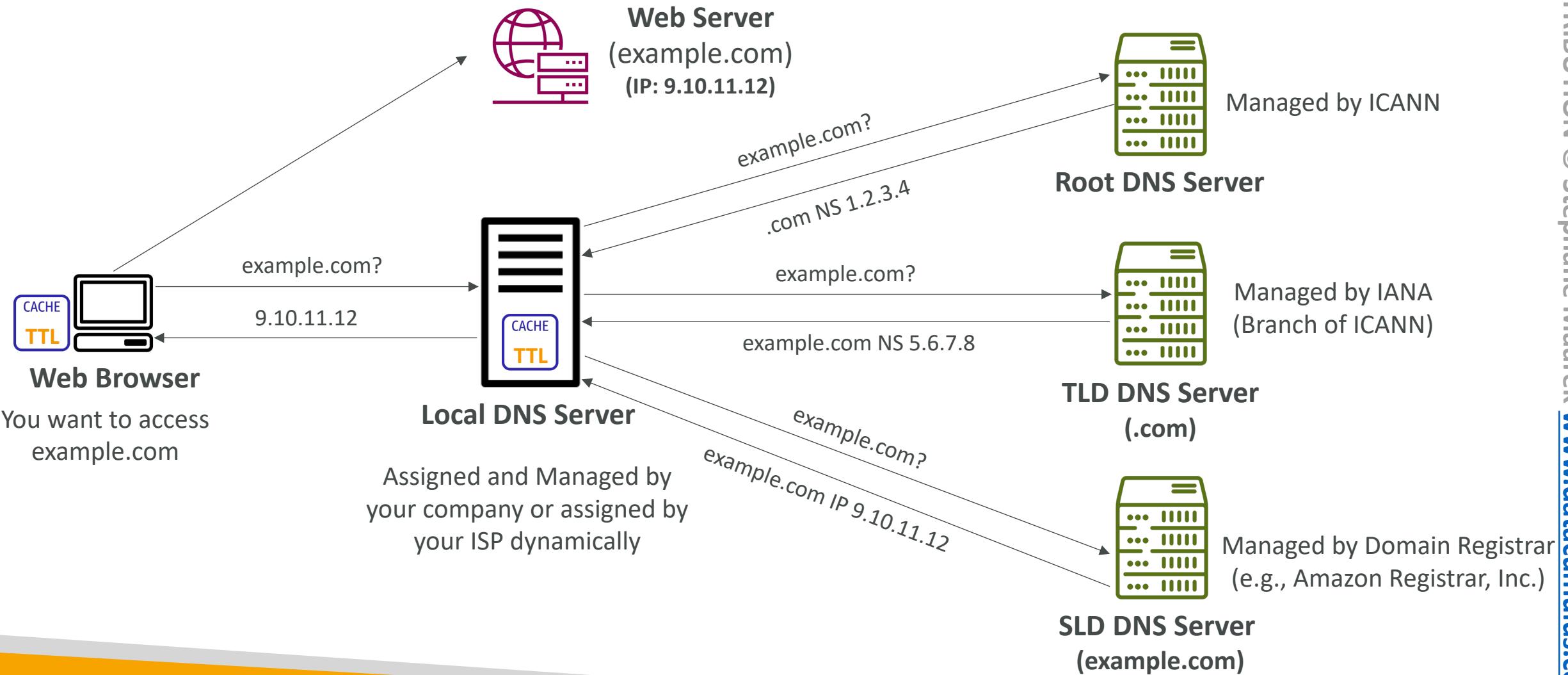
api.example.com

# DNS Terminologies

- Domain Registrar: Amazon Route 53, GoDaddy, ...
- DNS Records: A, AAAA, CNAME, NS, ...
- Zone File: contains DNS records
- Name Server: resolves DNS queries (Authoritative or Non-Authoritative)
- Top Level Domain (TLD): .com, .us, .in, .gov, .org, ...
- Second Level Domain (SLD): amazon.com, google.com, ...

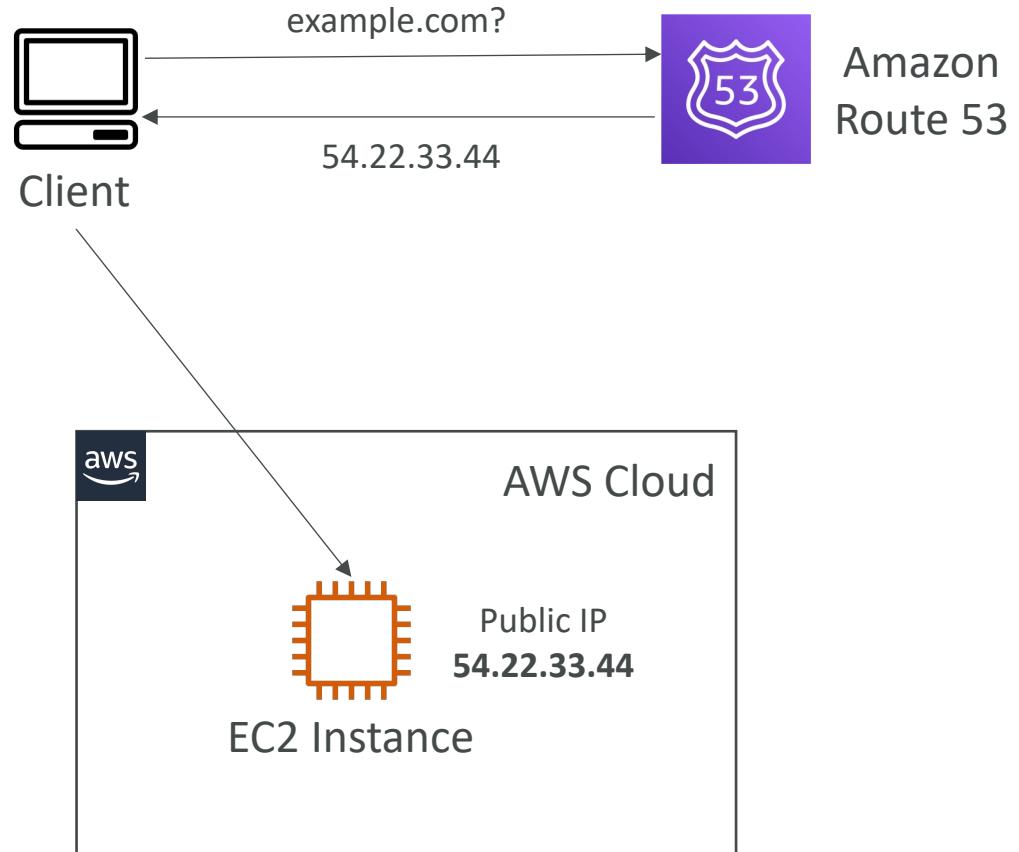


# How DNS Works



# Amazon Route 53

- A highly available, scalable, fully managed and Authoritative DNS
  - Authoritative = the customer (you) can update the DNS records
- Route 53 is also a Domain Registrar
- Ability to check the health of your resources
- The only AWS service which provides 100% availability SLA
- Why Route 53? 53 is a reference to the traditional DNS port

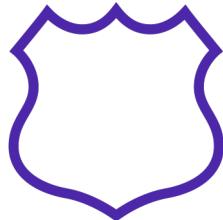


# Route 53 – Records

- How you want to route traffic for a domain
- Each record contains:
  - Domain/subdomain Name – e.g., example.com
  - Record Type – e.g., A or AAAA
  - Value – e.g., 123.456.789.123
  - Routing Policy – how Route 53 responds to queries
  - TTL – amount of time the record cached at DNS Resolvers
- Route 53 supports the following DNS record types:
  - (must know) A / AAAA / CNAME / NS
  - (advanced) CAA / DS / MX / NAPTR / PTR / SOA / TXT / SPF / SRV

# Route 53 – Record Types

- A – maps a hostname to IPv4
- AAAA – maps a hostname to IPv6
- CNAME – maps a hostname to another hostname
  - The target is a domain name which must have an A or AAAA record
  - Can't create a CNAME record for the top node of a DNS namespace (Zone Apex)
  - Example: you can't create for example.com, but you can create for www.example.com
- NS – Name Servers for the Hosted Zone
  - Control how traffic is routed for a domain

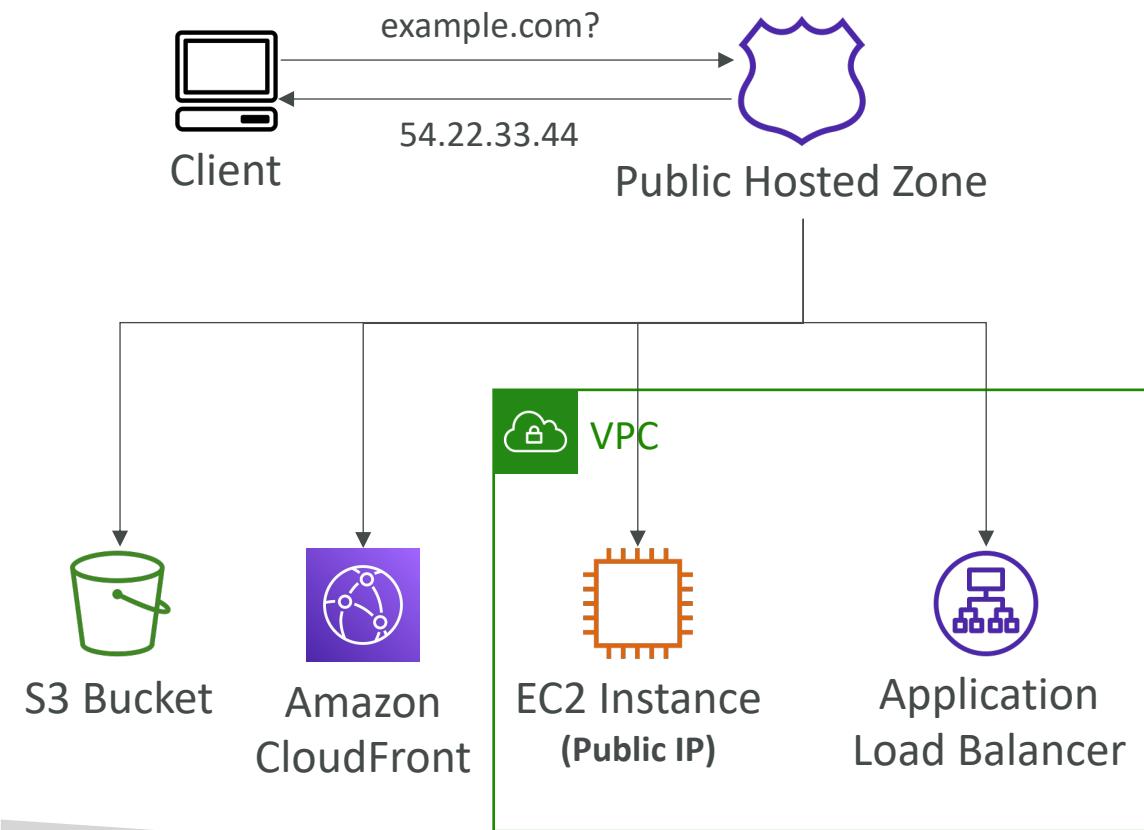


# Route 53 – Hosted Zones

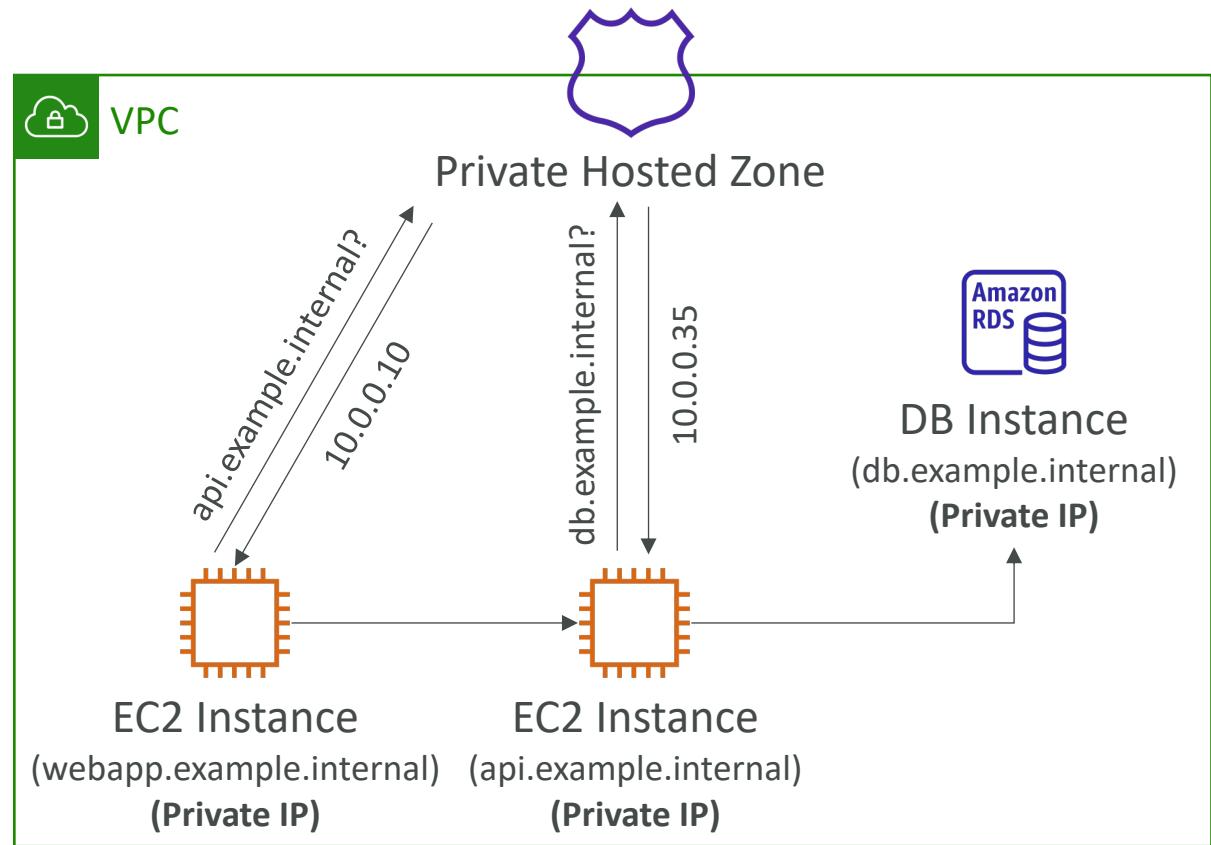
- A container for records that define how to route traffic to a domain and its subdomains
- **Public Hosted Zones** – contains records that specify how to route traffic on the Internet (public domain names)  
[application1.mypublicdomain.com](http://application1.mypublicdomain.com)
- **Private Hosted Zones** – contain records that specify how you route traffic within one or more VPCs (private domain names)  
[application1.company.internal](http://application1.company.internal)
- You pay \$0.50 per month per hosted zone

# Route 53 – Public vs. Private Hosted Zones

## Public Hosted Zone

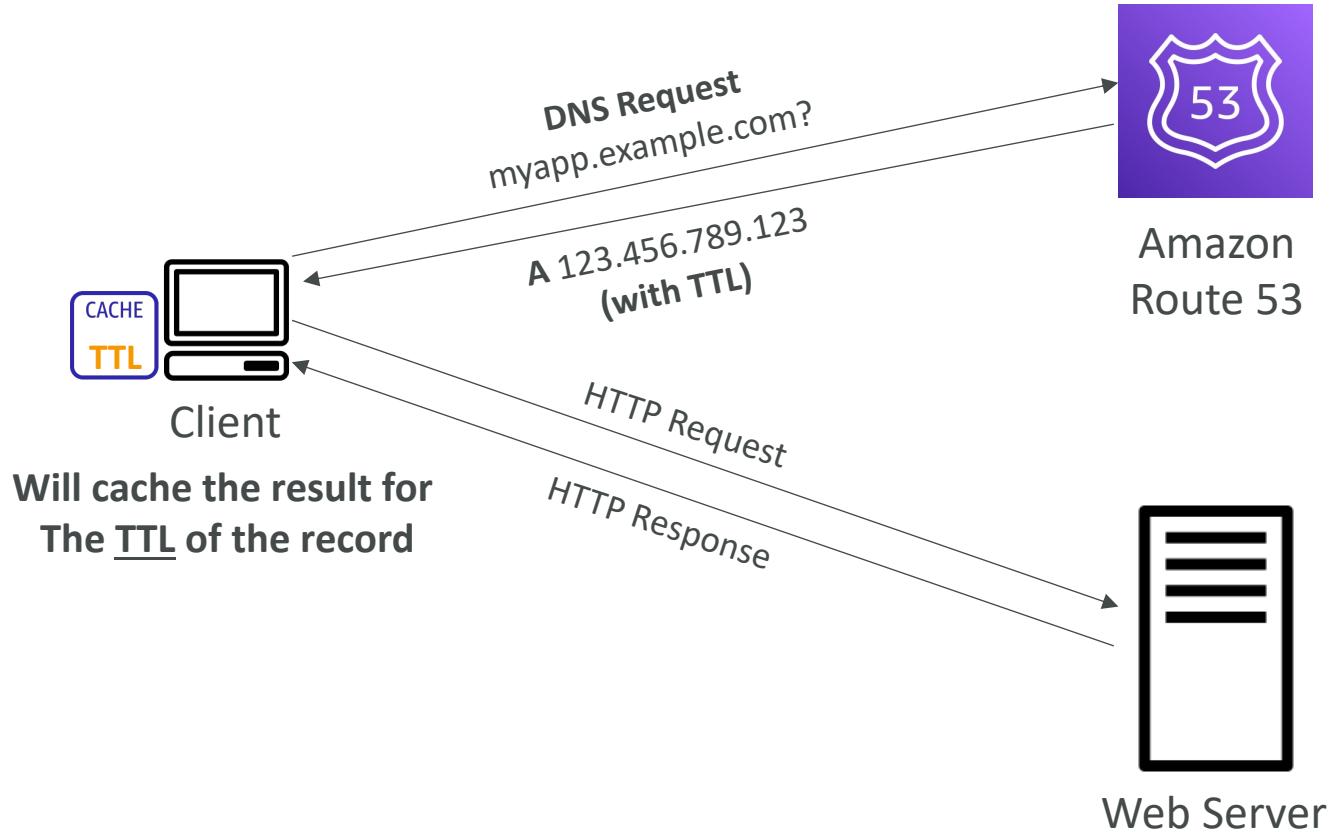


## Private Hosted Zone



# Route 53 – Records TTL (Time To Live)

- High TTL – e.g., 24 hr
  - Less traffic on Route 53
  - Possibly outdated records
- Low TTL – e.g., 60 sec.
  - More traffic on Route 53 (\$\$)
  - Records are outdated for less time
  - Easy to change records
- Except for Alias records, TTL is mandatory for each DNS record

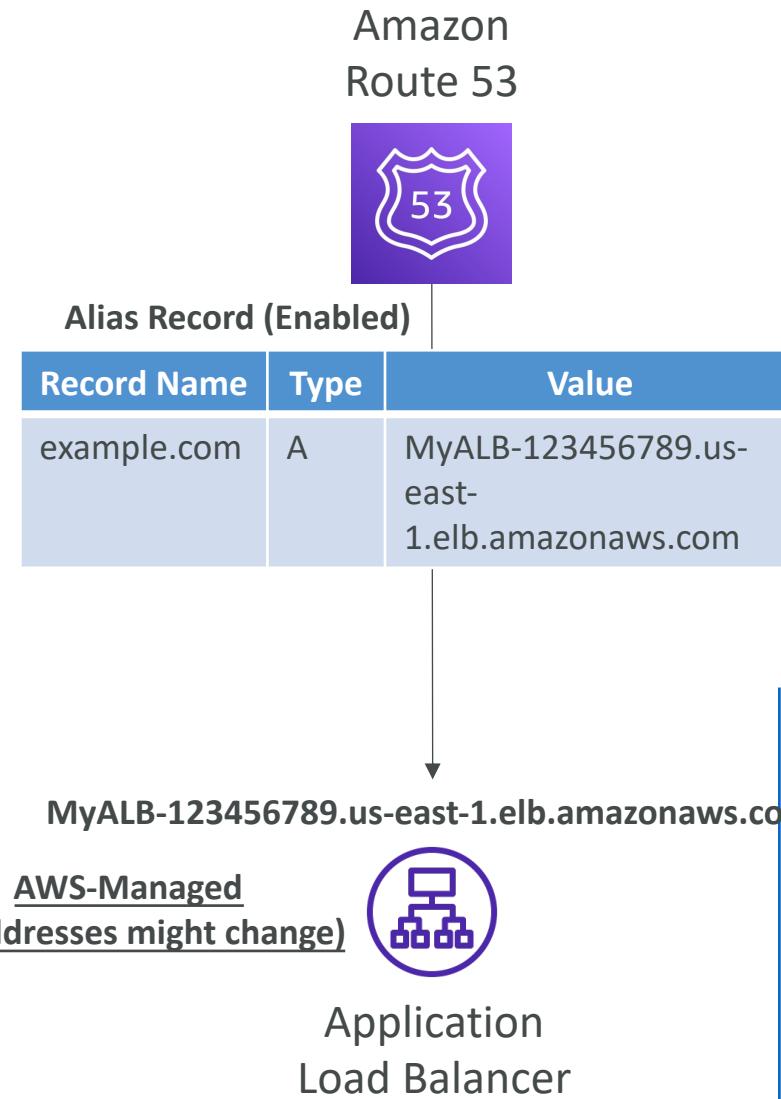


# CNAME vs Alias

- AWS Resources (Load Balancer, CloudFront...) expose an AWS hostname:
  - [l1-l234.us-east-2.elb.amazonaws.com](https://l1-l234.us-east-2.elb.amazonaws.com) and you want [myapp.mydomain.com](https://myapp.mydomain.com)
- CNAME:
  - Points a hostname to any other hostname. (app.mydomain.com => blabla.anything.com)
  - ONLY FOR NON ROOT DOMAIN (aka. something.mydomain.com)
- Alias:
  - Points a hostname to an AWS Resource (app.mydomain.com => blabla.amazonaws.com)
  - Works for ROOT DOMAIN and NON ROOT DOMAIN (aka mydomain.com)
  - Free of charge
  - Native health check

# Route 53 – Alias Records

- Maps a hostname to an AWS resource
- An extension to DNS functionality
- Automatically recognizes changes in the resource's IP addresses
- Unlike CNAME, it can be used for the top node of a DNS namespace (Zone Apex), e.g.: example.com
- Alias Record is always of type A/AAAA for AWS resources (IPv4 / IPv6)
- You can't set the TTL



# Route 53 – Alias Records Targets

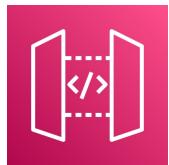
- Elastic Load Balancers
- CloudFront Distributions
- API Gateway
- Elastic Beanstalk environments
- S3 Websites
- VPC Interface Endpoints
- Global Accelerator accelerator
- Route 53 record in the same hosted zone
- You cannot set an ALIAS record for an EC2 DNS name



Elastic  
Load Balancer



Amazon  
CloudFront



Amazon  
API Gateway



Elastic Beanstalk



S3 Websites



VPC Interface  
Endpoints



Global Accelerator



Route 53 Record  
(same Hosted Zone)

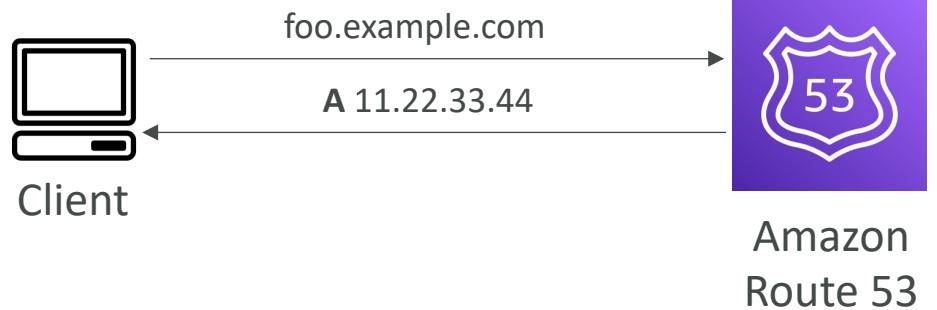
# Route 53 – Routing Policies

- Define how Route 53 responds to DNS queries
- Don't get confused by the word "Routing"
  - It's not the same as Load balancer routing which routes the traffic
  - DNS does not route any traffic, it only responds to the DNS queries
- Route 53 Supports the following Routing Policies
  - Simple
  - Weighted
  - Failover
  - Latency based
  - Geolocation
  - Multi-Value Answer
  - Geoproximity (using Route 53 Traffic Flow feature)

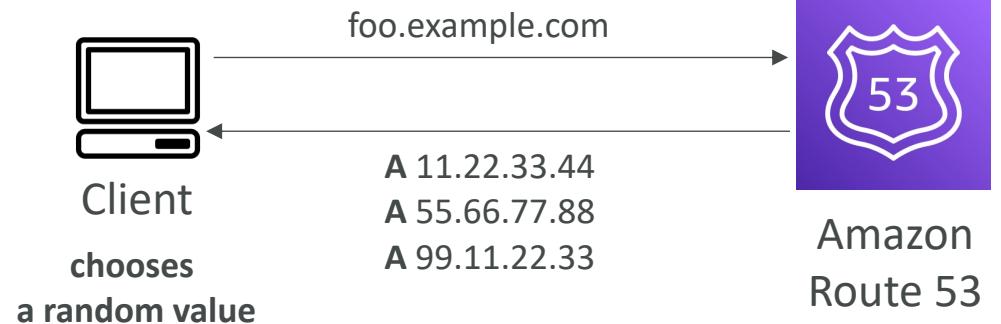
# Routing Policies – Simple

- Typically, route traffic to a single resource
- Can specify multiple values in the same record
- If multiple values are returned, a random one is chosen by the client
- When Alias enabled, specify only one AWS resource
- Can't be associated with Health Checks

## Single Value

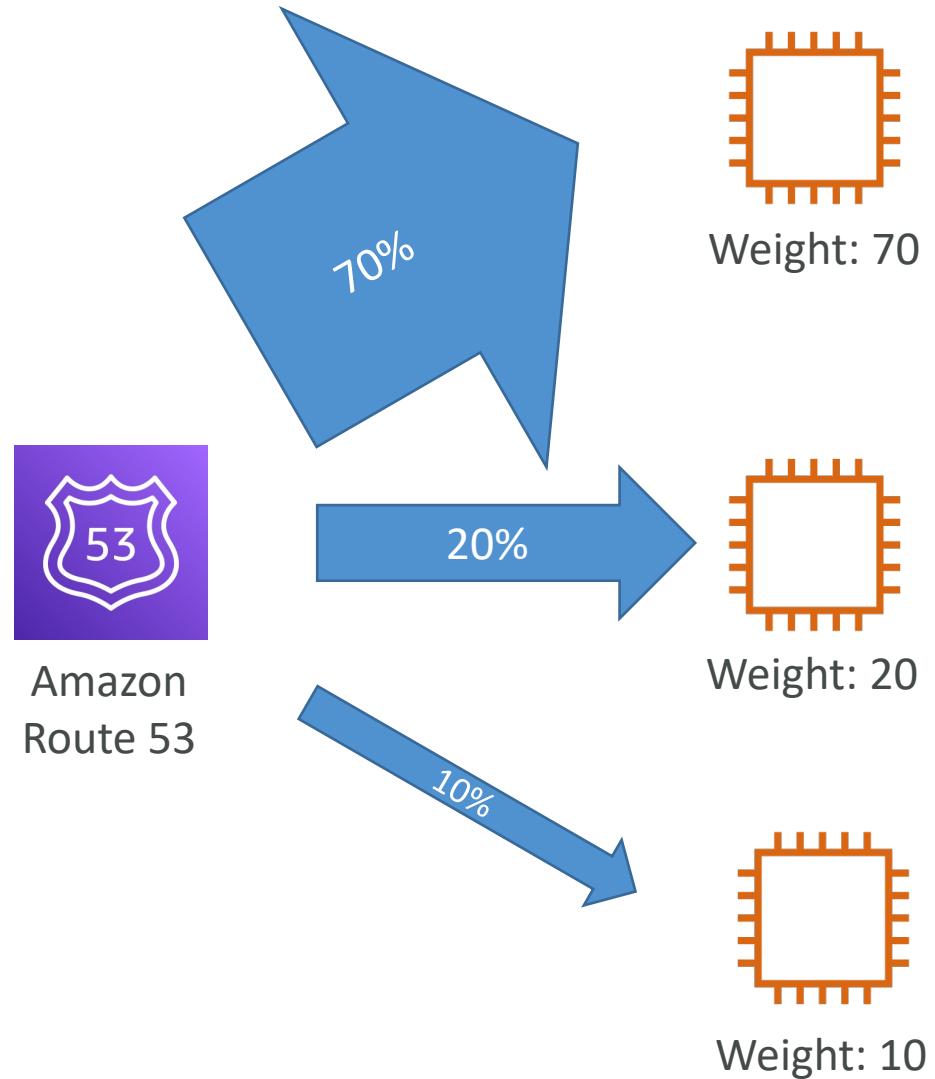


## Multiple Value



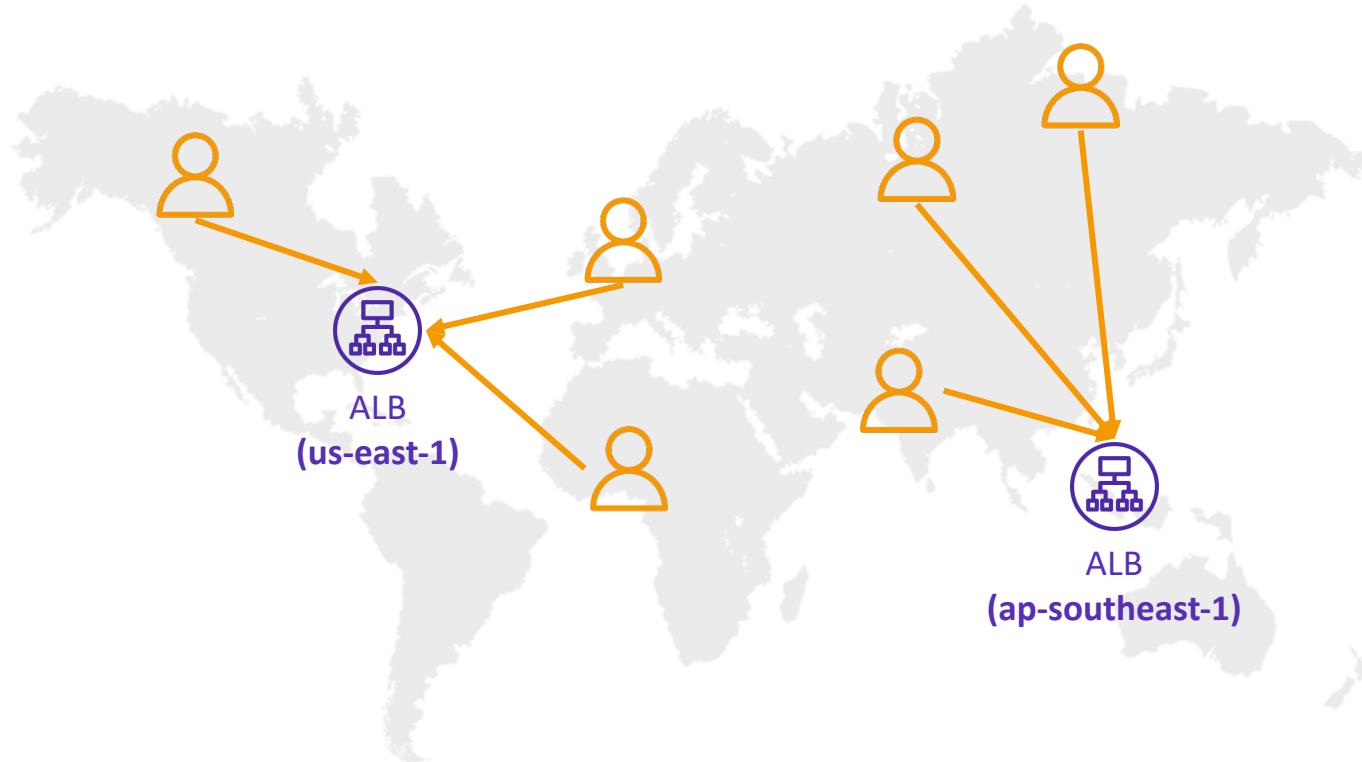
# Routing Policies – Weighted

- Control the % of the requests that go to each specific resource
- Assign each record a relative weight:
  - $traffic\ (%) = \frac{Weight\ for\ a\ specific\ record}{Sum\ of\ all\ the\ weights\ for\ all\ records}$
  - Weights don't need to sum up to 100
- DNS records must have the same name and type
- Can be associated with Health Checks
- Use cases: load balancing between regions, testing new application versions...
- Assign a weight of 0 to a record to stop sending traffic to a resource
- If all records have weight of 0, then all records will be returned equally



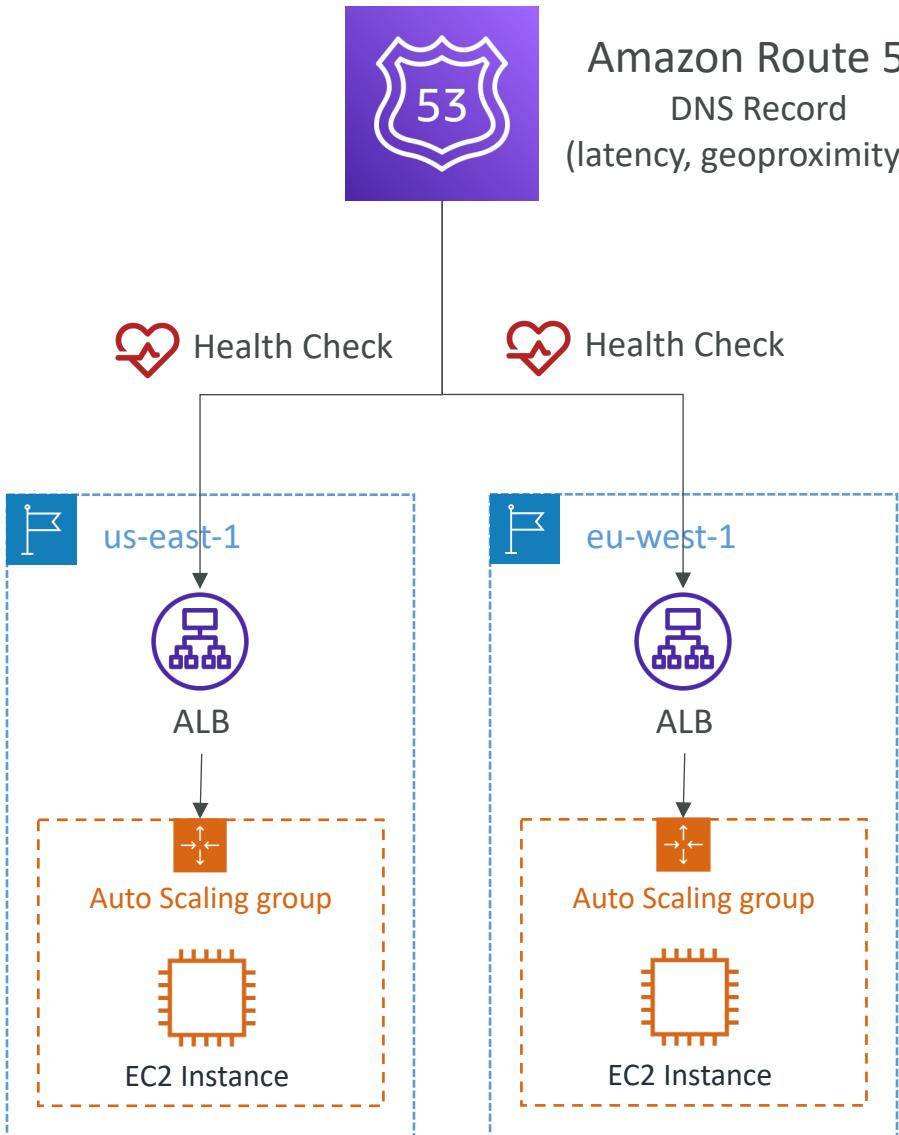
# Routing Policies – Latency-based

- Redirect to the resource that has the least latency close to us
- Super helpful when latency for users is a priority
- Latency is based on traffic between users and AWS Regions
- Germany users may be directed to the US (if that's the lowest latency)
- Can be associated with Health Checks (has a failover capability)



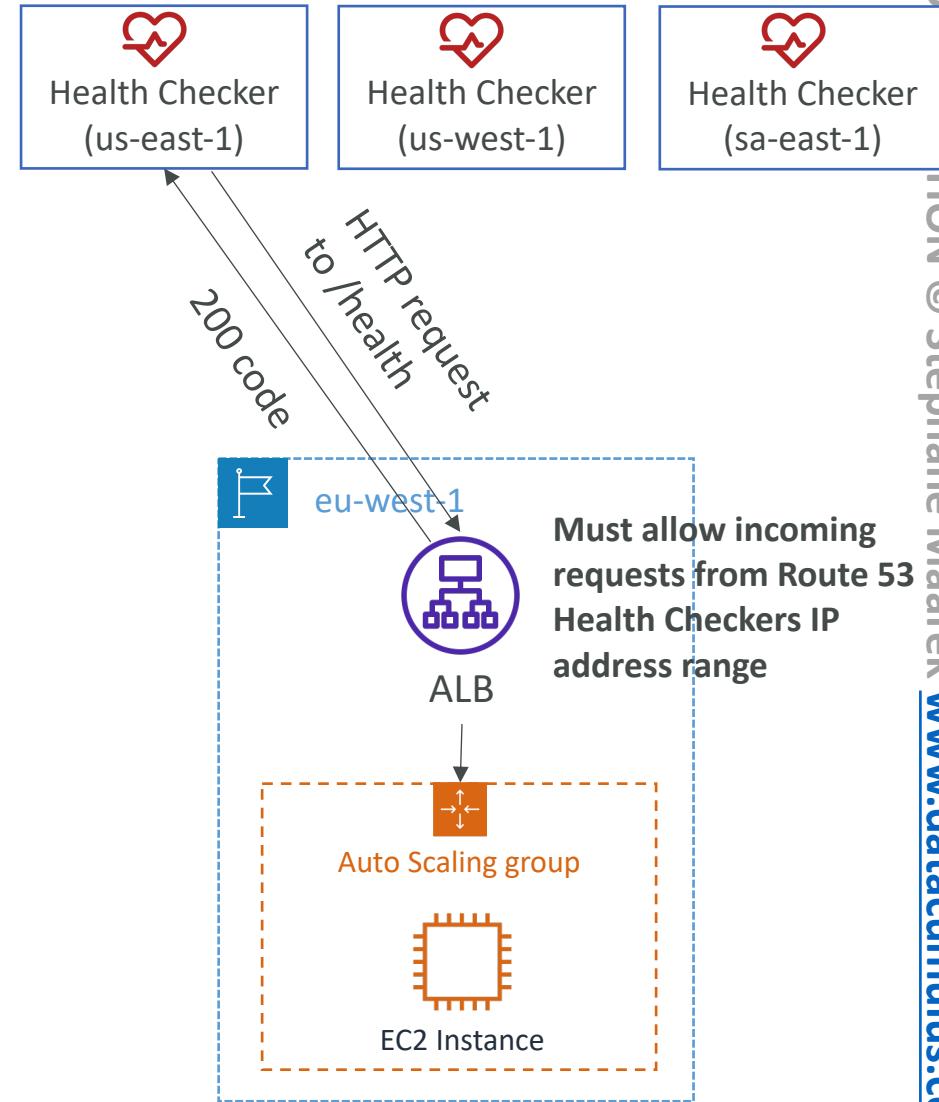
# Route 53 – Health Checks

- HTTP Health Checks are only for **public resources**
- Health Check => Automated DNS Failover:
  1. Health checks that monitor an endpoint (application, server, other AWS resource)
  2. Health checks that monitor other health checks (Calculated Health Checks)
  3. Health checks that monitor CloudWatch Alarms (full control !!) – e.g., throttles of DynamoDB, alarms on RDS, custom metrics, ... (helpful for private resources)
- Health Checks are integrated with CW metrics



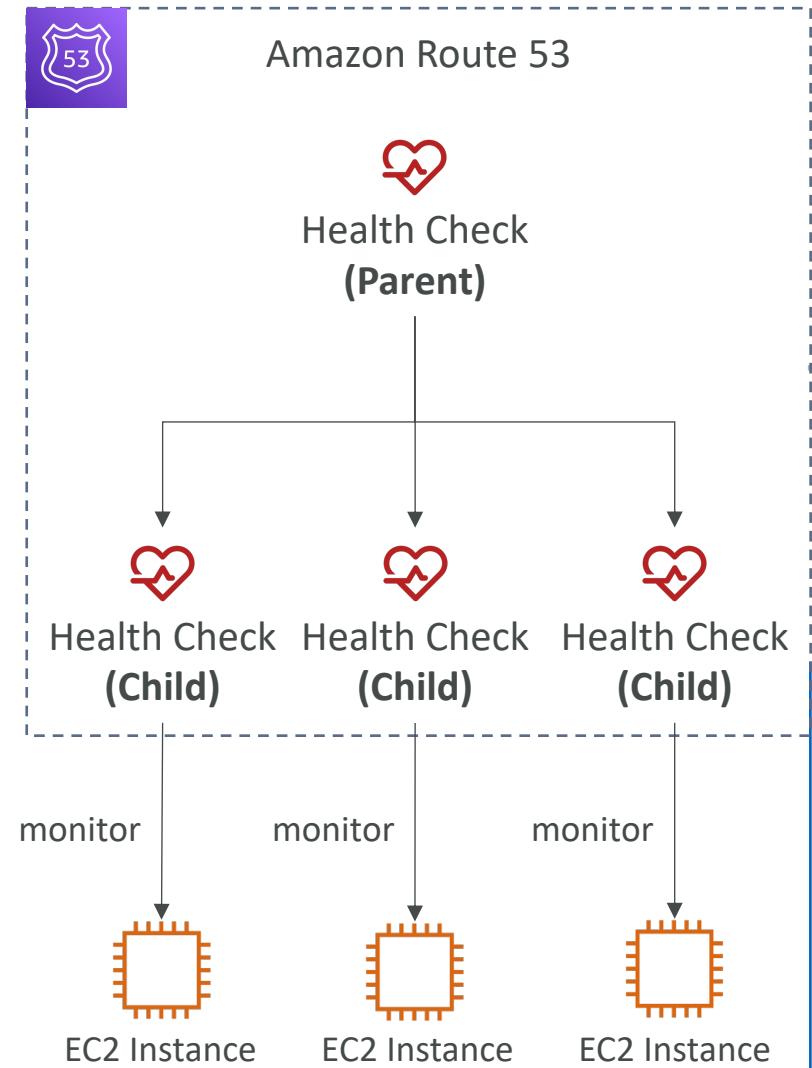
# Health Checks – Monitor an Endpoint

- About 15 global health checkers will check the endpoint health
  - Healthy/Unhealthy Threshold – 3 (default)
  - Interval – 30 sec (can set to 10 sec – higher cost)
  - Supported protocol: HTTP, HTTPS and TCP
  - If > 18% of health checkers report the endpoint is healthy, Route 53 considers it **Healthy**. Otherwise, it's **Unhealthy**
  - Ability to choose which locations you want Route 53 to use
- Health Checks pass only when the endpoint responds with the 2xx and 3xx status codes
- Health Checks can be setup to pass / fail based on the text in the first **5120 bytes** of the response
- Configure your router/firewall to allow incoming requests from Route 53 Health Checkers



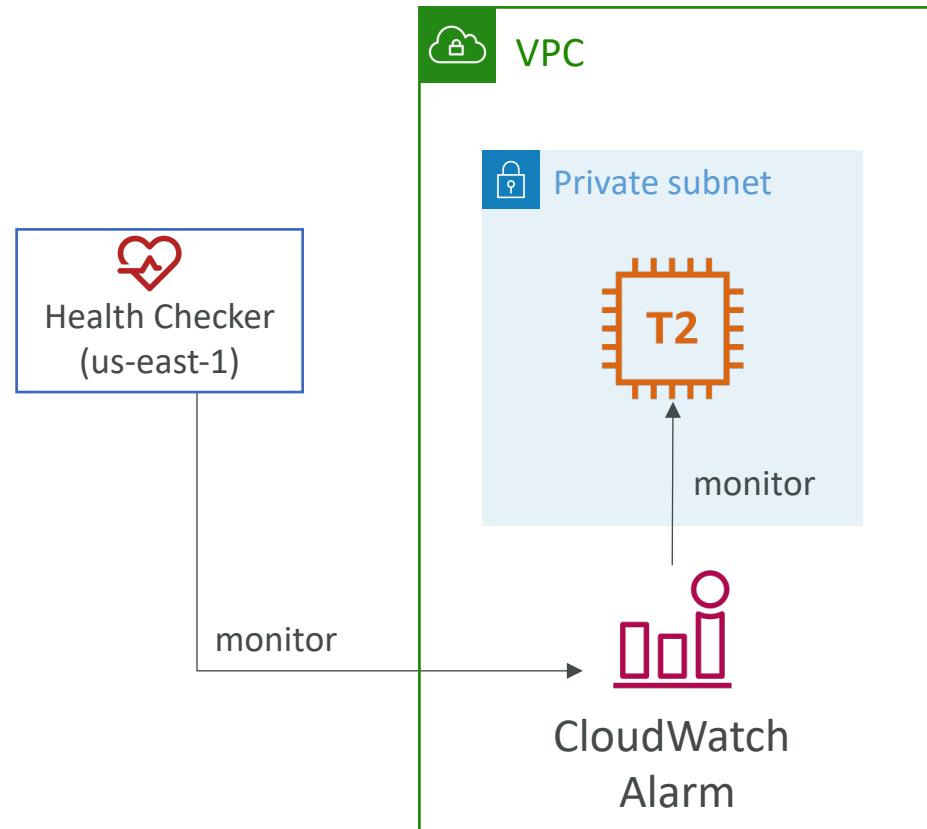
# Route 53 – Calculated Health Checks

- Combine the results of multiple Health Checks into a single Health Check
- You can use **OR**, **AND**, or **NOT**
- Can monitor up to 256 Child Health Checks
- Specify how many of the health checks need to pass to make the parent pass
- Usage: perform maintenance to your website without causing all health checks to fail

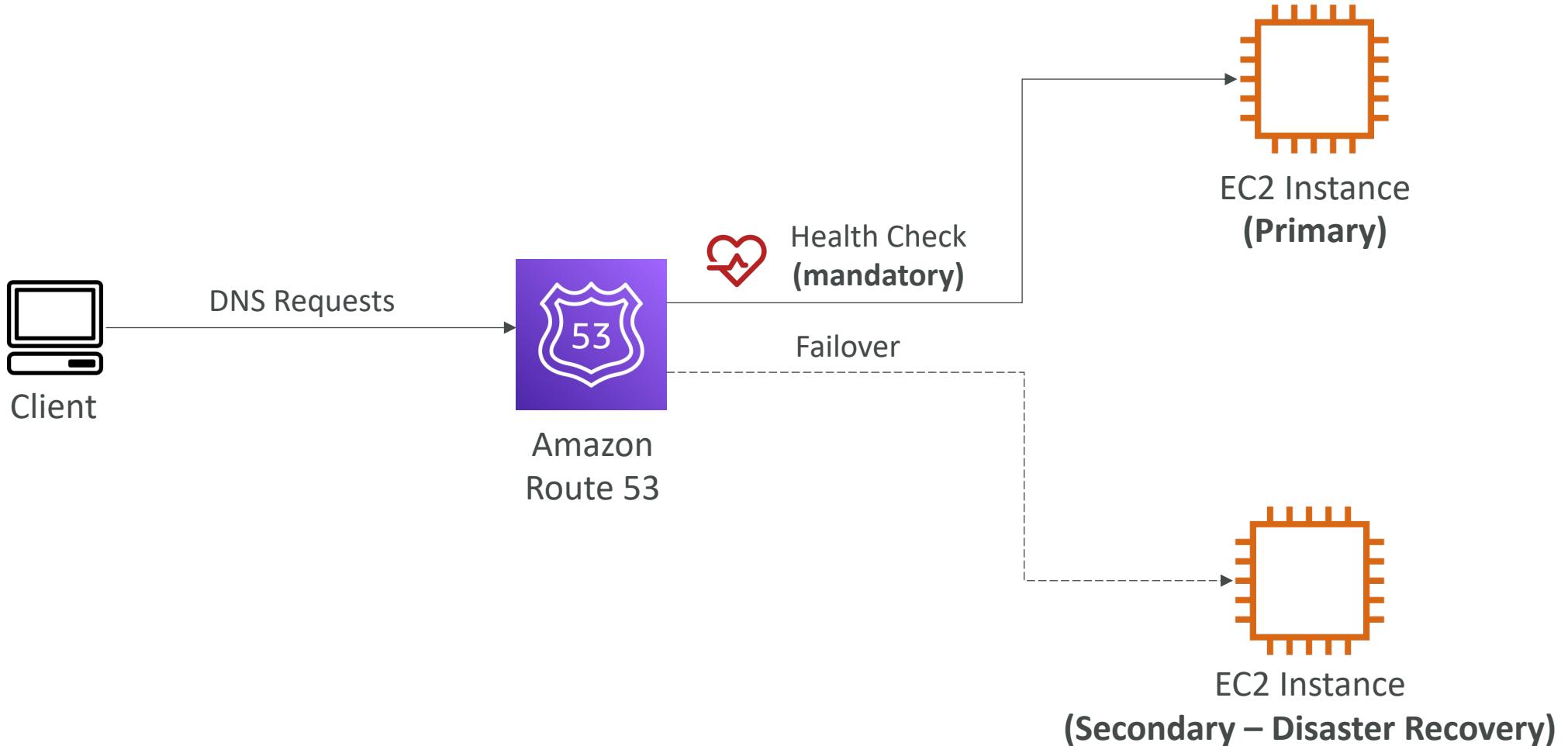


# Health Checks – Private Hosted Zones

- Route 53 health checkers are outside the VPC
- They can't access **private** endpoints (private VPC or on-premises resource)
- You can create a CloudWatch Metric and associate a CloudWatch Alarm, then create a Health Check that checks the alarm itself

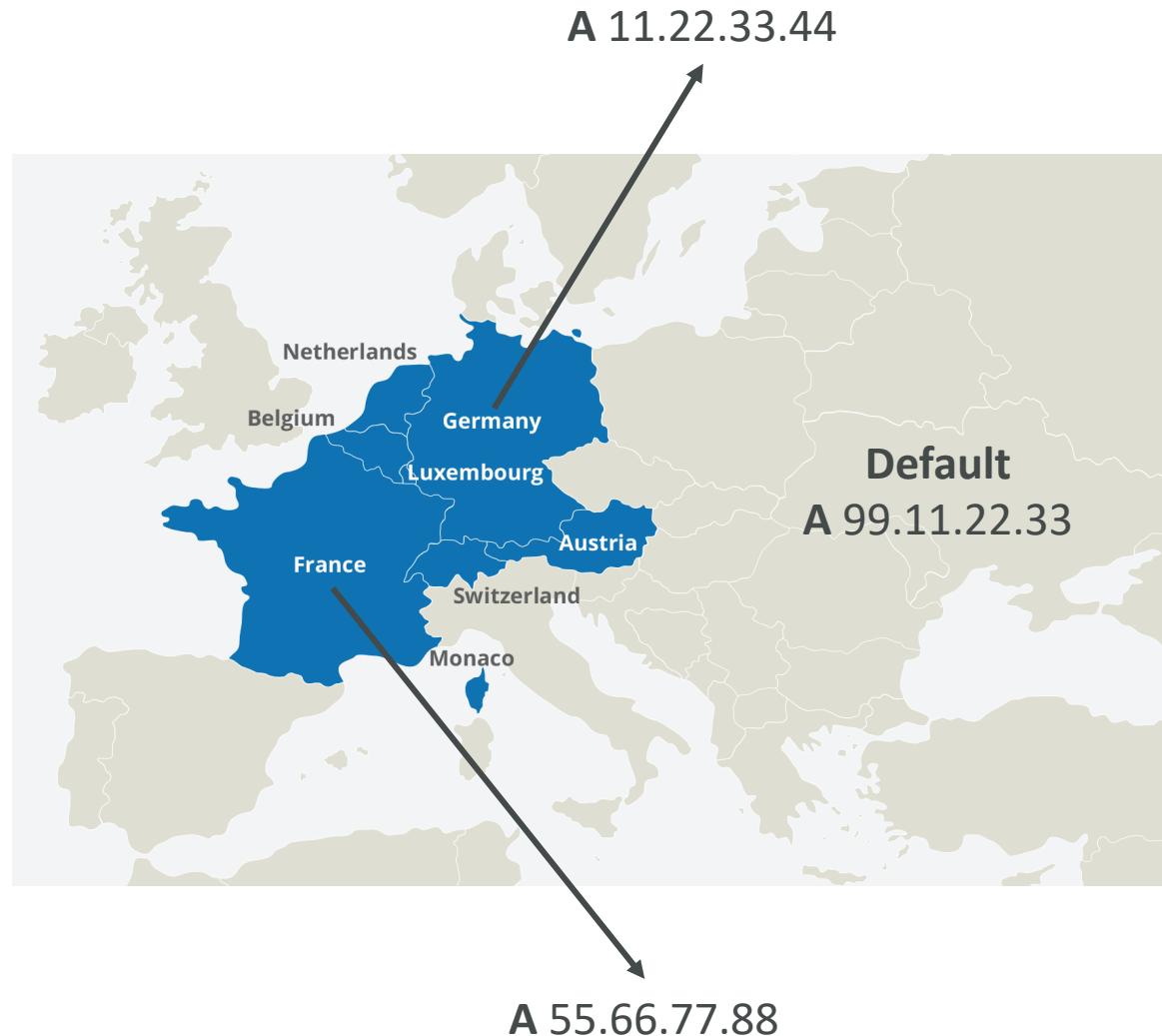


# Routing Policies – Failover (Active-Passive)



# Routing Policies – Geolocation

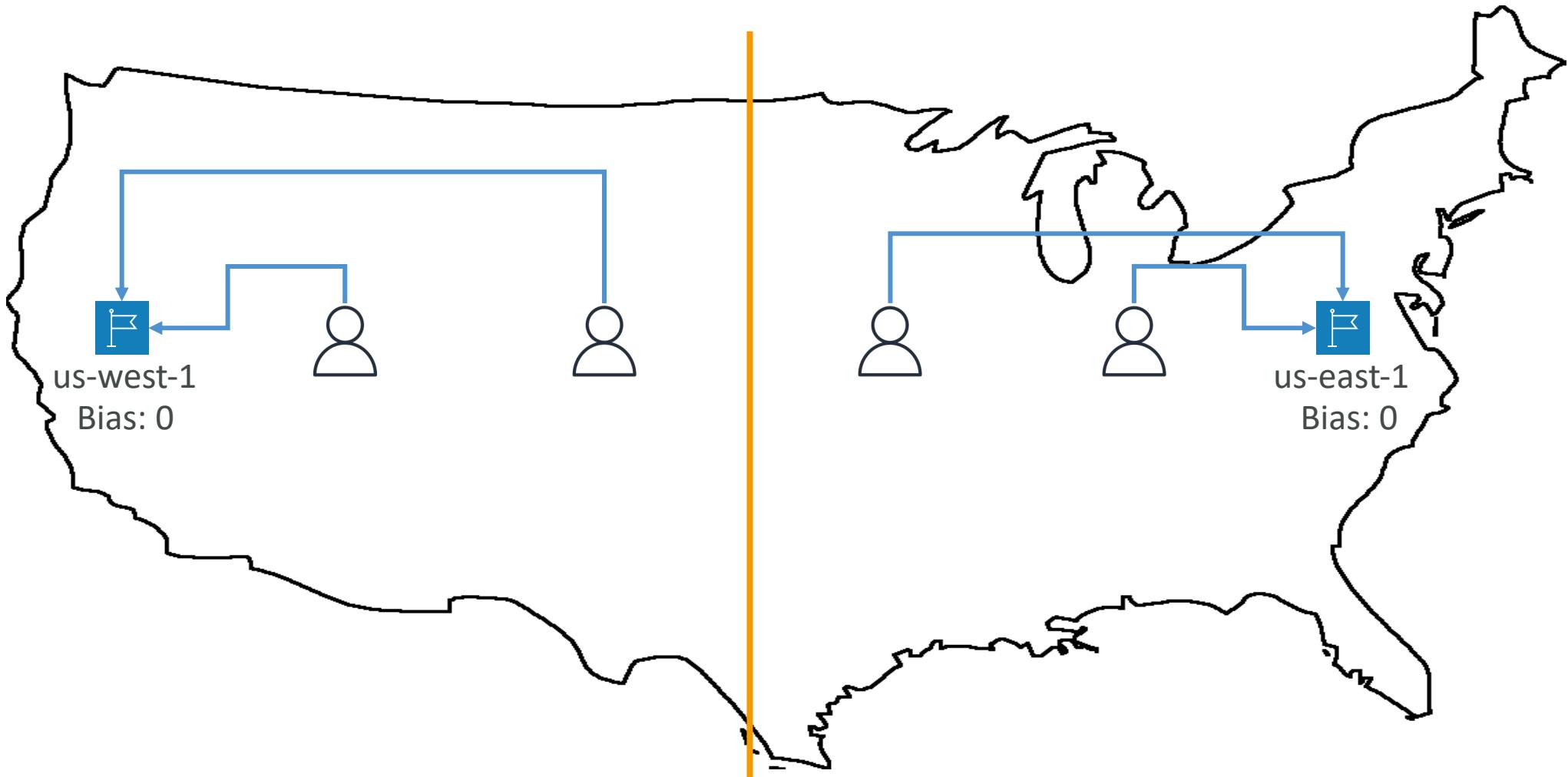
- Different from Latency-based!
- This routing is based on user location
- Specify location by Continent, Country or by US State (if there's overlapping, most precise location selected)
- Should create a “Default” record (in case there's no match on location)
- Use cases: website localization, restrict content distribution, load balancing, ...
- Can be associated with Health Checks



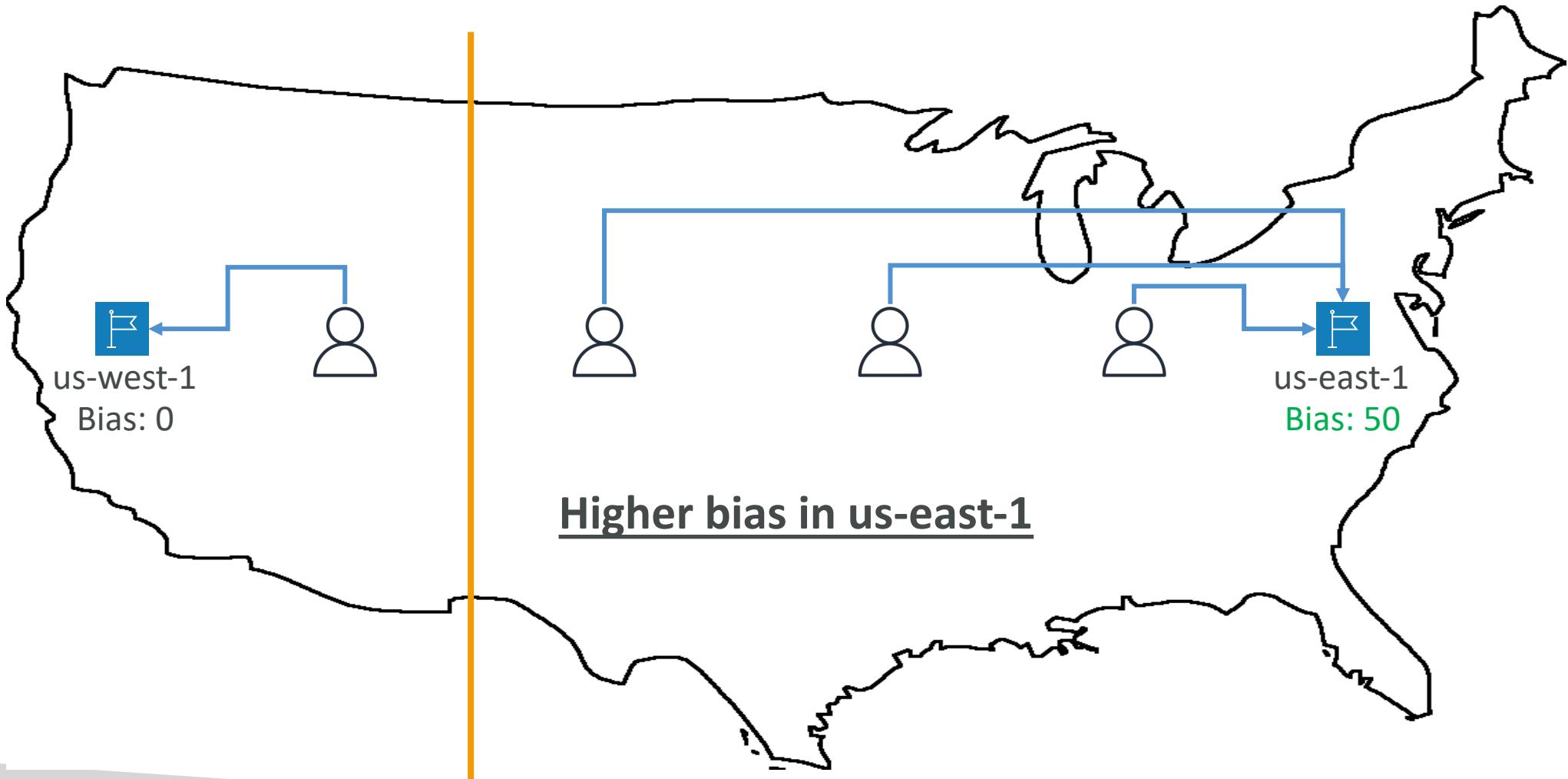
# Routing Policies – Geoproximity

- Route traffic to your resources based on the geographic location of users and resources
- Ability **to shift more traffic to resources based** on the defined bias
- To change the size of the geographic region, specify **bias** values:
  - To expand (1 to 99) – more traffic to the resource
  - To shrink (-1 to -99) – less traffic to the resource
- Resources can be:
  - AWS resources (specify AWS region)
  - Non-AWS resources (specify Latitude and Longitude)
- You must use Route 53 Traffic Flow to use this feature

# Routing Policies – Geoproximity

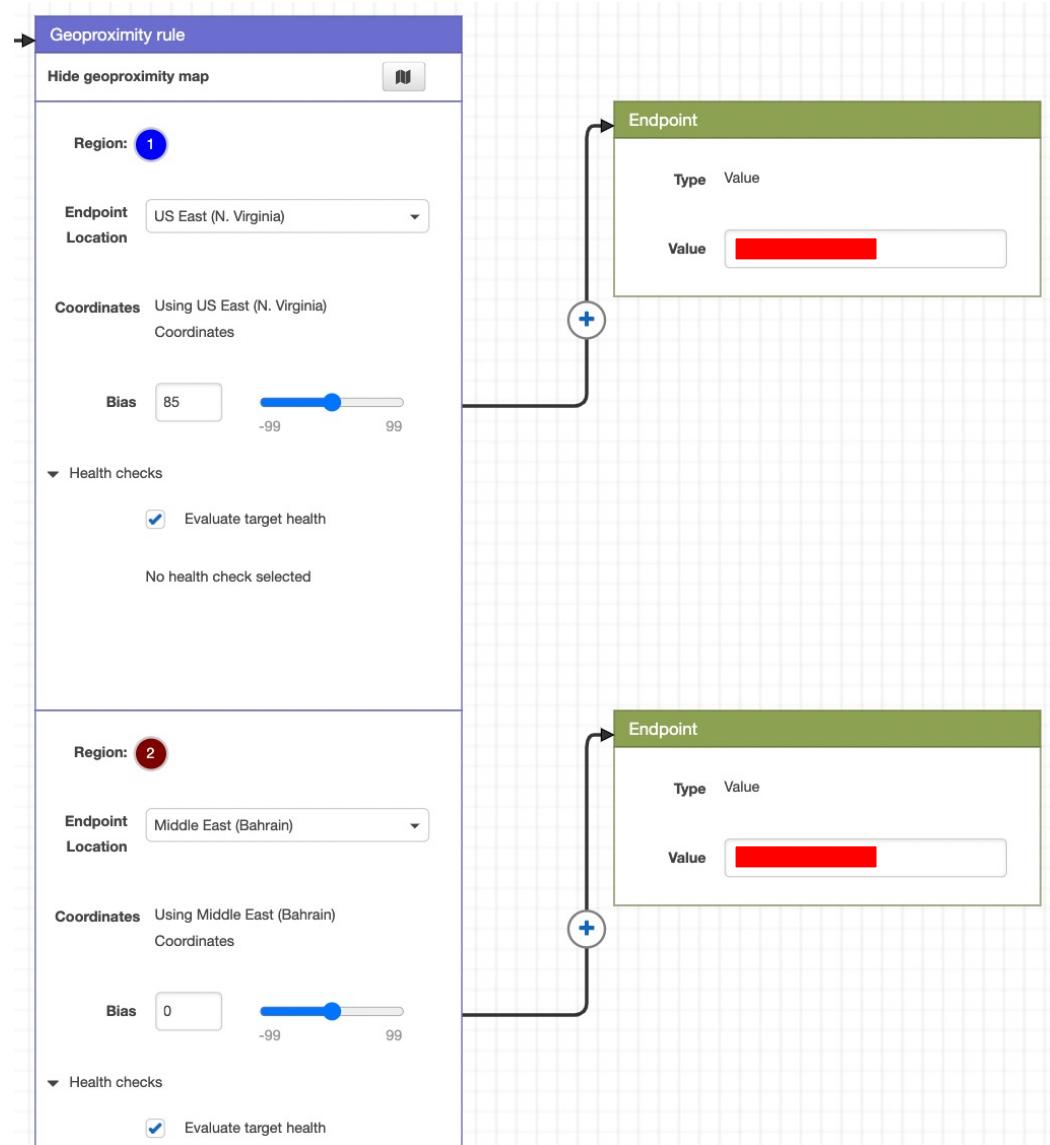


# Routing Policies – Geoproximity



# Route 53 – Traffic flow

- Simplify the process of creating and maintaining records in large and complex configurations
- Visual editor to manage complex routing decision trees
- Configurations can be saved as **Traffic Flow Policy**
  - Can be applied to different Route 53 Hosted Zones (different domain names)
  - Supports versioning



# Routing Policies – Multi-Value

- Use when routing traffic to multiple resources
- Route 53 return multiple values/resources
- Can be associated with Health Checks (return only values for healthy resources)
- Up to 8 healthy records are returned for each Multi-Value query
- Multi-Value is not a substitute for having an ELB

| Name            | Type     | Value        | TTL | Set ID | Health Check |
|-----------------|----------|--------------|-----|--------|--------------|
| www.example.com | A Record | 192.0.2.2    | 60  | Web1   | A            |
| www.example.com | A Record | 198.51.100.2 | 60  | Web2   | B            |
| www.example.com | A Record | 203.0.113.2  | 60  | Web3   | C            |

# Domain Registrar vs. DNS Service

- You buy or register your domain name with a Domain Registrar typically by paying annual charges (e.g., GoDaddy, Amazon Registrar Inc., ...)
- The Domain Registrar usually provides you with a DNS service to manage your DNS records
- But you can use another DNS service to manage your DNS records
- Example: purchase the domain from GoDaddy and use Route 53 to manage your DNS records



# GoDaddy as Registrar & Route 53 as DNS Service



## Records

We can't display your DNS information because your nameservers aren't managed by us.

## Nameservers

Using custom nameservers [Change](#)

| Nameserver              |
|-------------------------|
| ns-1083.awsdns-07.org   |
| ns-932.awsdns-52.net    |
| ns-1911.awsdns-46.co.uk |
| ns-481.awsdns-60.com    |



Amazon  
Route 53

**Public Hosted Zone**  
stephanetheteacher.com

▼ Hosted zone details [Edit hosted zone](#)

| Hosted zone ID                          | Type               | Name servers   |
|---|--------------------|--|
| Z30IJCCWPKZUV                           | Public hosted zone | ns-252.awsdns-31.com<br>ns-1468.awsdns-55.org<br>ns-633.awsdns-15.net<br>ns-1800.awsdns-33.co.uk |
| Description                             | Record count       |  |
| HostedZone created by Route53 Registrar | 22                 |  |
| Query log                               |                    |  |

# 3<sup>rd</sup> Party Registrar with Amazon Route 53

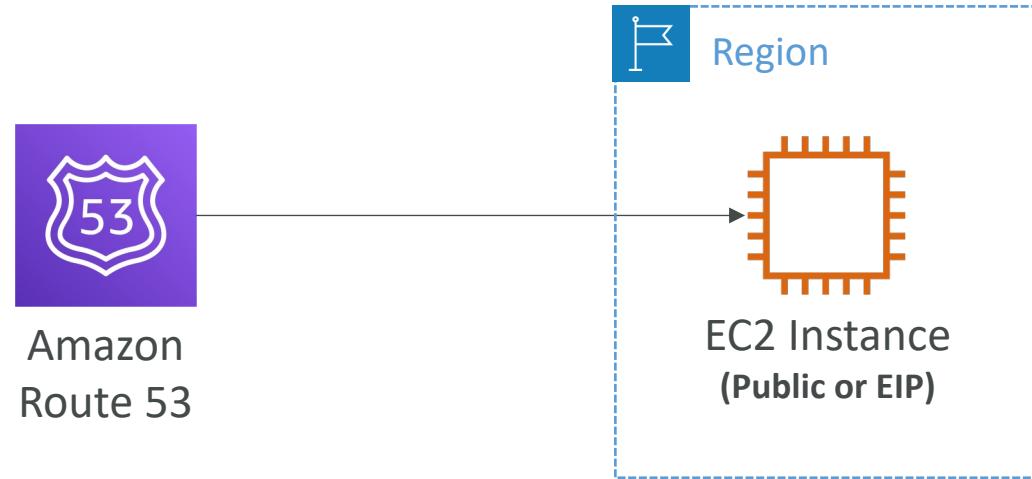
- If you buy your domain on a 3<sup>rd</sup> party registrar, you can still use Route 53 as the DNS Service provider
  - 1. Create a Hosted Zone in Route 53
  - 2. Update NS Records on 3<sup>rd</sup> party website to use Route 53 Name Servers
- Domain Registrar != DNS Service
- But every Domain Registrar usually comes with some DNS features

# Making Route 53 the DNS service for a domain that is in use (users are accessing it)

1. Get the current DNS configuration (records to duplicate)
2. Create a **public** hosted zone in Route 53
3. Create all records in the newly created zone
4. Lower TTL settings of NS record to 15 minutes (to roll back in case)
5. Wait two days to ensure the new NS record TTL has propagated
6. Update the NS record to use the Route 53 name servers
7. Monitor traffic for the domain
8. Change NS record TTL on Route 53 to a higher value (two days)
9. Transfer domain registration to Amazon Route 53 (optional)

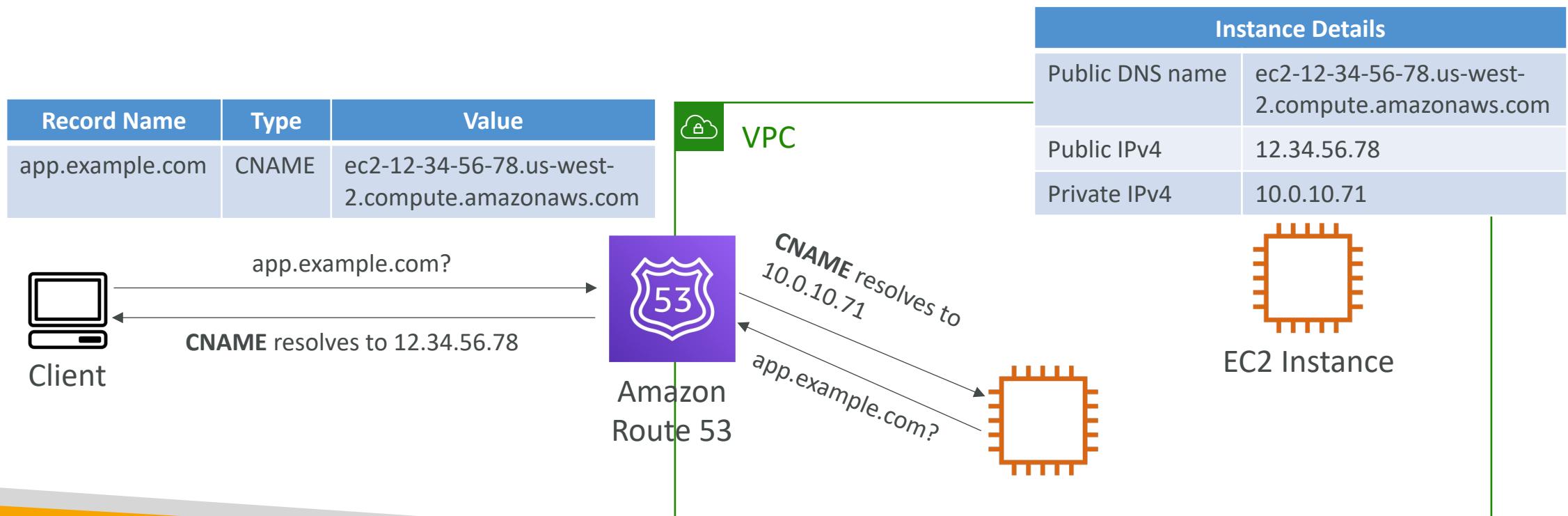
# Route 53 Scenarios – EC2 Instance

- Your domain points to an EC2 instances with public or Elastic IP
- Example:
  - example.com => 54.55.56.57 (A)



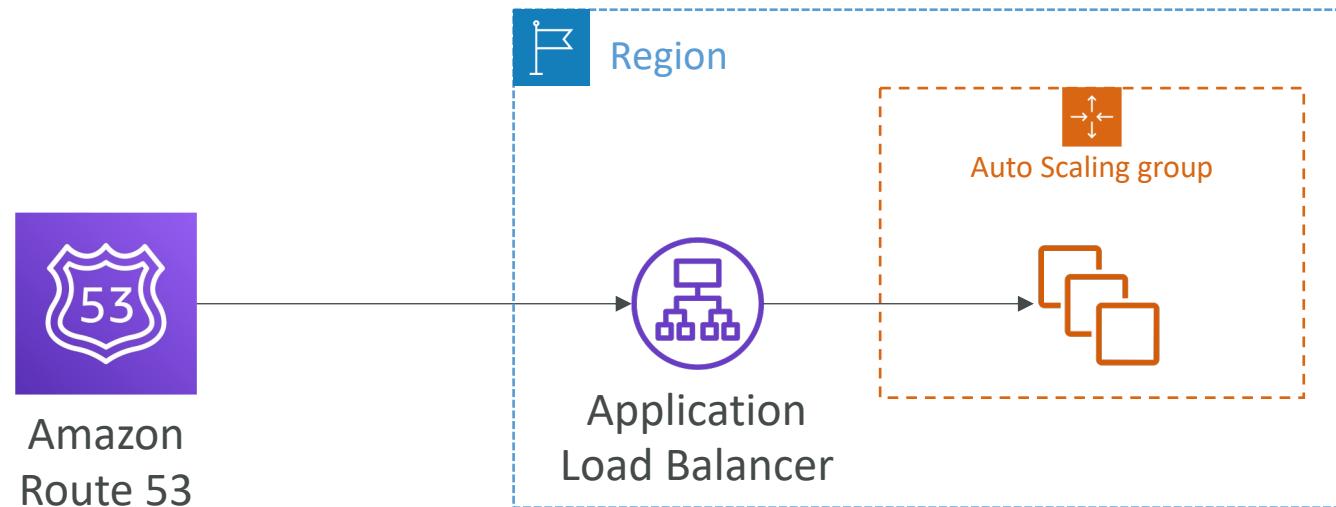
# Route 53 Scenarios – EC2 DNS name

- Example:
  - app.example.com => ec2-12-34-56-78.us-west-2.compute.amazonaws.com (**CNAME**)



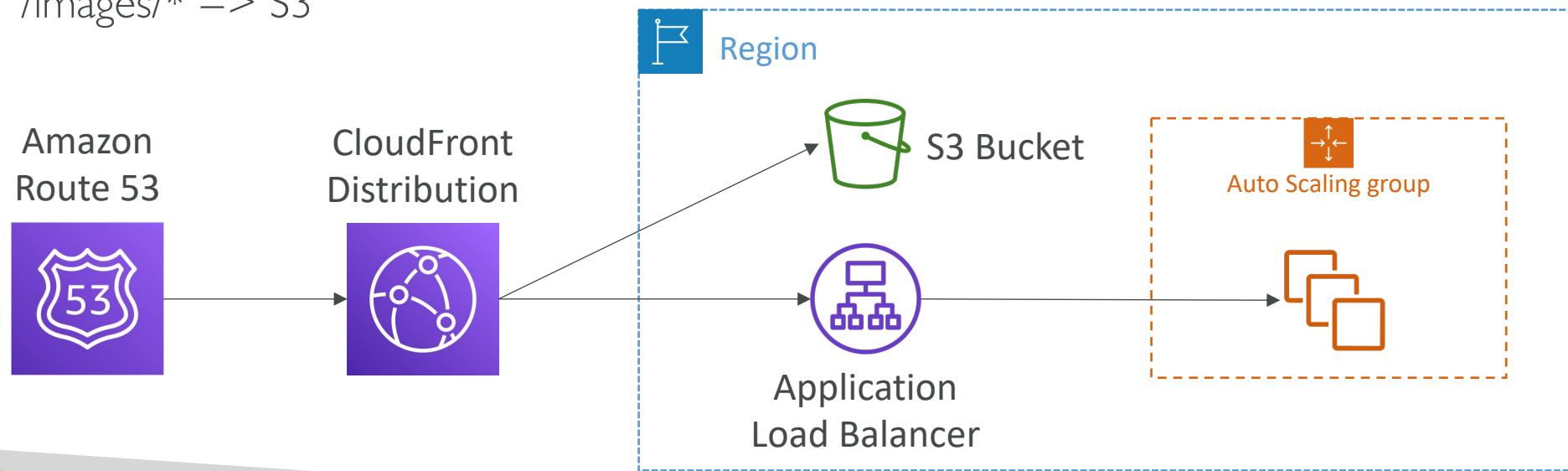
# Route 53 Scenarios – Application Load Balancer

- Your domain points to AWS provided ALB DNS name
- Example:
  - example.com => my-load-balancer-1234567890.us-west-2.elb.amazonaws.com (Alias)
  - lb.example.com => my-load-balancer-1234567890.us-west-2.elb.amazonaws.com (Alias or CNAME)



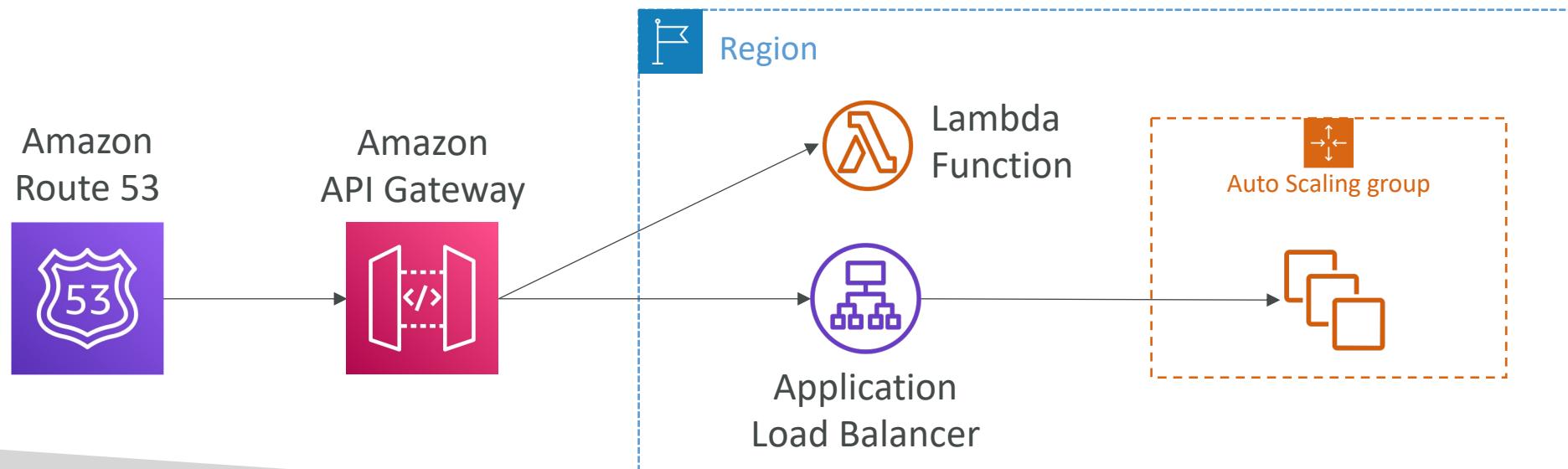
# Route 53 Scenarios – CloudFront Distribution

- Example:
  - example.com => d2222222abcdef8.cloudfront.net (Alias)
  - cdn.example.com => d2222222abcdef8.cloudfront.net (Alias or CNAME)
- CloudFront internally connects to multiple origins
  - /application/ => ELB
  - /images/\* => S3



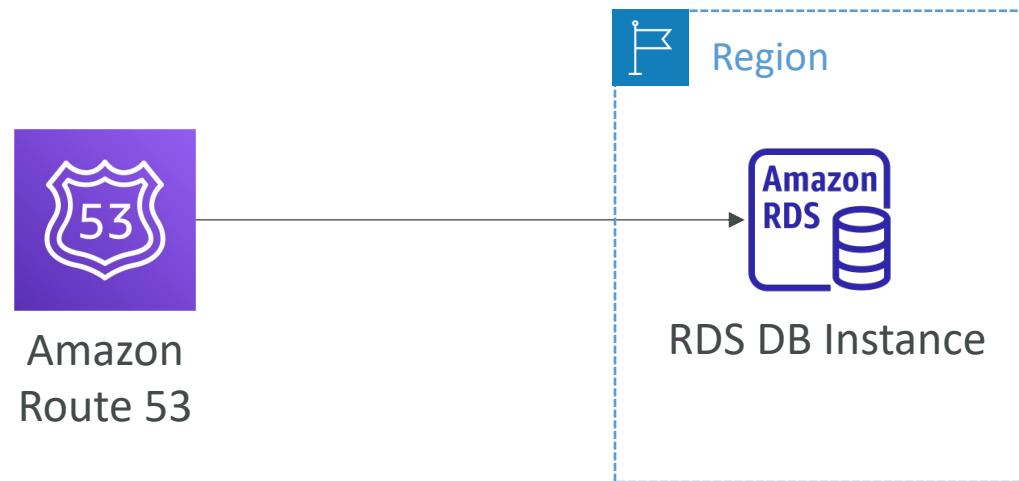
# Route 53 Scenarios – API Gateway

- Points to API Gateway Regional/Edge Optimized DNS name
- Example:
  - example.com => b123abcde4.execute-api.us-west-2.amazonaws.com (Alias)



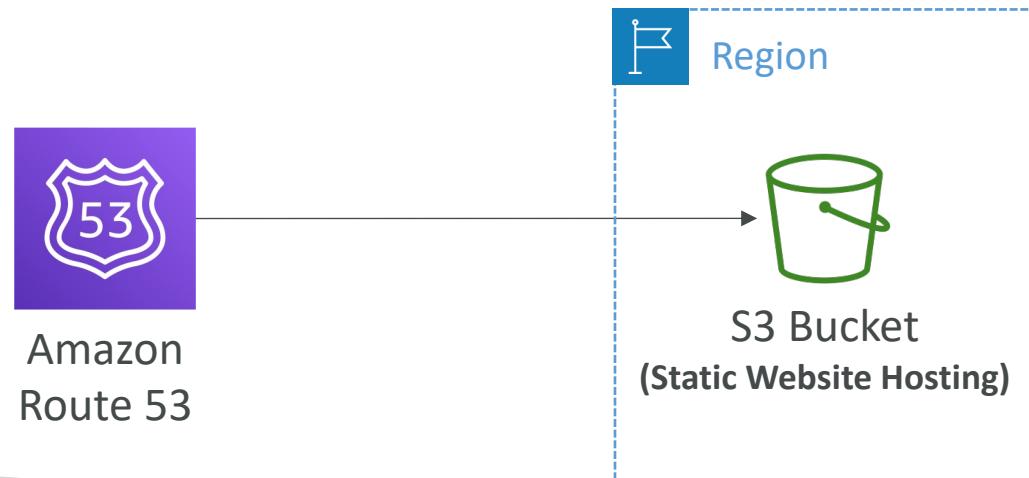
# Route 53 Scenarios – RDS DB Instance

- Your domain name points to RDS DB instance DNS name
- You must create a **CNAME** record (other record types are not supported)
- Example:
  - db.example.com => myexampledb.alb2c3d4wxyz.us-west-2.rds.amazonaws.com (**CNAME**)



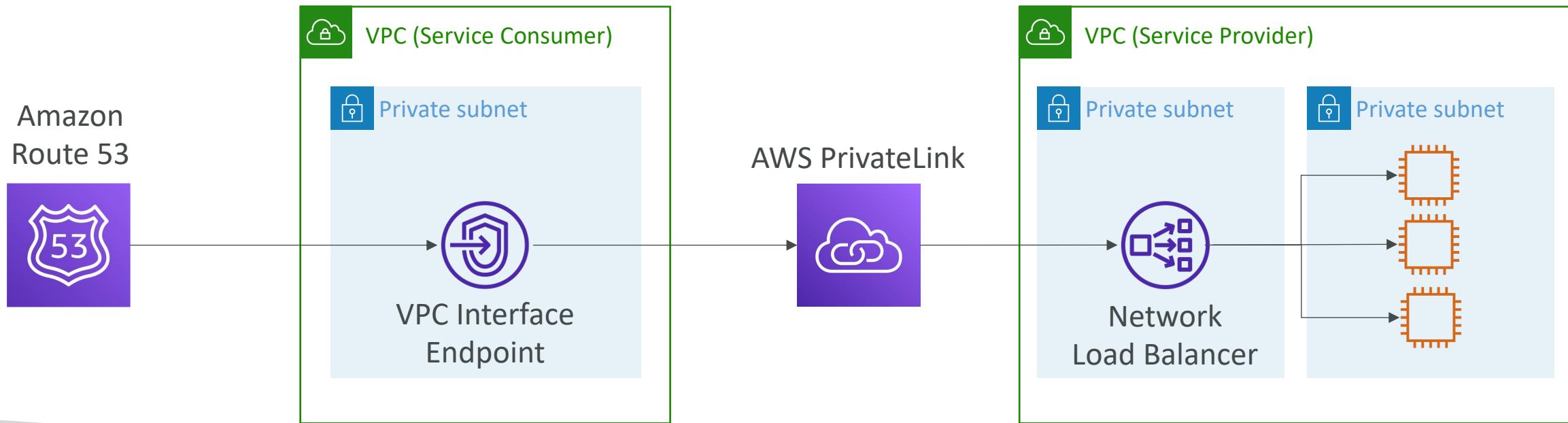
# Route 53 Scenarios – S3 Bucket

- Your domain name points to S3 website endpoint
- You must create an **Alias** record for S3 endpoints
- Bucket name must be the same as domain name
- Example:
  - example.com => s3-website-us-west-2.amazonaws.com (**Alias**)



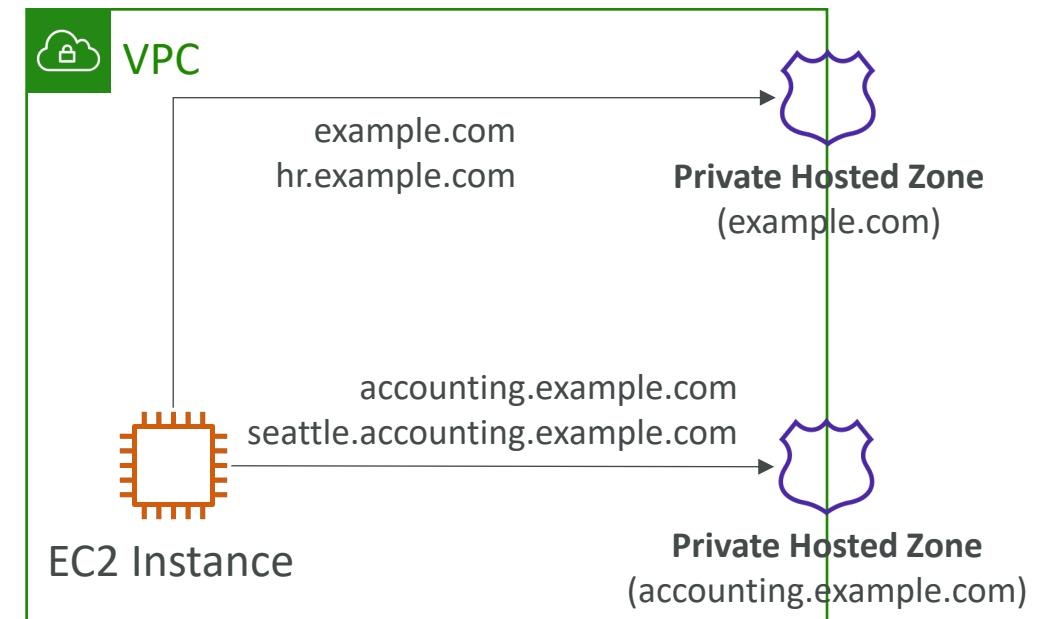
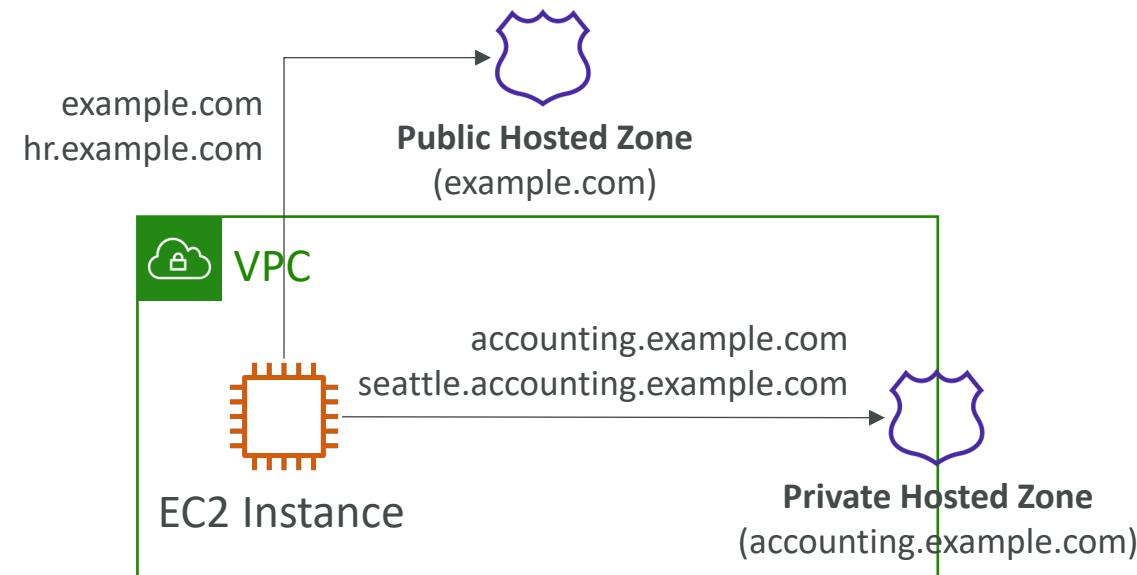
# Route 53 Scenarios – VPC Interface Endpoint

- Your domain name points to a VPC Interface Endpoint (AWS PrivateLink)
- Example:
  - example.com => vpce-1234-abcdev-us-east-1.vpce-svc-123345.us-east-1.vpce.amazonaws.com (Alias)



# Route 53 – Hosted Zones

- Route 53 automatically creates NS and SOA records
- For public/private and private Hosted Zones that have overlapping namespaces, Route 53 Resolvers routes traffic to the **most specific match**



# Route 53 – Routing Traffic For Subdomains

- Create a Hosted Zone for the Subdomain
- Known as, either:
  - “Delegation Responsibility for a Subdomain to a Hosted Zone”
  - “Delegating a Subdomain to Another Name Servers”
- Use cases:
  - different subdomains managed by different teams
  - Restrict access using IAM Permissions (you can't use IAM to control access to Route 53 records)



**Hosted Zone**  
(example.com)

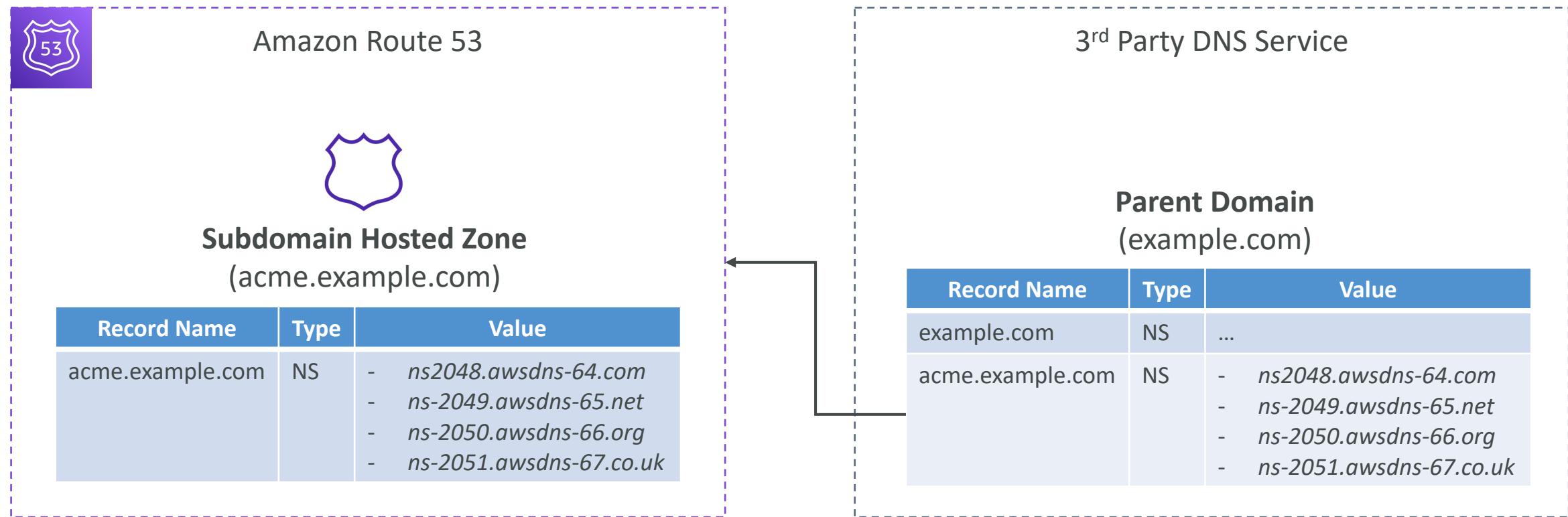
| Record Name      | Type | Value   |
|------------------|------|---|
| example.com      | NS   | ...   |
| acme.example.com | NS   | - ns2048.awsdns-64.com<br>- ns-2049.awsdns-65.net<br>- ns-2050.awsdns-66.org<br>- ns-2051.awsdns-67.co.uk |



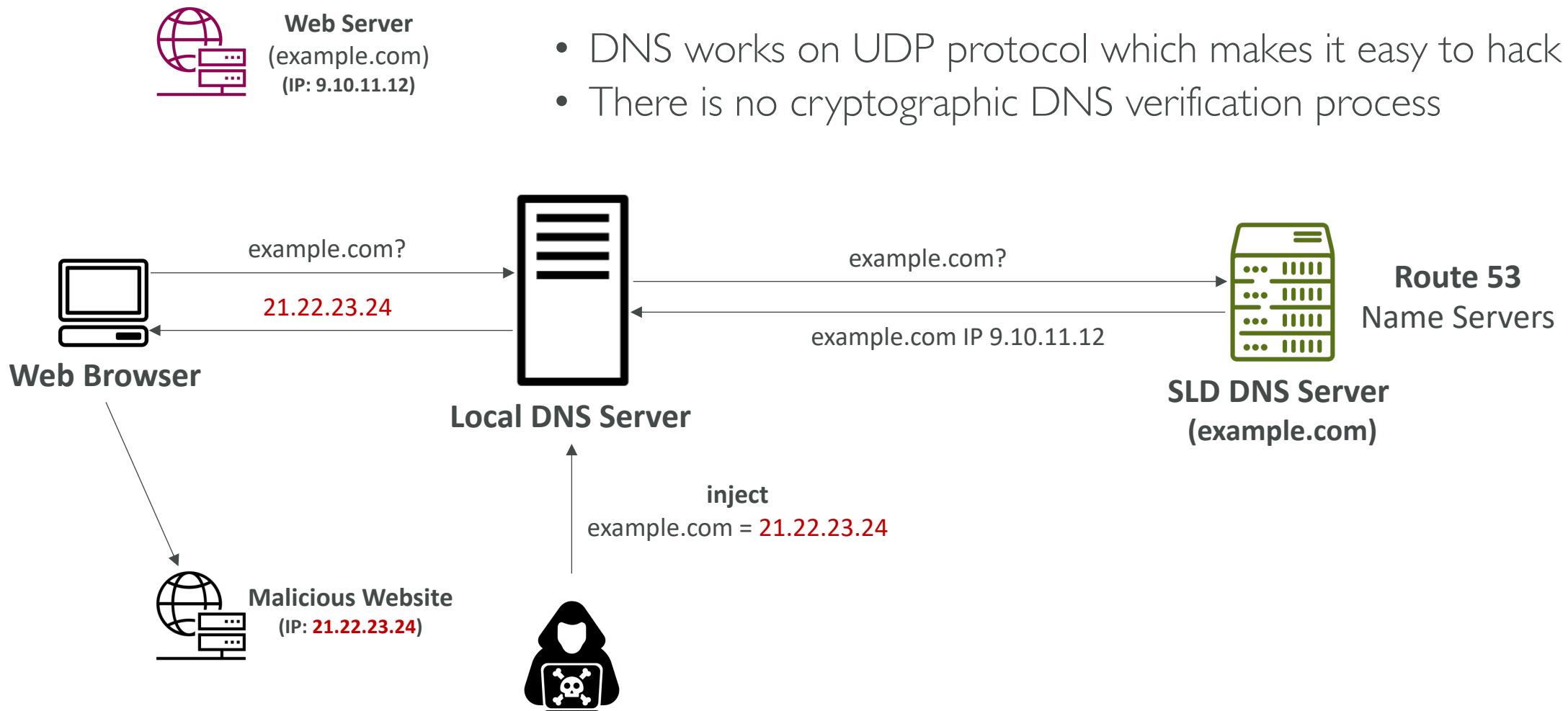
**Hosted Zone**  
(acme.example.com)

| Record Name      | Type | Value   |
|------------------|------|---|
| acme.example.com | NS   | - ns2048.awsdns-64.com<br>- ns-2049.awsdns-65.net<br>- ns-2050.awsdns-66.org<br>- ns-2051.awsdns-67.co.uk |

# Using Route 53 as the DNS Service for a Subdomain without Migrating the Parent Domain



# DNS Poisoning (Spoofing)



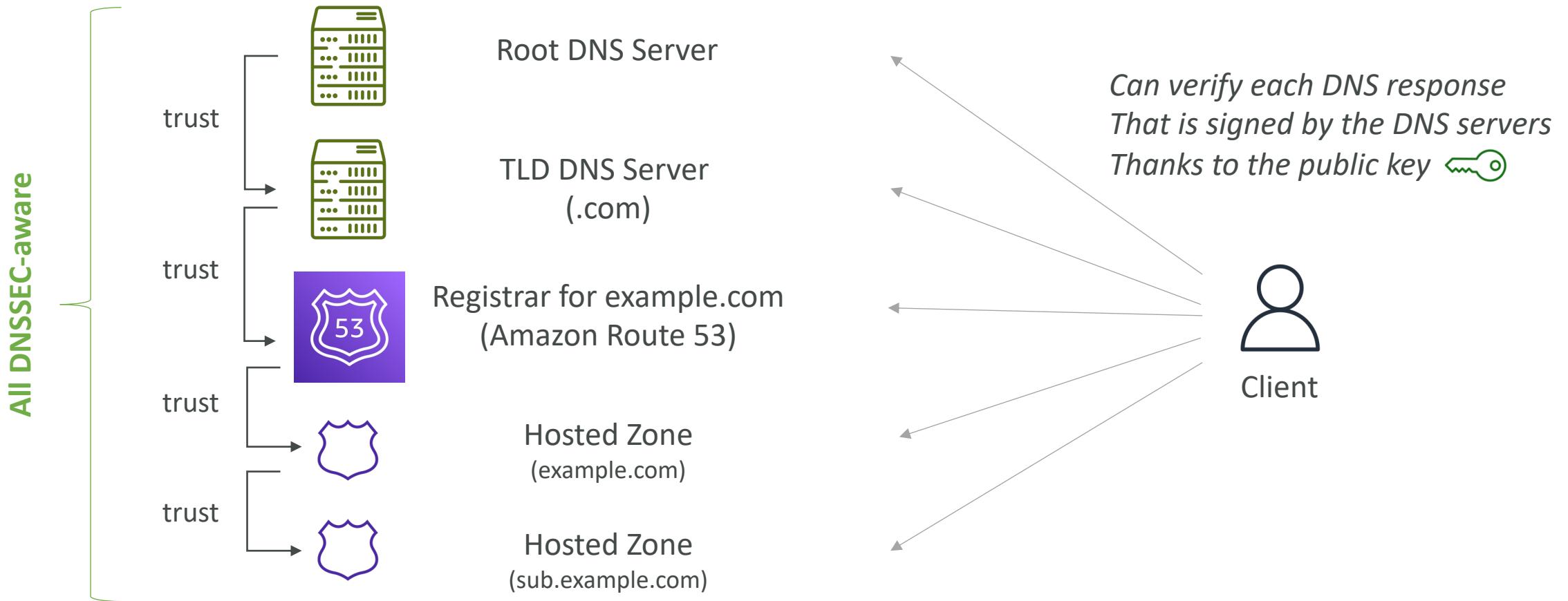
# Route 53 – DNS Security Extensions (DNSSEC)

- A protocol for securing DNS traffic, verifies DNS data integrity and origin
- Works only with **Public Hosted Zones**
- Route 53 supports both DNSSEC for Domain Registration and Signing
- **DNSSEC Signing**
  - Validate that a DNS response came from Route 53 and has not been tampered with
  - Route 53 cryptographically signs each record in the Hosted Zone
  - Two Keys:
    - Managed by you: Key-signing Key (KSK) – based on an asymmetric CMK in AWS KMS
    - Managed by AWS: Zone-signing Key (ZSK)
- When enabled, Route 53 enforces a TTL of one week for all records in the Hosted Zone (records that have TTL less than one week are not affected)

# Route 53 – Enable DNSSEC on a hosted zone

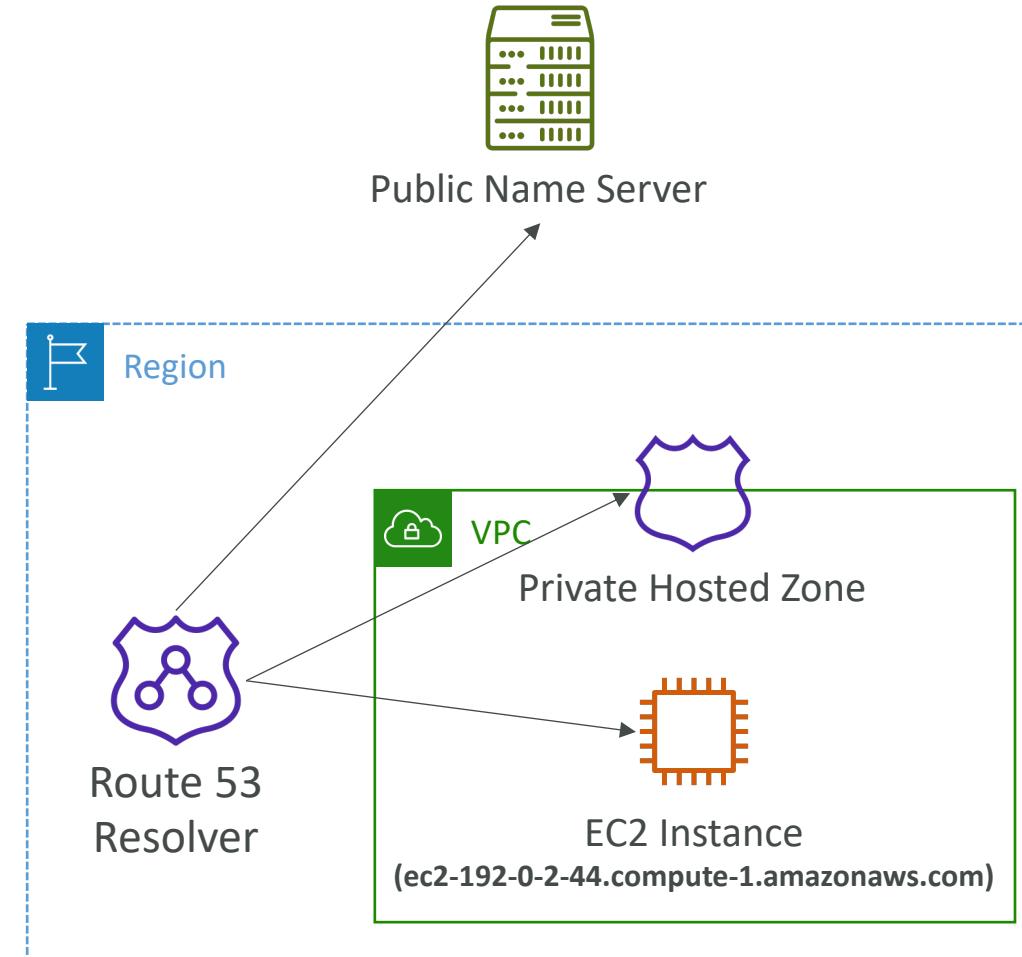
- Step 1 – Prepare for DNSSEC signing
  - Monitor zone availability (through customer feedback)
  - Lower TTL for records (recommended 1 hour)
  - Lower SOA minimum for 5 minutes
- Step 2 – Enable DNSSEC signing and create a KSK
  - Enable DNSSEC in Route 53 for your hosted zone (Console or CLI)
  - Make Route 53 create a KSK in the console and link it to a Customer managed CMK
- Step 3 – Establish chain of trust
  - Create a chain of trust between the hosted zone and the parent hosted zone
  - **By creating a Delegation Signer (DS) record in the parent zone**
  - It contains a hash of the public key used to sign DNS records
  - Your registrar can be Route 53 or a 3rd party registrar
- Step 4 – (good to have) Monitor for errors using CloudWatch Alarms
  - Create CloudWatch alarms for *DNSSECIInternalFailure* and *DNSSECKevSigningKeysNeedingAction*

# DNSSEC – Chain of Trust



# Route 53 – Hybrid DNS

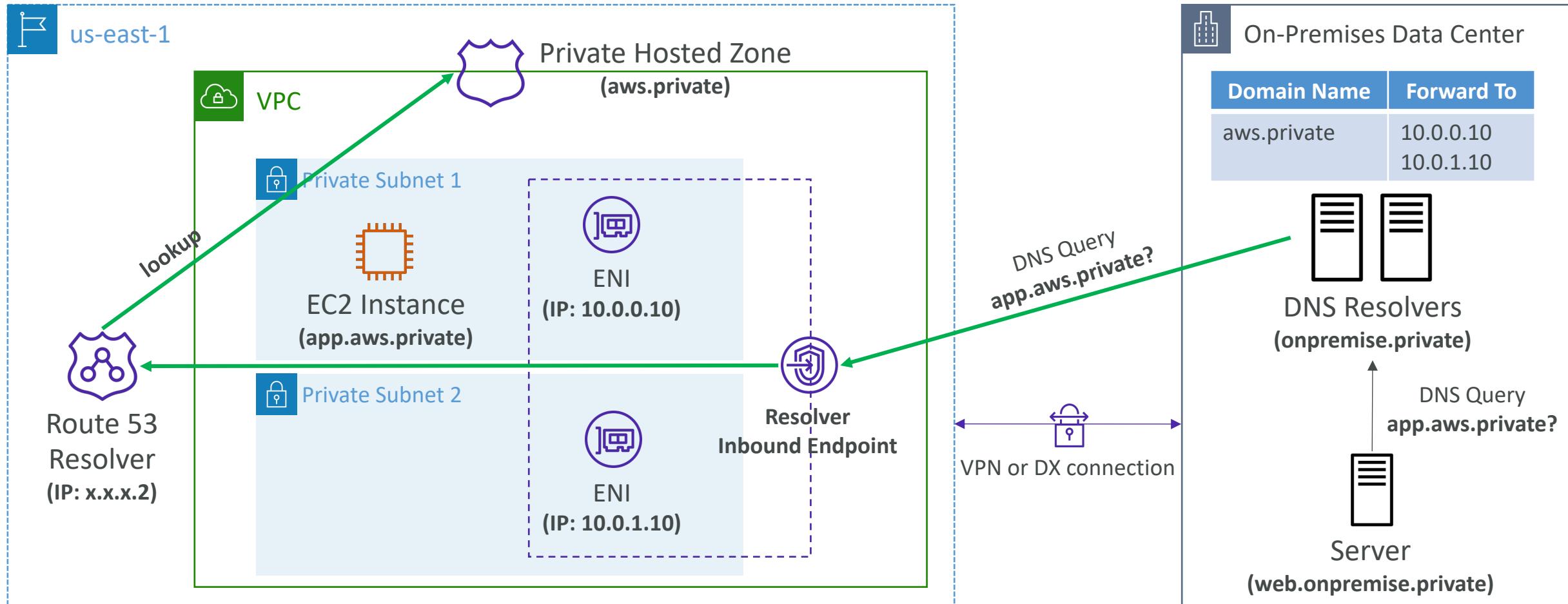
- By default, Route 53 Resolver automatically answers DNS queries for:
  - Local domain names for EC2 instances
  - Records in Private Hosted Zones
  - Records in public Name Servers
- **Hybrid DNS** – resolving DNS queries between VPC (Route 53 Resolver) and your networks (other DNS Resolvers)
- Networks can be:
  - VPC itself / Peered VPC
  - On-premises Network (connected through Direct Connect or AWS VPN)



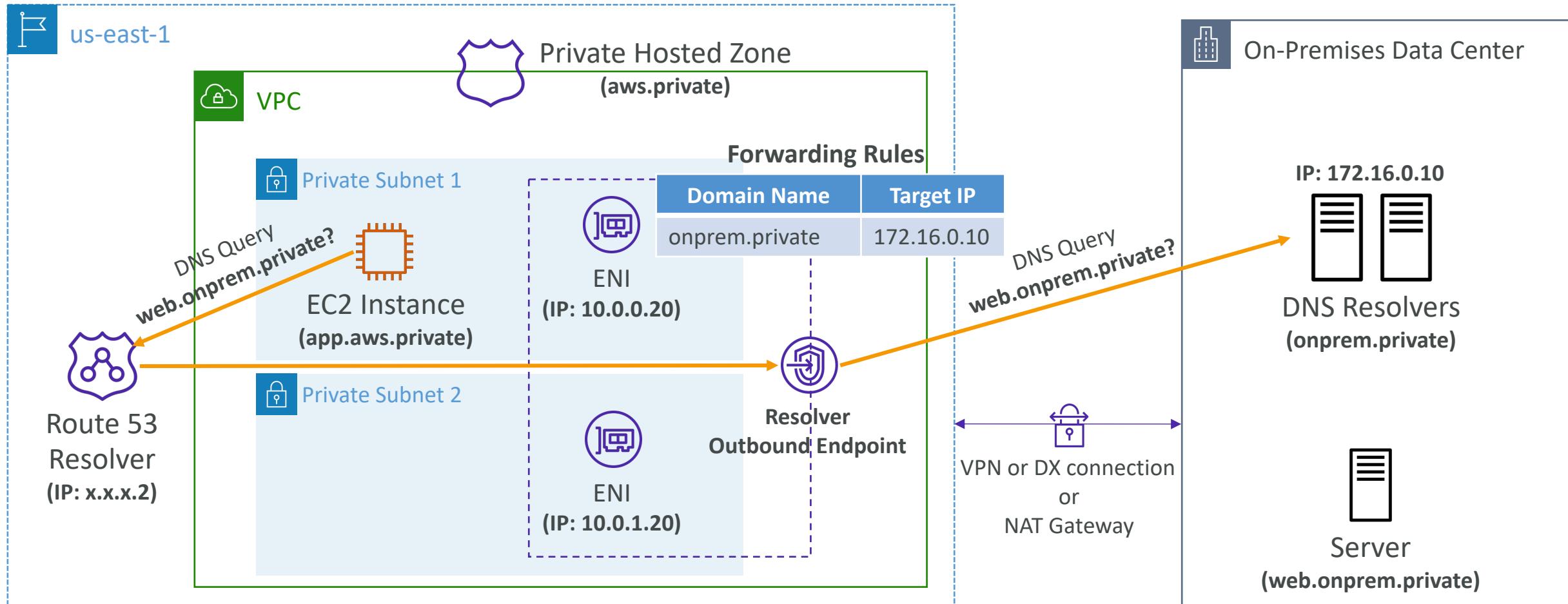
# Route 53 – Resolver Endpoints

- **Inbound Endpoint**
  - DNS Resolvers on your network can forward DNS queries to Route 53 Resolver
  - Allows your DNS Resolvers to resolve domain names for AWS resources (e.g., EC2 instances) and records in Route 53 Private Hosted Zones
- **Outbound Endpoint**
  - Route 53 Resolver conditionally forwards DNS queries to your DNS Resolvers
  - Use **Resolver Rules** to forward DNS queries to your DNS Resolvers
- Associated with one or more VPCs in the same AWS Region
- Create in two AZs for high availability
- Each Endpoint supports 10,000 queries per second per IP address

# Route 53 – Resolver Inbound Endpoints

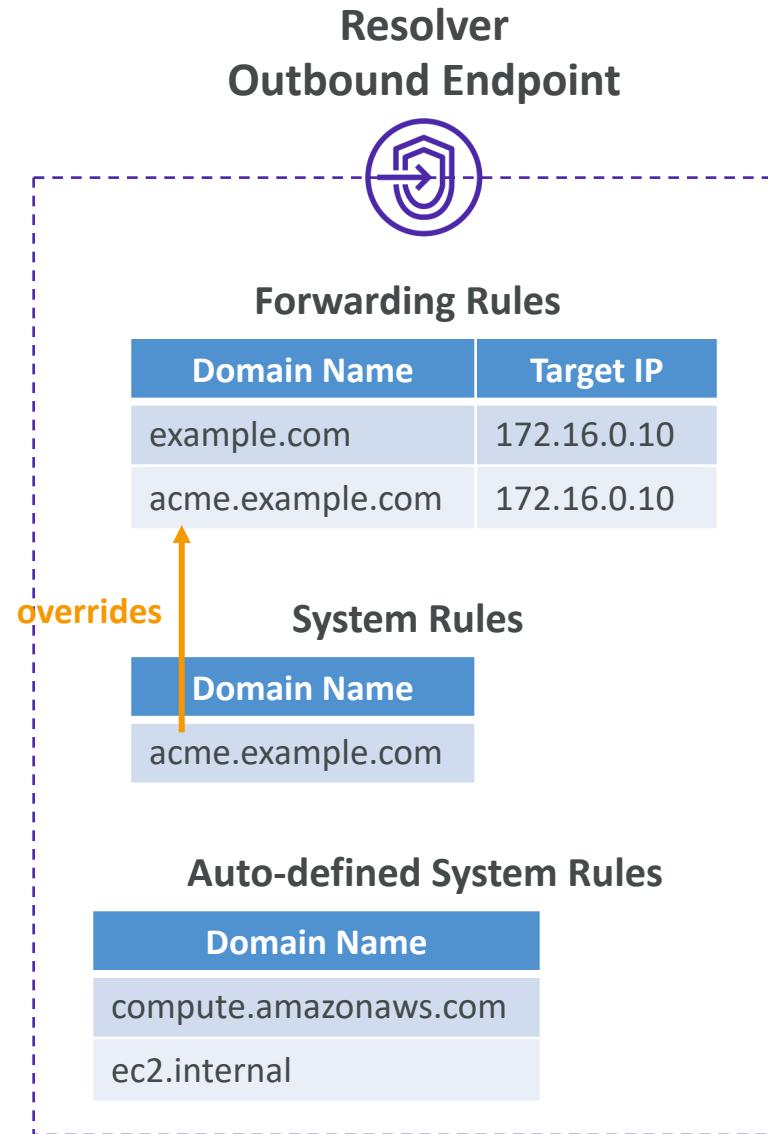


# Route 53 – Resolver Outbound Endpoints



# Route 53 – Resolver Rules

- Control which DNS queries are forwarded to DNS Resolvers on your network
- **Conditional Forwarding Rules (Forwarding Rules)**
  - Forward DNS queries for a specified domain and all its subdomains **to target IP addresses**
- **System Rules**
  - Selectively overriding the behavior defined in Forwarding Rules (e.g., don't forward DNS queries for a subdomain acme.example.com)
- **Auto-defined System Rules**
  - Defines how DNS queries for selected domains are resolved (e.g., AWS internal domain names, Private Hosted Zones)
- If multiple rules matched, Route 53 Resolver chooses the most specific match
- **Resolver Rules can be shared across accounts using AWS RAM**
  - Manage them centrally in one account
  - Send DNS queries from multiple VPC to the target IP defined in the rule



# Route 53 – DNS Query Logging

- Log information about public DNS queries Route 53 Resolver receives
- Only for Public Hosted Zones
- Can send logs to CloudWatch Logs (can export to S3)

| Log Format | Version                         | Hosted Zone ID       | Query Type              | Query Protocol           | Resolver IP Address                 |
|------------|---------------------------------|----------------------|-------------------------|--------------------------|-------------------------------------|
| 1.0        | 2017-12-13T08:16:02.130Z        | Z123412341234        | example.com A           | NOERROR UDP FRA6         | 192.168.1.1 -                       |
| 1.0        | <u>2017-12-13T08:15:50.235Z</u> | <u>Z123412341234</u> | <u>example.com AAAA</u> | <u>NOERROR TCP IAD12</u> | <u>192.168.3.1 192.168.222.0/24</u> |
| 1.0        | 2017-12-13T08:16:03.983Z        | Z123412341234        | example.com ANY         | NOERROR UDP FRA6         | 2001:db8::1234 2001:db8:abcd::/48   |
| 1.0        | 2017-12-13T08:15:50.342Z        | Z123412341234        | bad.example.com A       | NXDOMAIN UDP IAD12       | 192.168.3.1 192.168.111.0/24        |
| 1.0        | 2017-12-13T08:16:05.744Z        | Z123412341234        | txt.example.com TXT     | NOERROR UDP JFK5         | 192.168.1.2 -                       |

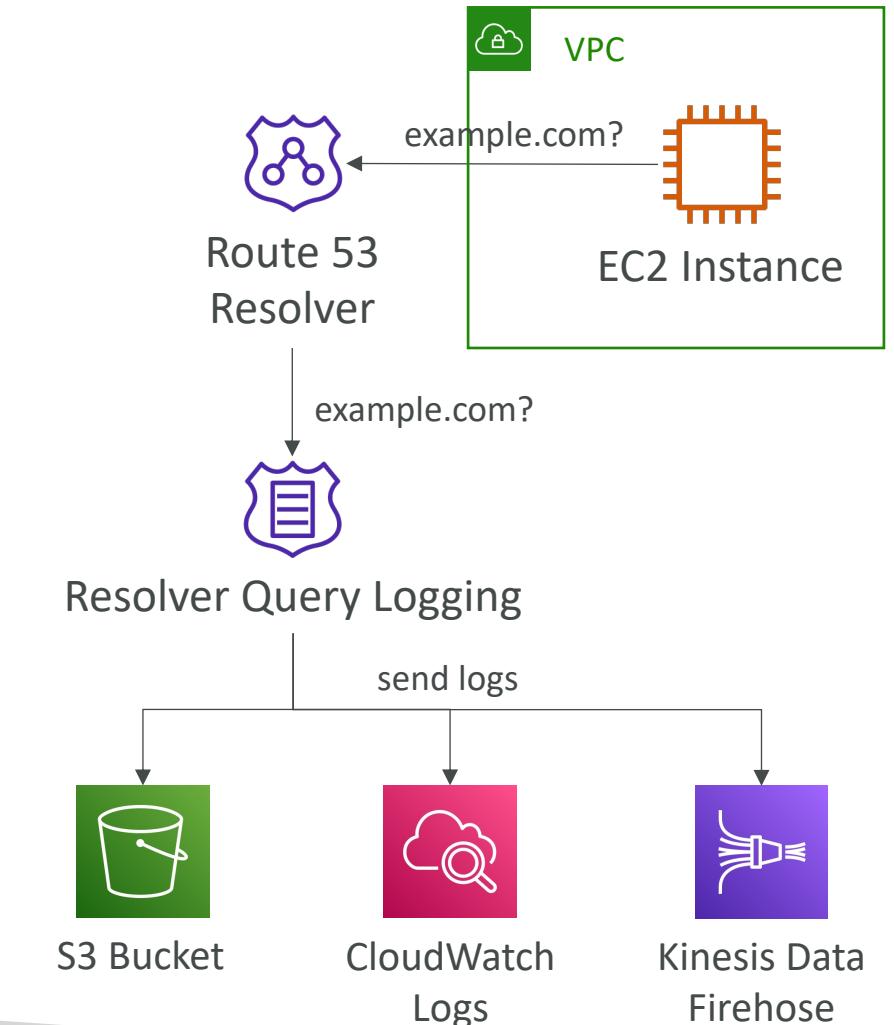
  

| Query Timestamp | Query Name | Response Code | Edge Location | EDNS Client Subnet |
|-----------------|------------|---------------|---------------|--------------------|
|-----------------|------------|---------------|---------------|--------------------|

# Route 53 – Resolver Query Logging



- Logs all DNS queries made by resources within a VPC
  - Private Hosted Zones
  - Resolver Inbound & Outbound Endpoints
  - Resolver DNS Firewall
- Can send logs to CloudWatch Logs, S3 bucket, or Kinesis Data Firehose
- Configurations can be shared with other AWS Accounts using AWS Resource Access Manager (AWS RAM)





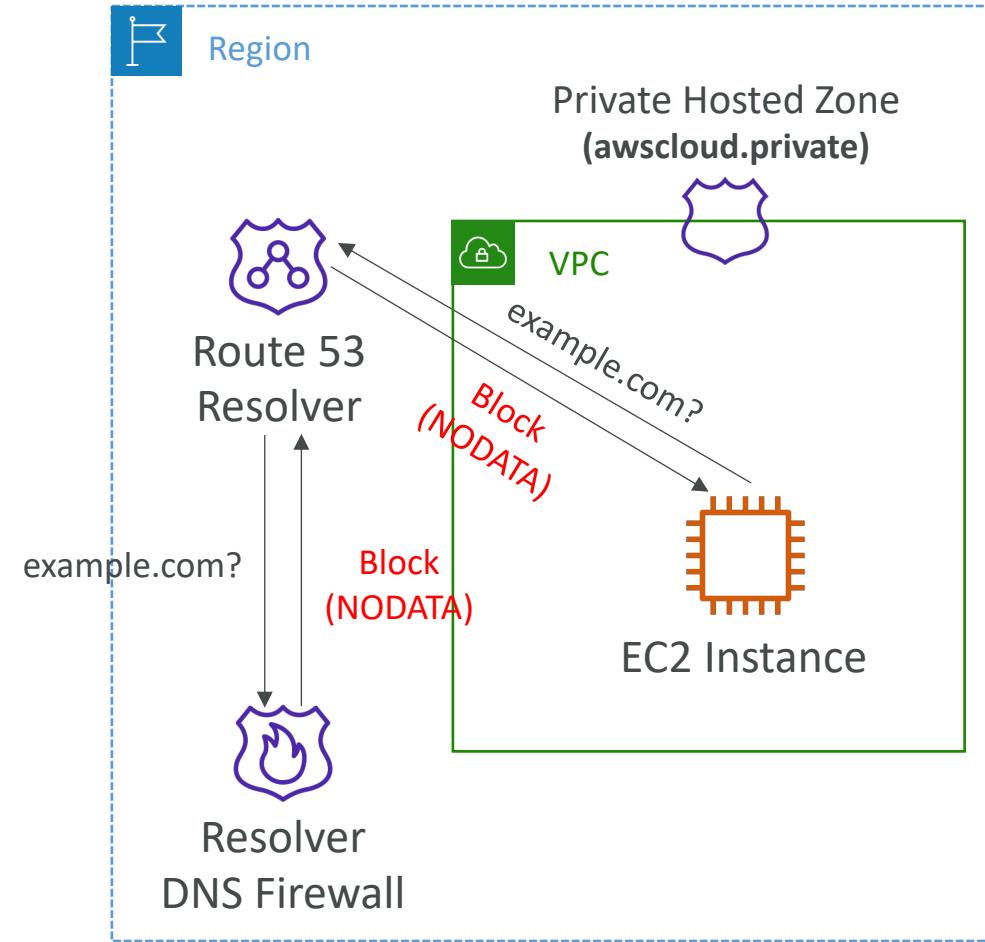
# Route 53 – Resolver Query Logging

```
{  
    "srcaddr": "4.5.64.102",  
    "vpc_id": "vpc-7example",  
    "answers": [  
        {  
            "Rdata": "203.0.113.9",  
            "Type": "PTR",  
            "Class": "IN"  
        }  
    ],  
    "firewall_rule_group_id": "rslvr-frg-01234567890abcdef",  
    "firewall_rule_action": "BLOCK",  
    "query_name": "15.3.4.32.in-addr.arpa.",  
    "firewall_domain_list_id": "rslvr-fdl-01234567890abcdef",  
    "query_class": "IN",  
    "srcids": {  
        "instance": "i-0d15cd0d3example"  
    },  
    "rcode": "NOERROR",  
    "query_type": "PTR",  
    "transport": "UDP",  
    "version": "1.100000",  
    "account_id": "111122223333",  
    "srcport": "56067",  
    "query_timestamp": "2021-02-04T17:51:55Z",  
    "region": "us-east-1"  
}
```



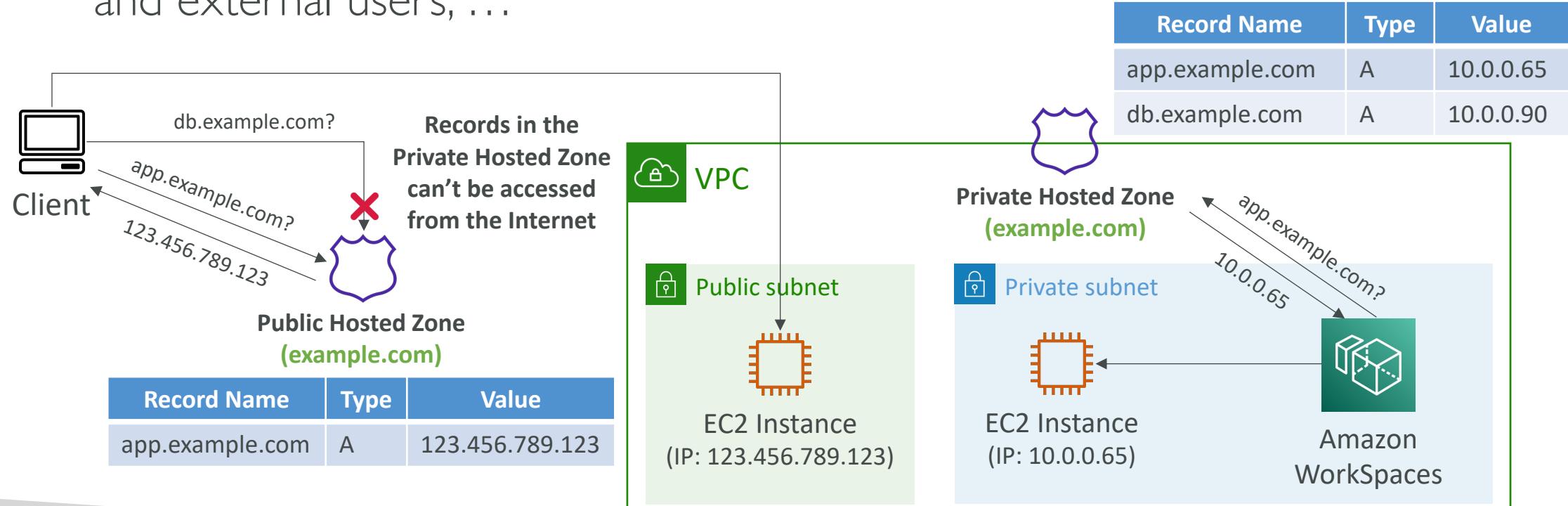
# Route 53 – Resolver DNS Firewall

- A managed firewall enables you to filter outbound DNS requests going out through Route 53 Resolver from your VPC
- Blacklist malicious domains or Whitelist trusted domains
- This is to prevent a compromised application within your VPC to send data out through DNS (to a malicious domain) – also called DNS exfiltration
- Can be managed/configured from AWS Firewall Manager
- Integrated with CloudWatch Logs and Route 53 Resolver Query Logs
- **Fail-close vs Fail-Open (DNS Firewall Configuration):**
  - **Fail-close:** Resolver blocks query if no response from DNS Firewall (security over availability)
  - **Fail-open:** Resolver allows query if no response from DNS firewall (availability over security)

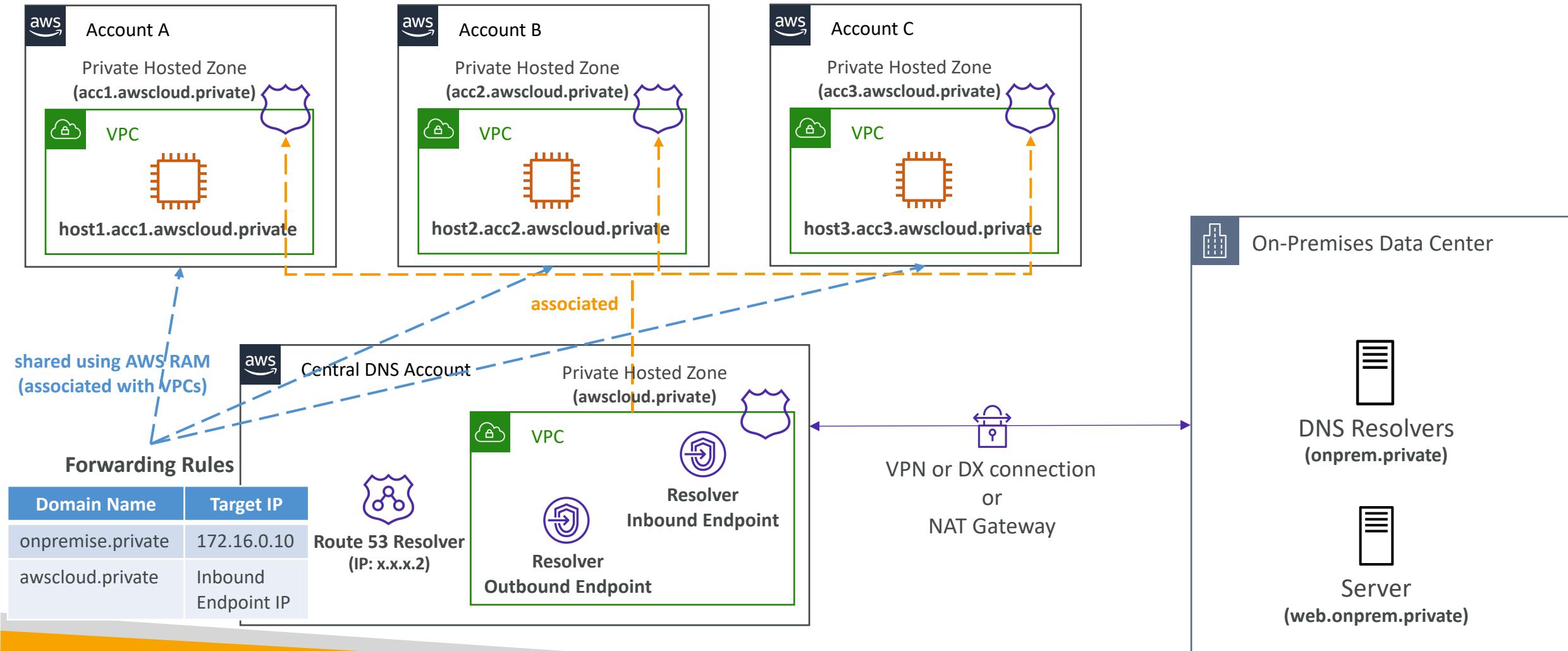


# Route 53 Solution Architecture Split-View DNS (Split-Horizon)

- Use the same domain for internal and external uses
- Public and Private hosted zones will have the same name (e.g., example.com)
- Use case: serve different content, require different authentication for internal and external users, ...

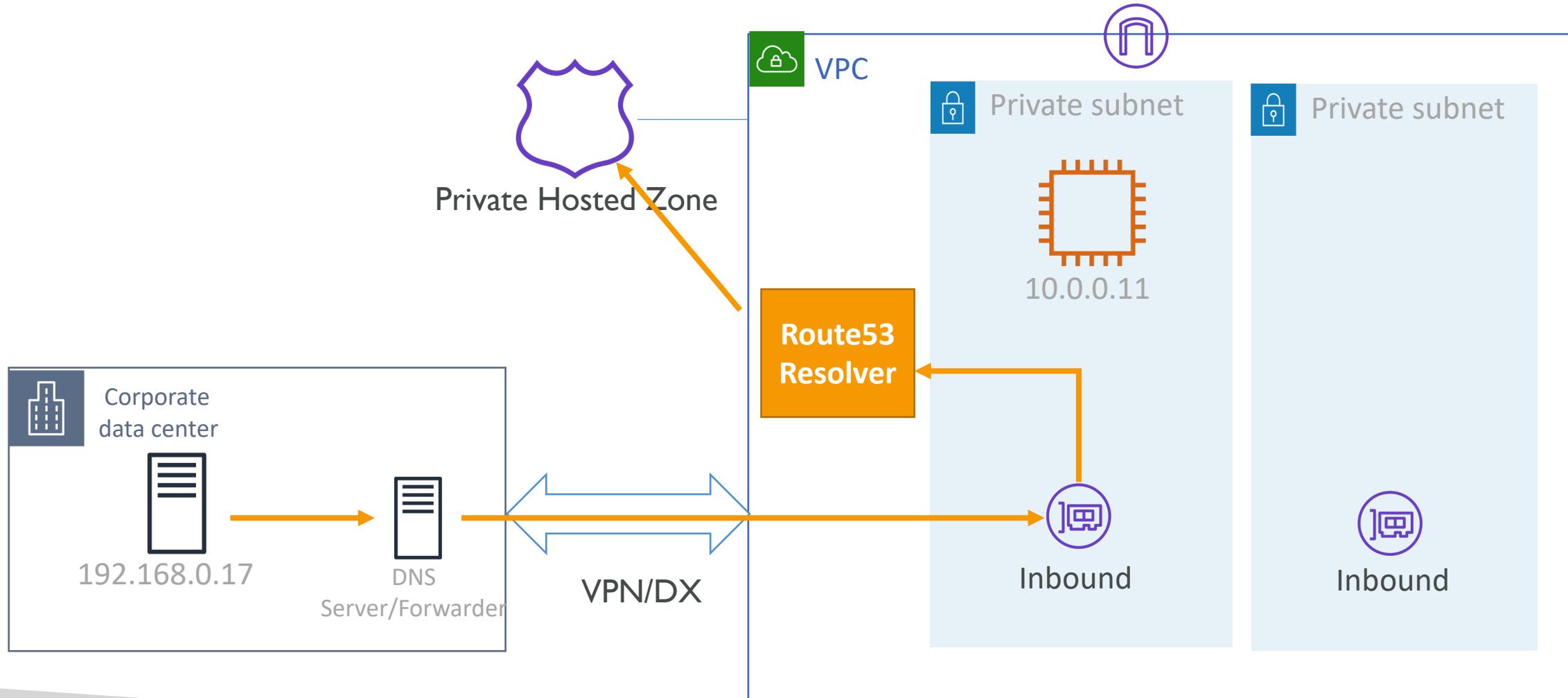


# Solution Architecture – Multi-Account DNS Management with Route 53 Resolver

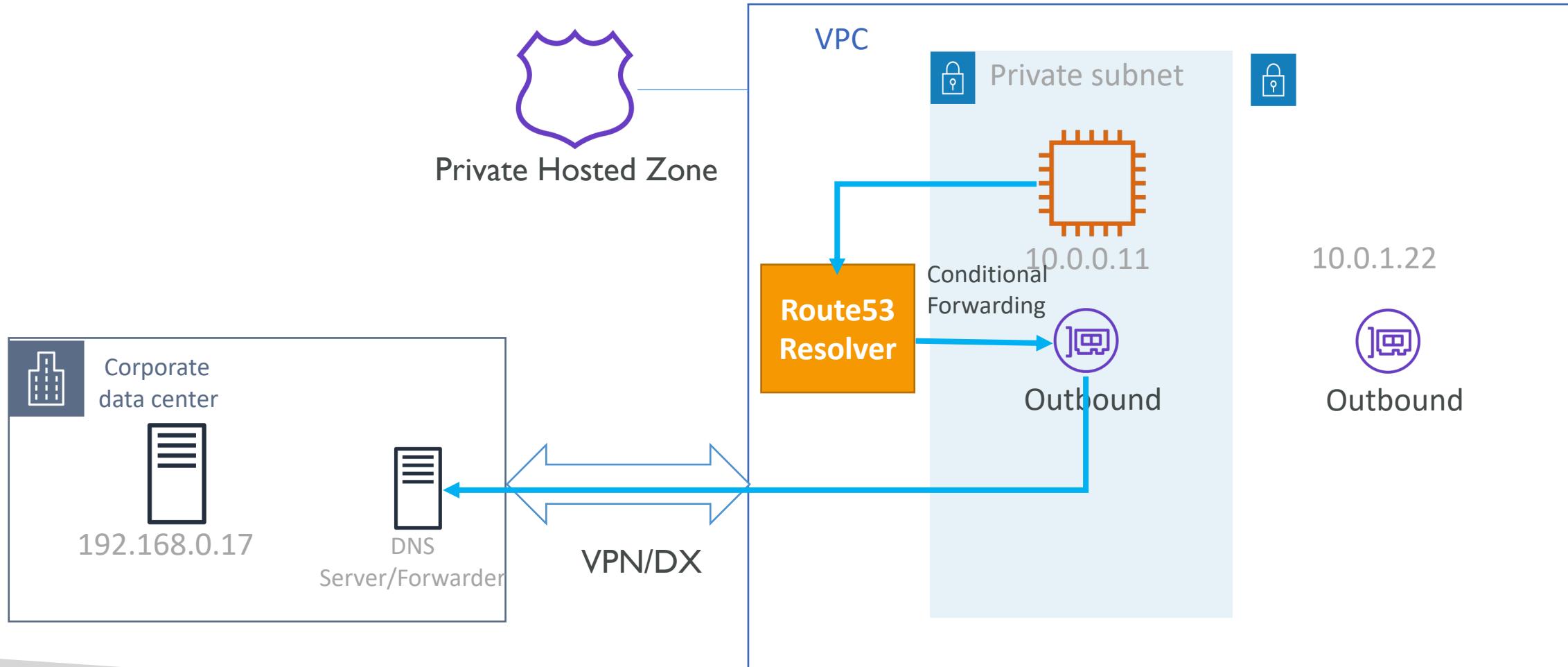


# Hands On: Route53 Hybrid DNS

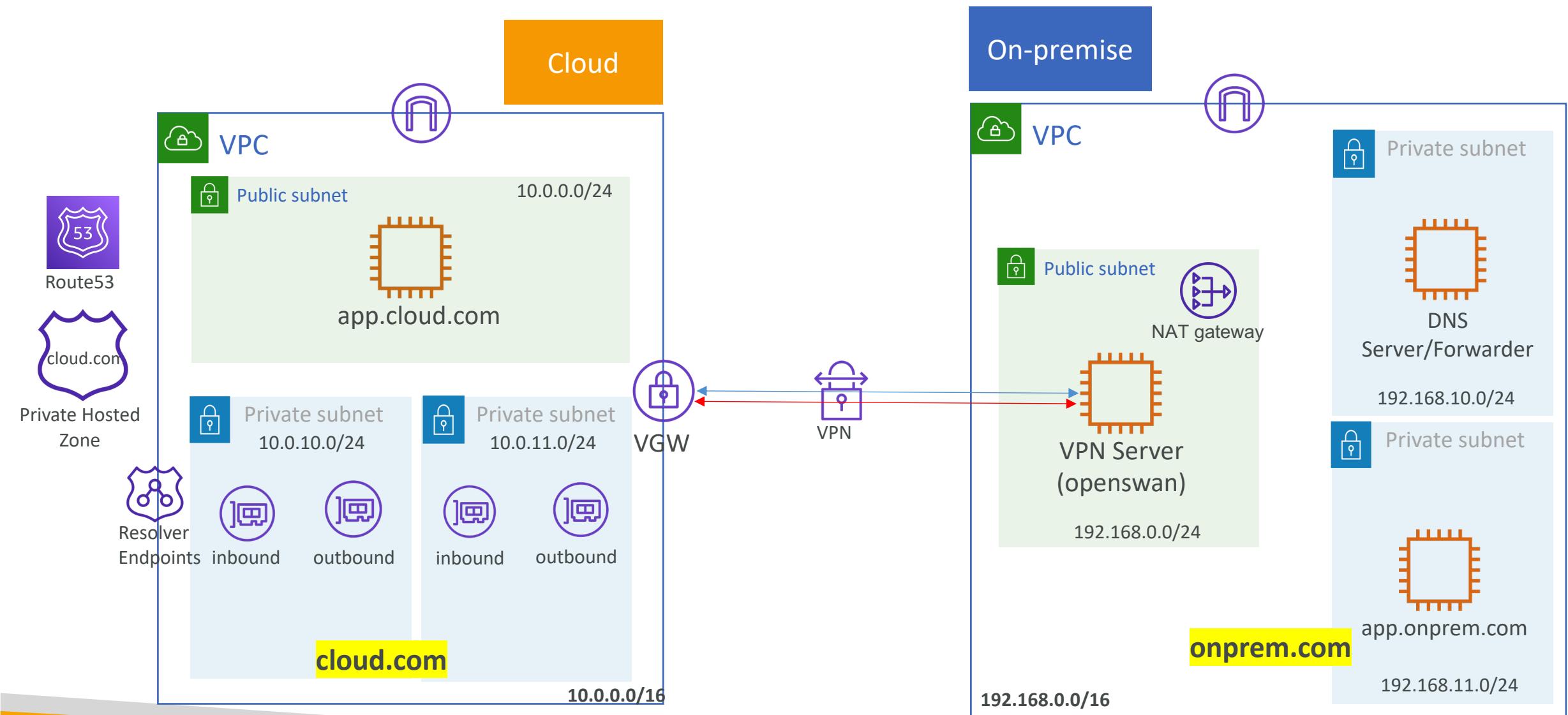
# Route53 Resolver Endpoint - Inbound



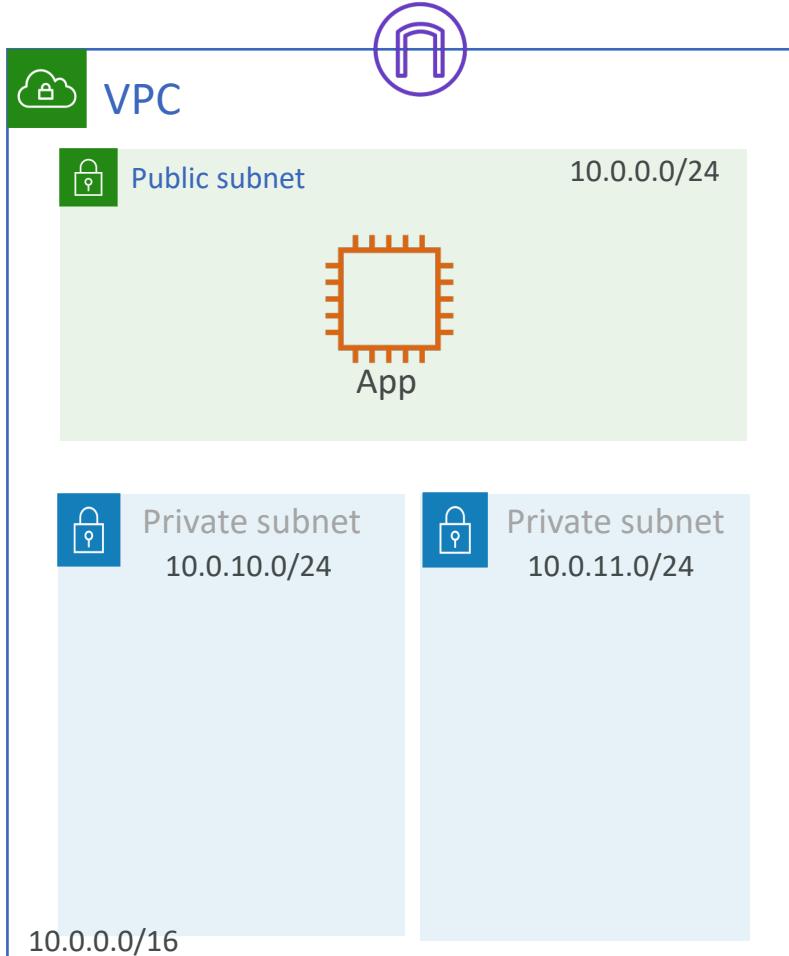
# Route53 Resolver Endpoint - Outbound



# Lab Architecture



# Step 1a – Setup basic cloud VPC network and app server



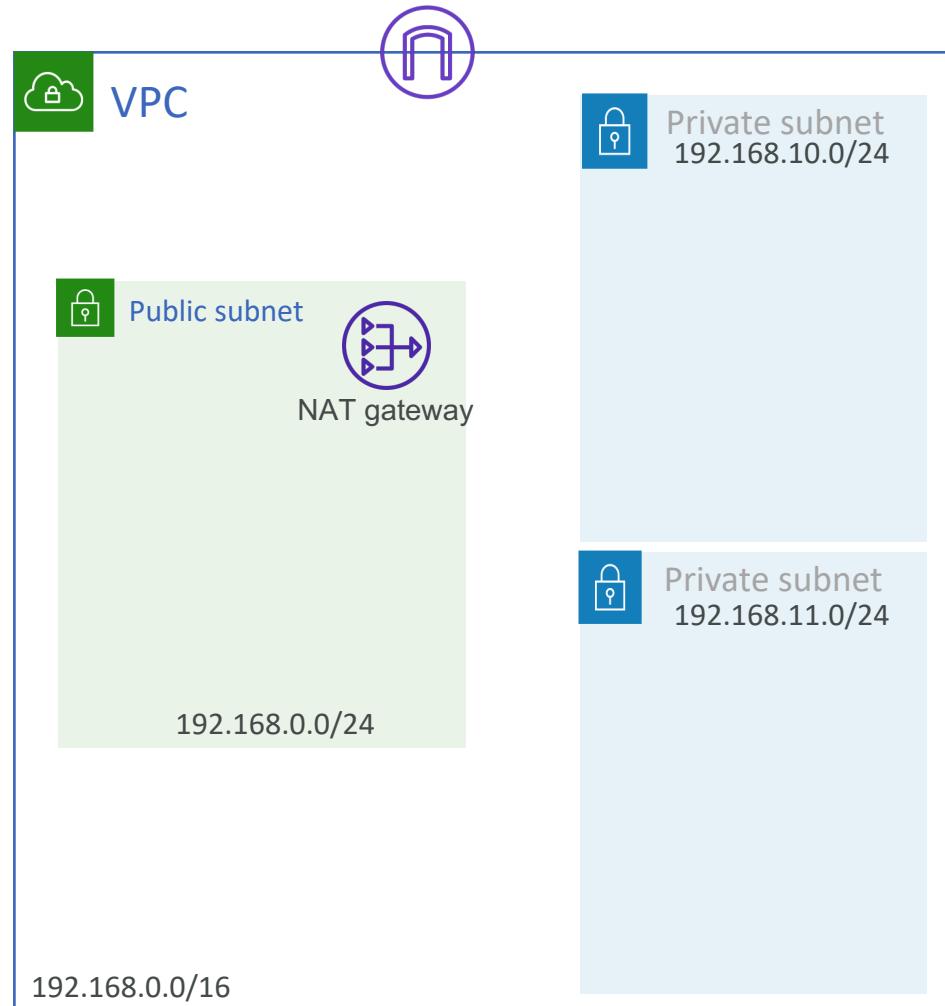
1. Create VPC (10.0.0.0/16), Internet Gateway, 1 Public subnet and 2 private subnets as shown
2. Create route tables for Public and Private subnets.
3. Launch App EC2 instance in a public subnet
4. Security group of App EC2 instance to allow
  1. SSH from 0.0.0.0/0 (to connect and configure)
  2. ICMP IPv4 from 192.168.0.0/16 (Ping from on-premises)

*Pre-created to save some time*

# Step 1b – Setup on-premise network

1. Create VPC (192.168.0.0/16), Internet Gateway, 1 Public subnet and 2 private subnets as shown
2. Create NAT gateway in Public subnet
3. Create route tables for Public and Private subnets.
4. Update private subnet Route table to route traffic for 0.0.0.0/0 through the NAT gateway.

**Pre-created to save some time**

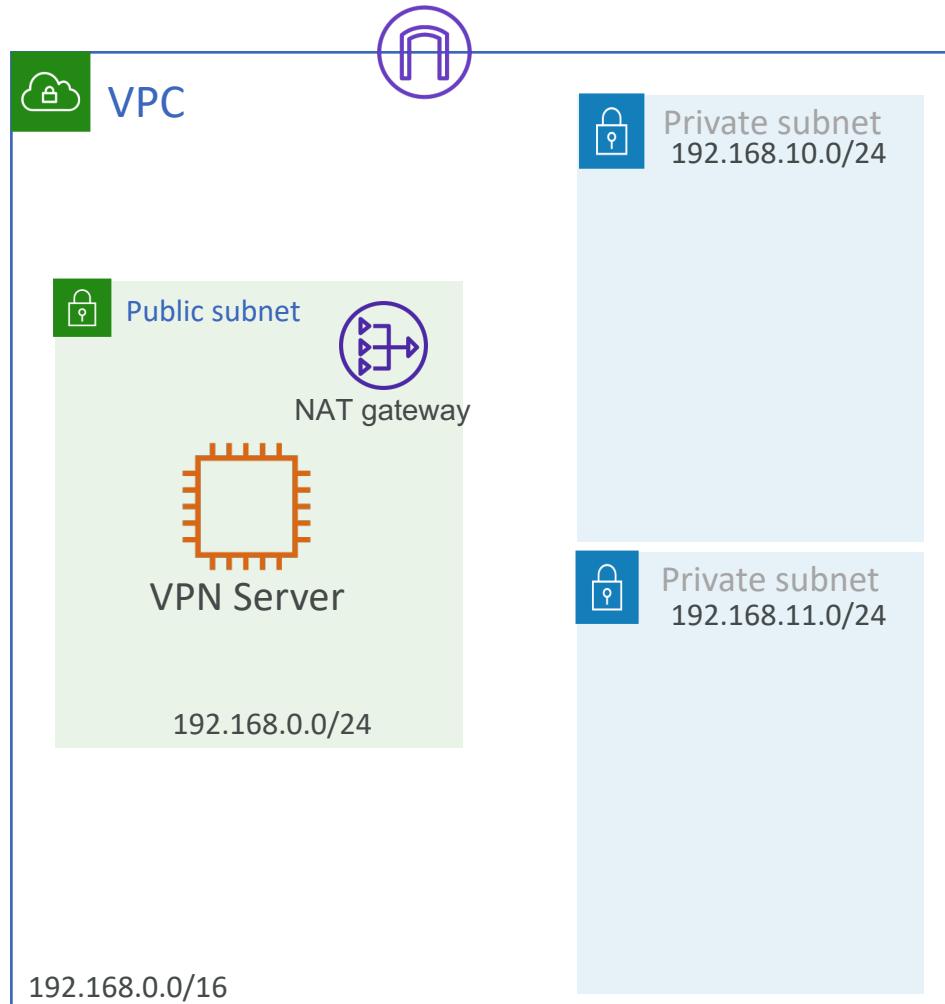


# Step 2a – Create on-premise VPN server

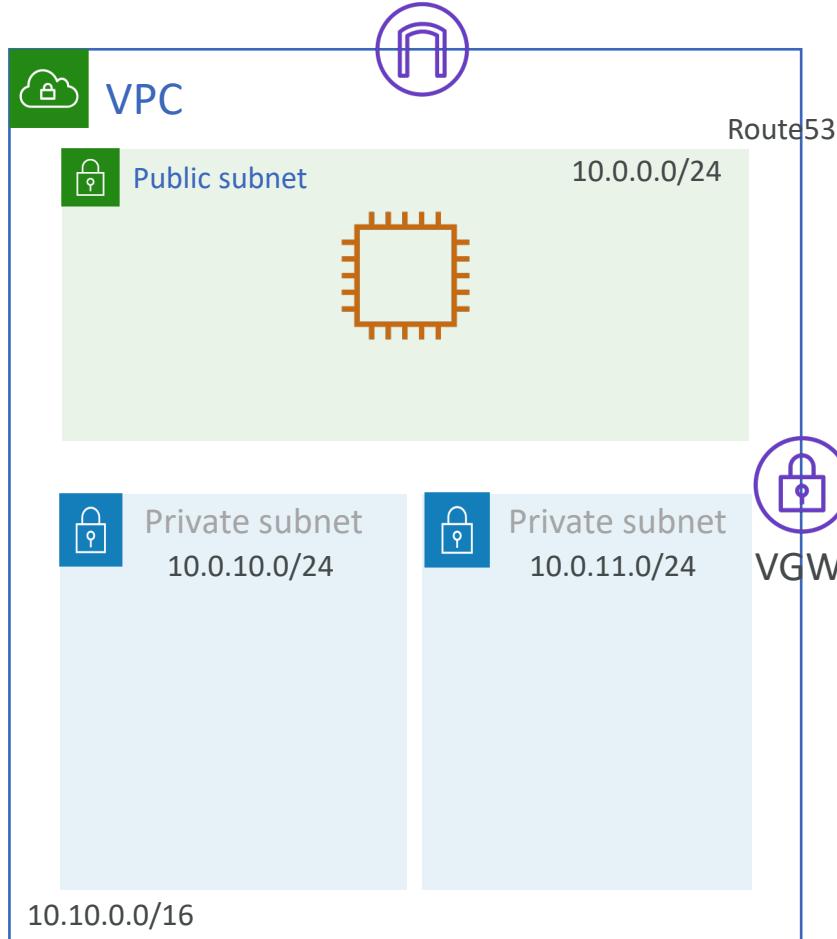
1. Launch EC2 instance in a public subnet (VPN server)
2. Security group of VPN server to allow
  1. SSH from 0.0.0.0/0 (to connect and configure)
  2. ICMP IPv4 from 192.168.0.0/16 (VPN server acts as router)
  3. DNS UDP 53 from 192.168.0.0/16 (VPN server acts as router)
3. Get new EIP and attach EIP to the VPN server

*Note: If VPN handshake is initiated from the other end, then you should have following inbound rules in VPN server firewall. In our case, it's initiated by Openswan server and SG by default has outbound traffic allowed. Due to SG's statefulness, we don't need to open these ports for inbound traffic. Return traffic is allowed.*

1. Custom Protocol 50 (ESP) from VGW Public IPs
2. UDP 500 from VGW Public IPs
3. UDP 4500 from VGW Public Ips if VPN server is behind NAT



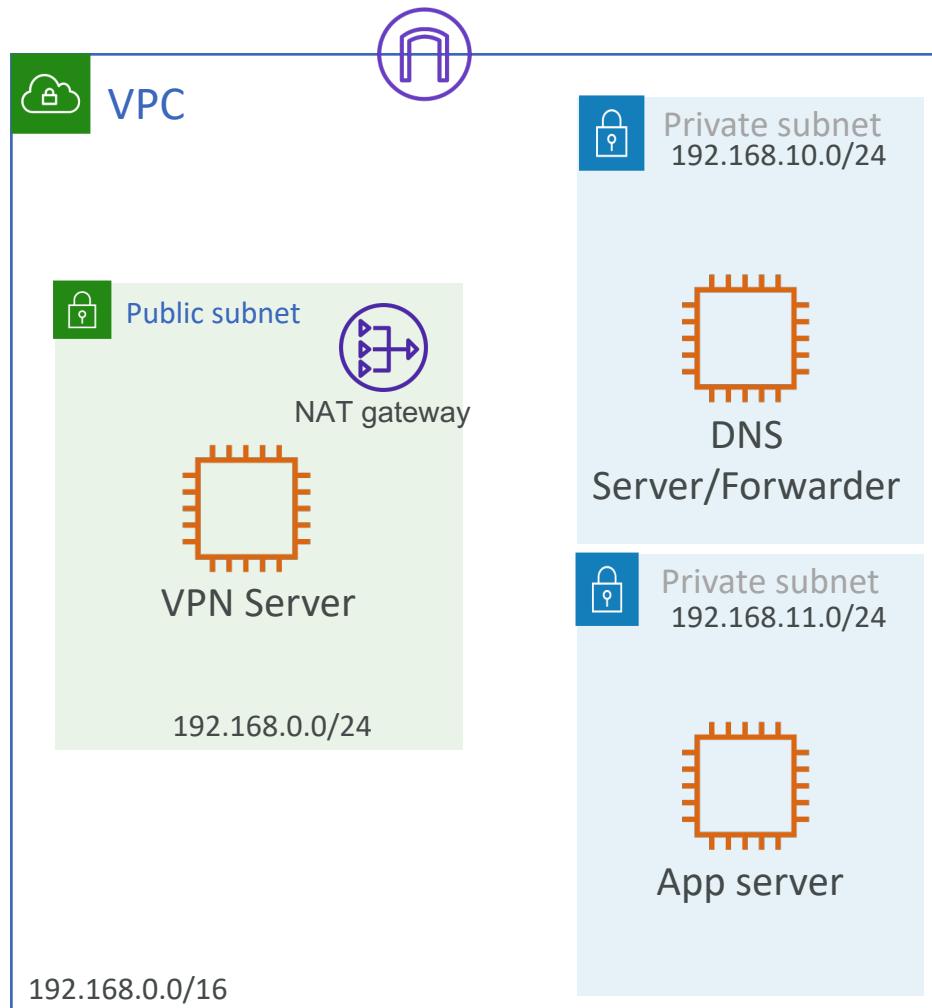
# Step 2b – Create VPN Connection



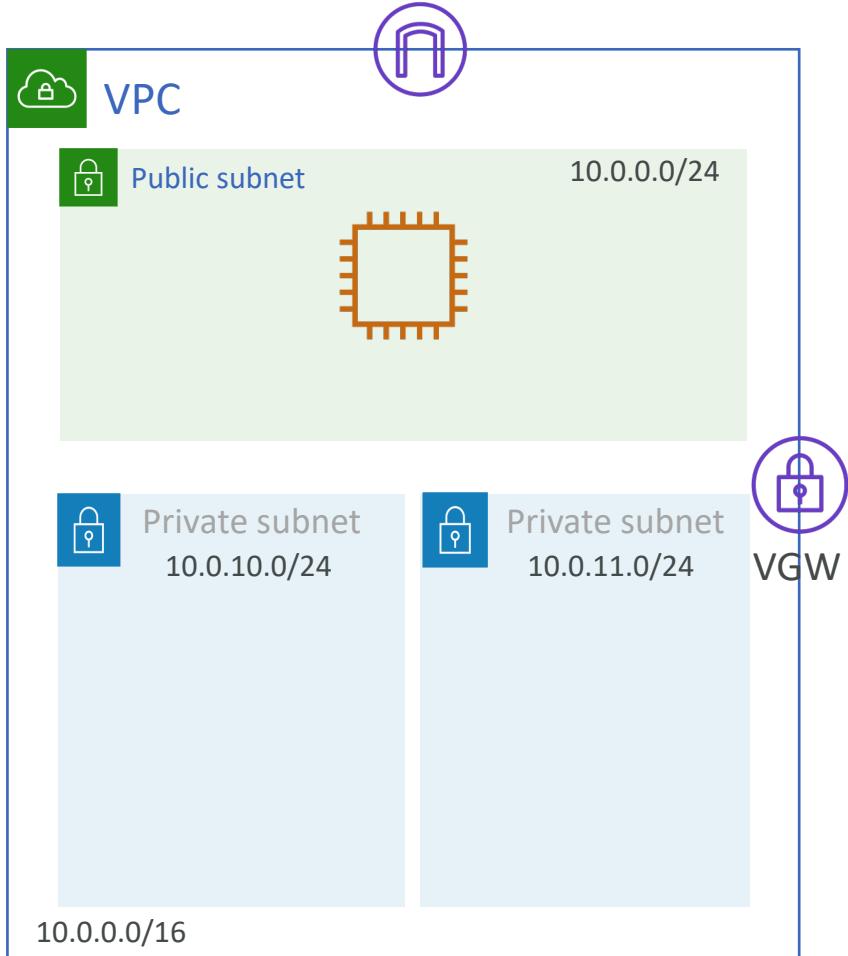
1. Create Virtual Private Gateway and attach to Cloud VPC
2. Create Customer gateway
  1. Use on-premise VPN server EIP
3. Create Site-to-Site VPN connection
  1. Use VGW and CGW created above
  2. Use static routing with IP prefixes – 192.168.0.0/16
  3. Use local IPv4 CIDR as 192.168.0.0/16 (on-prem side) and Remote IPv4 CIDR as 10.0.0.0/16 (Cloud side)
4. Create VPN connection
5. Wait for few minutes and download VPN configuration file for Openswan vendor.

# Step 2c – Configure on-premise VPN server

1. SSH into the VPN server
2. Install openswan: `sudo yum install openswan -y`
3. Open the VPN configuration file you downloaded and follow the instructions in the file to setup **Tunnel 1**
  - ✓ Make sure you remove auth=esp from the configuration
  - ✓ Make sure you change:
    - phase2alg=aes\_gcm
    - ike=aes256-sha1;modp1024
4. Start the IPSec service: `sudo service ipsec start`
5. Check status of IPSec service: `sudo service ipsec status`
6. Go back to AWS console and check the VPN tunnel status – Tunnel 1 should be UP (and Tunnel 2 should be down)
7. Disable Source/Destination check for VPN server

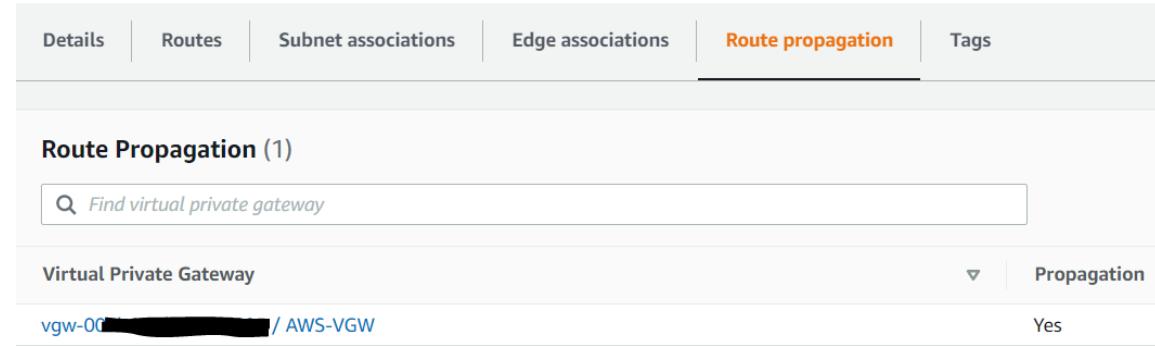


# Step 2d – Configure routes for cloud vpc to route VPN traffic through VPN connection



1. Update the Public and Private Subnets route table to propagate routes from the VGW

*(This should automatically add a route to the route table for destination 192.168.0.0/16)*

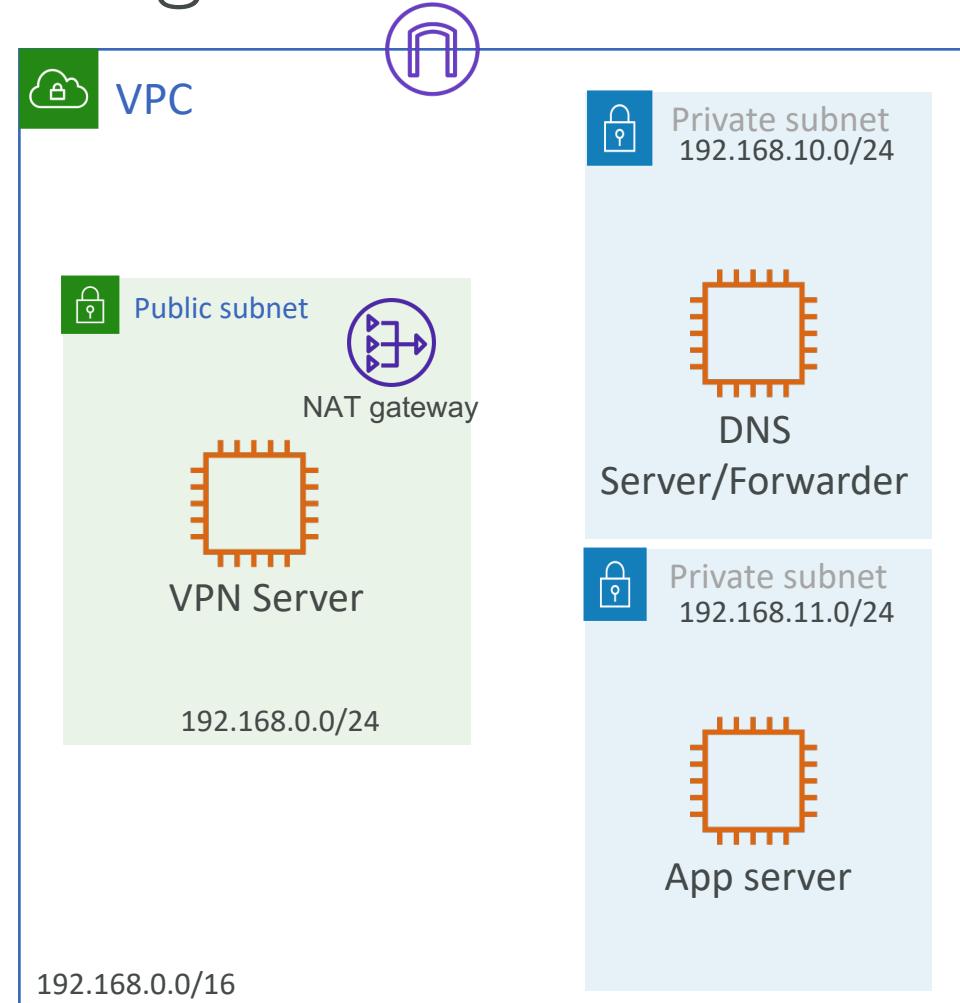


# Step 2e – Configure routes for on-premise network to route VPN traffic through VPN server

1. Update the private subnet route table to route the traffic to 10.0.0.0/16 through the VPN server eni.

| Routes (2)                                   |   |
|--|---|
| <input type="button" value="Filter routes"/> |   |
| Destination                                  | Target  |
| 0.0.0.0/0                                    | eni-Of [REDACTED] <input type="button" value="Edit"/> |
| 192.168.0.0/16                               | local   |

Note: Both private subnets are using the same route table.



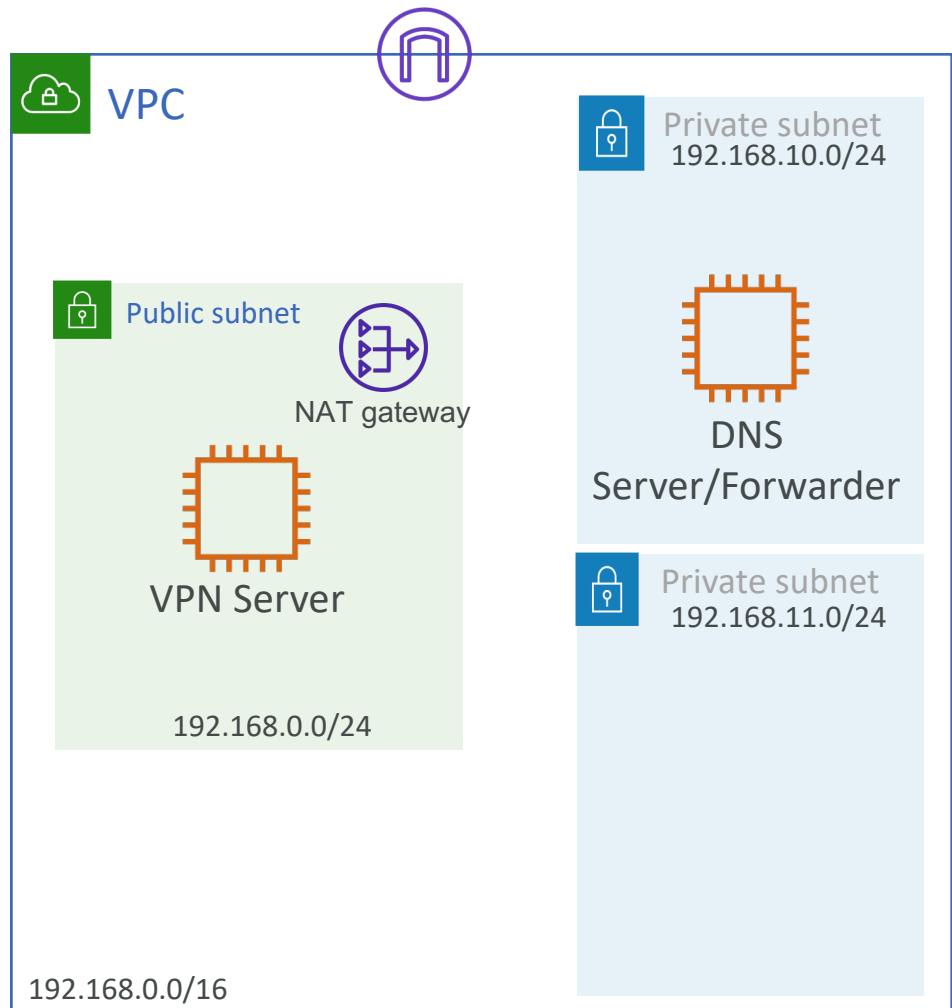
# Step 2f – Verify that VPN is working

- From Cloud EC2 instance – Ping to on-premises App server private IP  
Cloud EC2 -> VGW -> VPN Tunnel | -> VPN server -> App server
- Login to on-premises App server (via SSH into VPN server first and then SSH to app server) – Ping to Cloud EC2 instance private IP  
On-prem App server -> On-prem VPN router -> VPN tunnel | -> VGW -> Cloud EC2
- From on-premises App server ping public website e.g. google.com. Should be able to reach. This traffic goes out through the NAT gateway.  
On-prem App server -> NAT gateway -> public website

**Well done. VPN working fine.**

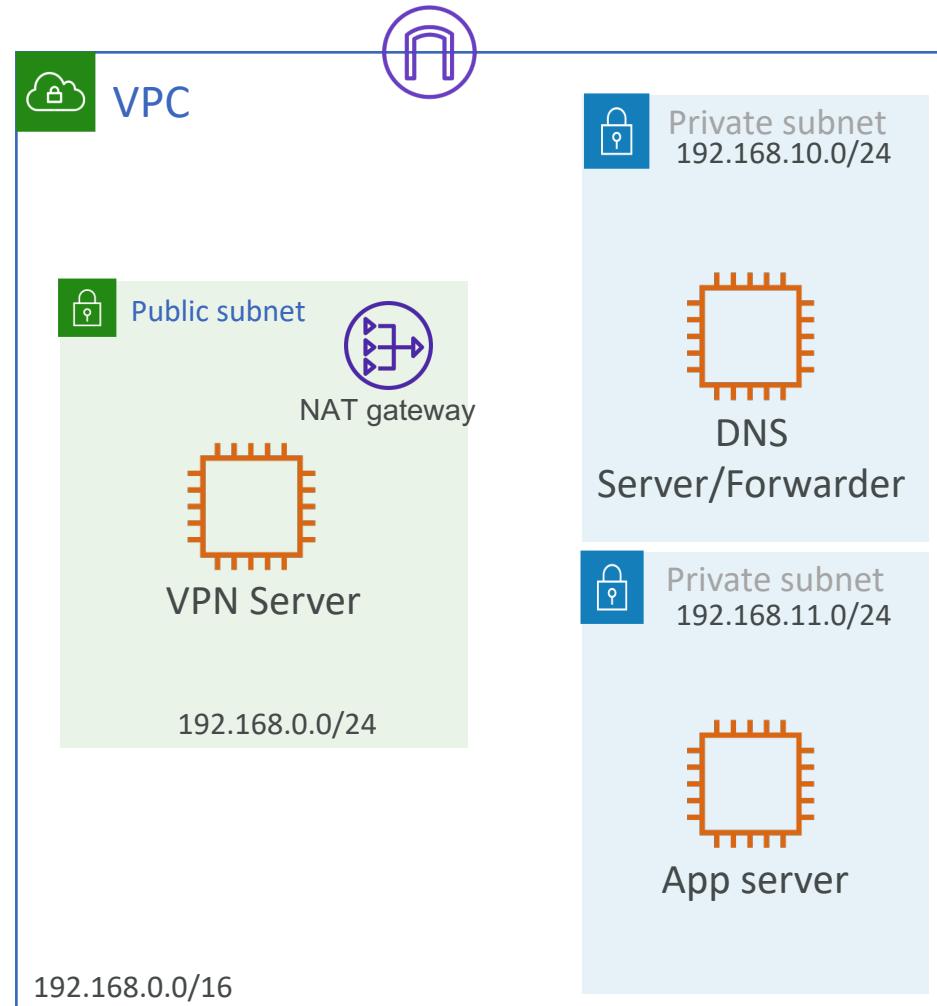
# Step 3a – Setup on-premise DNS server

1. Launch EC2 instance in one of the private subnet
2. Security group of DNS server to allow
  1. SSH (TCP 22) from 192.168.0.0/16 (to connect and configure)
  2. DNS (UDP 53) from 192.168.0.0/16 (to receive DNS queries from within same VPC on-premise servers)
  3. DNS (UDP 53) from 10.0.0.0/16 (to receive DNS queries from cloud servers)



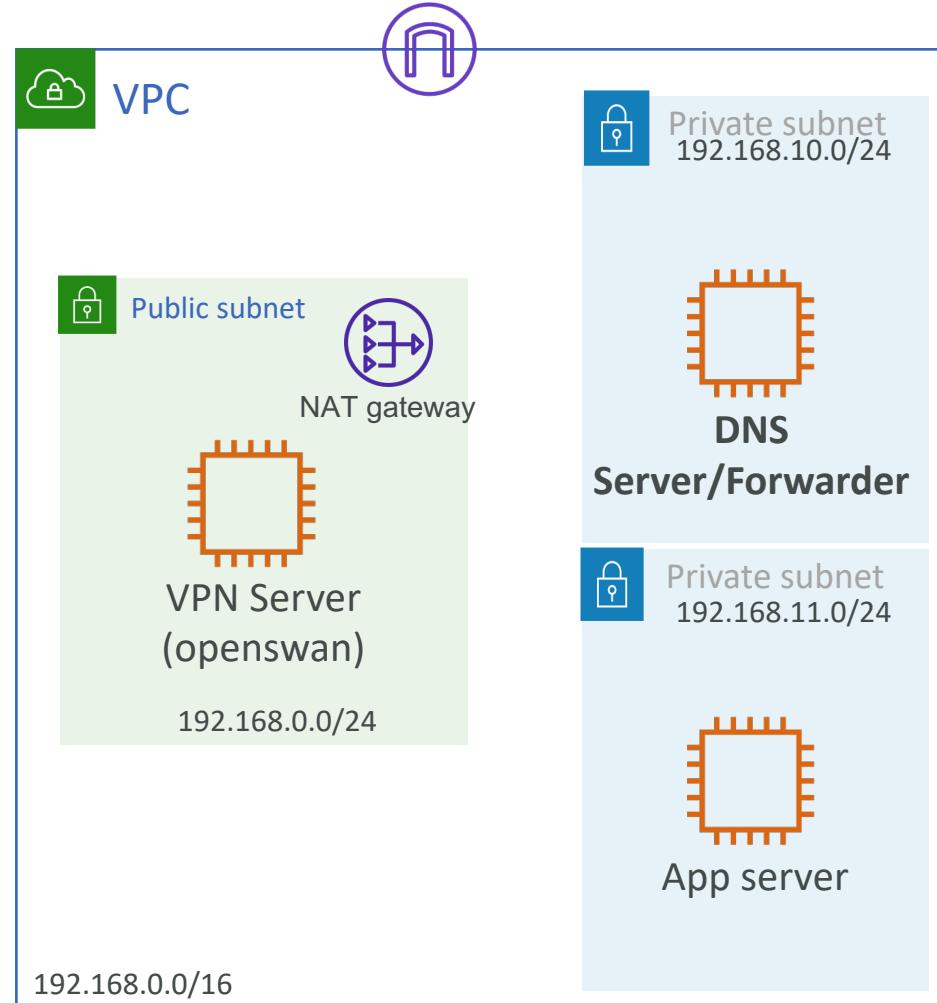
# Step 3b – Setup on-premise App server

1. Launch an EC2 instance in another private subnet
2. Security group of App server to allow
  1. SSH (TCP 22) from 192.168.0.0/16 (to connect and configure)
  2. ICMP IPv4 from 10.0.0.0/16

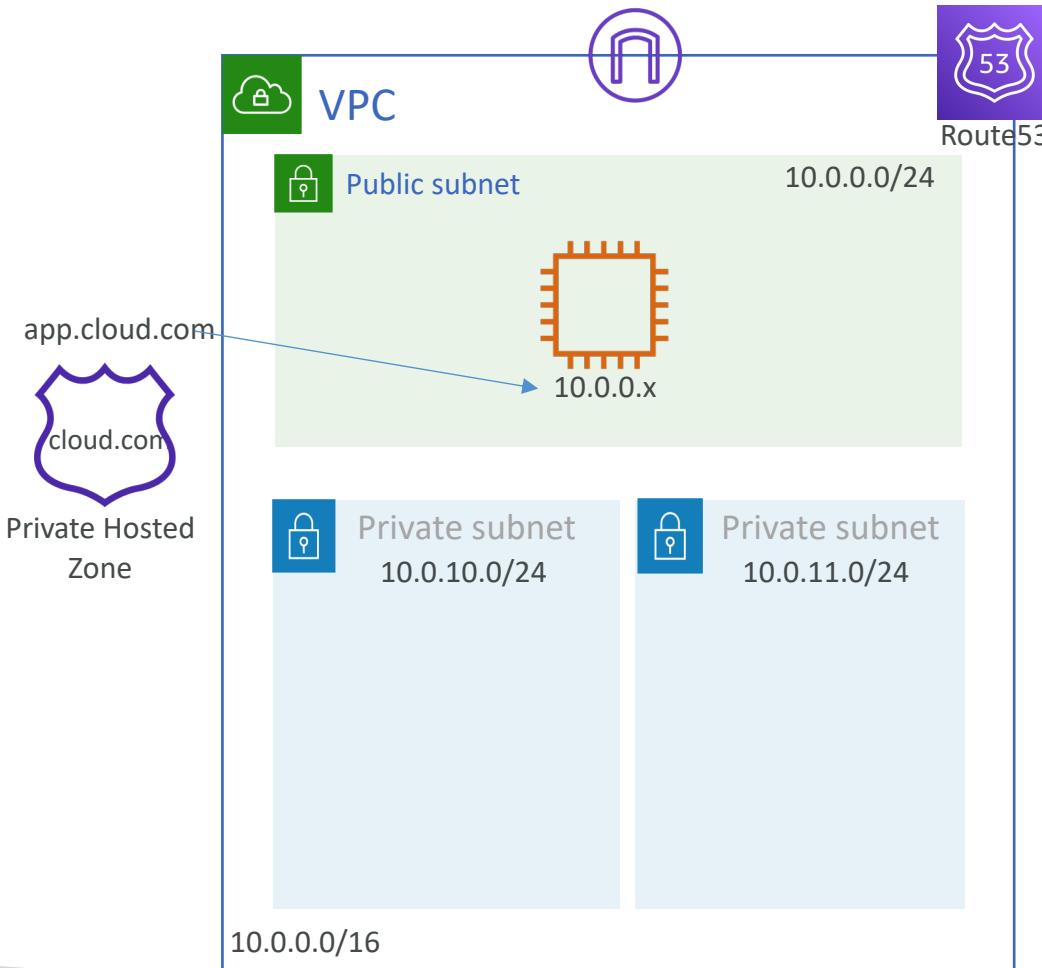


# Step 4 – Configure Cloud and on-premise DNS servers

We will now configure the DNS Server for both cloud and on-premises network



# Step 4a – Setup cloud DNS using Route53 Private hosted zone



1. For Cloud VPC – **Enable DNS resolution and Enable DNS hostname** (required to use Private hosted zone)
2. Create a Private Hosted Zone (cloud.com) and attach to the Cloud VPC
3. Create A record with name app.cloud.com with Cloud EC2 instance private IP address
4. Login to Cloud EC2 instance and verify if the domain name app.cloud.com is resolving to private IP of itself

\$nslookup app.cloud.com

*(may have to wait for couple of minutes)*

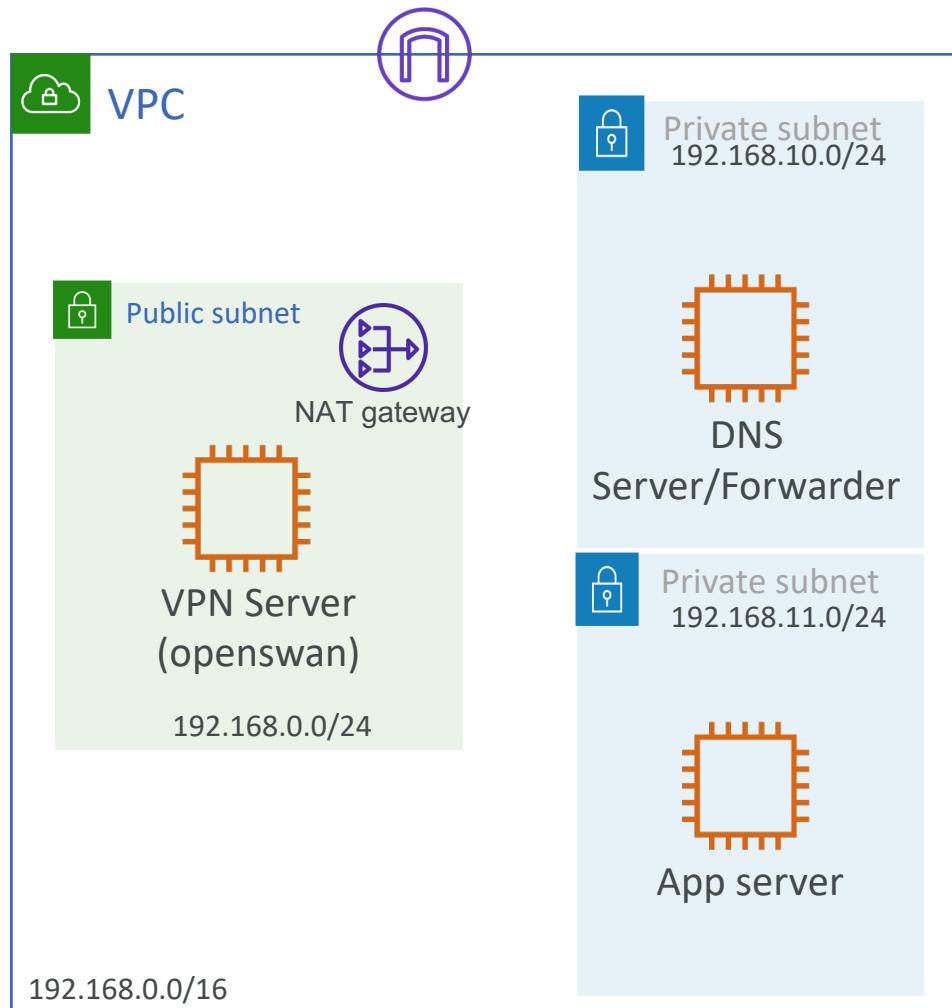
# Step 4b – Setup on-premise DNS server

1. Login to on-premise DNS server (via SSH into VPN server first)
2. Install DNS server packages

```
$sudo su
```

```
$yum update -y
```

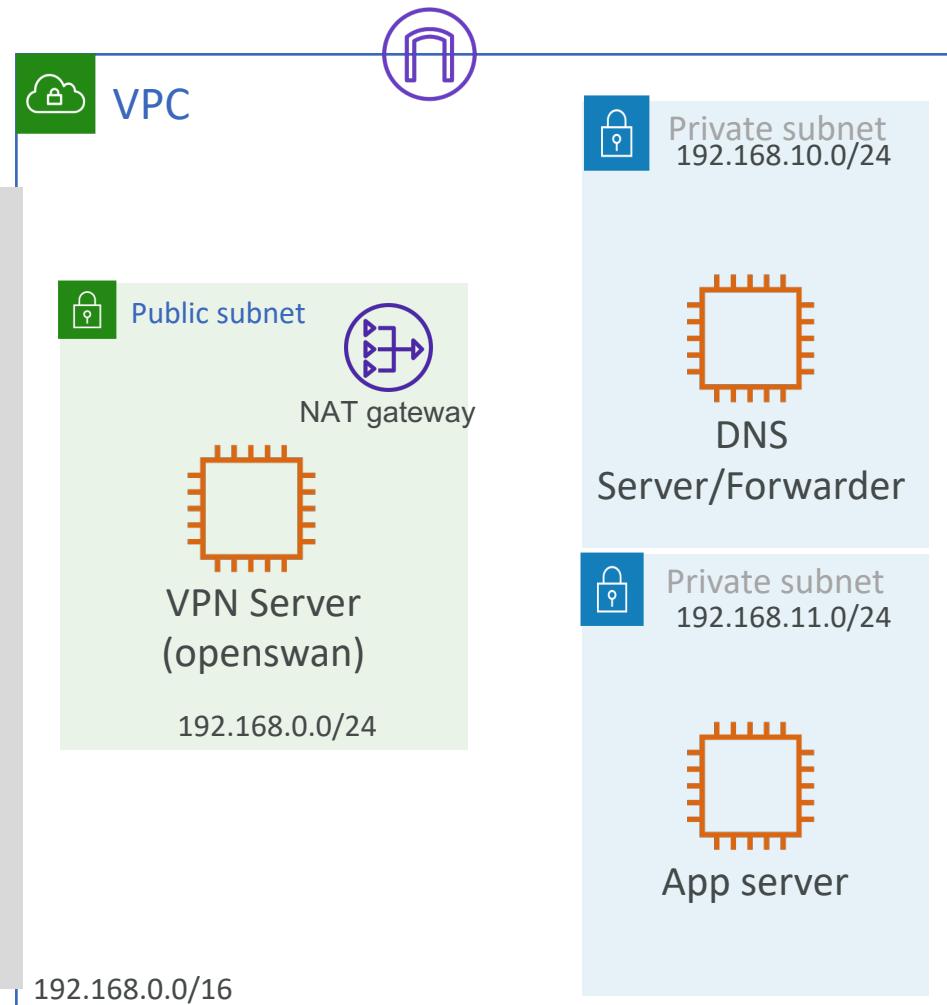
```
$yum install bind bind-utils -y
```



# Step 4b – Configure on-premise DNS server

3. Create file /var/named/onprem.com.zone [Replace X.X with your App server IP]

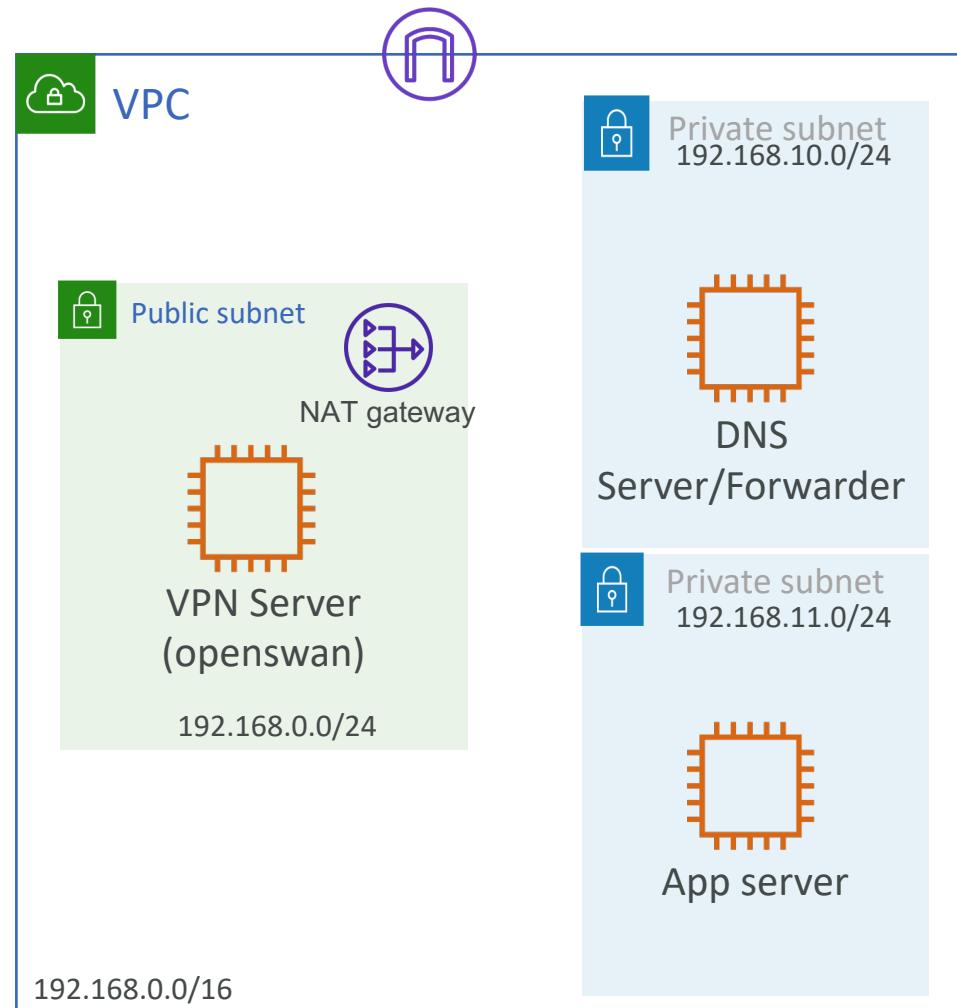
```
$TTL 86400
@
IN SOA ns1.onprem.com. root.onprem.com. (
2013042201 ;Serial
3600 ;Refresh
1800 ;Retry
604800 ;Expire
86400 ;Minimum TTL
)
; Specify our two nameservers
IN NS dnsA.onprem.com.
IN NS dnsB.onprem.com.
; Resolve nameserver hostnames to IP, replace with your two droplet IP addresses.
dnsA
IN A 1.1.1.1
dnsB
IN A 8.8.8.8
; Define hostname -> IP pairs which you wish to resolve
@
IN A 192.168.X.X
app IN
A 192.168.X.X
```



# Step 4b – Configure on-premise DNS server

## 4. Create file /etc/named.conf [Replace X.X with your DNS server IP]

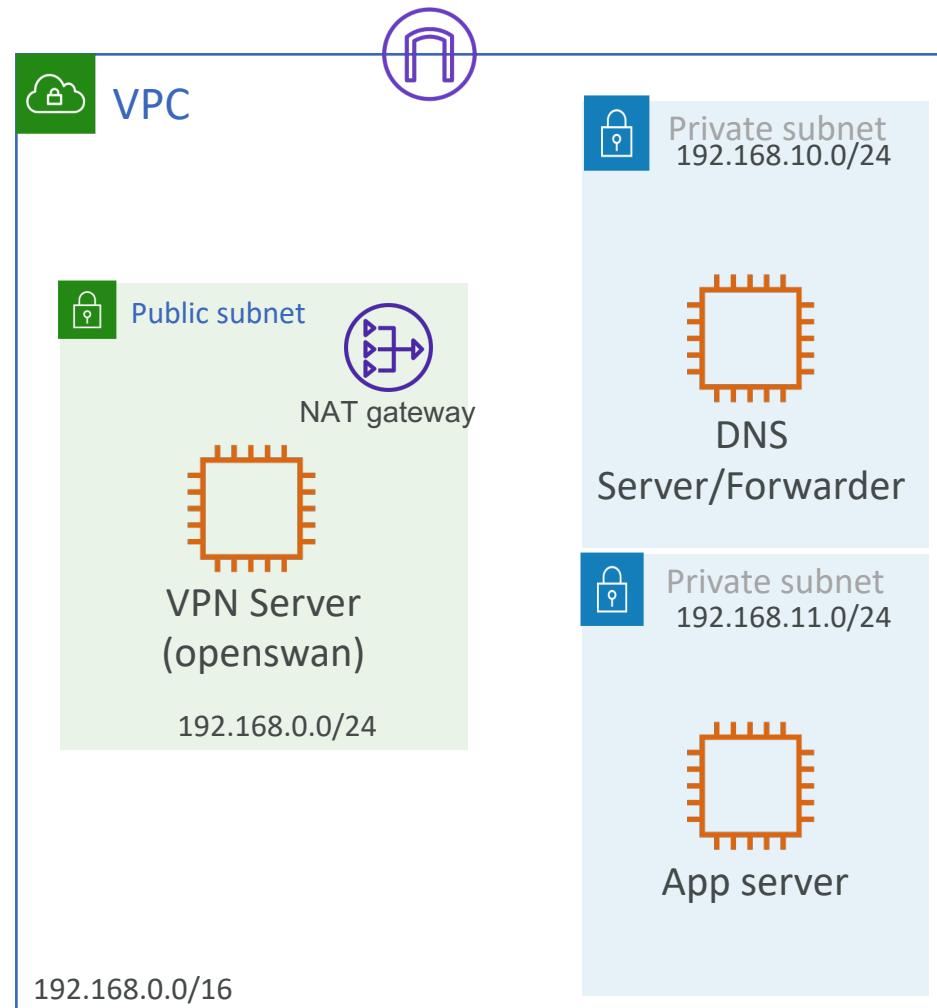
```
options {  
    directory "/var/named";  
    dump-file "/var/named/data/cache_dump.db";  
    statistics-file "/var/named/data/named_stats.txt";  
    memstatistics-file "/var/named/data/named_mem_stats.txt";  
    allow-query { any; };  
    allow-transfer { localhost; 192.168.X.X; };  
    recursion yes;  
    forward first;  
    forwarders {  
        192.168.0.2;  
    };  
    dnssec-enable yes;  
    dnssec-validation yes;  
    dnssec-lookaside auto;  
    /* Path to ISC DLV key */  
    bindkeys-file "/etc/named.iscdlv.key";  
    managed-keys-directory "/var/named/dynamic";  
};  
zone "onprem.com" IN {  
    type master;  
    file "onprem.com.zone";  
    allow-update { none; };  
};
```



# Step 4b – Configure on-premise DNS server

## 5. Restart **named** service

```
$service named restart  
$chkconfig named on
```



# Step 4c – Configure App server to send all DNS requests to DNS server

You need to configure the App server to send the DNS queries to your DNS server

1. Add the DNS server details to /etc/sysconfig/network-scripts/ifcfg-eth0

DNS1=<IP address of on-premise DNS server>

2. Reboot App server

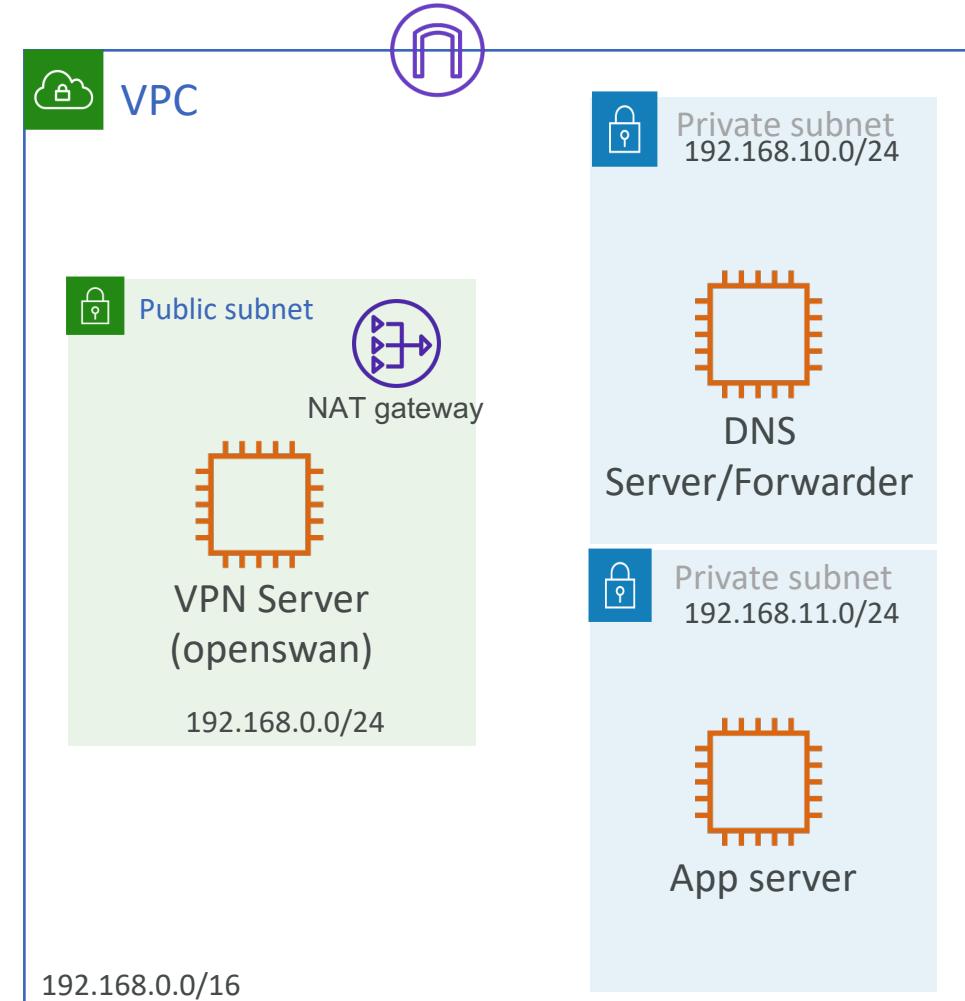
\$sudo reboot

3. Login to App server again and verify that you are able to ping to app.onprem.com

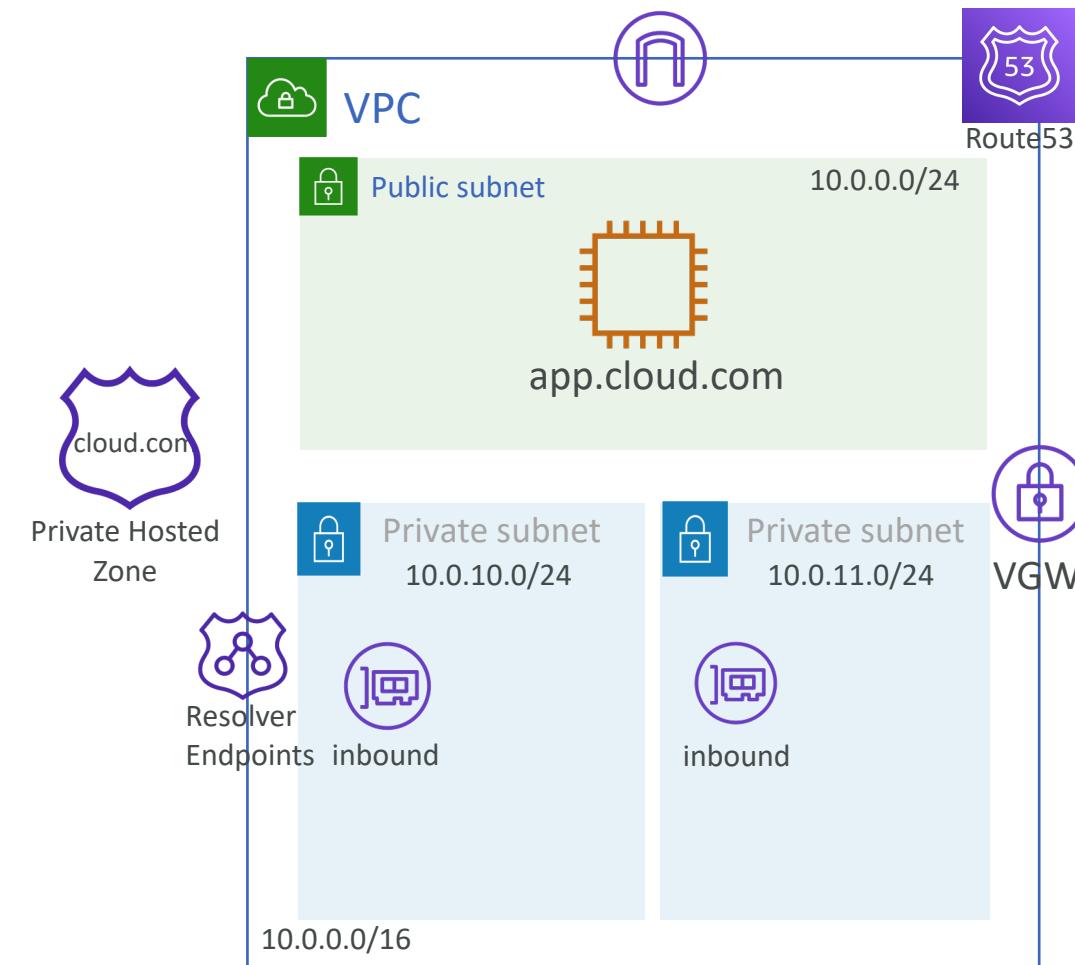
\$ping app.onprem.com

[At this moment, you **can not** ping to app.cloud.com from on-premise app server and vice-a-versa for obvious reason that we haven't configured hybrid DNS so far]

Well done. Separate DNS configurations done for Cloud and On-premises environments.



# Step 5a – Hybrid DNS – From on-premises to AWS



## Create Route53 Resolver inbound endpoint

1. Create a Security group for Route53 resolver inbound endpoint
  - Allow DNS (UDP 53) from on-premise network 192.168.0.0/16
2. Create Route53 Resolver inbound endpoint
  1. Make sure you choose same region
  2. Select Cloud VPC
  3. Select 2 private subnets (across 2 AZs) created earlier.

# Step 5b – Hybrid DNS – From on-premises to AWS

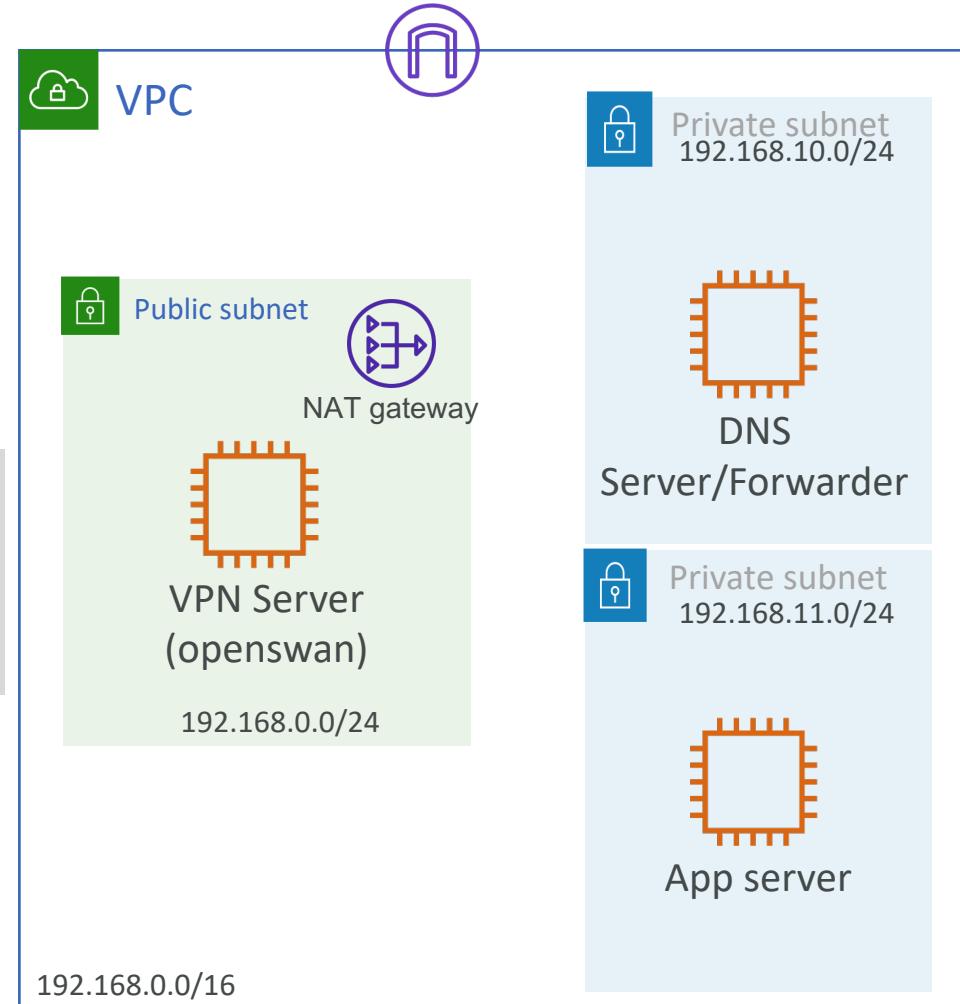
**Configure on-premise DNS server to forward DNS queries for cloud.com to Route53 inbound resolver endpoint IPs**

1. Add following to /etc/named.conf. Replace ENDPOINT IPs with Route53 inbound resolver IPs.

```
zone "cloud.com" {  
    type forward;  
    forward only;  
    forwarders { INBOUND_ENDPOINT_IP1; INBOUND_ENDPOINT_IP2; };  
};
```

2. Restart named service

```
$service named restart
```



# Step 5c – Verify DNS resolution

You should be able to resolve `app.cloud.com` from the on-premise App Server

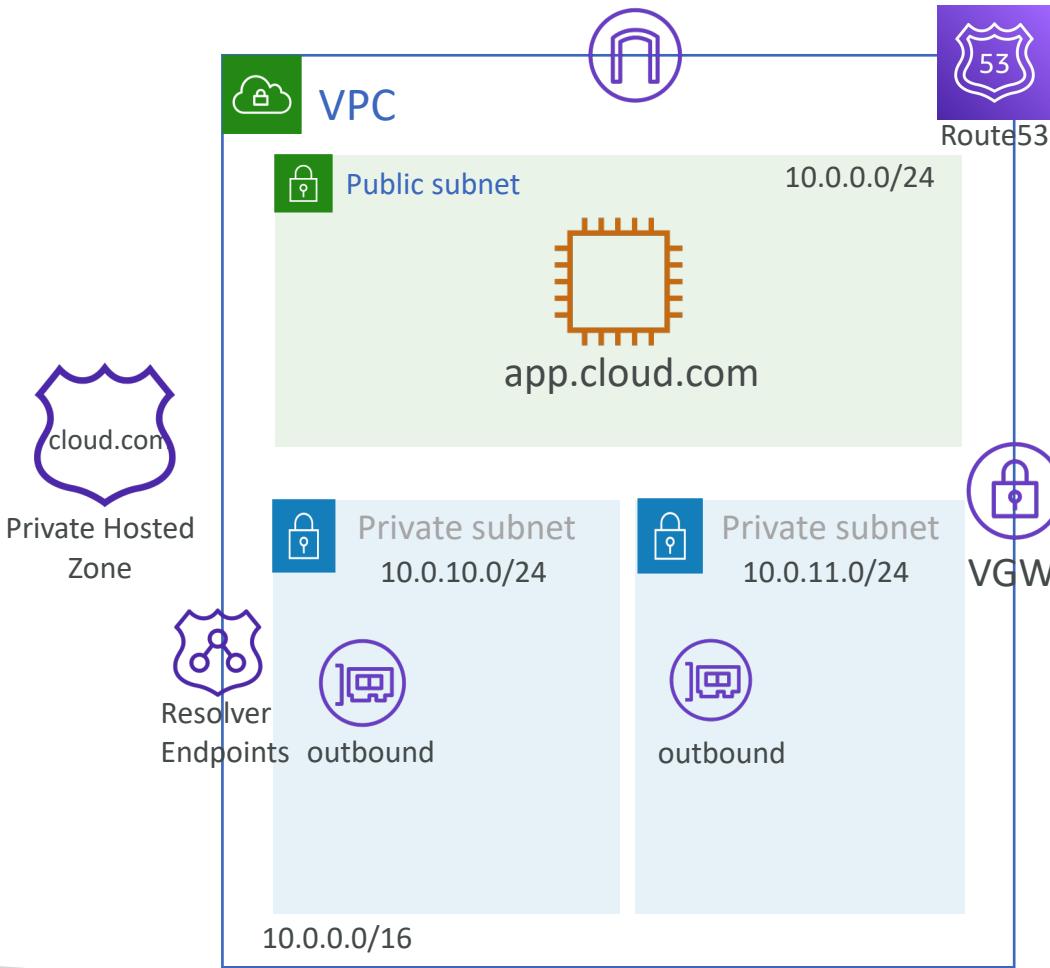
1. From on-premise App Server

```
$nslookup app.cloud.com  
$ping app.cloud.com
```



Well done ! On-premises to AWS DNS resolution working as expected.

# Step 6a – Hybrid DNS – From cloud to on-premises (outbound)



## Create Route53 Resolver outbound endpoint

1. Create a Security group for Route53 resolver outbound endpoint
  - Outbound Rule: Allow DNS (UDP 53) to on-premise network 192.168.0.0/16
2. Create Route53 Resolver outbound endpoint
  1. Make sure you choose same region
  2. Select Cloud VPC
  3. Select 2 private subnets (across 2 AZs) created earlier.
3. Create Outbound Rule to forward the DNS requests for onprem.com to DNS server IP
  1. Domain name as onprem.com
  2. Select Cloud VPC
  3. Target IP address of on-premises DNS server IP (192.168.X.X)

# Step 6b – Verify DNS resolution

You should be able to resolve app.onprem.com from the Cloud EC2 instance

1. From Cloud EC2 instance

```
$nslookup app.onprem.com  
$ping app.onprem.com
```



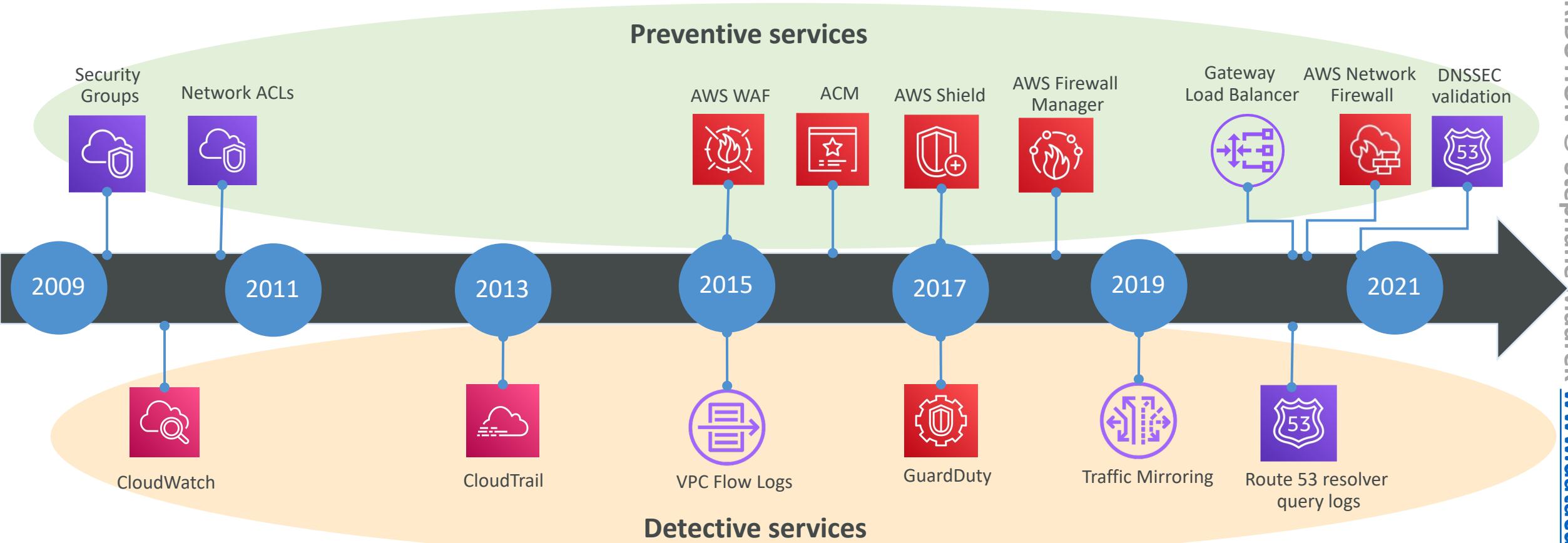
Amazing ! AWS to on-premises DNS resolution is also working!

# Step 7 - Cleanup

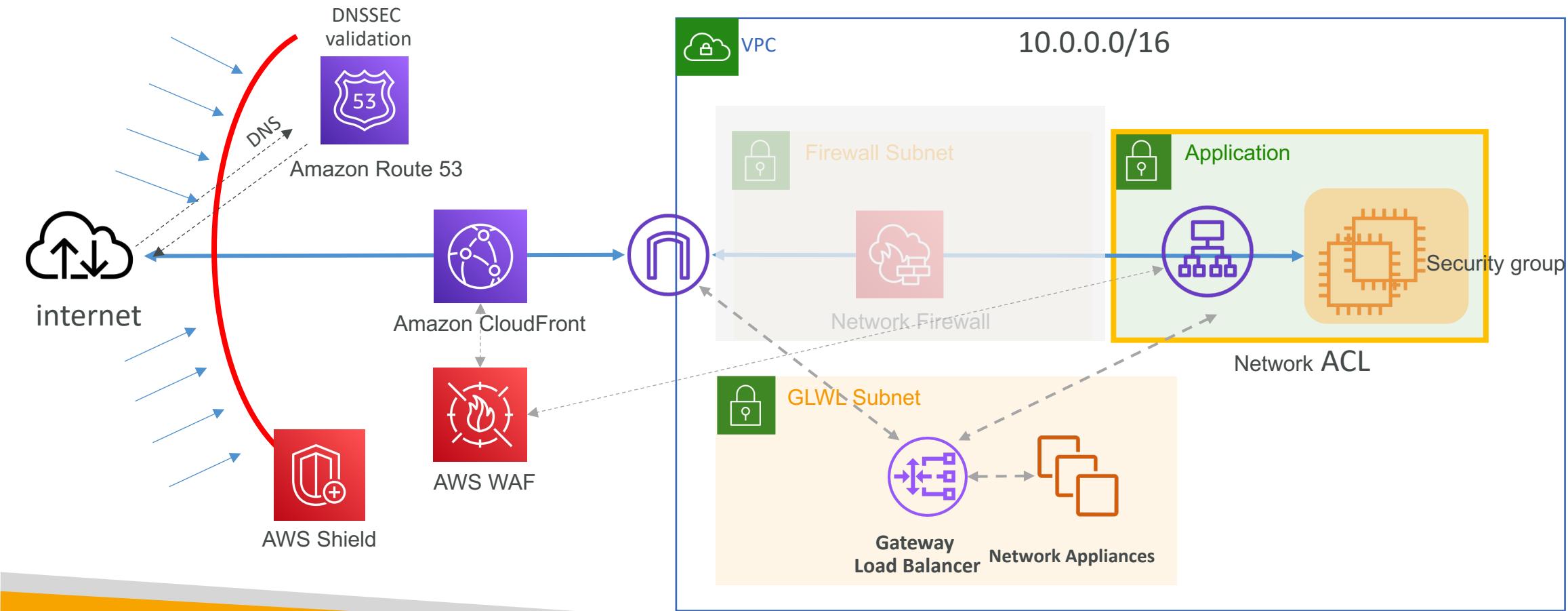
- Terminate all EC2 instances created in this lab.
- Delete VPN Connection, VGW and CGW
- Delete NAT Gateway
- Delete Route53 resolver rules followed by resolver endpoints
- Delete Route53 A record followed by a private hosted zone
- Delete both VPCs

# AWS Network Security

# AWS Network Security services



# Security at different layers

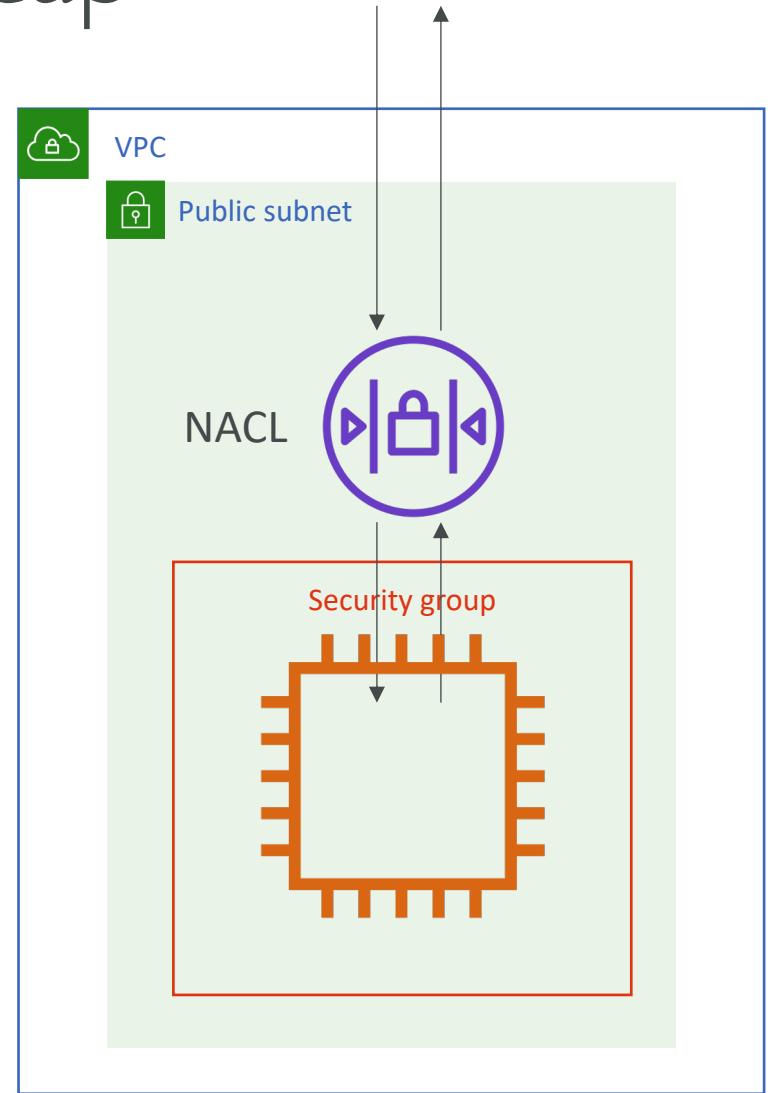


# AWS Firewall services

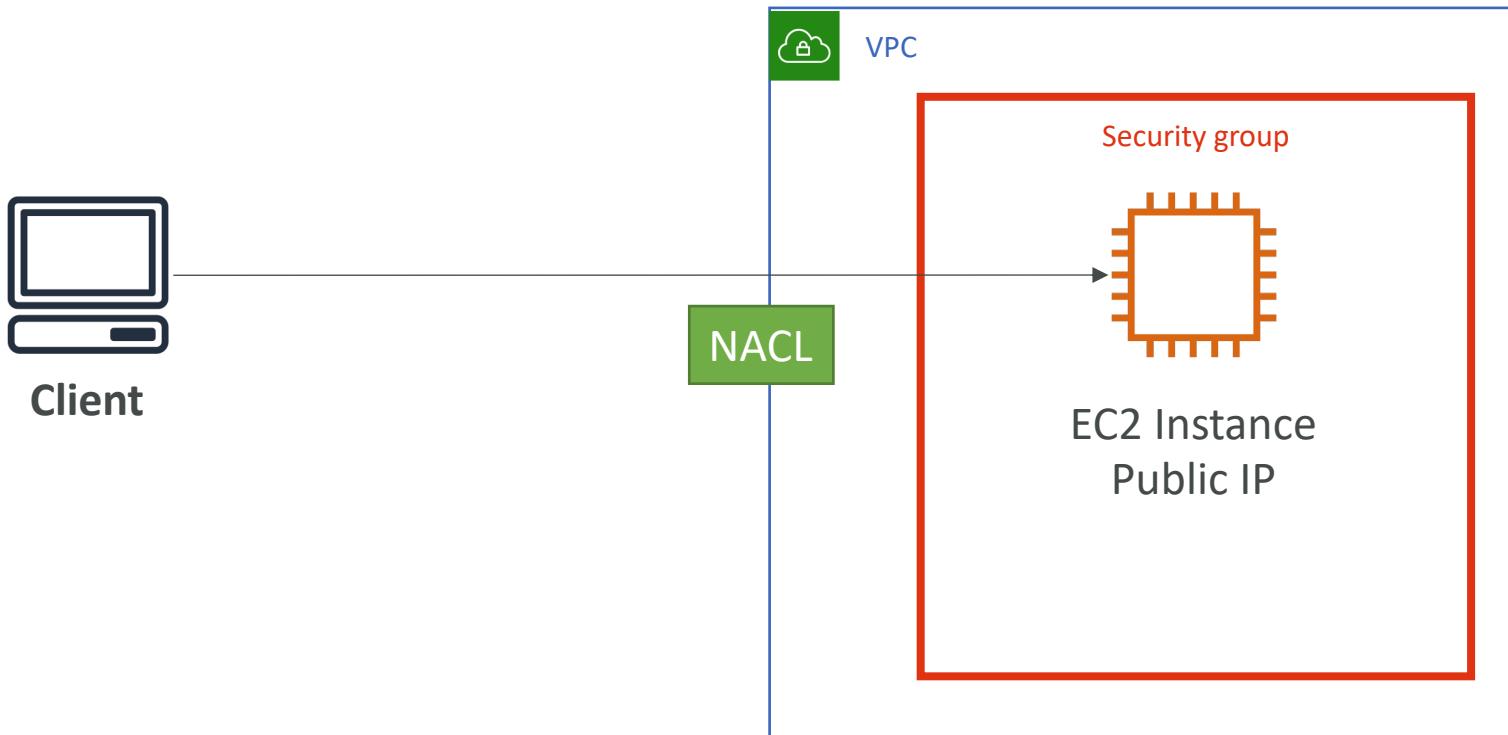
# Recap – Security Group and Network ACL

# Security Group and NACL recap

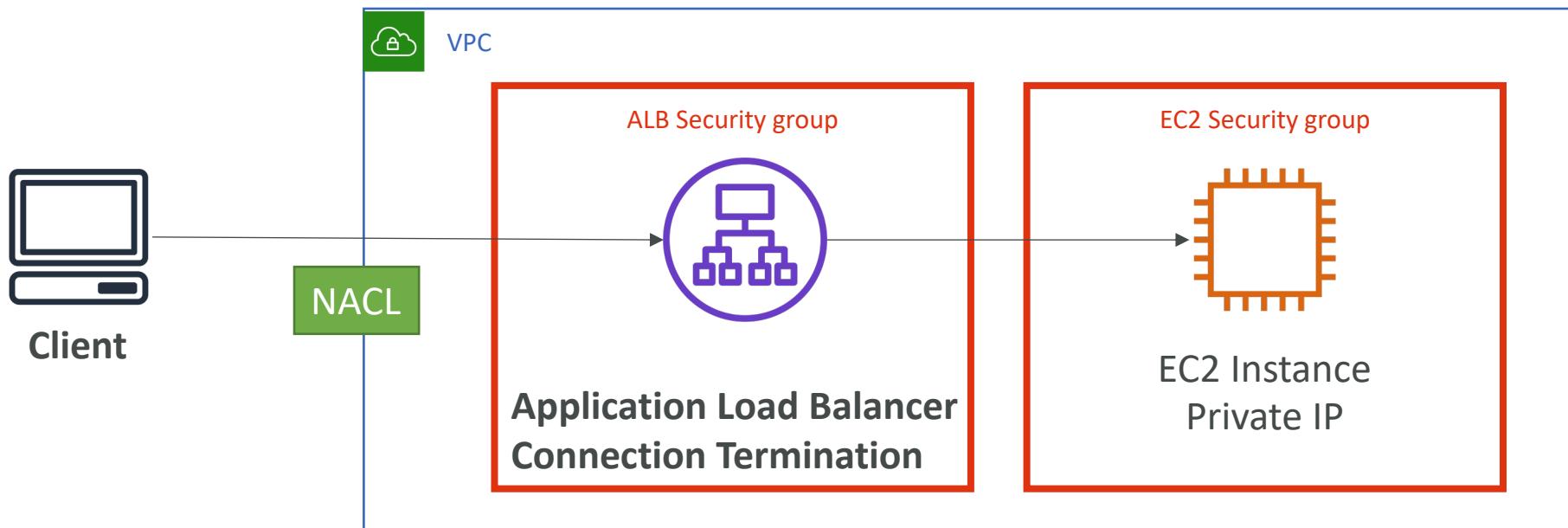
- **Security Groups**
  - Attached to ENI (Elastic Network Interfaces) – EC2, RDS, Lambda in VPC, etc
  - Are stateful (any traffic in is allowed to go out, any traffic out can go back in)
  - Can reference by CIDR and security group id
  - Supports security group references for VPC peering
  - Default: inbound denied, outbound all allowed
- **NACL (Network ACL):**
  - Attached at the subnet level
  - Are stateless (inbound and outbound rules apply for all traffic)
  - Can only reference a CIDR range (no hostname)
  - Default: denies all inbound, denies all outbound



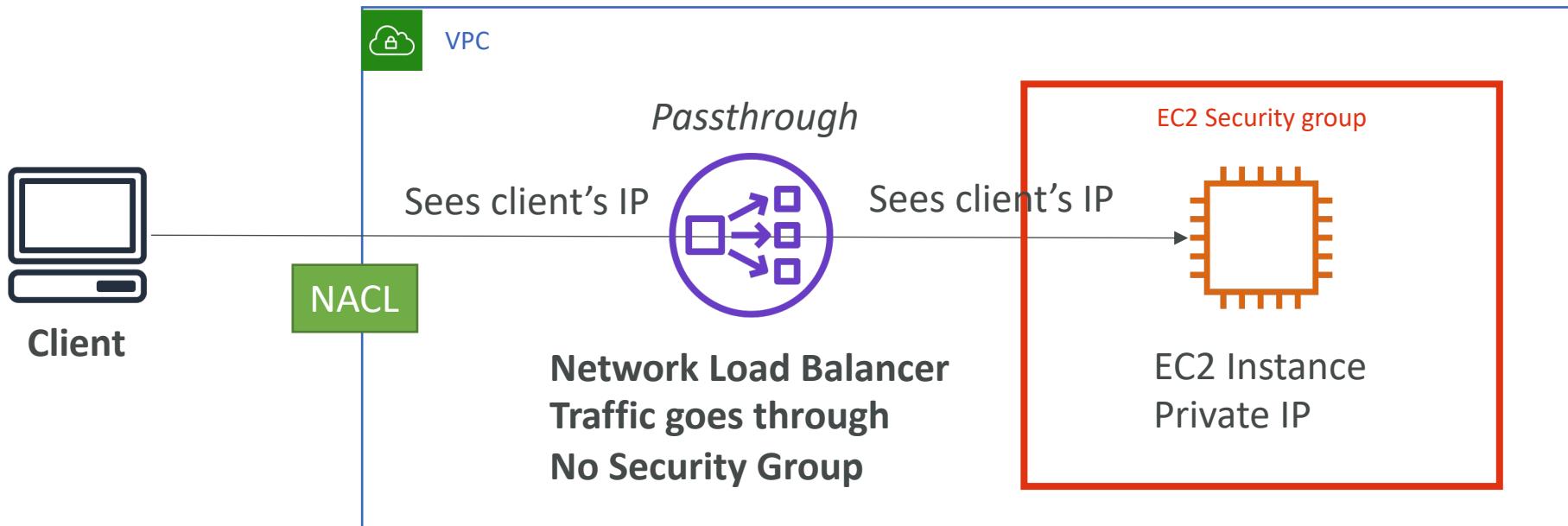
# Blocking an IP address



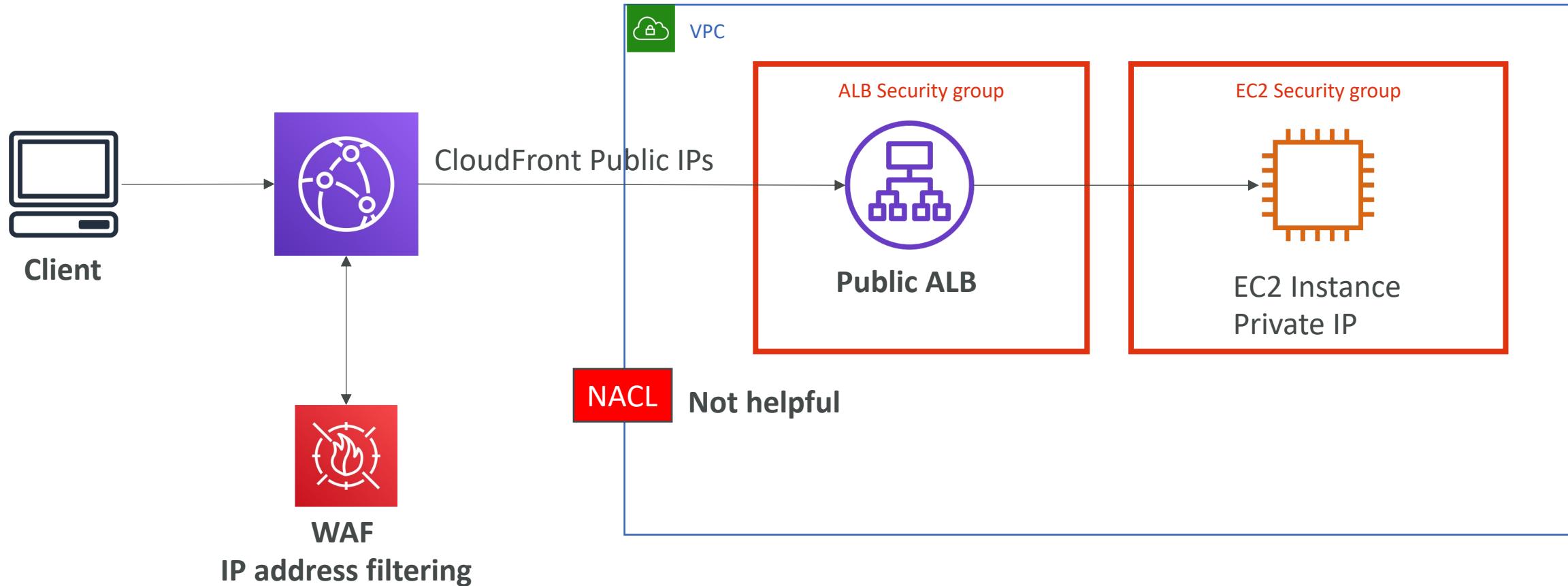
# Blocking an IP address – with an ALB



# Blocking an IP address – with an NLB

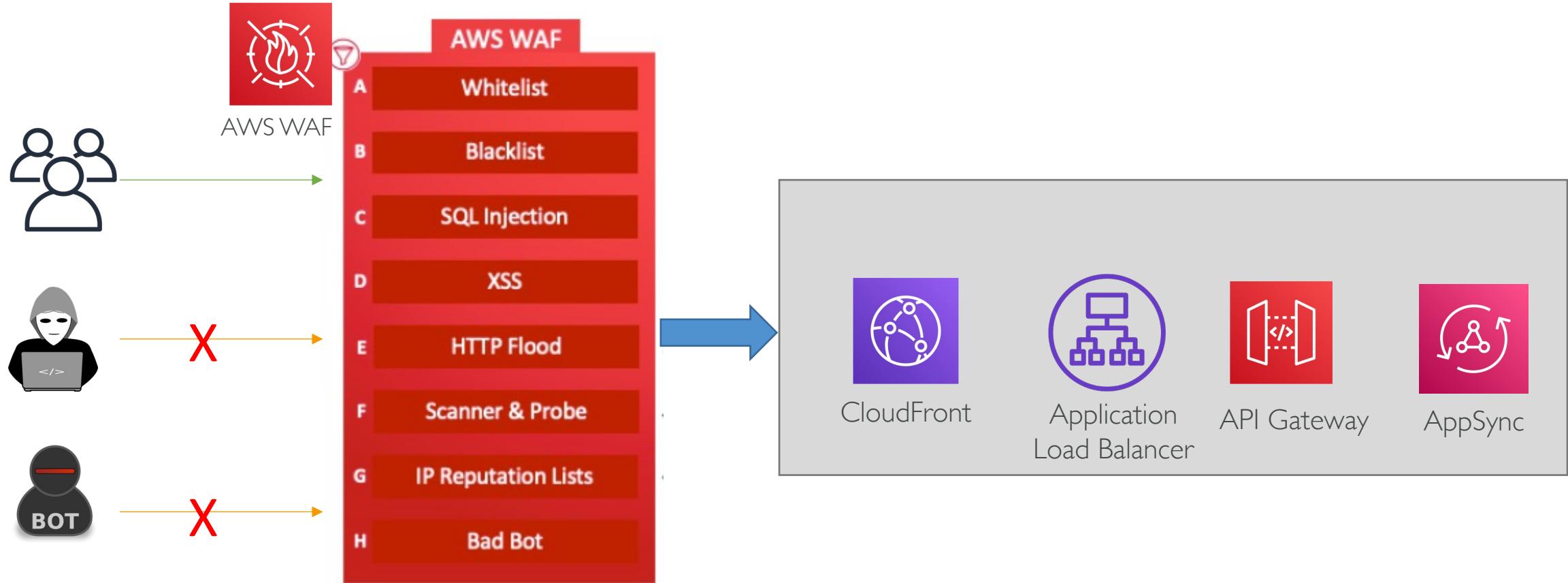


# Blocking an IP address - CloudFront



# Web Application Firewall (WAF)

# Amazon Web Application Firewall (WAF)



# AWS WAF – Web Application Firewall



- Protects web applications from common web exploits (Layer 7)
- Deploy on **Application Load Balancer**, **API Gateway** or **AppSync GraphQL APIs** and **CloudFront** (rules running globally on edge locations)
- Define Web ACL (Web Access Control List) with which you can allow, block or monitor web requests by inspecting IP addresses, HTTP headers, HTTP body, or URI strings, Size constraints, Geo match, String match etc.
- Web ACLs also support Rate-based rules (to count occurrences of events)
- When you evaluate the body of a request, only the first 8,192 bytes (8KB) are inspected.
- On blocking the malicious traffic WAF returns HTTP 403 status code (Forbidden)
- It takes less than a minute for the WAF rules to propagate worldwide

# Amazon Web Application Firewall (WAF)

Web ACL regular rules support following conditions

1. **Cross-site Scripting (xss)** : Searched in HTTP method, header, query string, Uniform Resource Identifier (URI), or body
2. **IP addresses**: Allows you to match IPv4 and IPv6 addresses
3. **Size**: Allows you to match request on basis of length for common parts of request data
4. **SQL Injection (SQLi)**: Can be applied on common parts of request data.
5. **Geographic match**: Allow or block requests based on the country from which the request originates
6. **String/Regex match**: Allows to match common parts of request data based on string content including regular expression

# WAF Labs – XSS example

## I. Deploy simple application which is susceptible to the XSS attack

- Launch an EC2 instance and install httpd (\$sudo yum install httpd -y)
- Install git (\$sudo yum install git -y)
- Clone the xss-example repo & copy the content into /var/www/html/ dir
  - \$git clone <https://github.com/academind/xss-example.git>
- Start the web server (\$sudo service httpd start)
- Open the web browser and connect to EC2 Public IP.
- In the image field enter something like:

example.com/any-image.jpg" onerror="alert('This computer is hacked')"

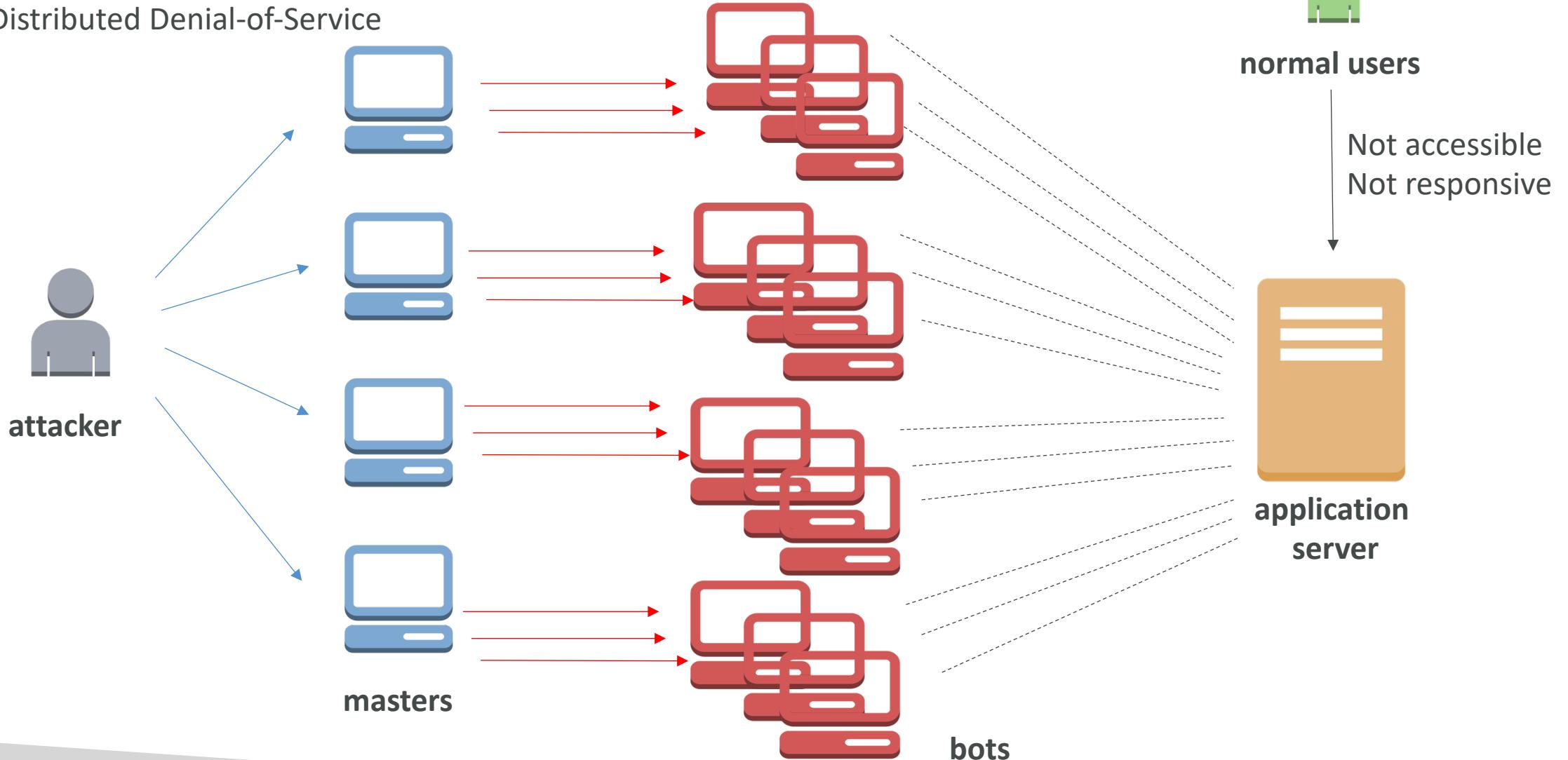
# WAF Labs – Use WAF to prevent XSS

1. Use the existing EC2 instance but this time we will have simple hello web page
2. Just backup the existing index.html file under /var/www/html dir and create another index.html file with simple HTML text
3. Create ALB and add attach EC2 instance in the target group. Make sure instance shows healthy.
4. Create WAF Web ACL and enable the core rules. Associate with ALB.
5. Now try to access the website using EC2 Public IP followed by some XSS script  
*http://x.x.x.x/<script>alert('Hacked')</script>*
6. You should get 403 Forbidden error. Also check the WAF console, it should show the matched rule for XSS.

# AWS Shield

# What's a DDOS attack?

\*Distributed Denial-of-Service

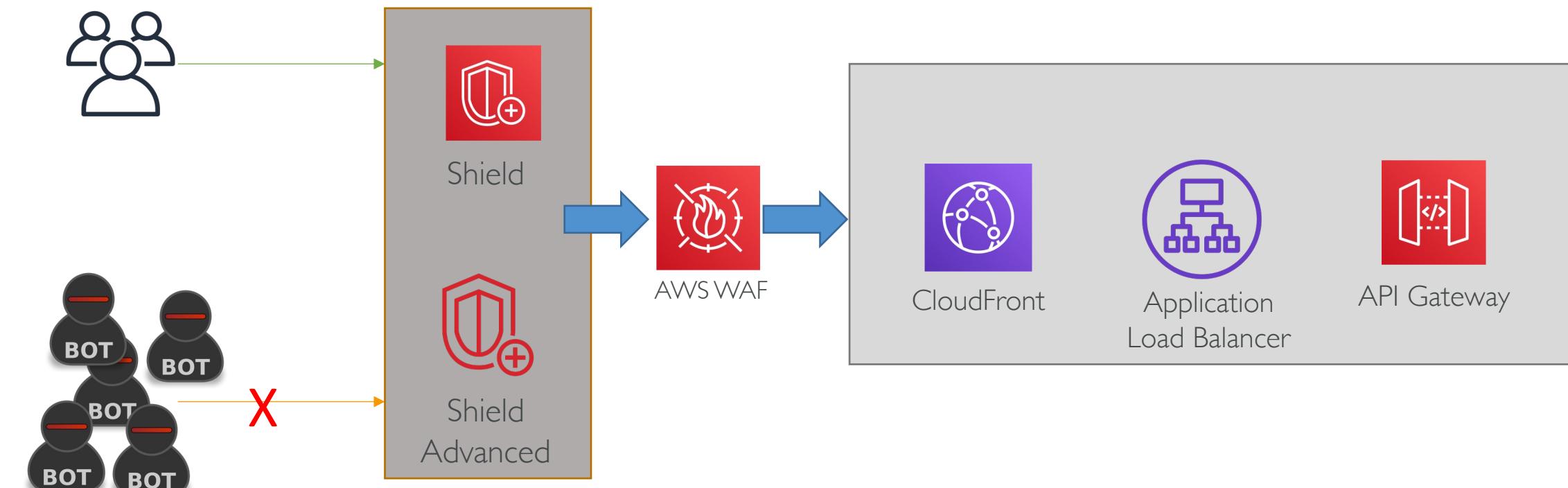


# Common DDoS attacks

- SYN Flood attack: Too many half open TCP connections
- UDP Flood attack: Too many UDP requests
- UDP Reflection attack: Spoof the victim server IP as a source for UDP packet. Victim server receives the unexpected responses.
- DNS Flood attack: Overwhelm the DNS so legitimate users can't find the site
- Slow Loris attack (Layer7): A lot of HTTP connections are opened and maintained
- Cache Busting attack: Request un-cached data from CDN

# AWS Shield

- Provides protection against DDoS attack at Network Layer (layer3) and Transport Layer (layer4)



# AWS Shield



- AWS Shield Standard:
  - Free service that is activated for every AWS customer
  - Provides protection from attacks such as SYN/UDP Floods, Reflection attacks and other layer 3/layer 4 attacks
- AWS Shield Advanced:
  - Optional DDoS mitigation service (\$3,000 per month per organization)
  - Protect against more sophisticated attack on [Amazon EC2](#), [Elastic Load Balancing \(ELB\)](#), [Amazon CloudFront](#), [AWS Global Accelerator](#), and [Route 53](#)
  - 24/7 access to AWS Shield Response Team (SRT)
  - Protect against higher fees during usage spikes due to DDoS

# AWS Shield Advanced

- Provides additional DDoS protection for Route53 hosted zones, CloudFront, ELB and resources attached to Elastic IPs like EC2, NAT
- Also provides intelligent DDoS attack detection and mitigation for application layer (Layer 7) attacks
- Customer receives attack forensic report and they can also contact 24x7 AWS Shield Response Team (SRT) for assistance during an attack.
- To use the services of the Shield Response Team (SRT), you must be subscribed to the [Business Support plan](#) or the [Enterprise Support plan](#).
- Get AWS credits\* against Route53, CloudFront, ELB and EC2 in case of billing spike due to DDoS
- You also get access to AWS WAF for free and you can work with AWS SRT team to identify layer 7 attacks/patterns and allow them to deploy those on WAF (you must grant IAM role to SRT team)

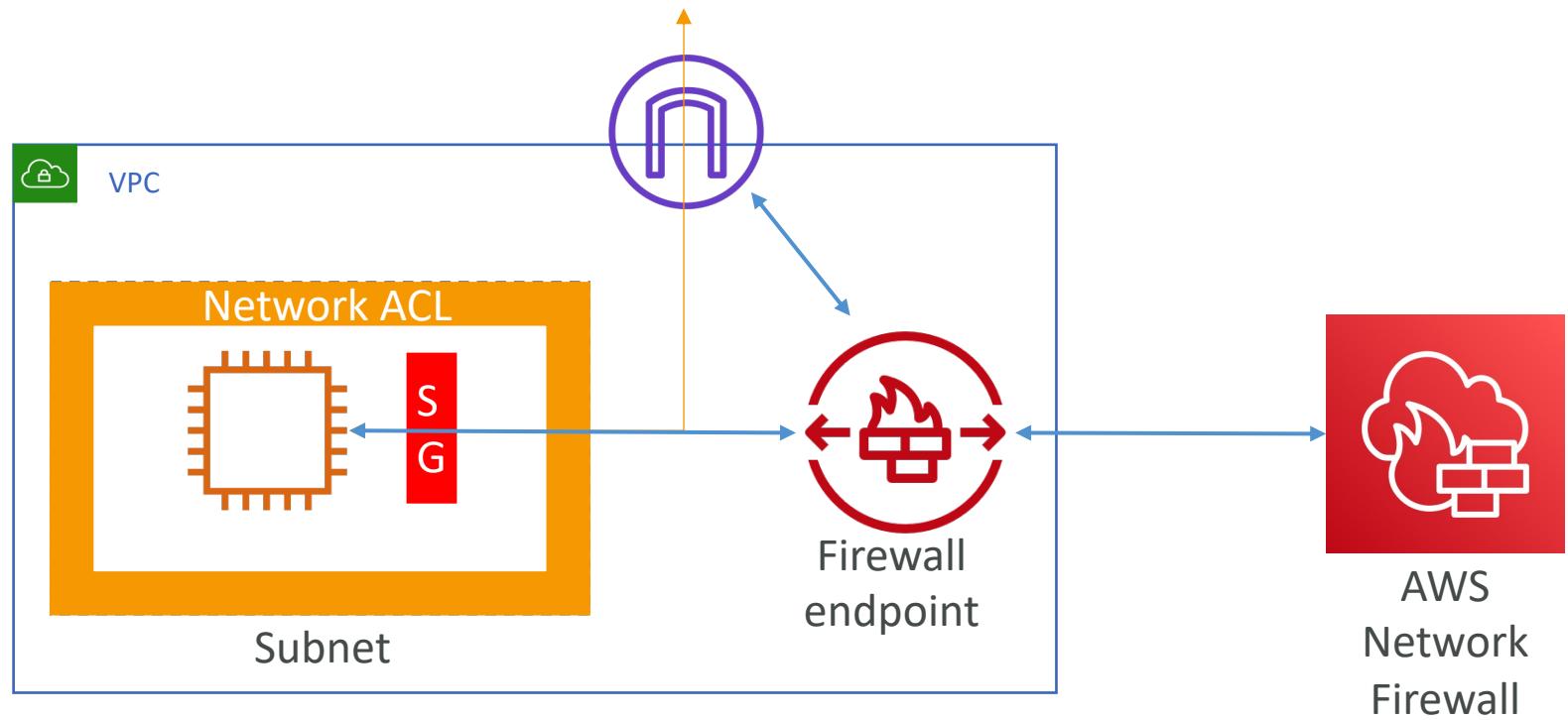
# Additional levers for DDoS protection

- Use CloudFront and Route 53
  - Edge locations provides the built in protection for network layer3 and layer4 traffic
  - The infrastructure is continuously analyzing traffic for anomalies, and it provides built-in defense against common DDoS attacks
  - Amazon Route 53 uses shuffle sharding and anycast striping to isolate availability challenges and mitigate impact
- Be ready to scale – leverage AWS Auto Scaling
- Separate static resources (S3/CloudFront) from dynamic ones (EC2/ALB)
- Read the whitepaper:  
[https://dl.awsstatic.com/whitepapers/Security/DDoS\\_White\\_Paper.pdf](https://dl.awsstatic.com/whitepapers/Security/DDoS_White_Paper.pdf)

# AWS Network Firewall

# What is AWS Network Firewall?

A stateful network firewall and intrusion detection and prevention service for VPC



# SG, NACL, WAF ... so why this new service?

|                              | <b>Security Group</b>                      | <b>Network ACL</b>                       | <b>WAF</b>   | <b>AWS Network Firewall</b>   |
|------------------------------|--|--|--|---|
| <b>Protection at</b>         | EC2 instance level                         | Subnet level                             | Endpoint level (ALB, CloudFront etc)                       | VPC level based on routes   |
| <b>Stateful or Stateless</b> | Stateful                                   | Stateless                                | Stateless  | Both  |
| <b>OSI layer</b>             | Layer3/4                                   | Layer3/4                                 | Layer7   | Layer3-7  |
| <b>Features</b>              | IP, Port, Protocol filtering               | IP, Port, Protocol filtering             | Application layer filtering                                | Stateless/ACL L3 rules, stateful/L4 rules, IPS-IDS/L7 rules, FQDN filtering, Protocol detection, Large IP block/allow lists |
| <b>Flows</b>                 | All ingress/egress flows at instance level | All ingress/egress flows at subnet level | Ingress only from internet to API Gateway, ALB, CloudFront | All ingress/egress flows at perimeter of VPC (e.g. IGW, VGW, DX, VPN, VPC-VPC)  |

# Stateful vs Stateless firewalls

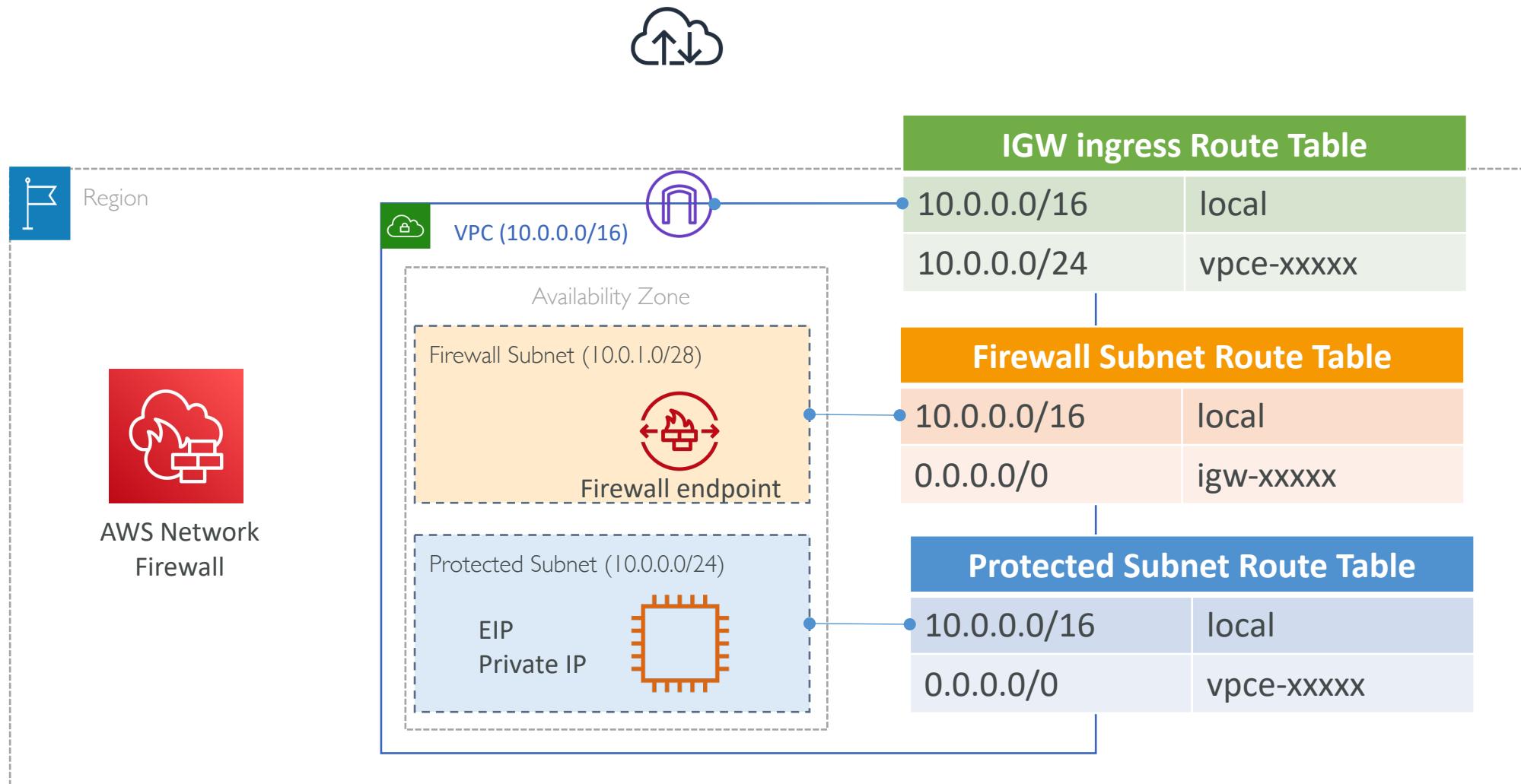
- Stateful Firewalls
  - Filters traffic based on network packet context and connection state
  - Context involves metadata of the packet including IP addresses, ports, packet length, information about reassembly and defragmentation, flags, TCP sequence number etc
  - State refers to the state of the connection e.g for TCP connection the state of connection is SYN, ACK, FIN and RST
- Stateless Firewalls
  - Inspects each packet in isolation
  - Performs well in case of heavy traffic

# AWS Network Firewall

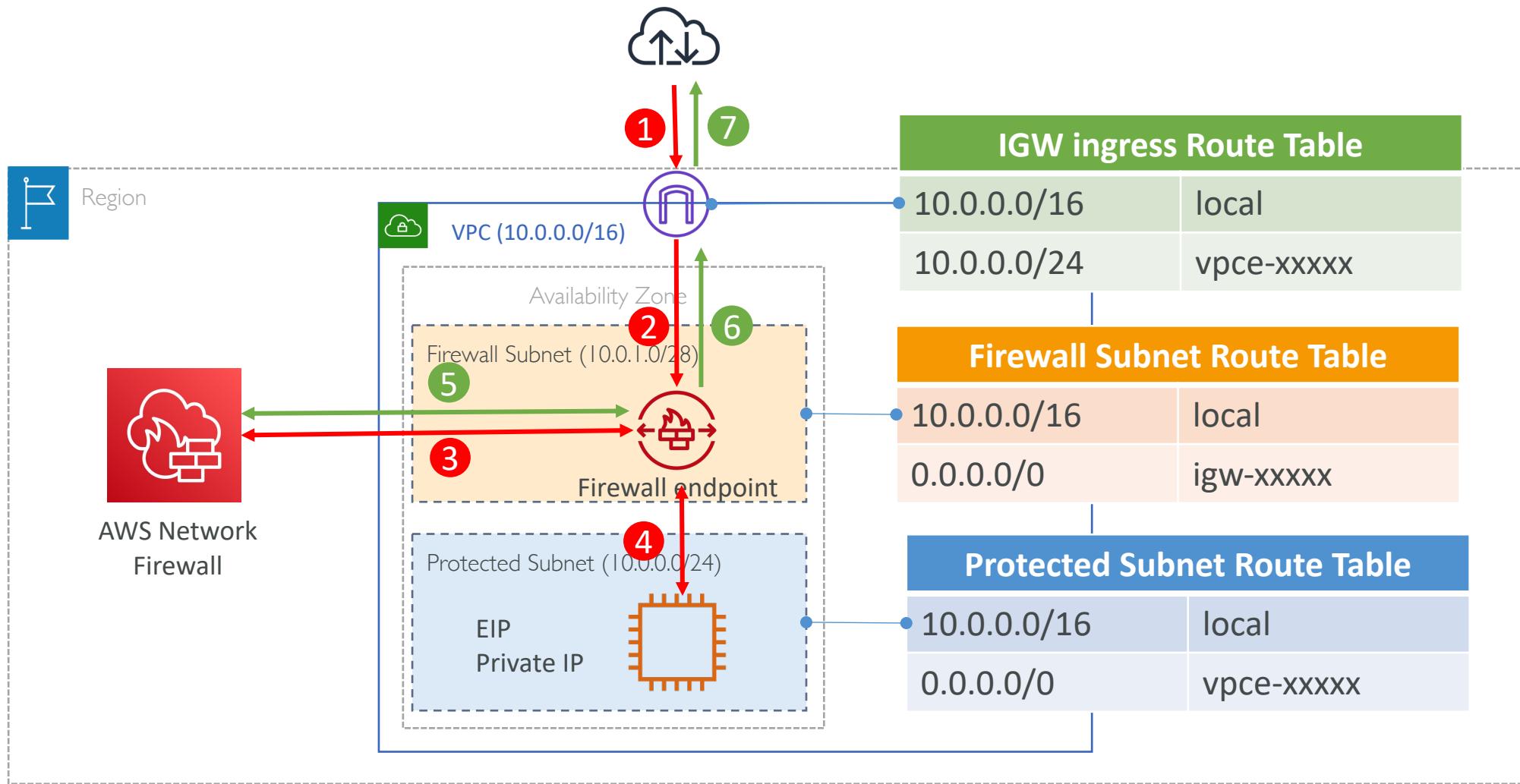


- Filters traffic at the perimeter of the VPC such as traffic going to or coming from internet gateway, NAT gateway, over VPN or Direct connect
- Uses the Suricata for stateful inspection
- Suricata is an open source-based intrusion detection system and intrusion prevention system developed by the Open Information Security Foundation
- Allows domain name filtering e.g only AWS service endpoints
- Block access to bad domains and limit the types of domain names
- Deep packet inspection on traffic entering or leaving your VPC
- Stateful protocol detection to filter protocols like HTTPS (independent of the port used)

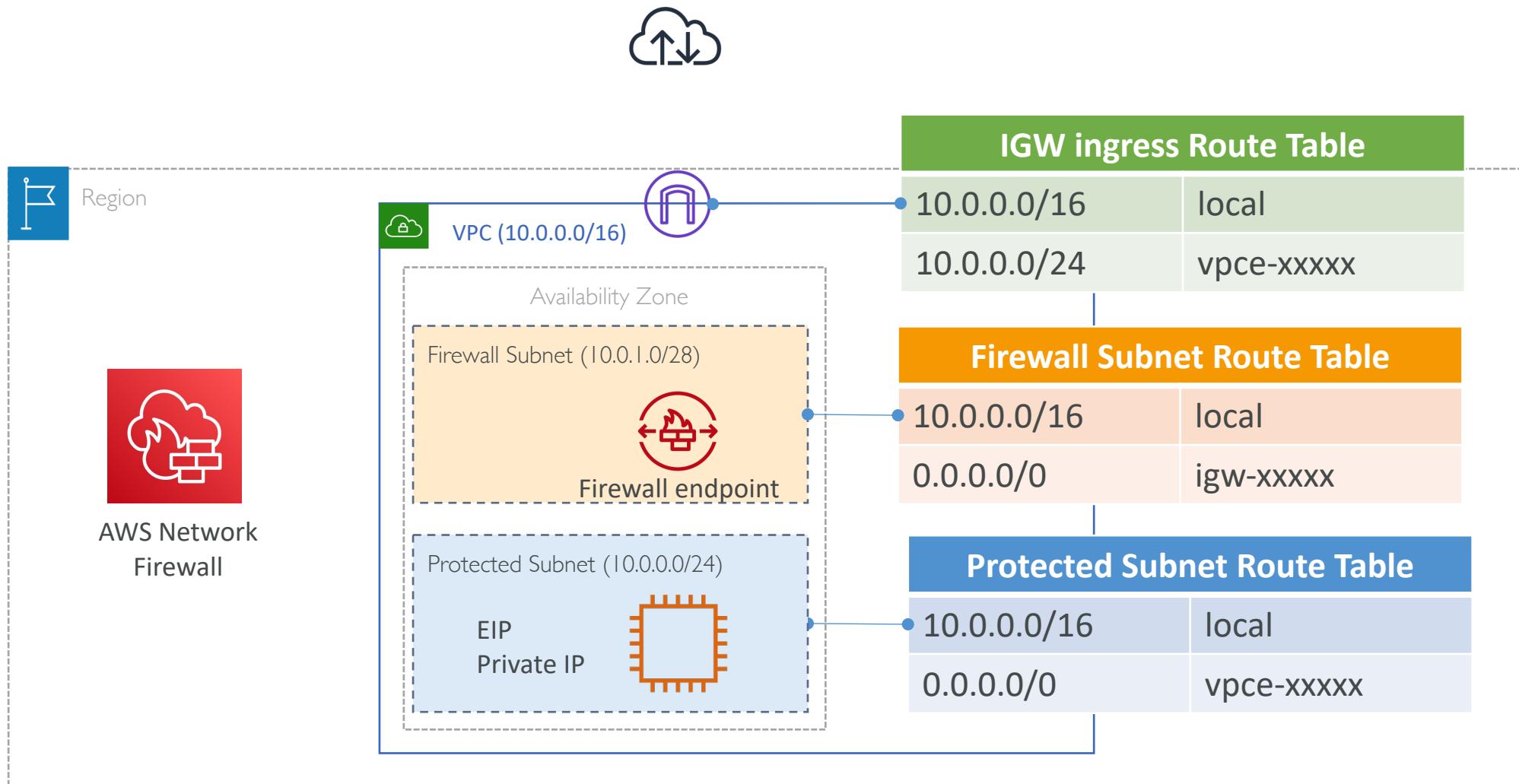
# Network firewall traffic flow - Inbound



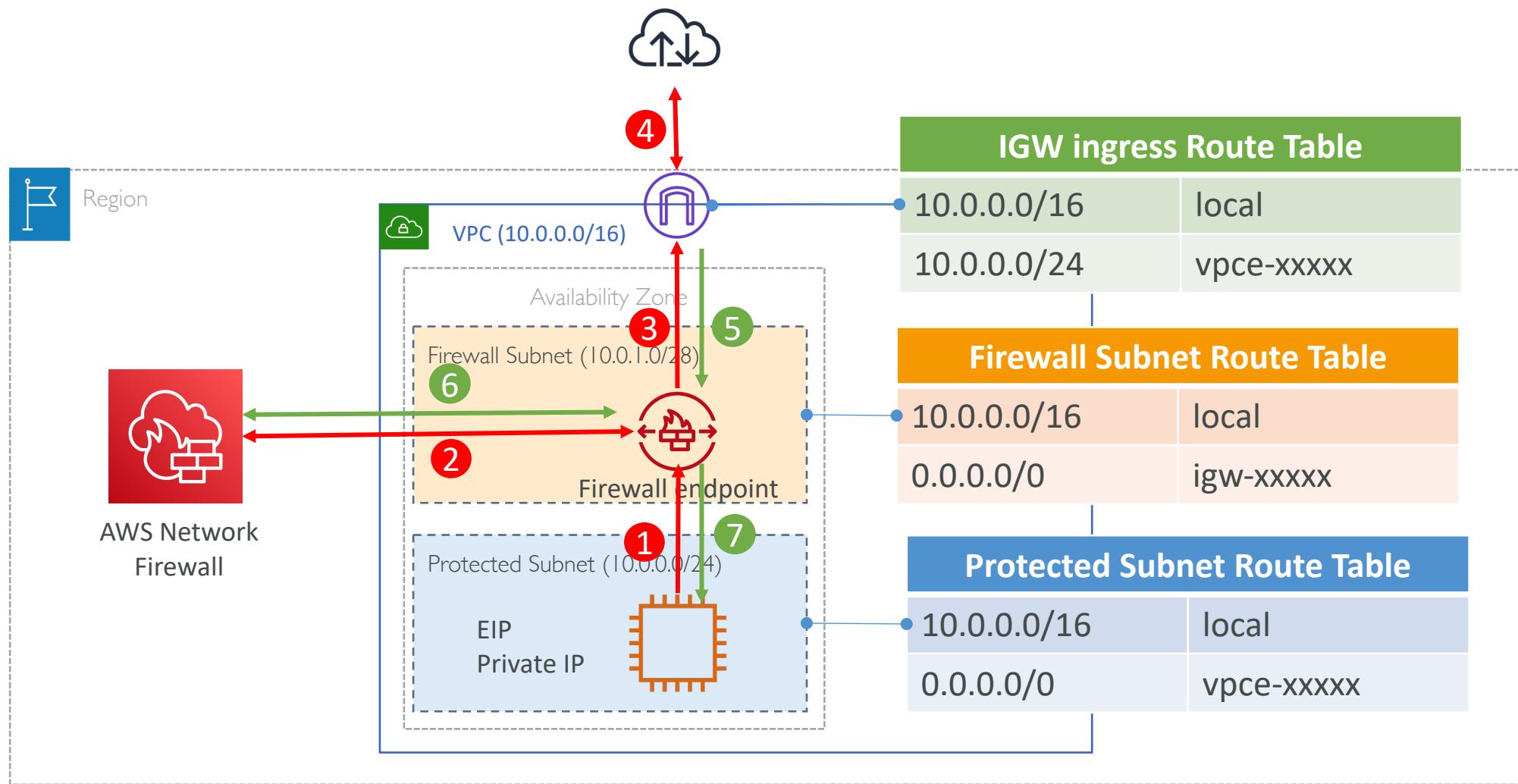
# Network firewall traffic flow - Inbound



# Network firewall traffic flow - Outbound



# Network firewall traffic flow - Outbound



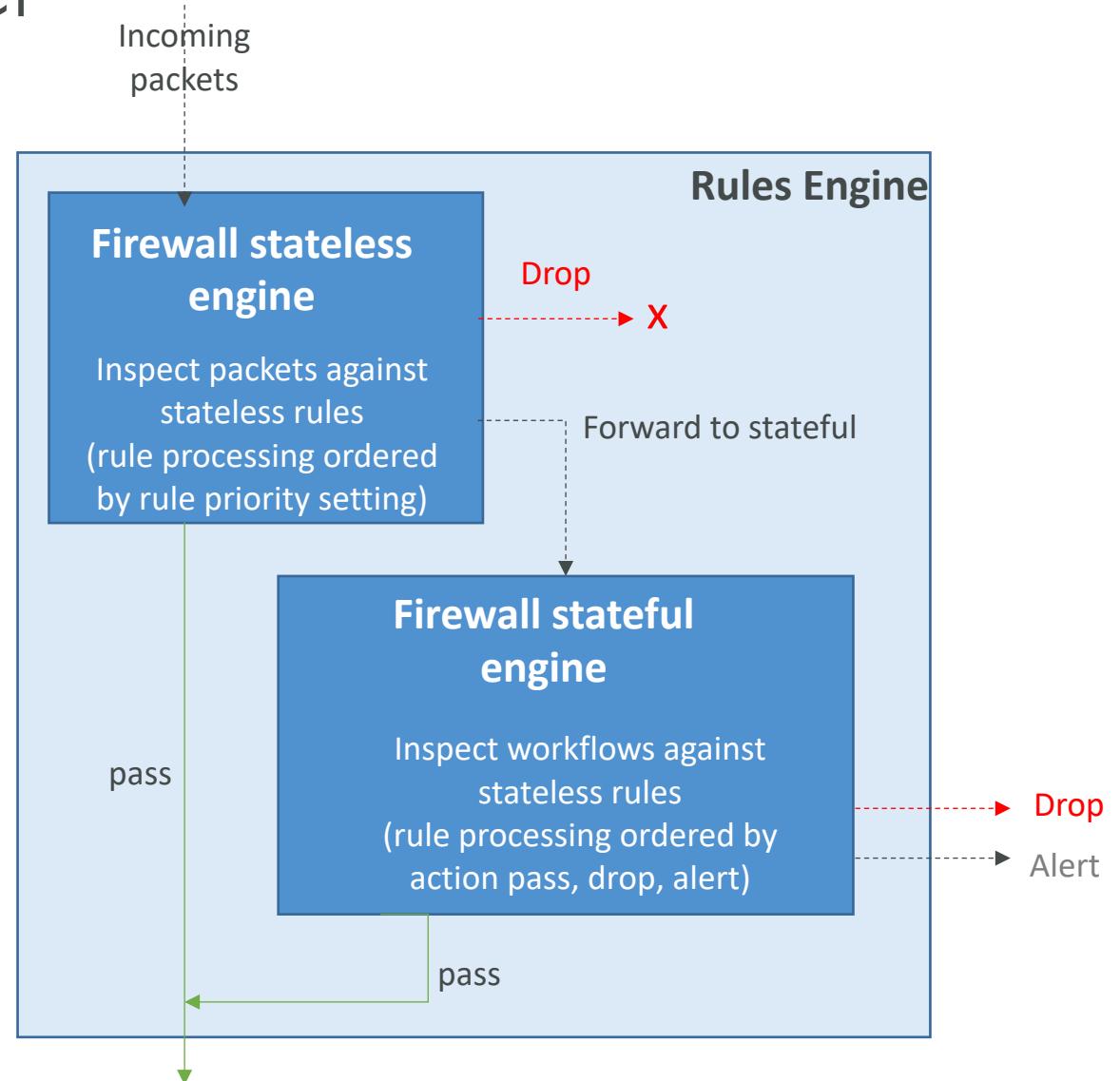
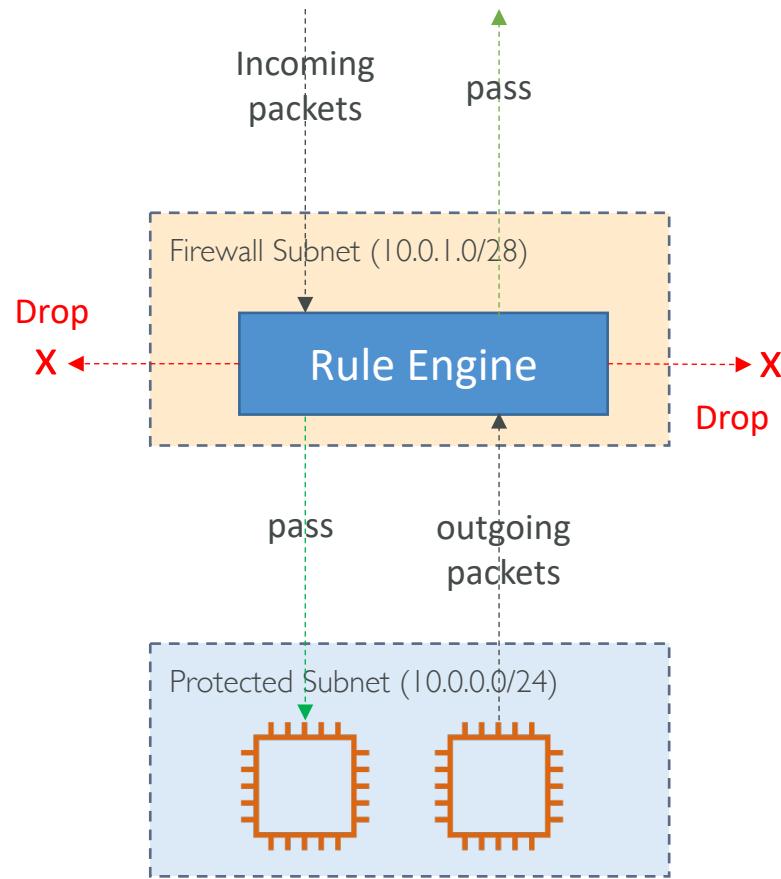
# Network Firewall components

- Firewall
  - A firewall connects the VPC that you want to protect to the protection behavior that's defined in a firewall policy.
- Firewall policy
  - Defines the behavior of the firewall in a collection of stateless and stateful rule groups and other settings.
  - You can associate each firewall with only one firewall policy, but you can use a firewall policy for more than one firewall.
- Rule group
  - A rule group is a collection of stateless or stateful rules that define how to inspect and handle network traffic.
  - Rules configuration includes 5-tuple and domain name filtering.

# Rule engine

- Stateful
  - Inspects packets in the context of their traffic flow and direction of the traffic
  - Supports rules compatible with Suricata (an open source IPS)
  - Processes rules in the order of their action setting, with pass rules processed first, then drop, then alert and then stops processing when it finds a match
- Stateless
  - Inspects each packet in isolation
  - Does not take into consideration factors such as the direction of traffic, or whether the packet is part of an existing, approved connection
  - Standard 5-tuple connection criteria – Protocol, Source IP range, Source port range, destination IP range, destination port range
  - Processes rules in the order that you prioritize them and stops processing when it finds a match
  - Similar in behavior and use to Amazon VPC network access control lists (ACLs)

# Rules evaluation order



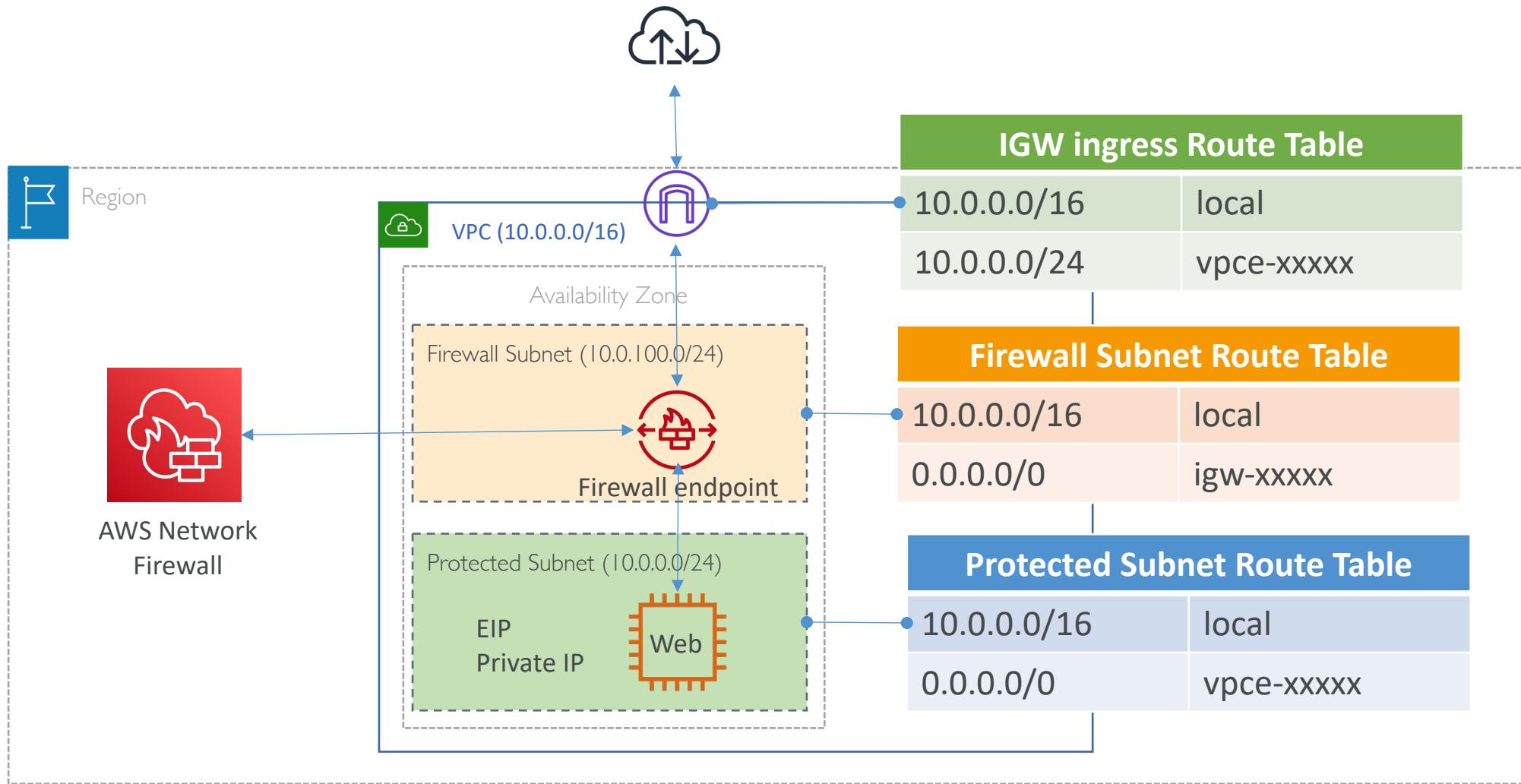
# Lab: AWS Network Firewall

# Lab: AWS Network Firewall

- What do we want to achieve?
  - Launch the simple webserver running on EC2 instance
  - Stateless rule to block the inbound ICPM (ping request) traffic to the web server
  - Stateful rule to allow outbound traffic from the webserver to a particular domain name e.g aws.amazon.com over TLS and block all other outgoing traffic

<https://aws.amazon.com/blogs/security/hands-on-walkthrough-of-the-aws-network-firewall-flexible-rules-engine/>

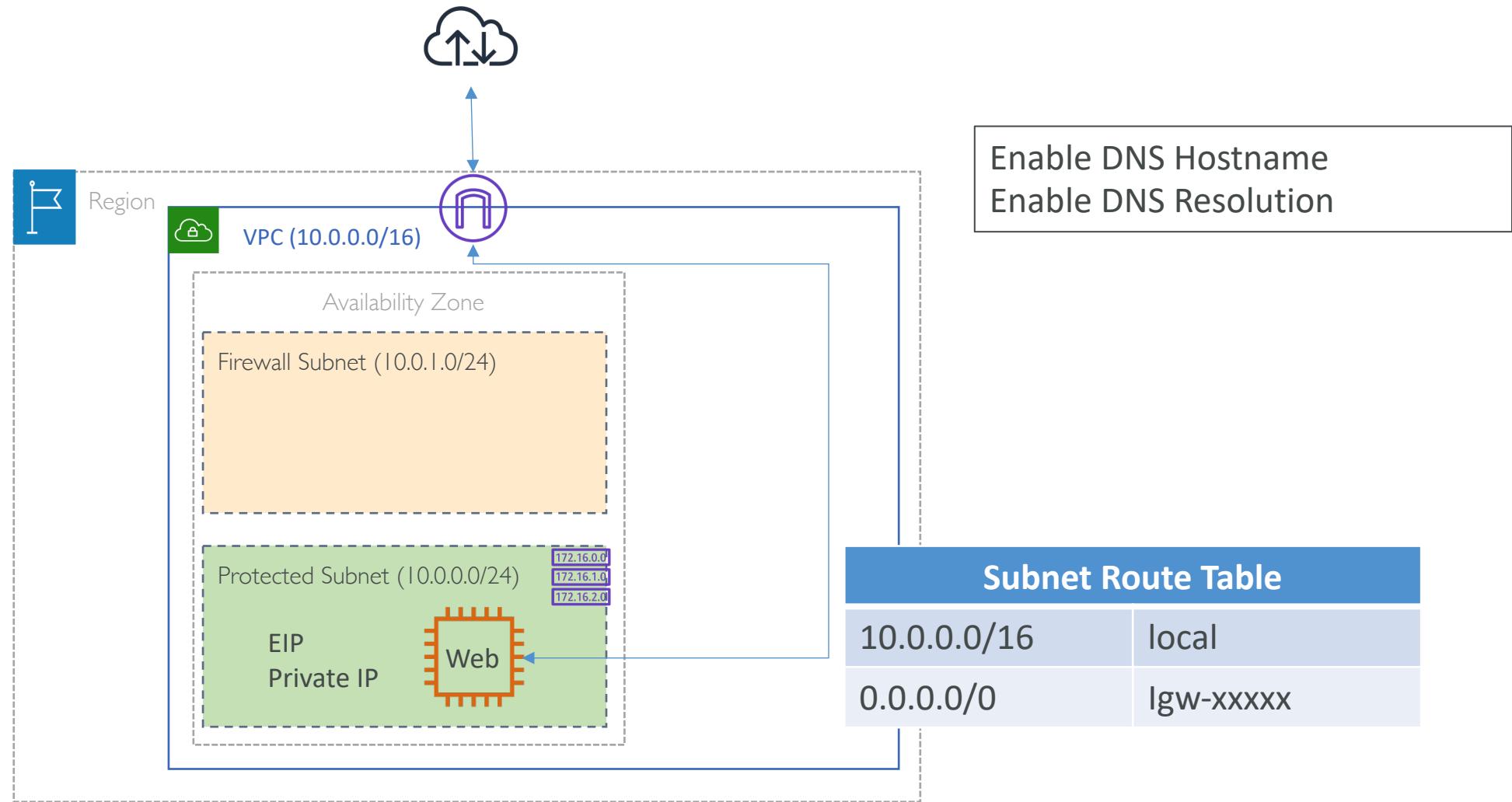
# Lab: Architecture



# Lab: Firewall rules

- Stateless Rules
  - 1. Drop all ICMP traffic from 0.0.0.0/0 to 0.0.0.0/0 (priority 10)
  - 2. Forward all other traffic to Stateful rule group (priority 20)
- Stateful Rules
  - 1. pass tcp any any -> any 22 (msg:"Allow TCP 22"; sid:1000001; rev:1;)
  - 2. pass http any any -> any any (http.host; dotprefix; content:".amazonaws.com"; endswith; msg:"Permit HTTP access to the web server"; sid:1000002; rev:1;)
  - 3. pass tls any any -> any any (tls.sni; content:"aws.amazon.com"; startswith; nocase; endswith; msg:"Permit HTTPS access to aws.amazon.com"; sid:1000003; rev:1;)
  - 4. drop tcp any any -> any any (flow:established,to\_server; msg:"Deny all other TCP traffic"; sid:1000004; rev:1;)

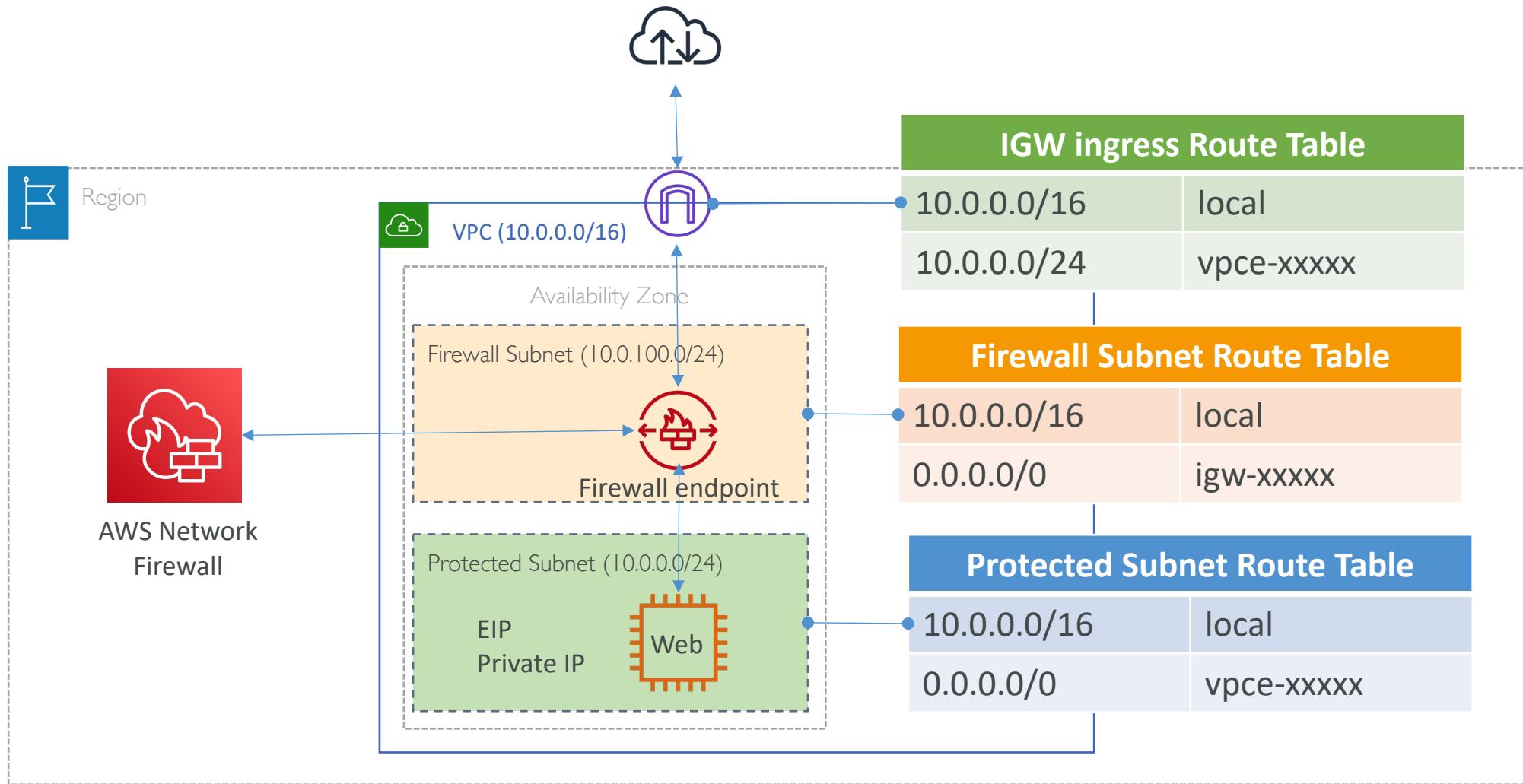
# Lab: Initial Architecture to start with



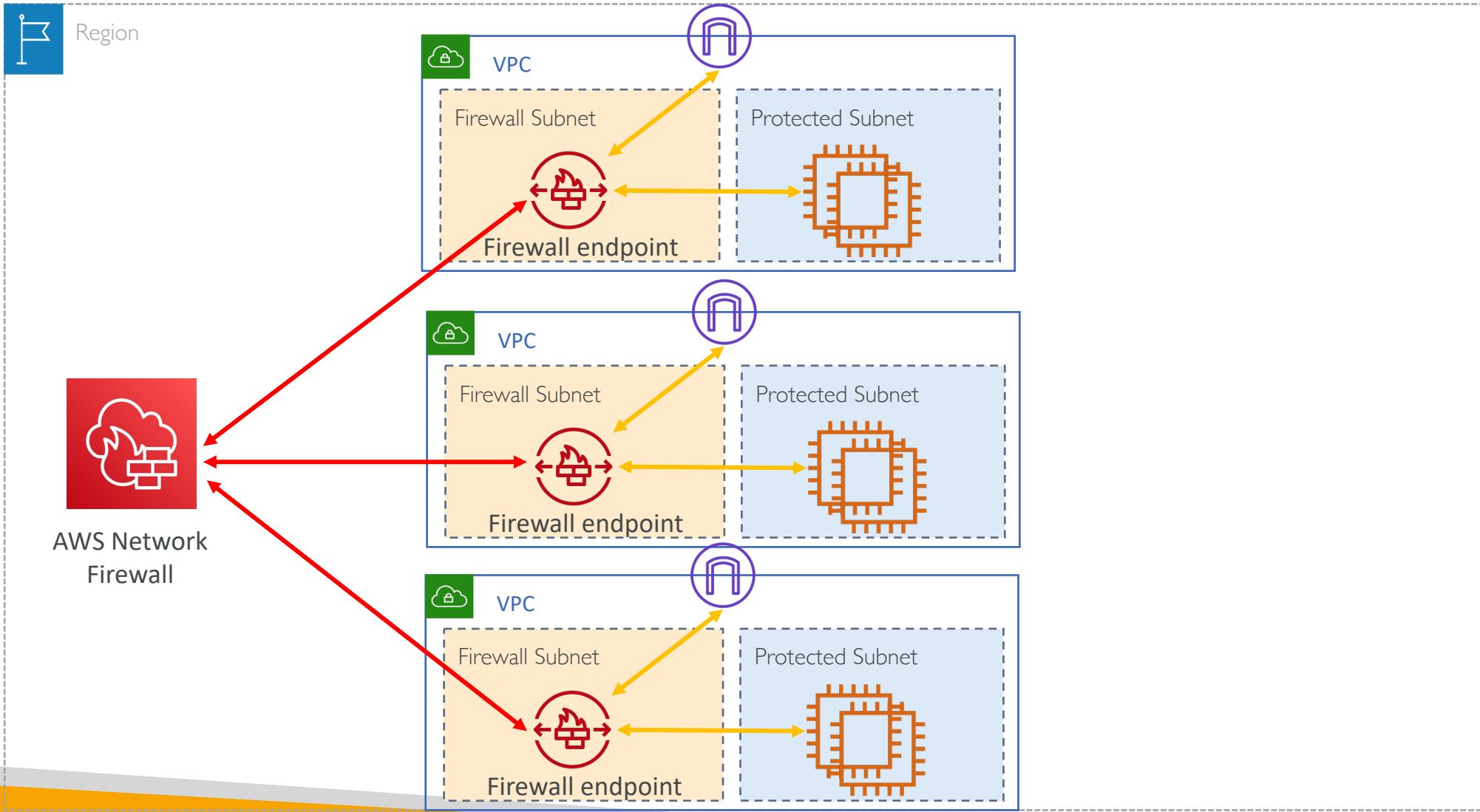
# Lab: Next steps

- SSH to EC2 instance and install HTTPD web server (In security group allow SSH, HTTP and ICMP traffic from anywhere 0.0.0.0/0)
- Verify that you are able to access the web server over a browser using EC2 instances public IP or public DNS
- Verify that you are able to ping to EC2 instance from your workstation
- Now create AWS Network Firewall in the Firewall subnet and create firewall policy and rules as shown earlier
- Modify route tables for both the subnets and internet gateway
- Test connectivity
  - Ping to webserver should not work
  - SSH to webserver and from webserver you should only be able to download content from <https://aws.amazon.com>. All other traffic should be blocked.

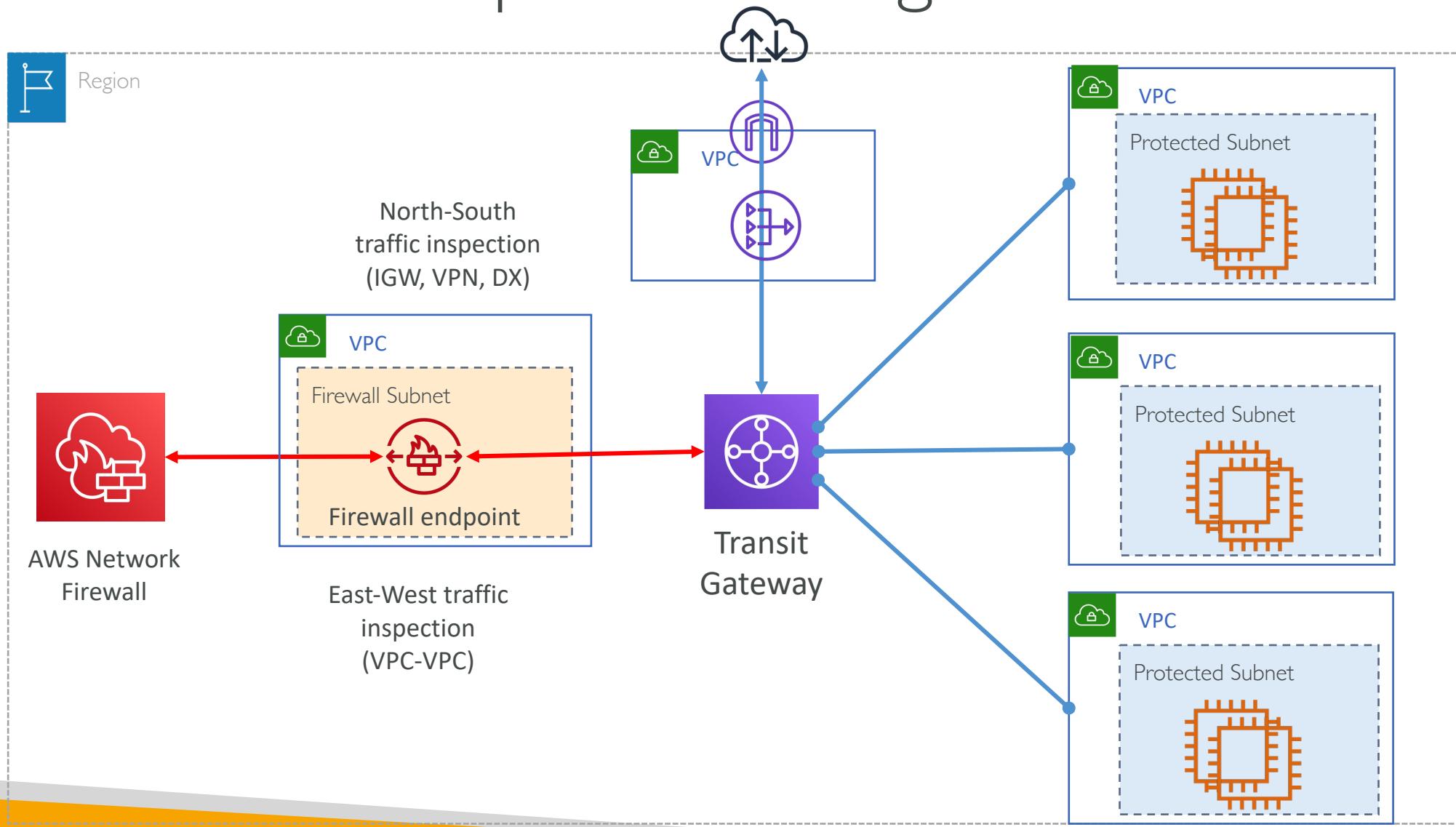
# Lab: Route tables



# Distributed Inspection



# Centralized Inspection using TGW



# AWS Firewall Manager



- Security management tool which simplifies security administration and maintenance tasks across multiple AWS accounts and resources
- Manages the rules for AWS WAF, AWS Shield Advanced, Amazon VPC security groups, AWS Network Firewall, and Amazon Route 53 Resolver DNS Firewall
- New accounts added under the AWS Organizations are automatically protected
- Provides centralized monitoring of DDoS attacks across AWS organization

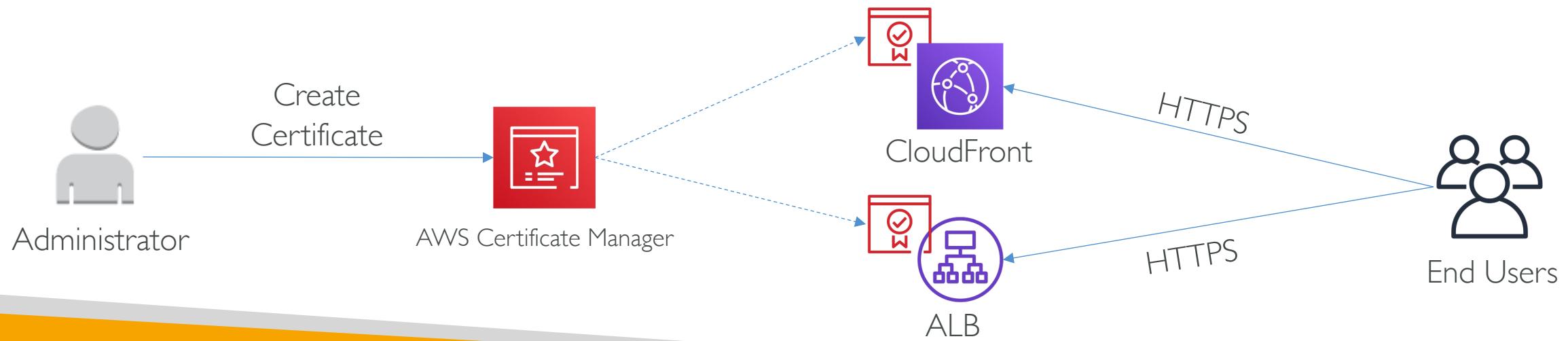
# How to decide?

|  |                                    |
|--|------------------------------------|
| Need an easy and quick to deploy firewall for protecting Web applications from Layer7 attacks?   | AWS Web Application Firewall (WAF) |
| Need to have protection against DDoS with support from AWS to handle the attack?   | AWS Shield Advanced                |
| Need the centralized governance for managing the firewall protections across accounts?   | AWS Firewall Manager               |
| Need more sophisticated managed firewall (IPS/IDS) for all the traffic flowing in/out of VPC resources?                                | AWS Network Firewall               |
| Need sophisticated 3 <sup>rd</sup> party firewall (IPS/IDS) or network monitoring for all the traffic flowing in/out of VPC resources? | Gateway Load Balancer              |
| <i>These services are not mutually exclusive. Ex. You can use WAF, Shield and AWS Firewall manager together.</i>                       |                                    |

# Other AWS Security Services – ACM, Route53 DNSSEC, GuardDuty, Inspector

# Amazon Certificate Manager (ACM)

- Provides the free SSL/TLS Certificate which you can deploy on CloudFront, ELB, ElasticBeanstalk, API Gateway
- You can also request wildcard certificate e.g \*.example.com
- ACM is a regional service and you need to generate certificate in each region. However for **CloudFront** you must request certificate in **us-east-1** i.e North Virginia region.



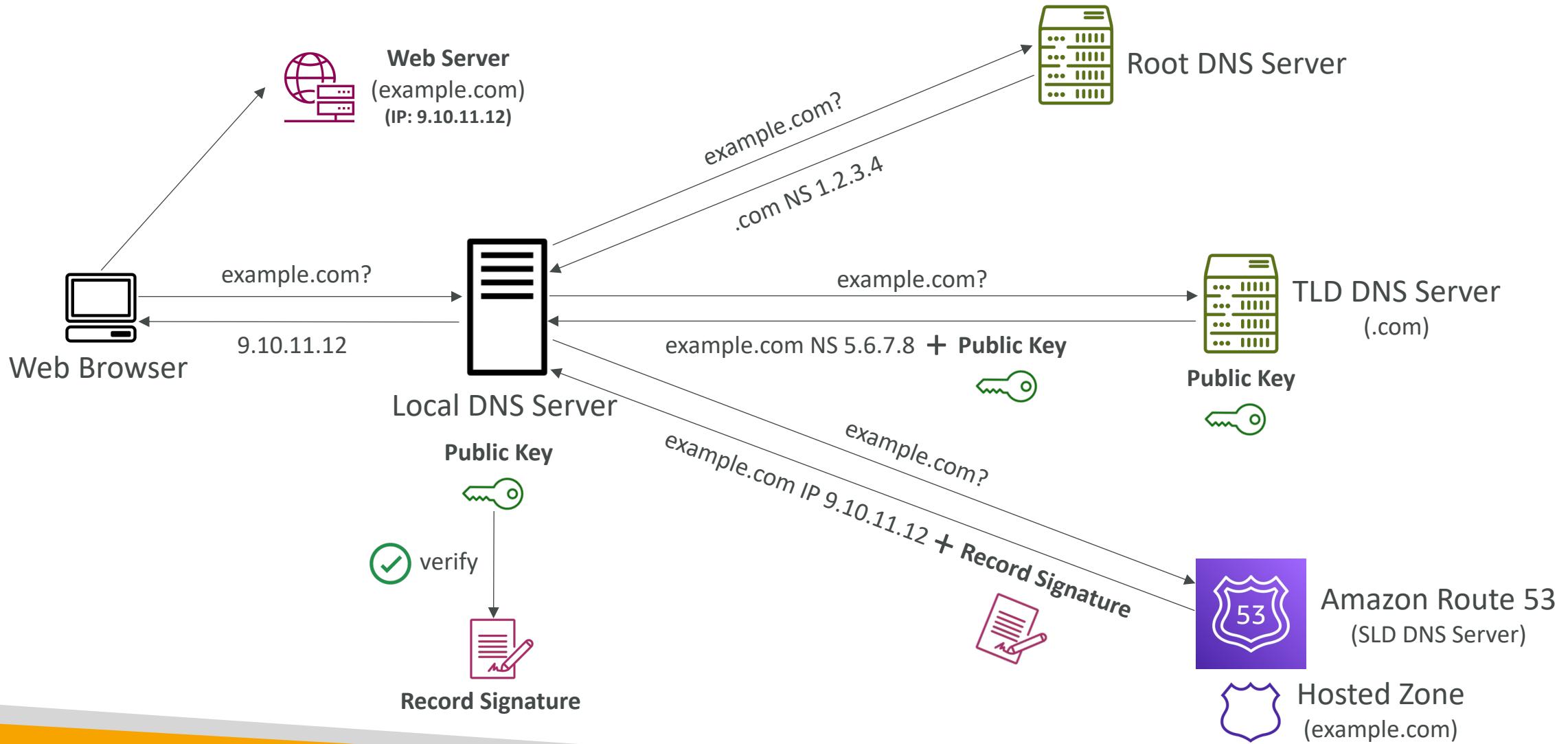
# Amazon Certificate Manager (ACM)

- You can not have wildcard in the middle of the domain name
  - Example: app.\*.example.com is not allowed
  - Example: \*.app.example.com is allowed
- Public certificates issued through ACM are valid for 13 months (395 days) However, while creating private CA, you may chose the certificate validity period
- ACM begins the renewal process up to 60 days prior to the certificate's expiration date
- Certificates managed in ACM use RSA keys with a 2048-bit modulus and SHA-256

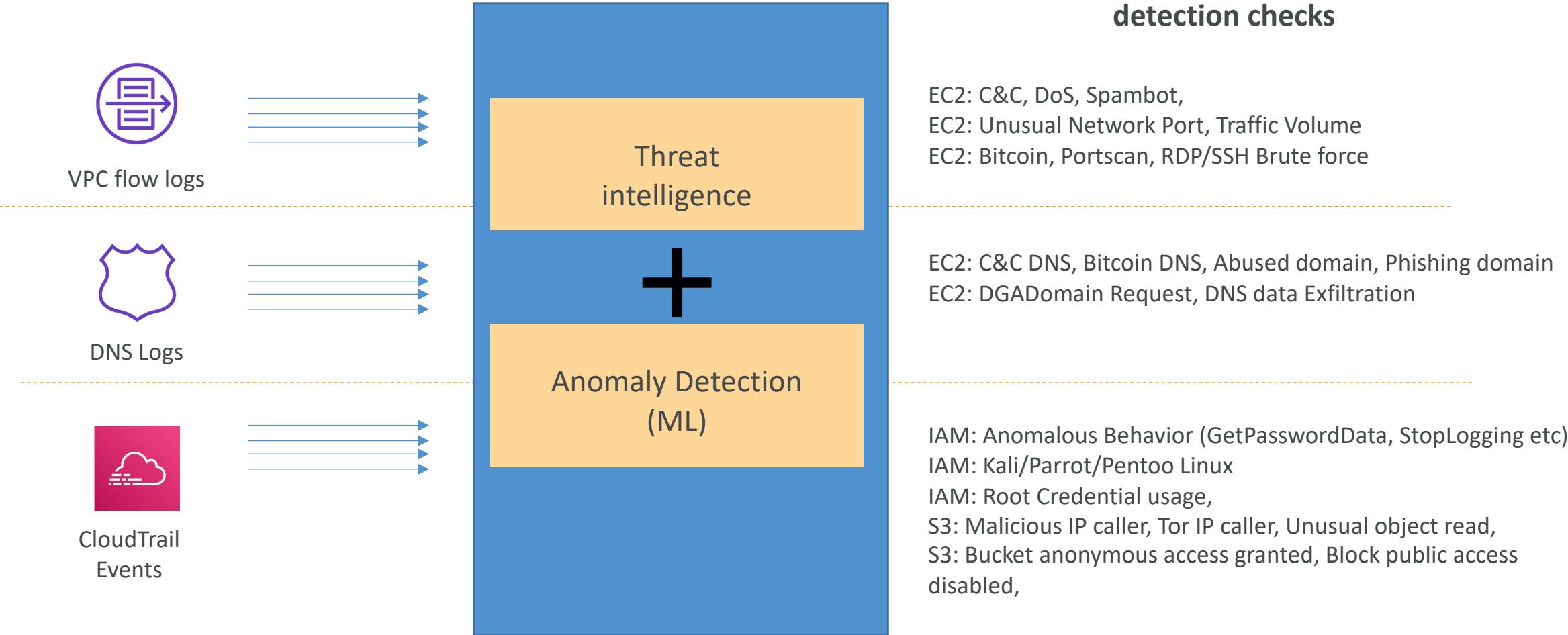
# Route 53 – DNS Security Extensions (DNSSEC)

- A protocol for securing DNS traffic, verifies DNS data integrity and origin
- Works only with **Public Hosted Zones**
- Route 53 supports both DNSSEC for Domain Registration and DNSSEC Signing
- **DNSSEC Signing**
  - Validate that a DNS response came from Route 53 and has not been tampered with
  - Route 53 cryptographically signs each record in the Hosted Zone
  - Two Keys:
    - Key-signing Key (KSK) – based on an asymmetric CMK in AWS KMS (managed by you)
    - Zone-signing Key (ZSK) – managed by Route 53
- When enabled, Route 53 enforces a TTL of one week for all records in the Hosted Zone (records that have TTL less than one week are not affected)

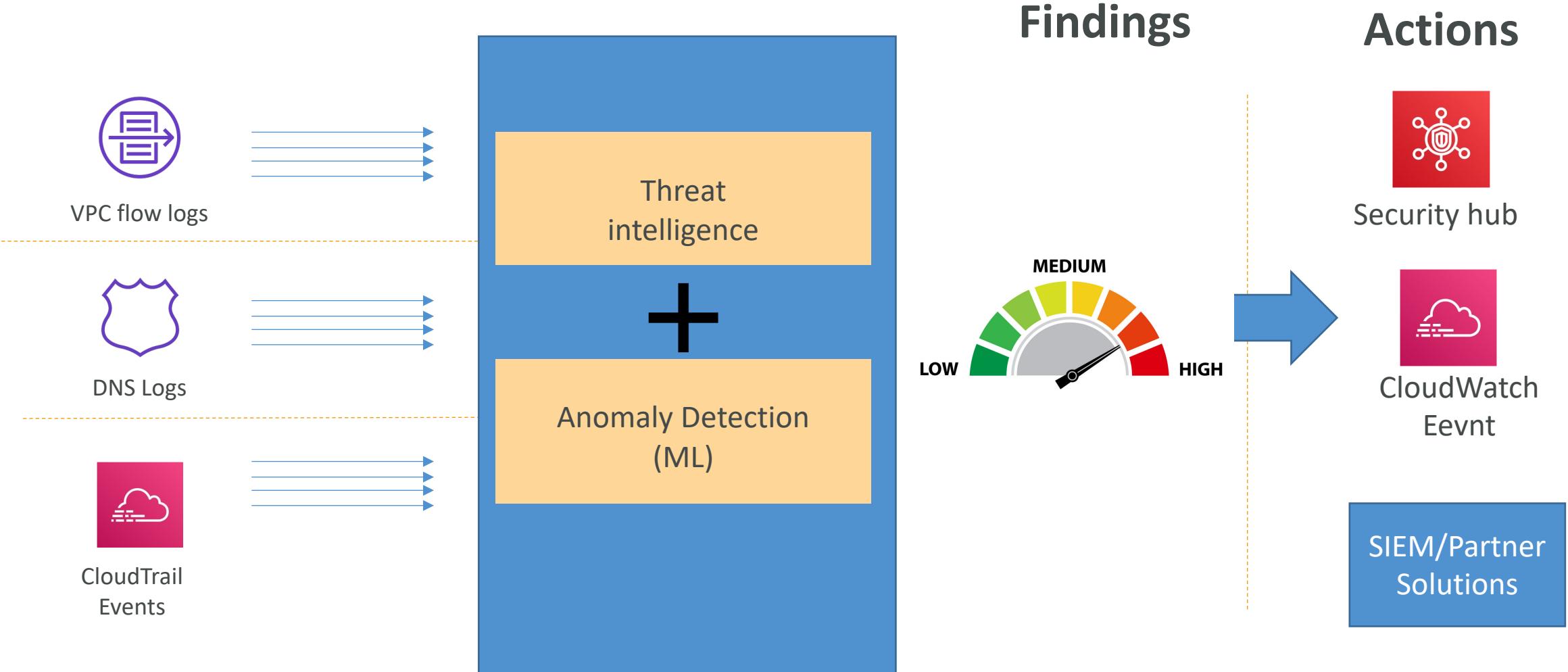
# DNSSEC – How It Works?



# AWS GuardDuty



# AWS GuardDuty



# AWS GuardDuty

- Intelligent threat detection service
- GuardDuty analyzes billions of events from AWS CloudTrail, VPC Flow Logs, and DNS Logs
- There are no agents, sensors, or network appliances to deploy and absolutely no footprint in your AWS account
- There is no risk of performance or availability impact to existing workloads
- GuardDuty uses threat intelligence feeds, such as lists of malicious IPs and domains, and **machine learning** to detect threats more accurately
- When a threat is detected, GuardDuty delivers a detailed security finding to the console and AWS CloudWatch Events, making alerts actionable

# AWS Inspector



Assessment Template



Assessment Targets



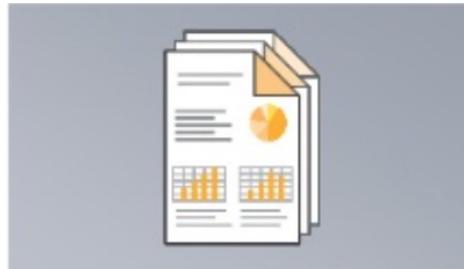
Rules Packages



Inspector Agent



Triggers and Rules



Findings and Reports

# AWS Inspector



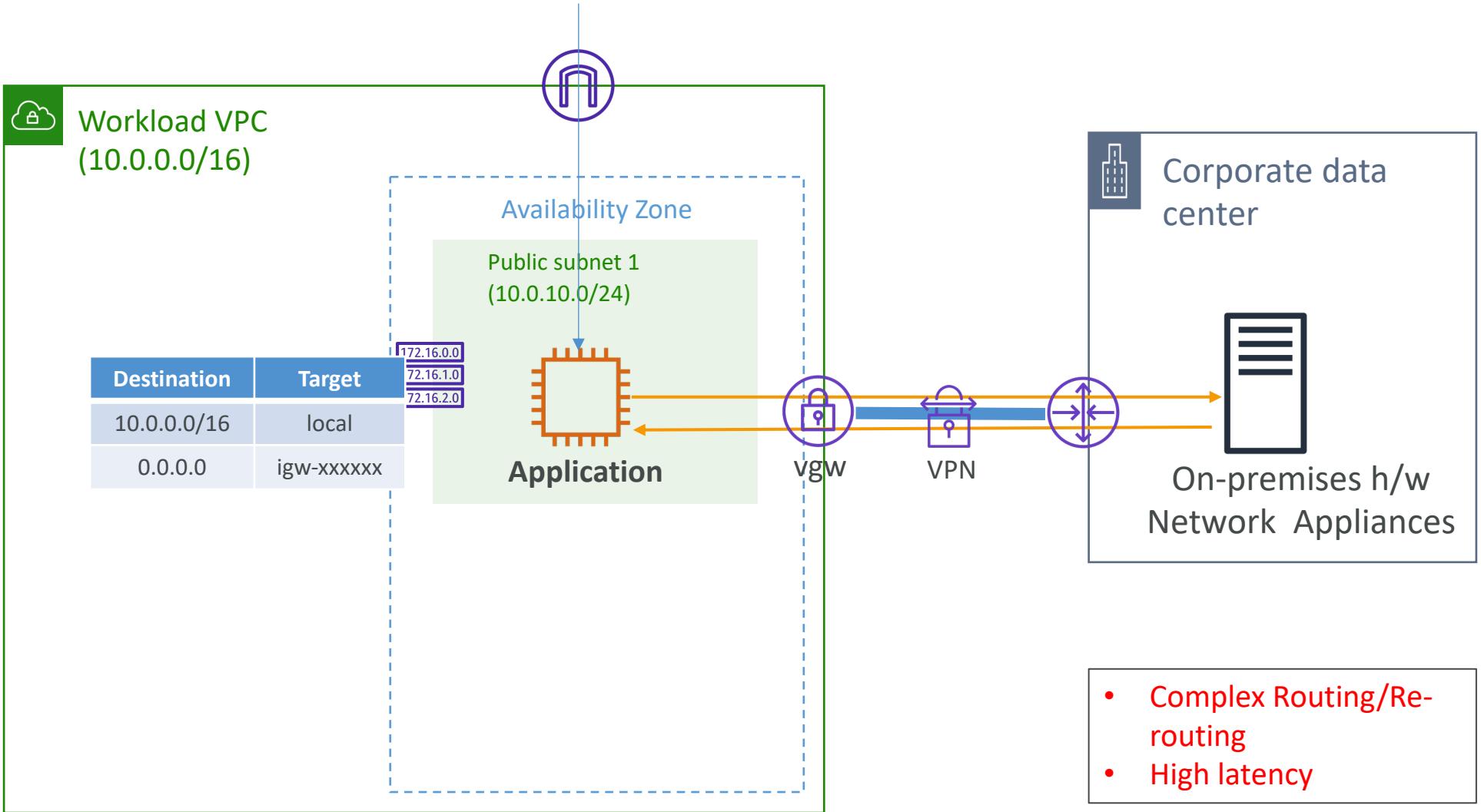
- Simple Vulnerability Assessment for EC2 instances
- Common Vulnerability and Exposures (CVE)
- Center of Internet Security (CIS) Benchmark
- Security Best Practices
- Network Reachability
- The CVEs, Vulnerability rules and security best practices are automatically updated over the time

# AWS Inspector Classic (re:invent 2021)

- Support for scanning Container images for software vulnerabilities
- Support for multi-account management via AWS Organization
- Just use Systems manager agent (SSM Agent)
- Automated and Continual scanning
- Risk score
- Integration with Security Hub, Event Bridge, ECR etc.

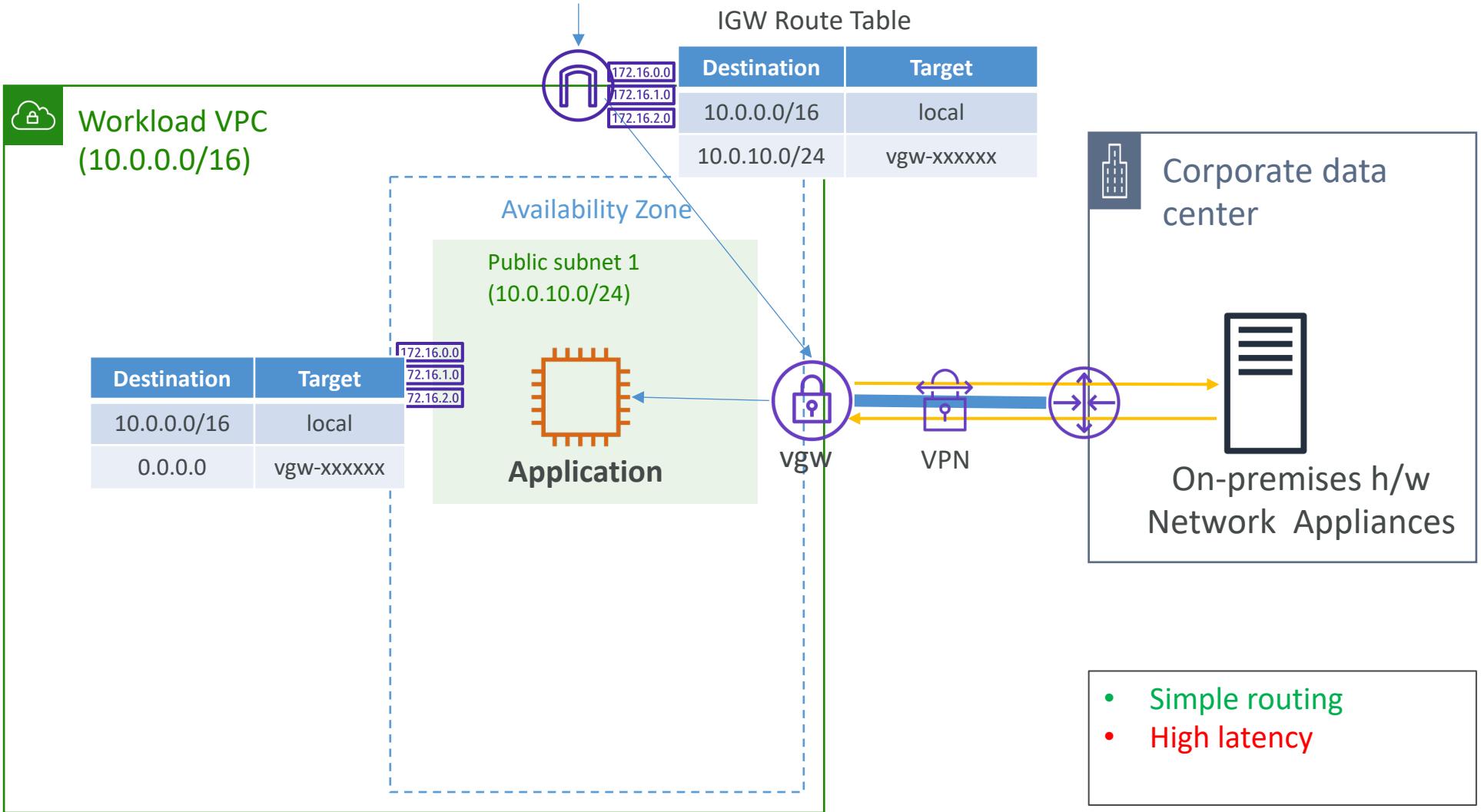
# Gateway Load Balancer

# Network appliances – Hardware based

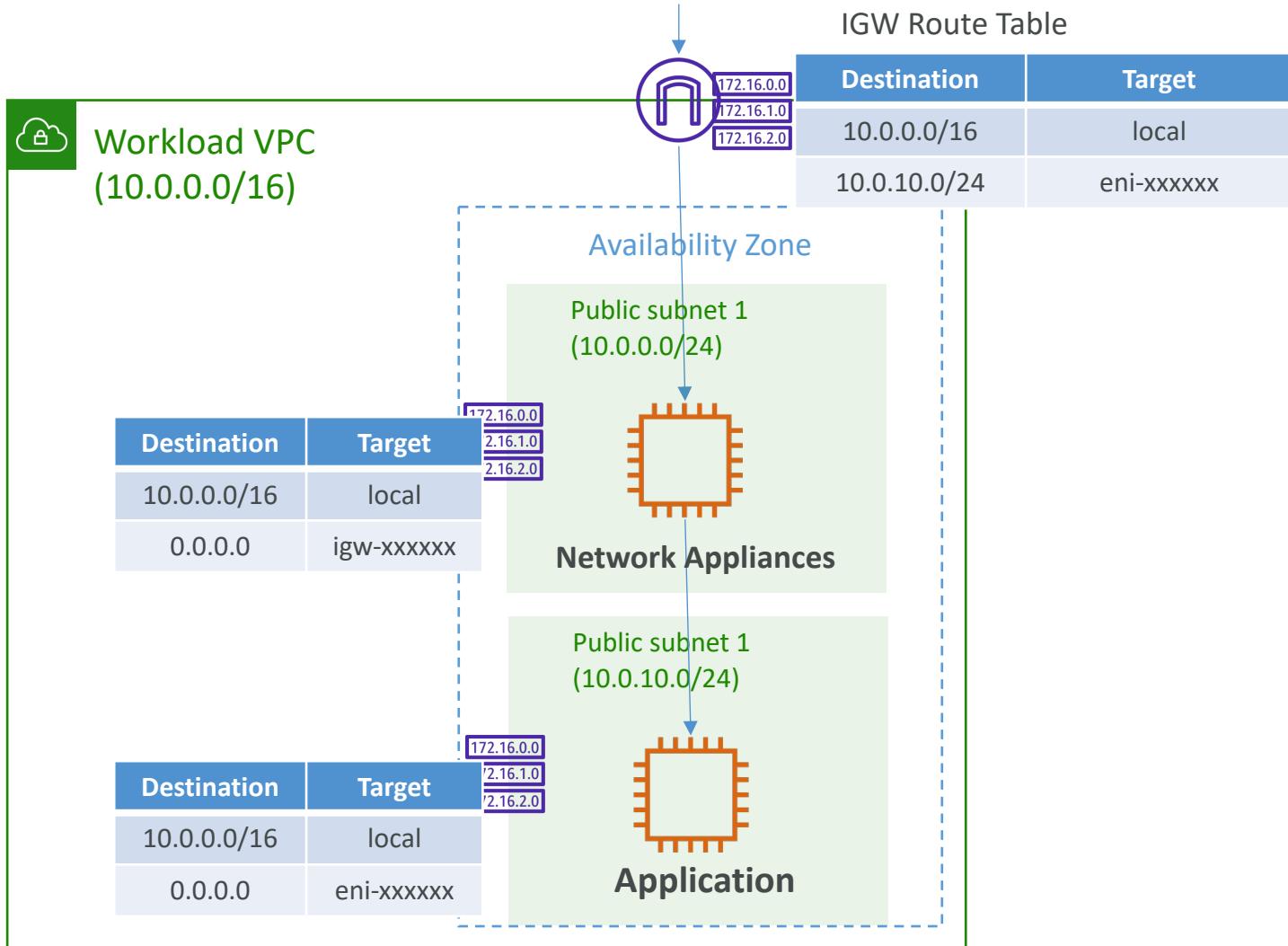


- Complex Routing/Re-routing
- High latency

# Network appliances – VPC ingress routing

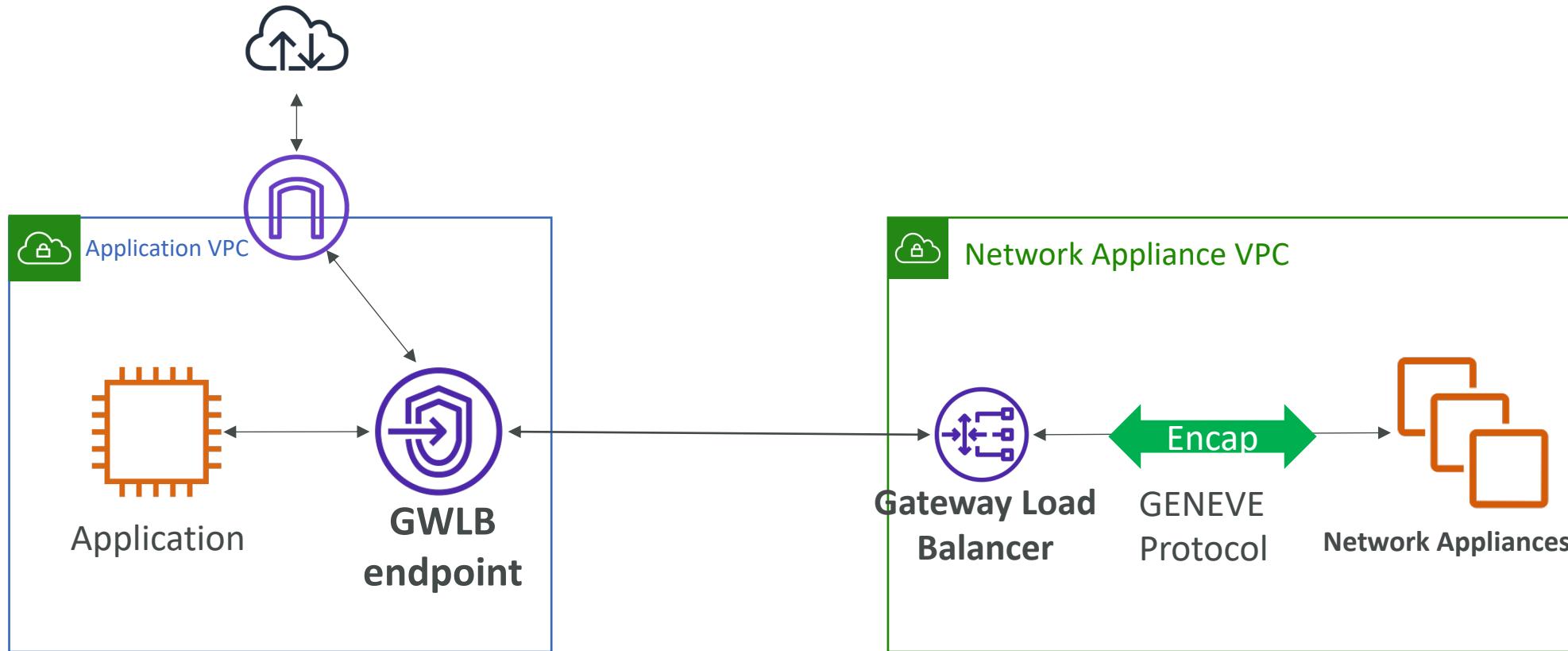


# Network appliances – Virtual Appliance



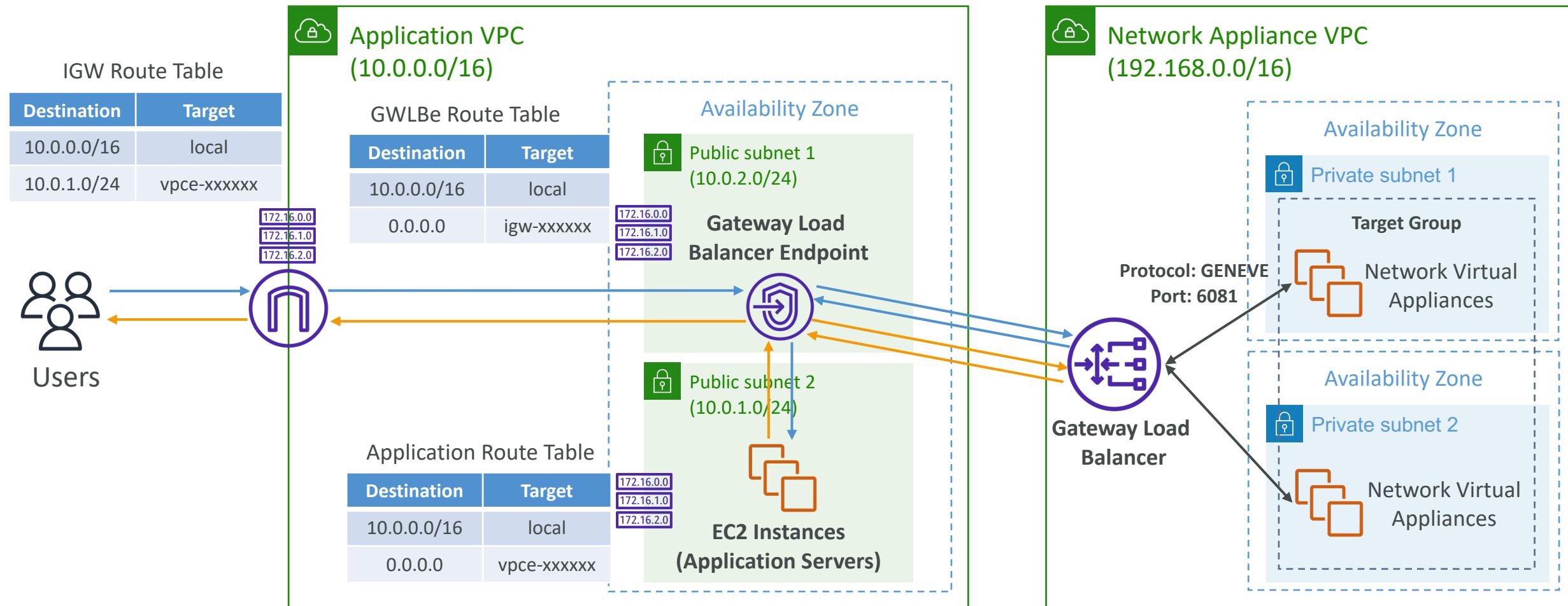
- Simple
- Low Latency
- Non-scalable
- Non-HA

# Gateway Load Balancer



- Simple
- Low Latency
- Scalable
- HA

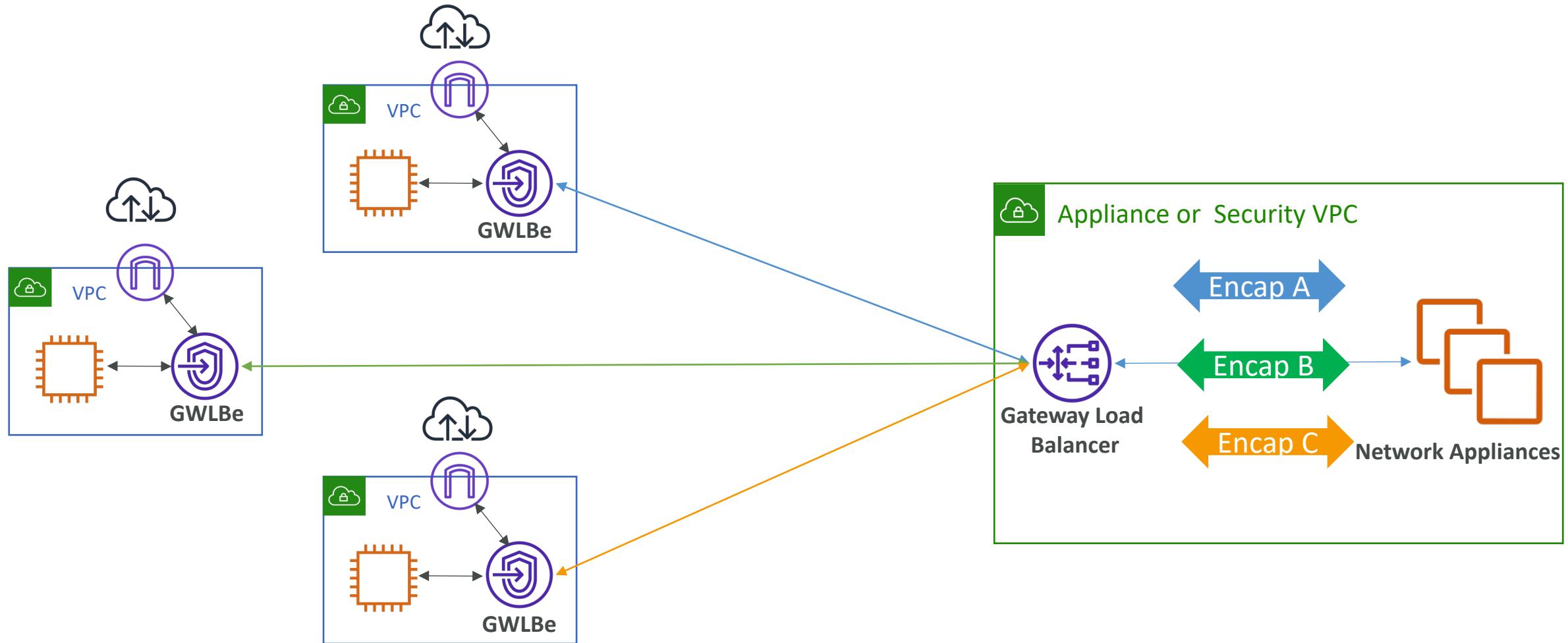
# Gateway Load Balancer – How It Works



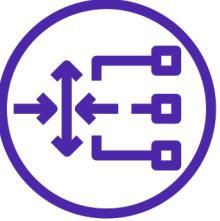
**NOTE: GWLBE and Application Servers must be in different subnets**

# Gateway Load Balancer Architectures

# Centralized Inspection using GWLB



# Summary



- GWLB is used to deploy, scale, and manage a fleet of 3<sup>rd</sup> party network appliances in AWS.
- Example: Firewalls, Intrusion Detection and Preventions Systems, Deep Packet Inspection Systems, payload manipulation etc.
- GWLB operates at Layer 3 (Network Layer) – As opposed to ALB (Layer7) and NLB (Layer4)
- It's a transparent network gateway which acts as single entry and exit point for all traffic
- Load balances the traffic and scales virtual appliances on demand
- GWLB integrates with industry leading partners Aviatrix, Cisco Systems, Fortinet, Palo Alto Networks, ...
  - <https://aws.amazon.com/elasticloadbalancing/partners>

# Exam Essentials

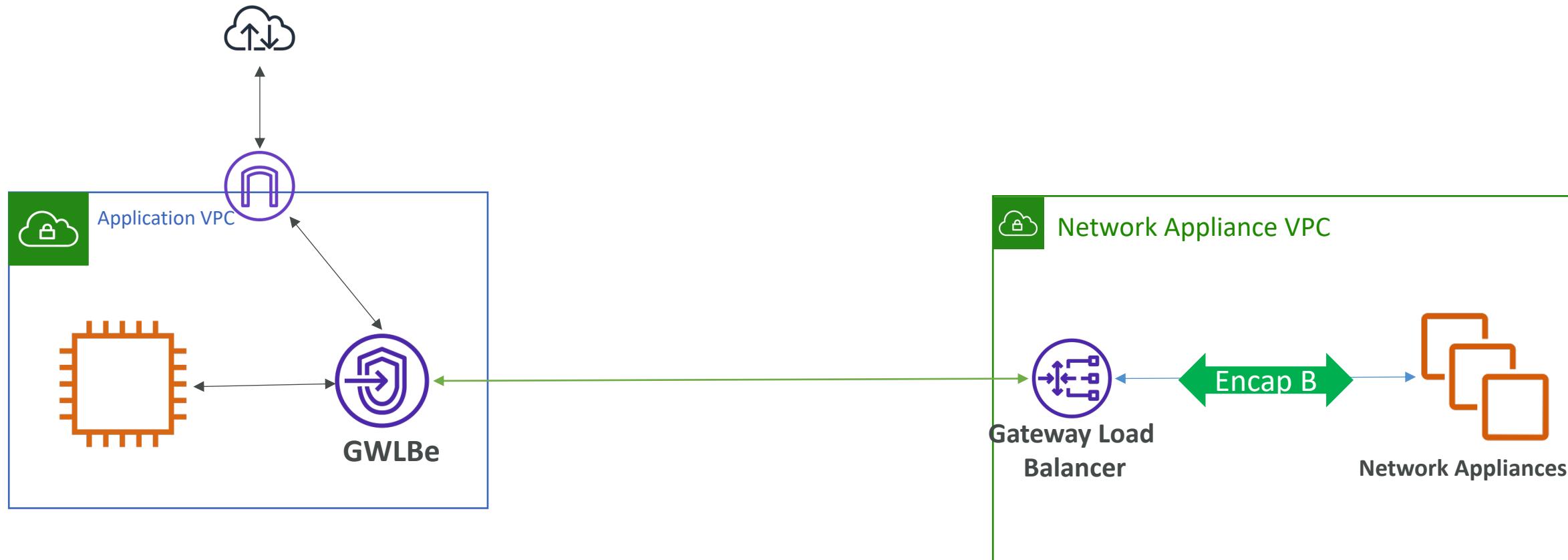
- GWLB Listener
  - Listen for all IP packets across all ports (can't specify a protocol or port)
  - You only specify a rule for routing requests to target groups
  - Can't be deleted
- Target Groups
  - EC2 Instances – can be managed by an ASG
  - IP Addresses – Private IP only
- Traffic between GWLB and its targets exchanged using the **GENEVE protocol** on UDP port 6081
- It maintains stickiness of flows to a specific target appliance using 5-tuple (for TCP/UDP flows) or 3-tuple (for non-TCP/UDP flows)
- Health Checks supported protocols HTTP, HTTPS, and TCP

# Exam Essentials

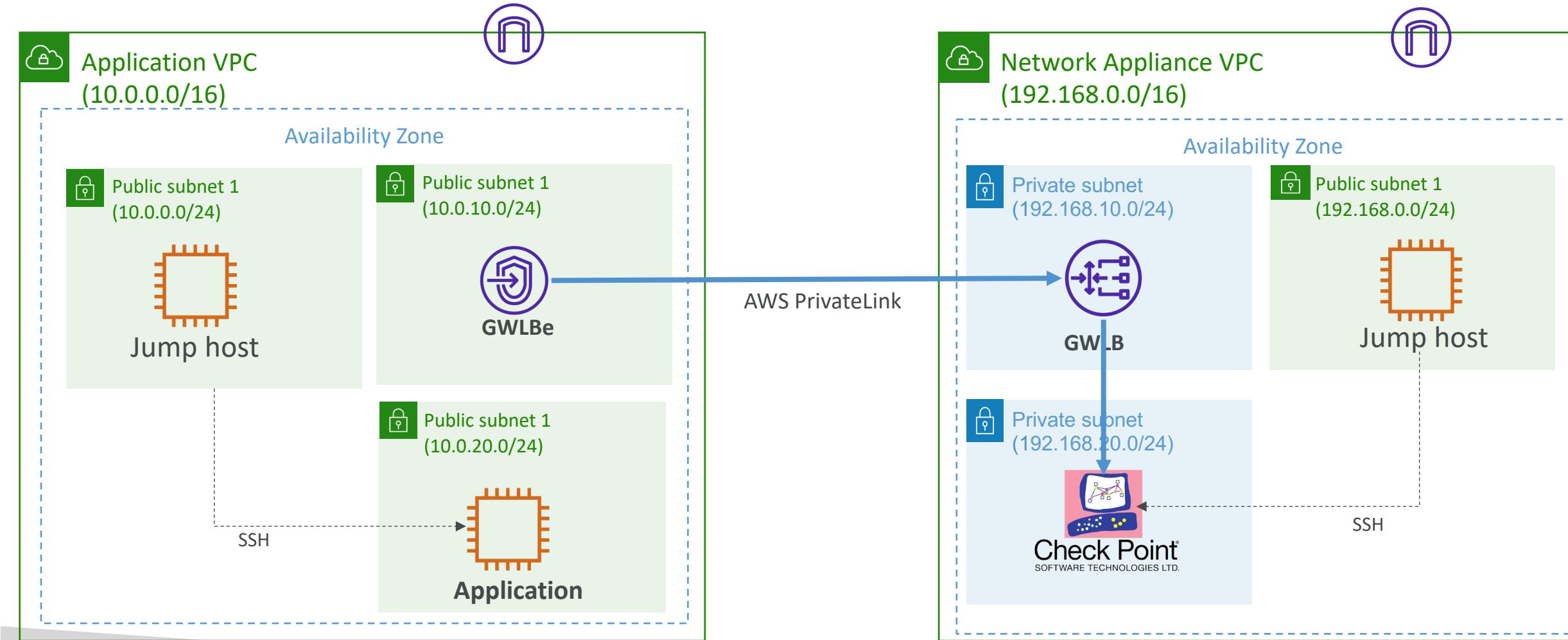
- Doesn't support public GWLB (internal only, no public DNS)
- Doesn't support dual stack mode (IPv4 only)
- You can't associate security groups to GWLBs (SGs associated with targets must use IP addresses to allow traffic from GWLB)
- Supports MTU size of 8500 bytes

# Lab: Gateway Load Balancer

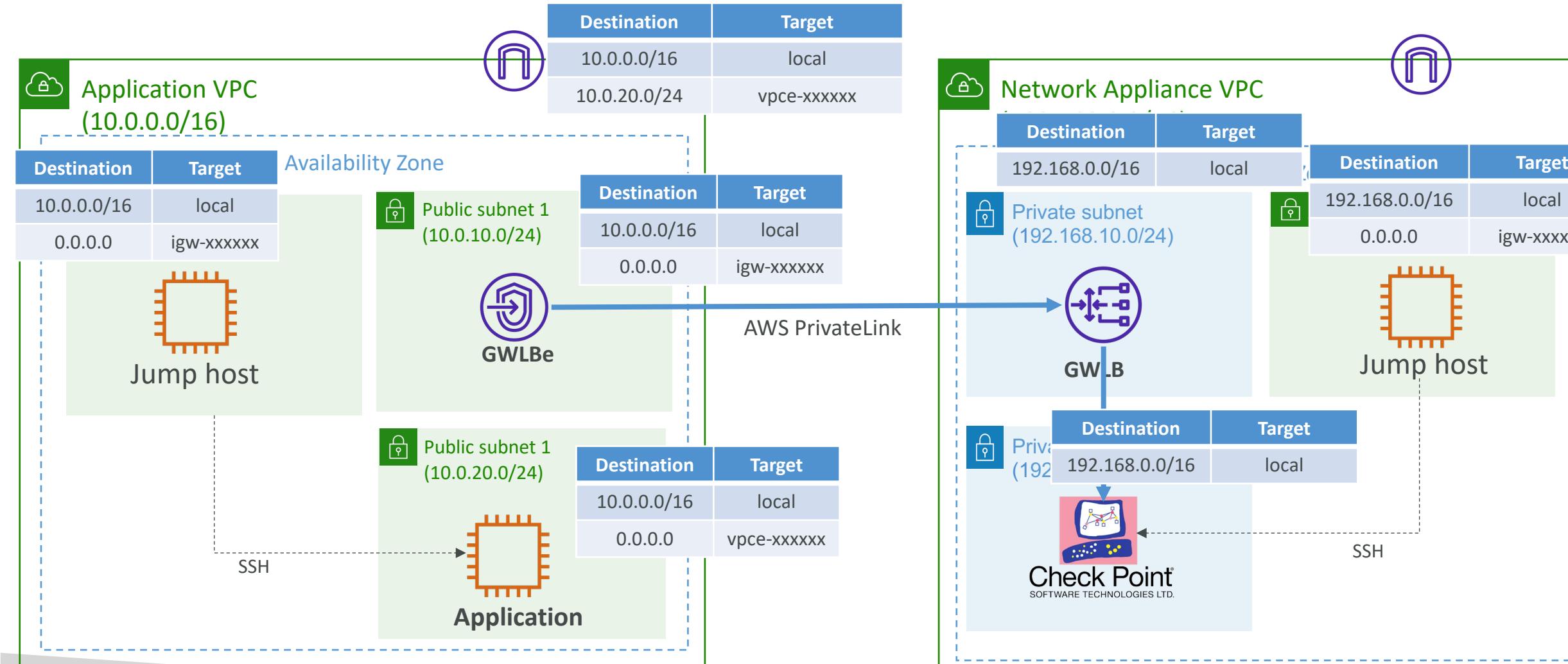
# Lab setup



# Lab: Network architecture



# Lab: Network setup



# Steps - I

- I. Create 2 VPCs – Application VPC and Network Appliance VPC
2. Create 3 subnets and corresponding 3 route tables in Application VPC
  - I. Public subnet for jump host instance
  2. Public subnet for Application instance
  3. Public subnet for Gateway load balancer endpoint
3. Create 3 subnets and corresponding 3 route tables in Network VPC
  - I. Public subnet for Jump host instance
  2. Private subnet for Gateway load balancer (ideally we should use multiple AZs)
  3. Private subnet for the Network appliance instance (ideally we should have multiple appliance instances across multiple AZs)

# Steps - 2

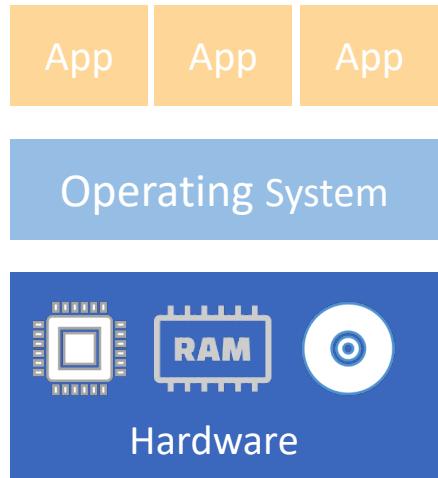
4. Launch jumps hosts in both the VPCs public subnets
5. Launch an Application host in the Application VPC public subnet
6. Launch Checkpoint CloudGuard instance in Network Appliance VPC
  - I. For this you need to first subscribe to the AWS marketplace AMI.
  2. SG should allow Traffic on GENEVE port 6081 from the GWLB IPs (192.168.10.0/24)
7. Create GWLB and Target group
  - I. Add the Checkpoint instance as a target with healthcheck on TCP port 443
8. Create VPC endpoint service in Network Appliance VPC using GWLB
9. Create VPC endpoint in Application VPC for the GWLB service created above
10. Configure all the route tables as shown in the network diagram

# Steps - 3

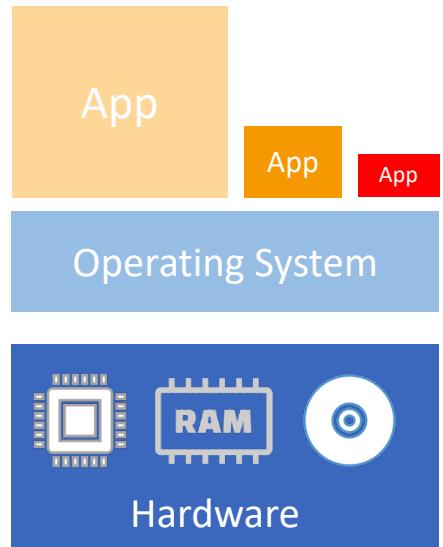
- I 0. SSH to jump host in Network Appliance VPC and from there SSH to Checkpoint instance (use the username: admin)
  - Get into the Expert mode: \$expert
  - Run the tcpdump command: \$tcpdump –nwv ‘port 6081’
- I I. SSH to jump host in Application VPC and from there SSH to application host
  - From Application host: \$ping [www.amazon.com](http://www.amazon.com)
- I 2. You should see the ICMP traffic captured in the checkpoint instance tcpdump

# Containers, Microservices & Kubernetes

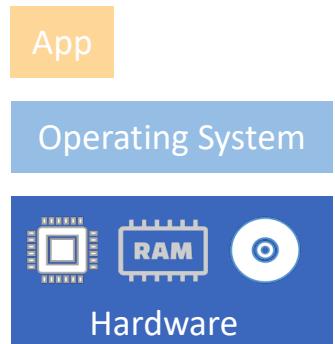
# Why containers?



Traditional Deployment on Physical Servers



Resource allocation issue



Hardware Utilization

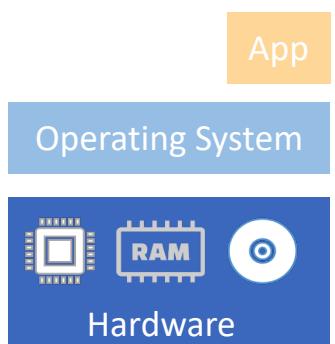
Operating system overhead

Application Portability

Deployment Automation

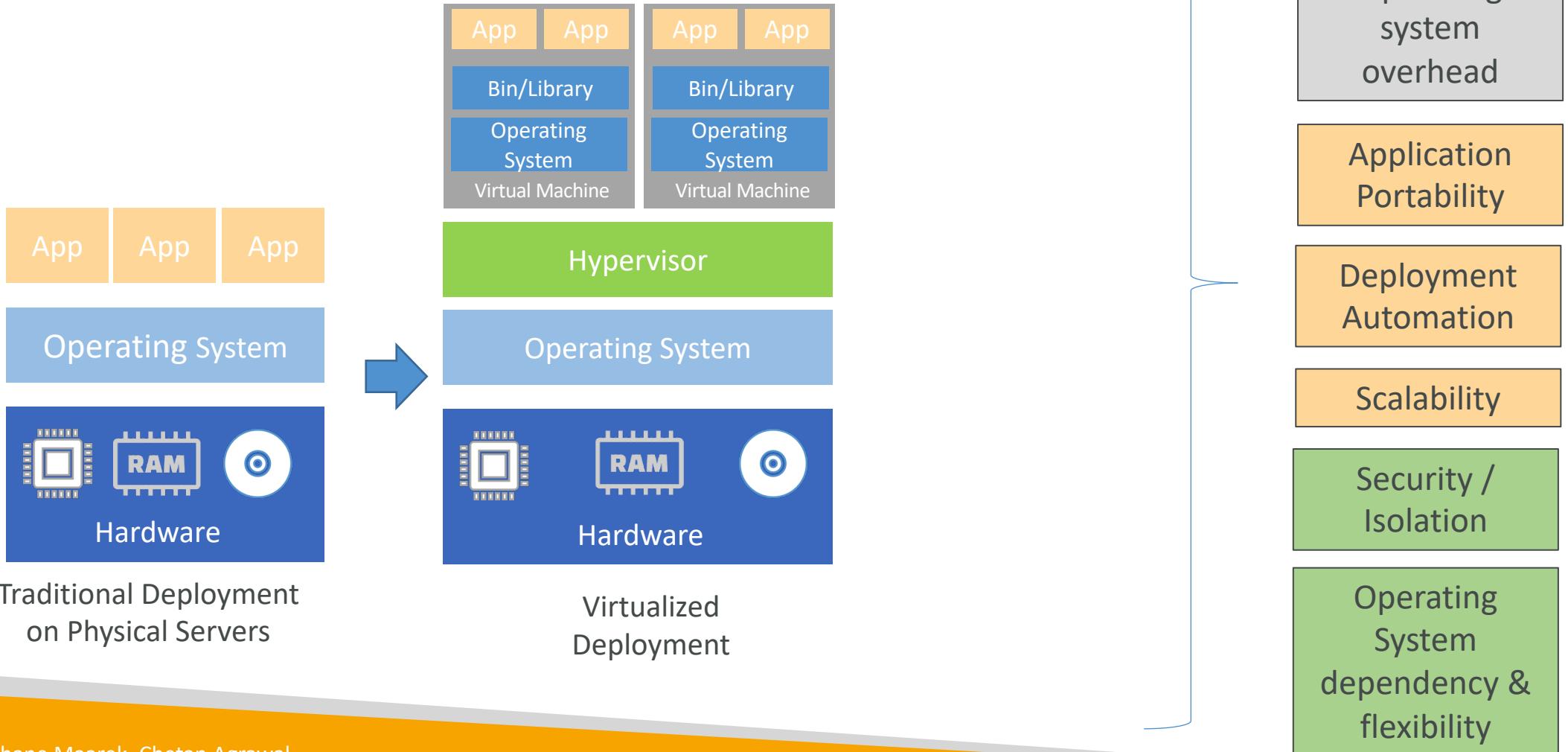
Scalability

Security / Isolation

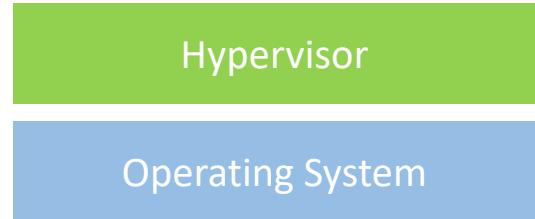
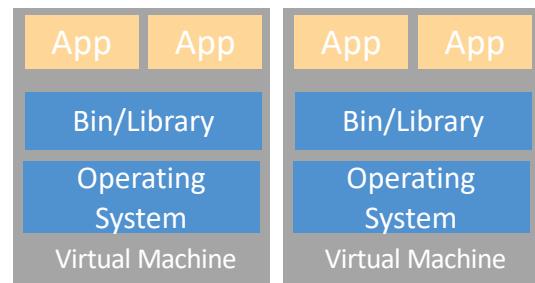


Operating System dependency & flexibility

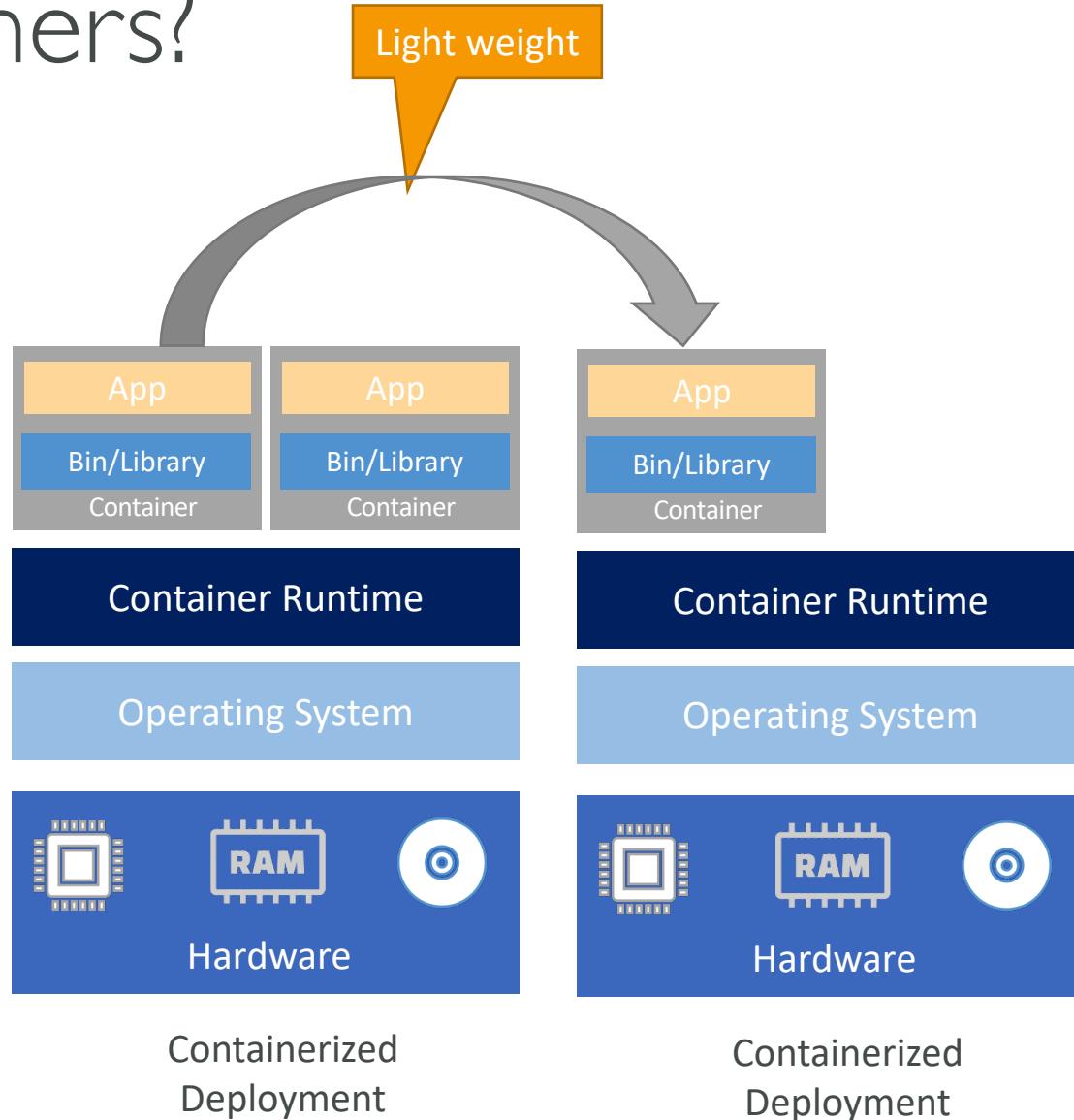
# Why containers?



# Why containers?

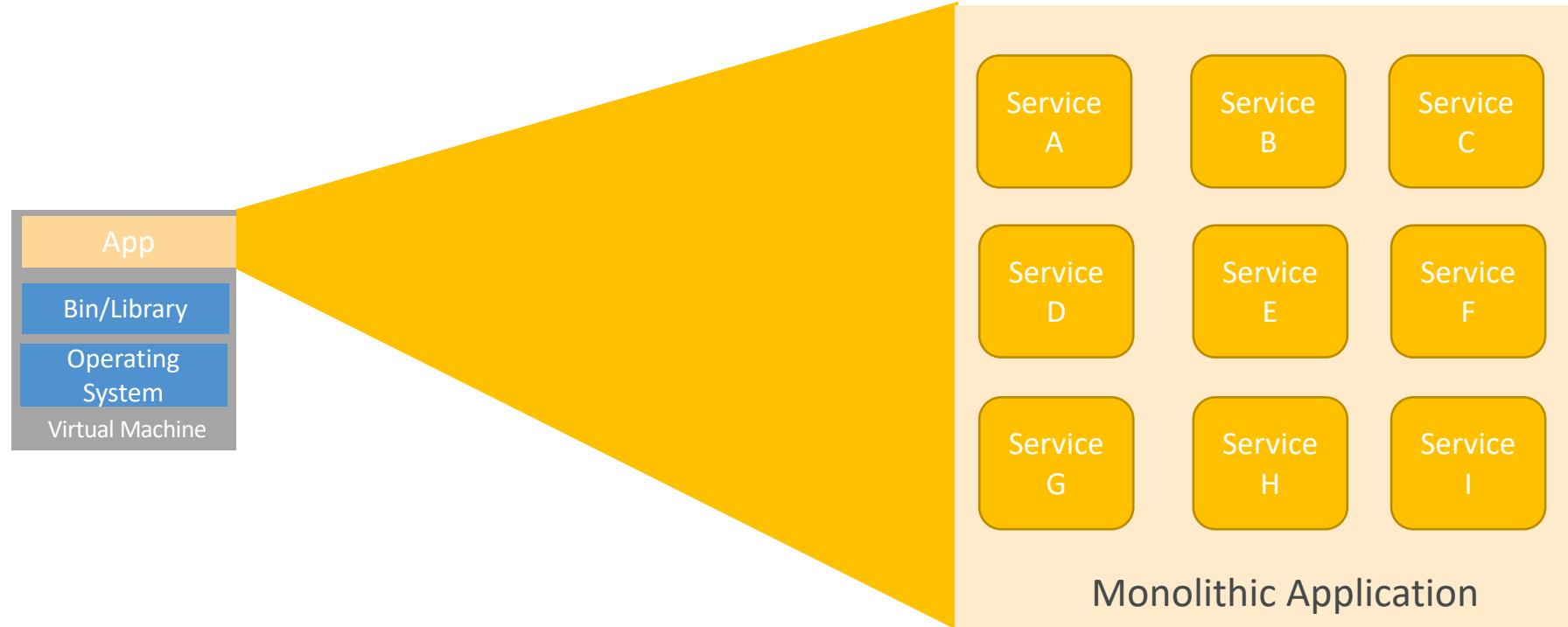


Virtualized Deployment

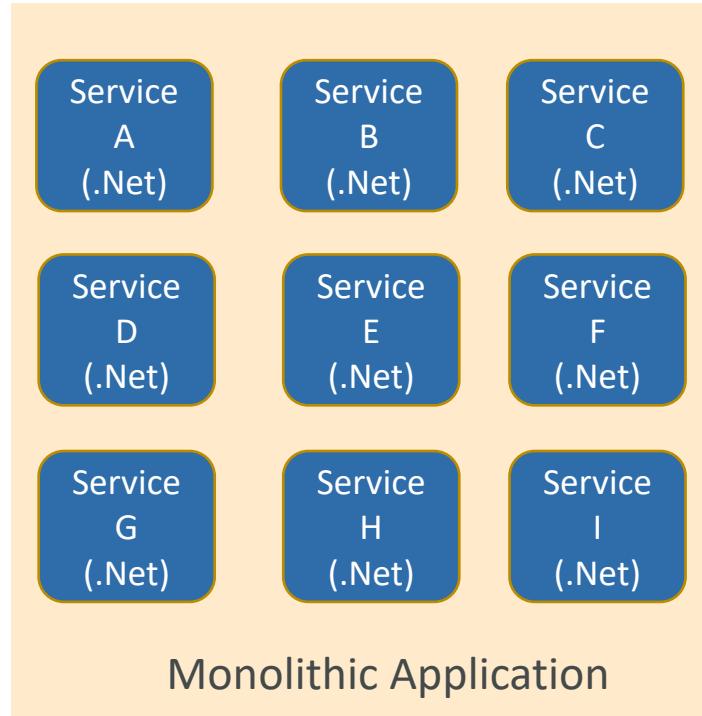
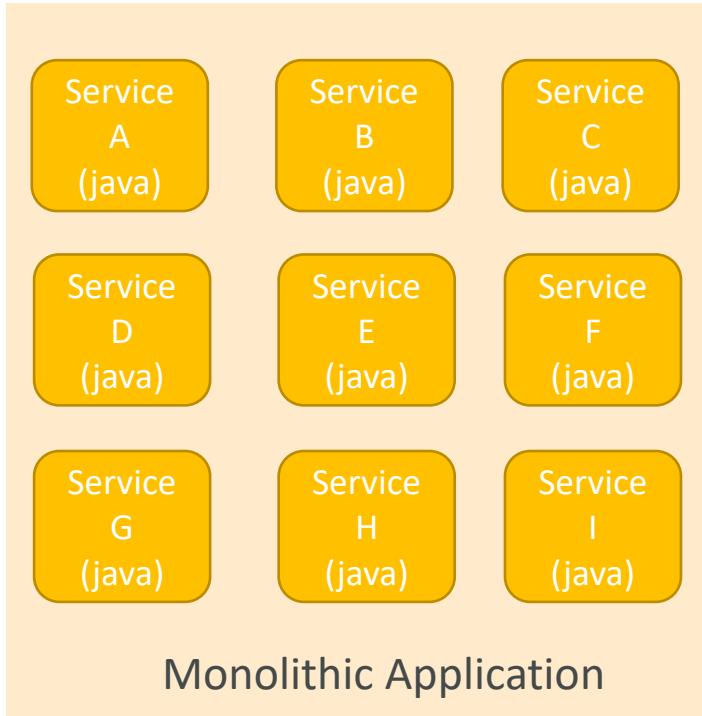


- Hardware Utilization
- Operating system overhead
- Application Portability
- Deployment Automation
- Scalability
- Security / Isolation
- Operating System dependency & flexibility

# Why microservices?



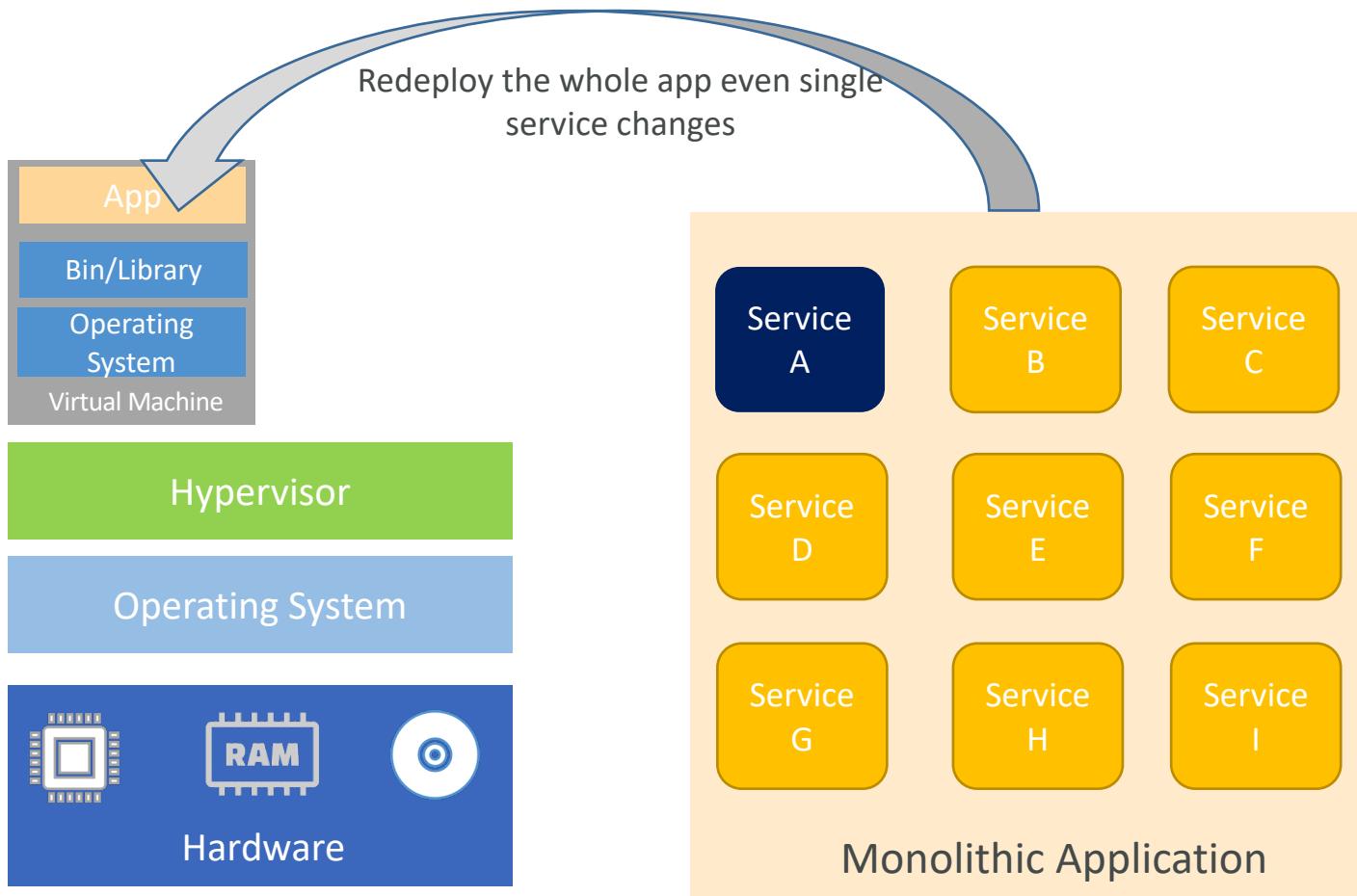
# Why microservices?



Lack of flexibility with choice of development language & framework

All services use common underlying libraries – changes in one service may break another service

# Why microservices?

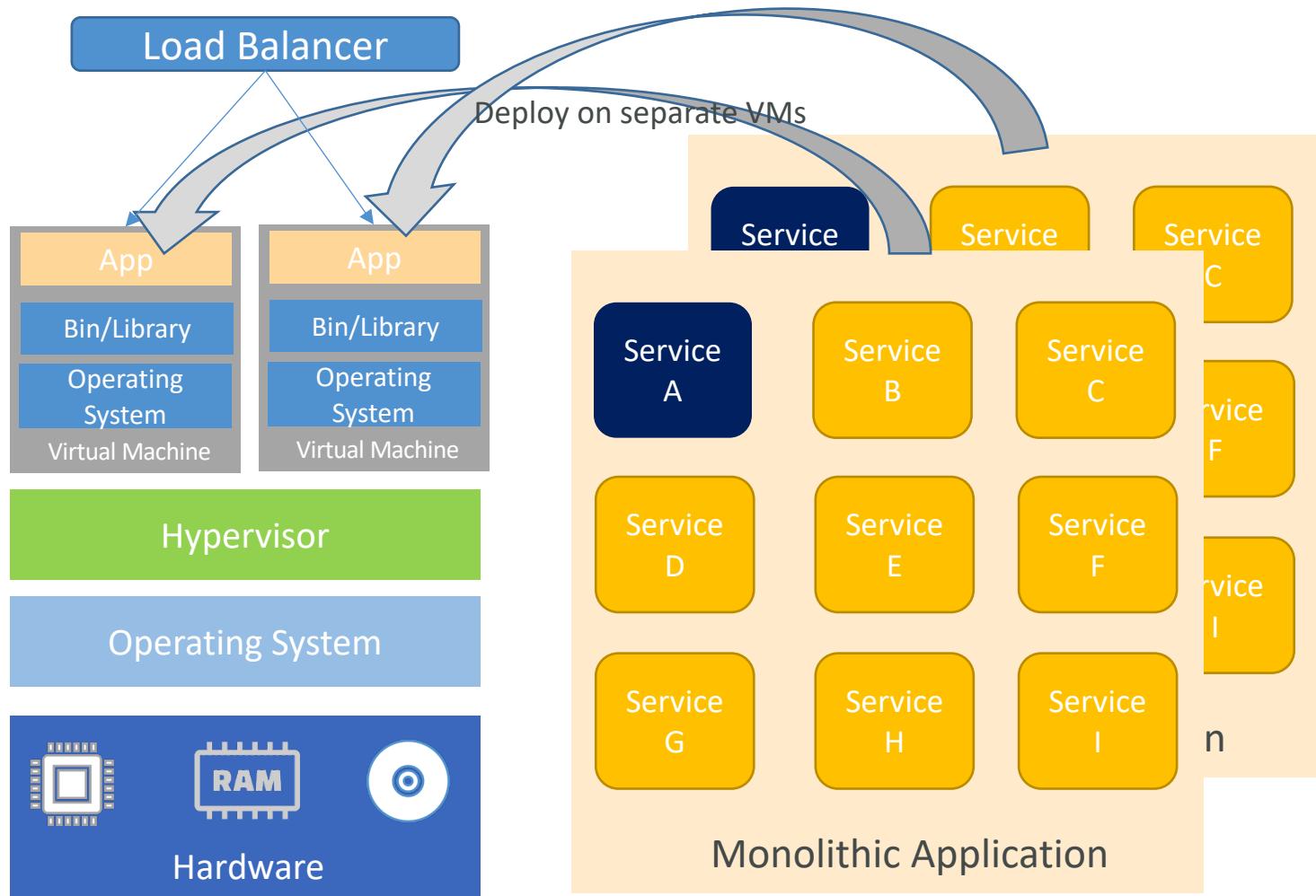


Lack of flexibility with choice of development language & framework

All services use common underlying libraries – changes in one service may break another service

A change in a single service needs a deployment of an entire application

# Why microservices?



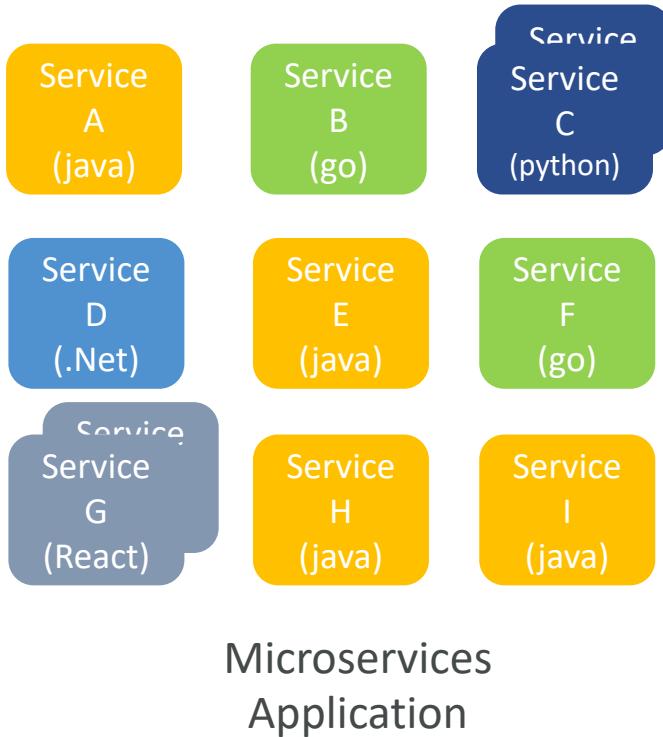
Lack of flexibility with choice of development language & framework

All services use common underlying libraries – changes in one service may break another service

A change in a single service needs a deployment of an entire application

Not possible to scale individual service. Need to create another instance of whole application.

# Welcome to Microservices !



Lack of flexibility with choice of development language & framework

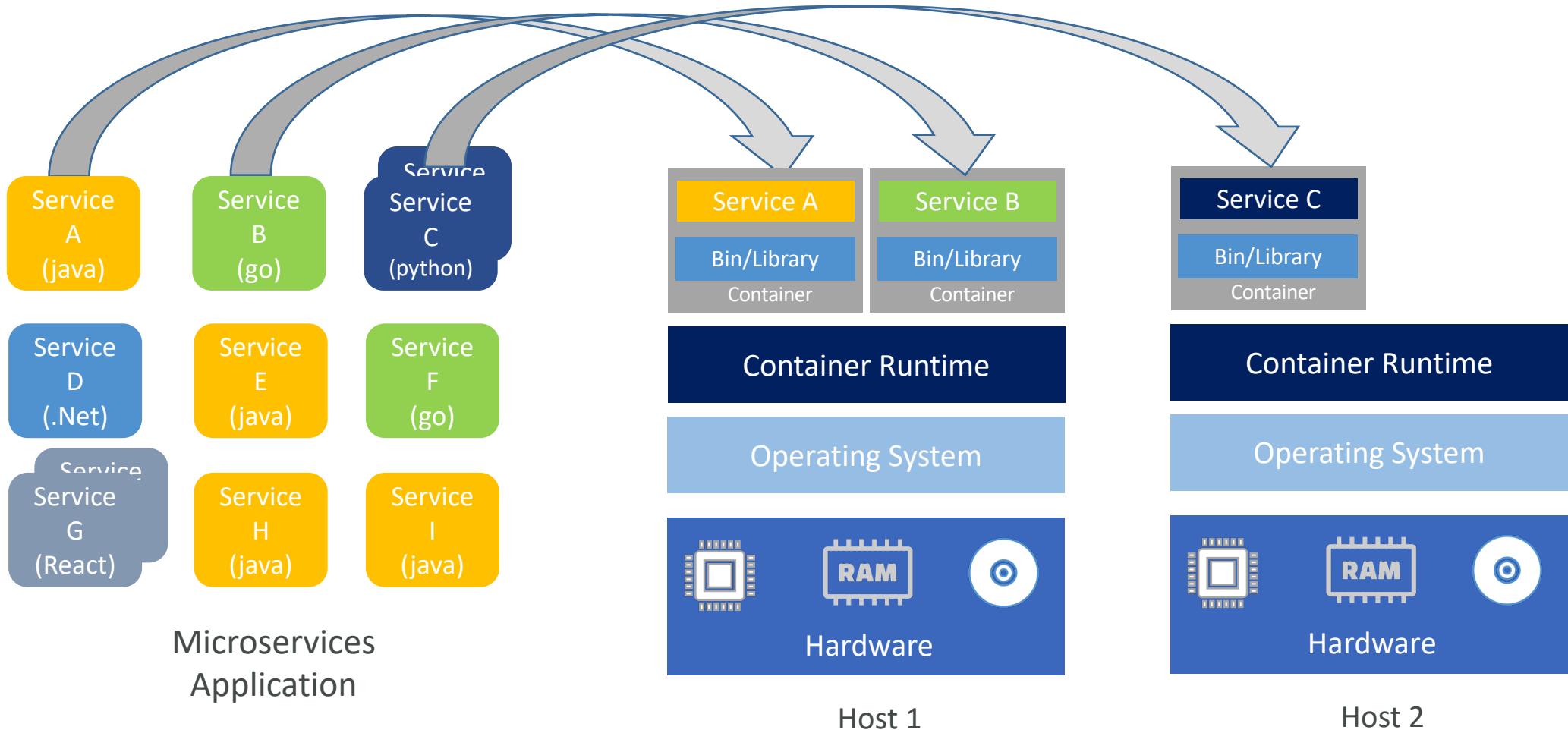
All services **do not** use common underlying libraries – changes in one service **does not** break another service

A change in a single service **does not need** a deployment of an entire application

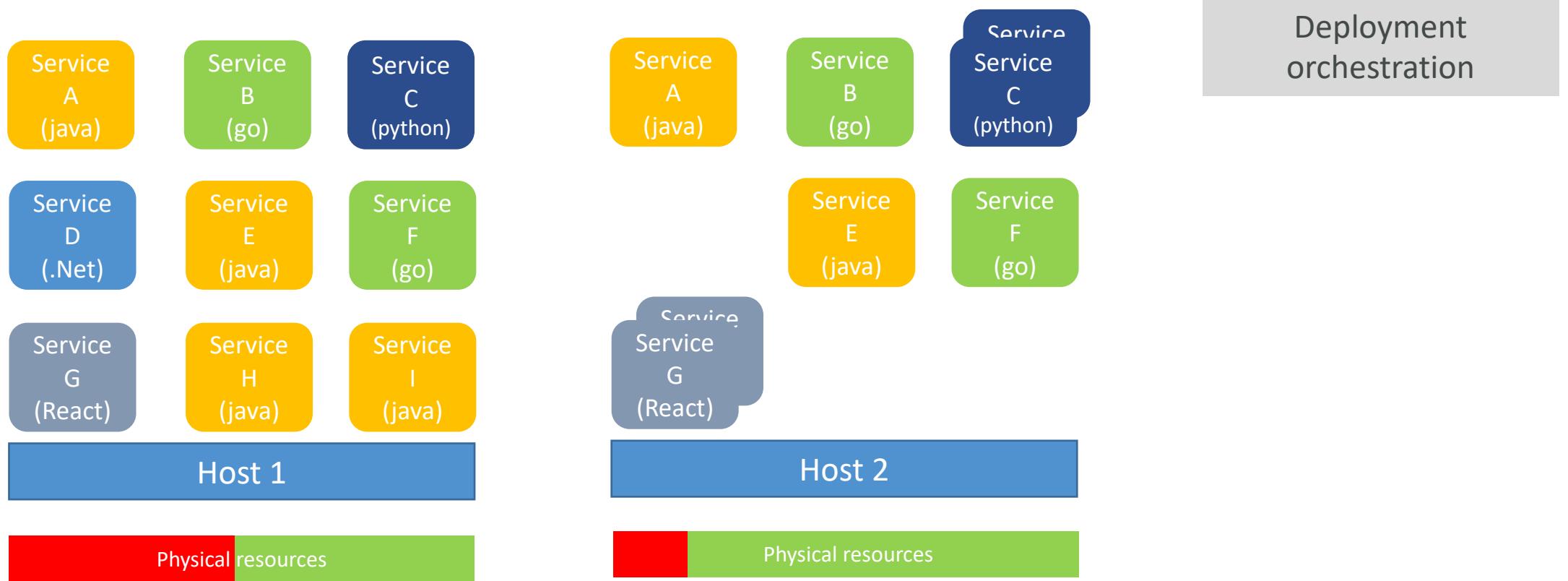
Not possible to scale individual service.  
**No** Need to create another instance of whole application.

But where should we run Microservices? Physical HW ? VM ? Containers ?

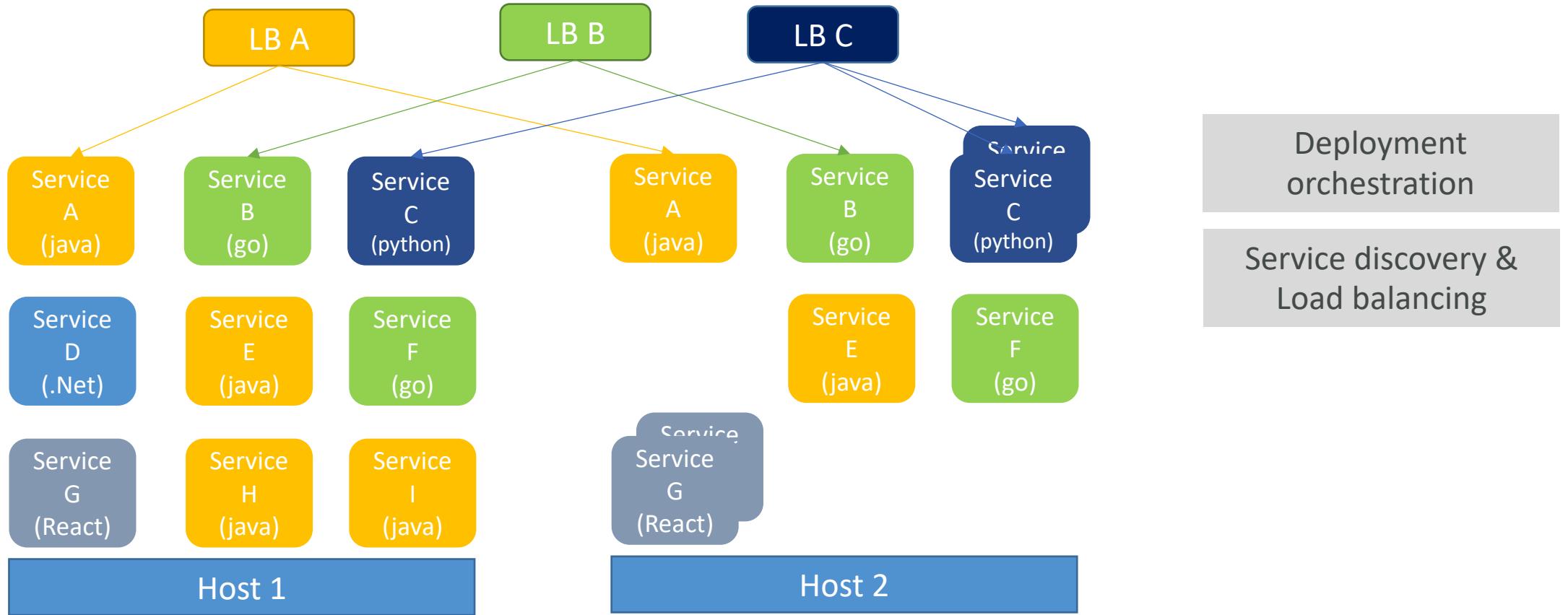
# Microservices & Containers



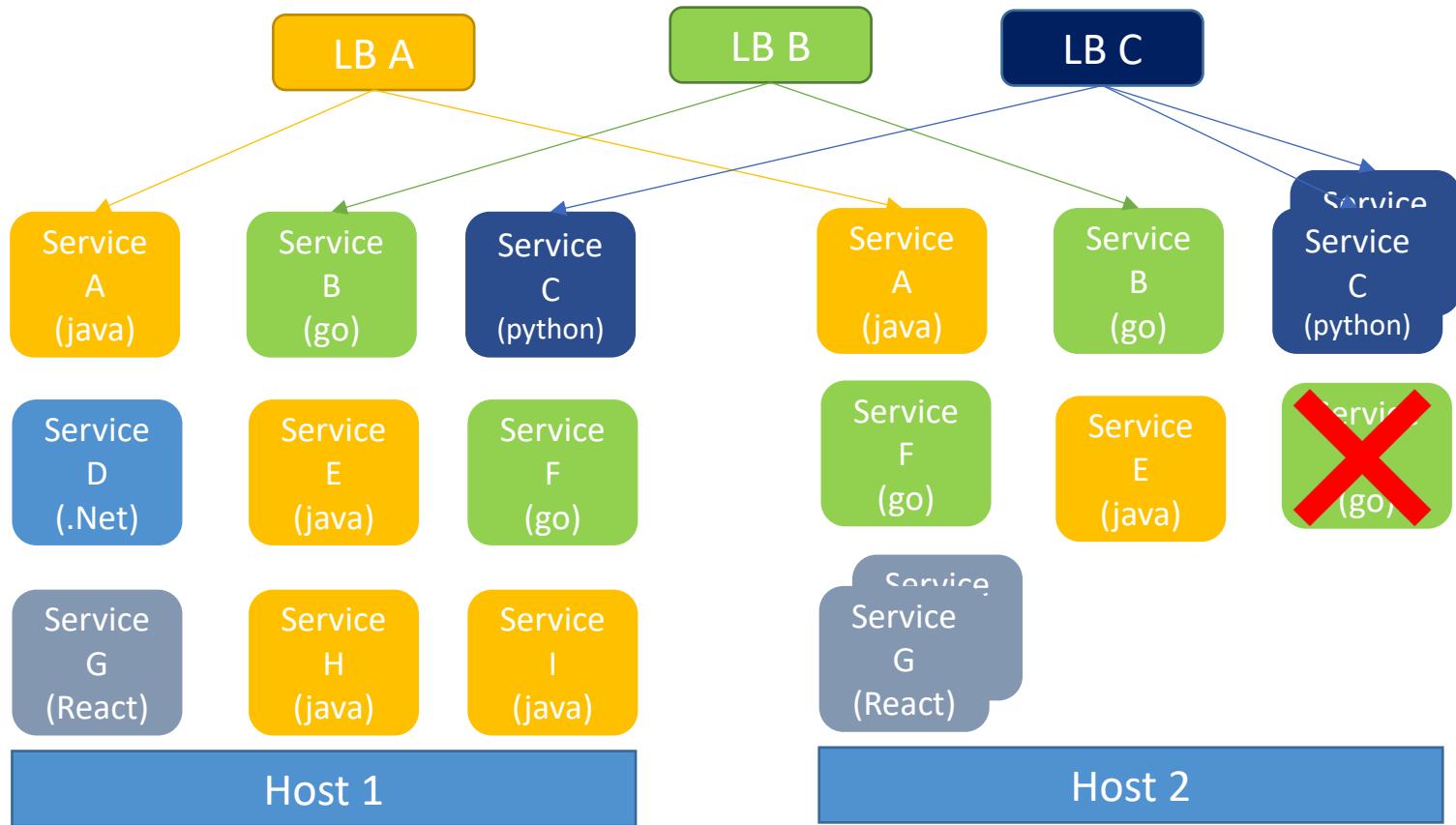
# Microservices+containers needs..



# Microservices+containers needs..



# Microservices+containers needs..

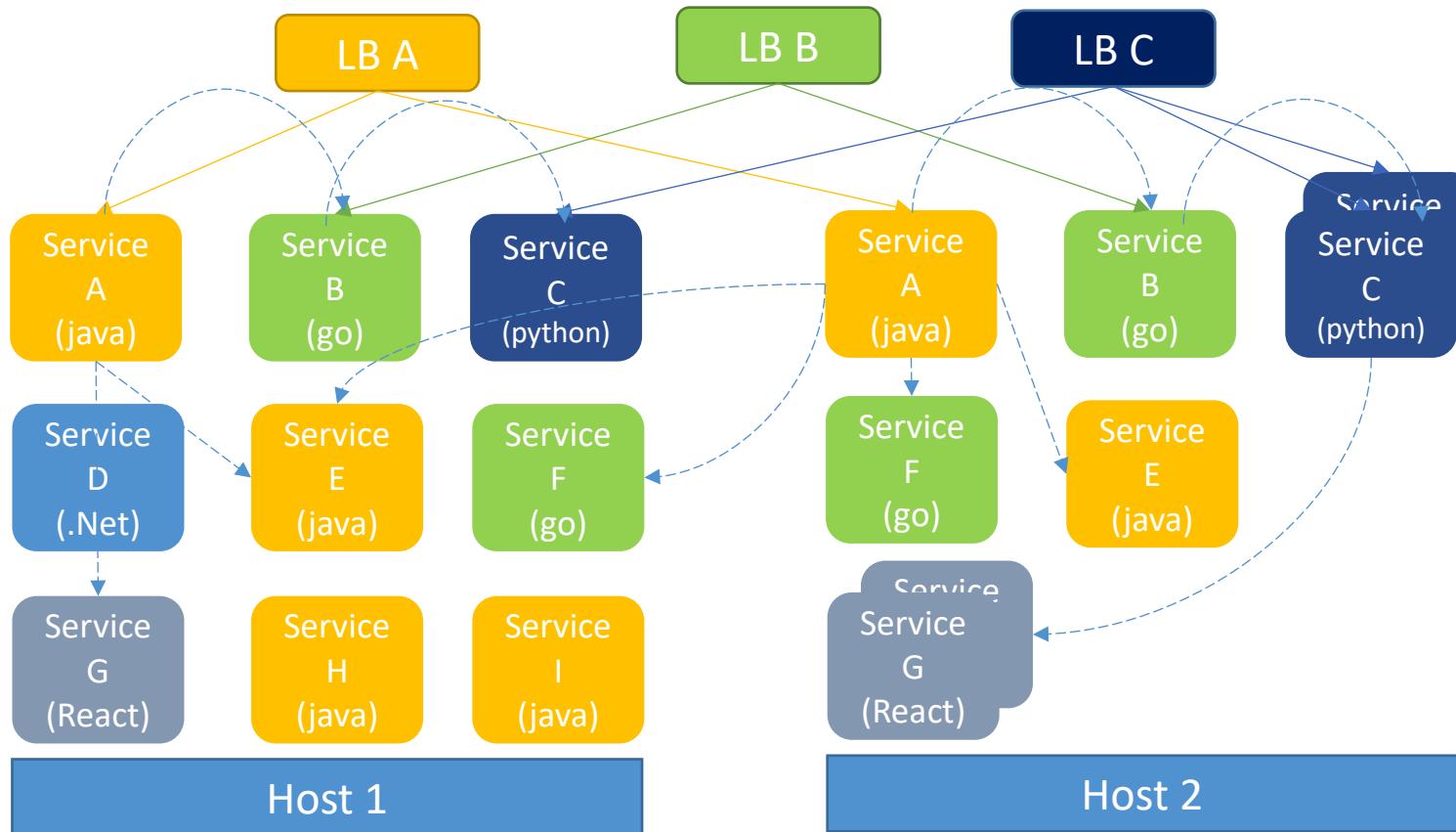


Deployment  
orchestration

Service discovery &  
Load balancing

Self-healing

# Microservices+containers needs..



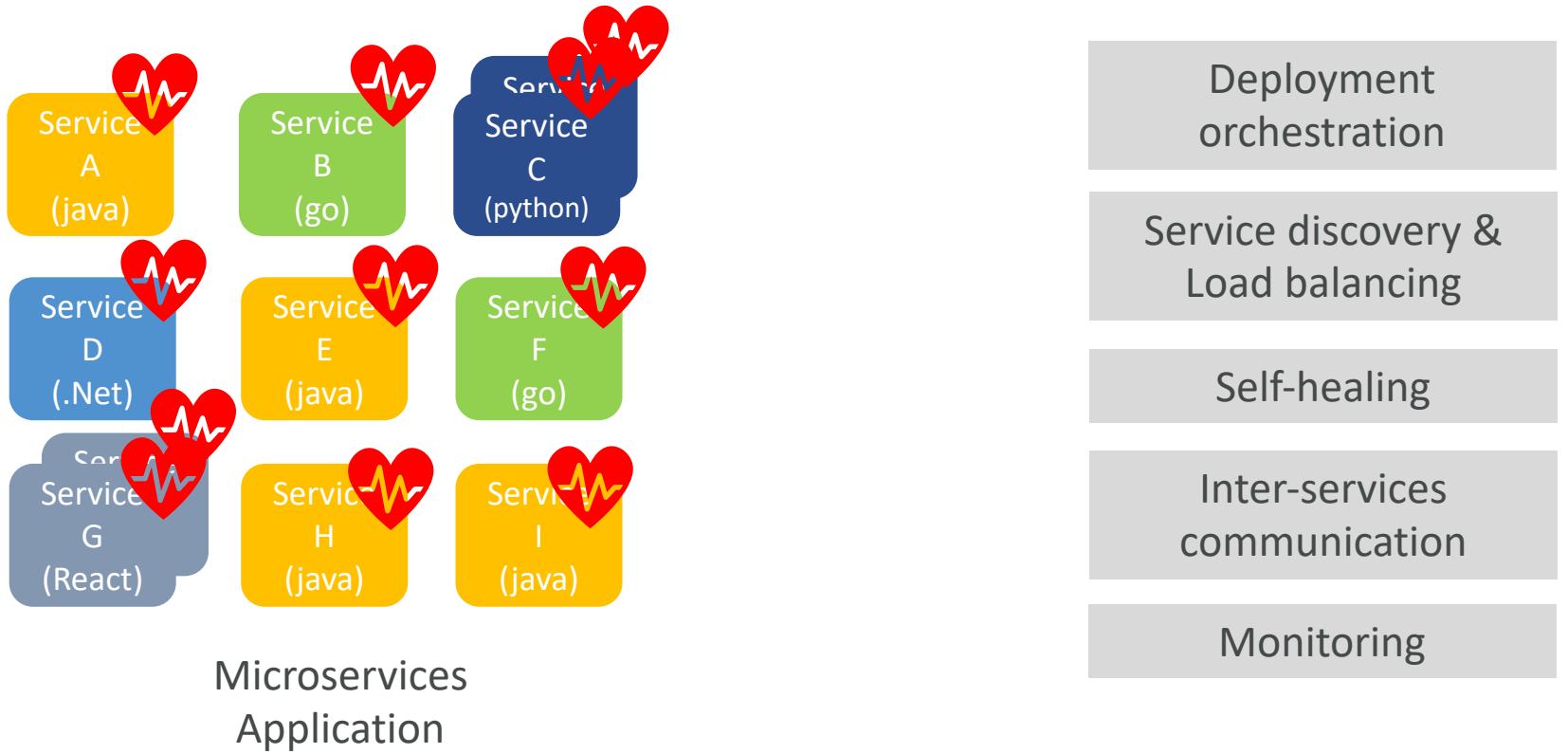
Deployment  
orchestration

Service discovery &  
Load balancing

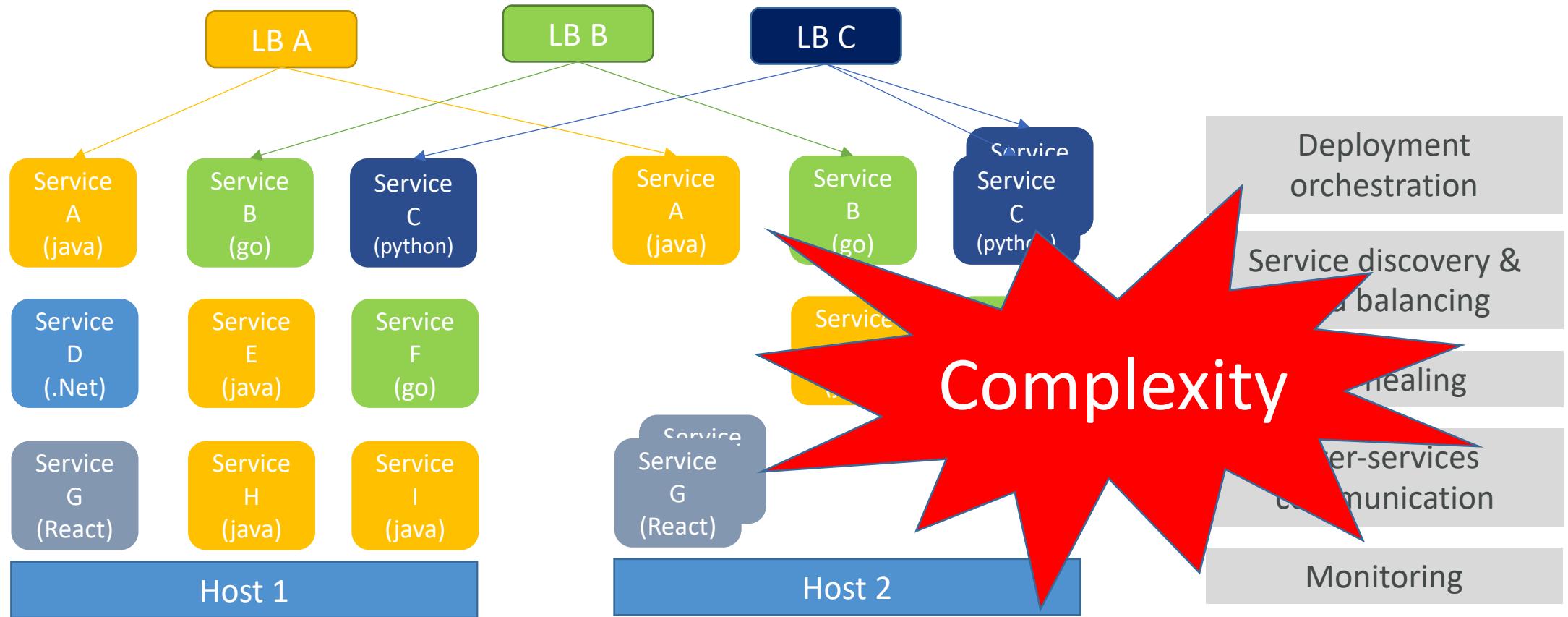
Self-healing

Inter-services  
communication

# Microservices+containers needs..



# Putting it together..



# Welcome to Kubernetes

*Kubernetes is a portable, extensible, open source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation.*

- Service discovery and load balancing
- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing
- Secret and configuration management
- Storage orchestration

+Much more..



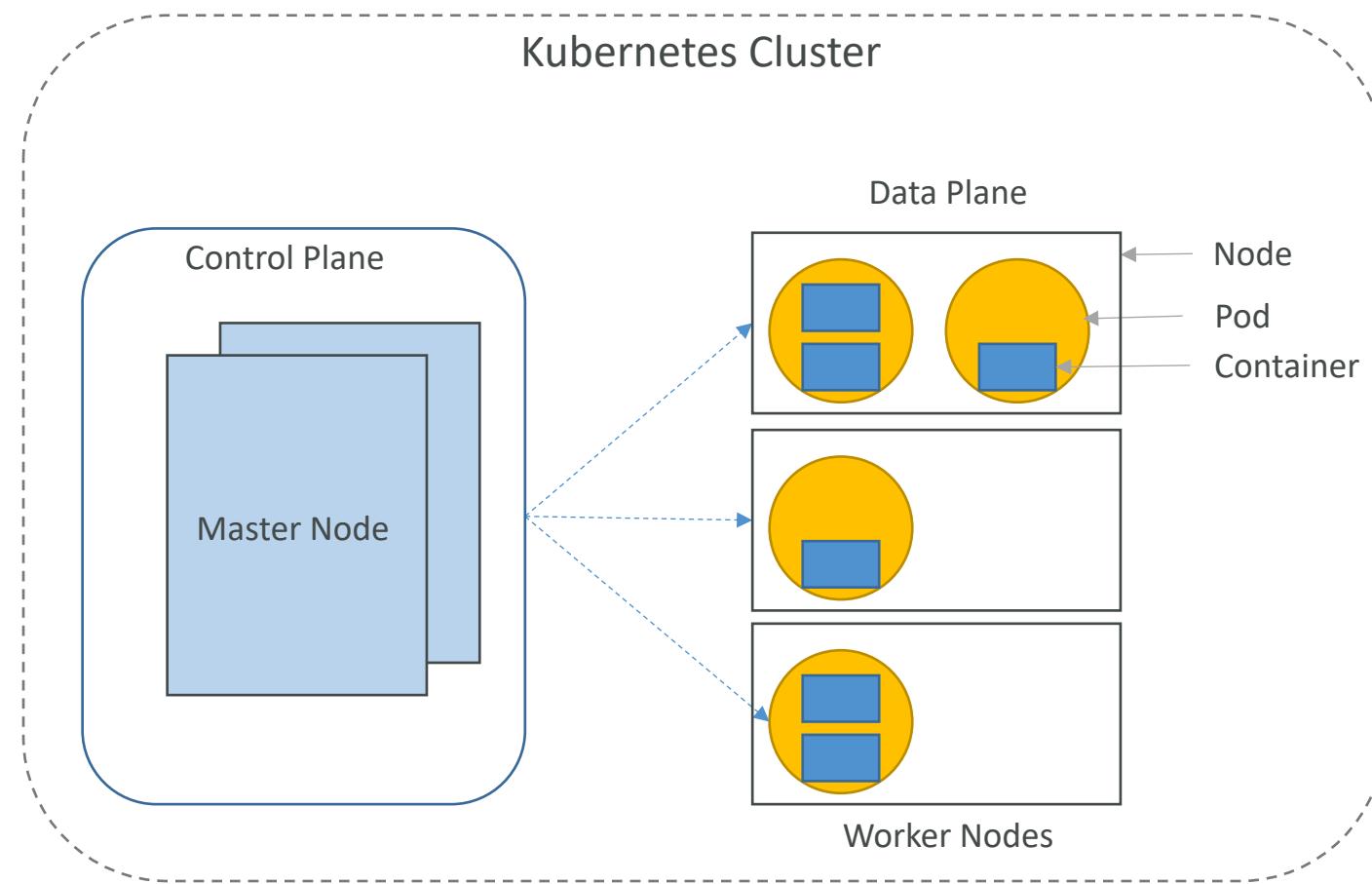
K u b e r n e t e s  
K \_ \_ \_ 8 \_ \_ \_ s

<https://kubernetes.io/docs/concepts/overview/>

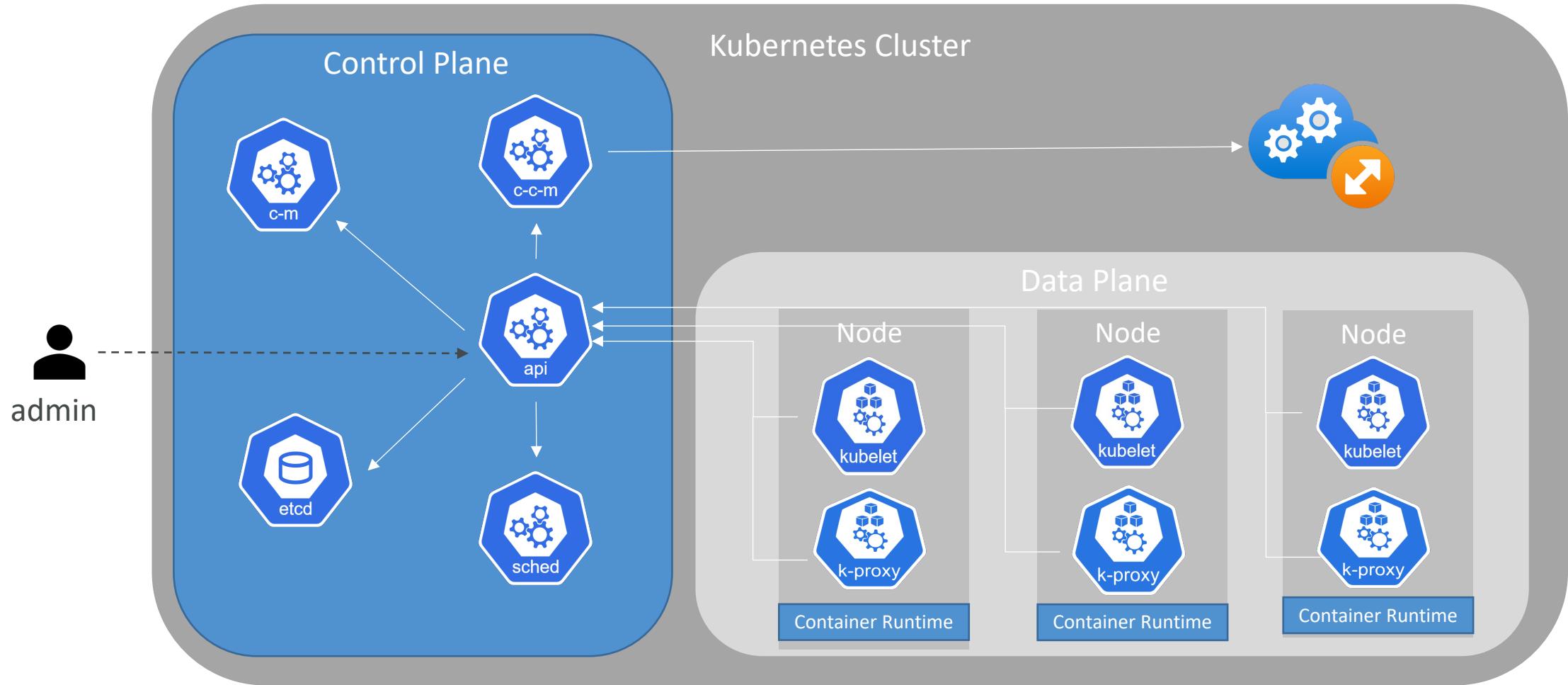
# Kubernetes Architecture

# Kubernetes building blocks

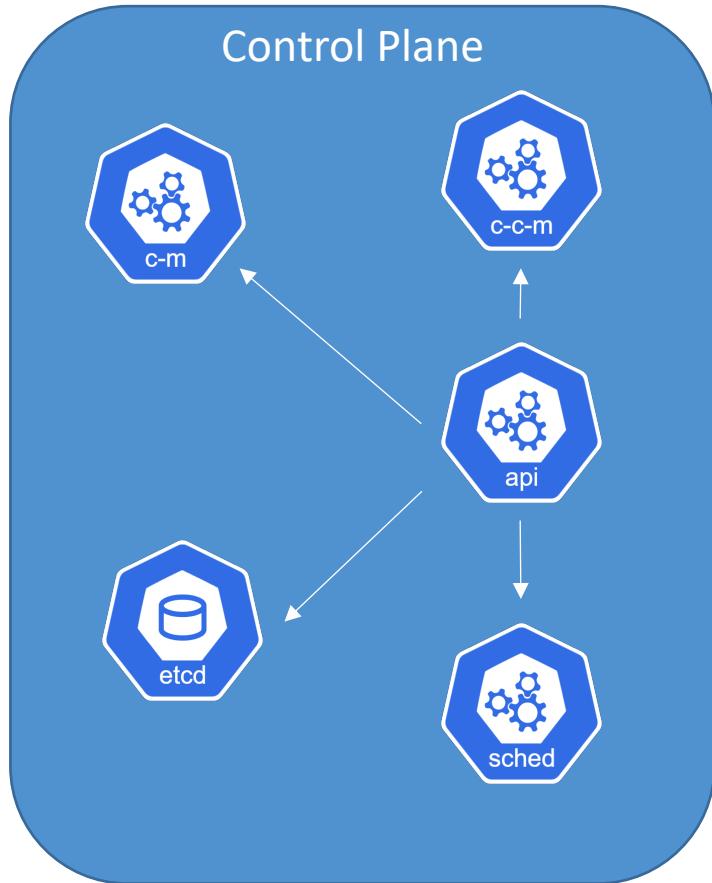
- Kubernetes Cluster consists of Control Plane and Data Plane
- Control plane consists of set of control processes hosted on master nodes
- Data plane consist of set of worker nodes called Nodes
- Nodes host the Pods
- A Pod represents a group of one or more application containers



# Kubernetes architecture

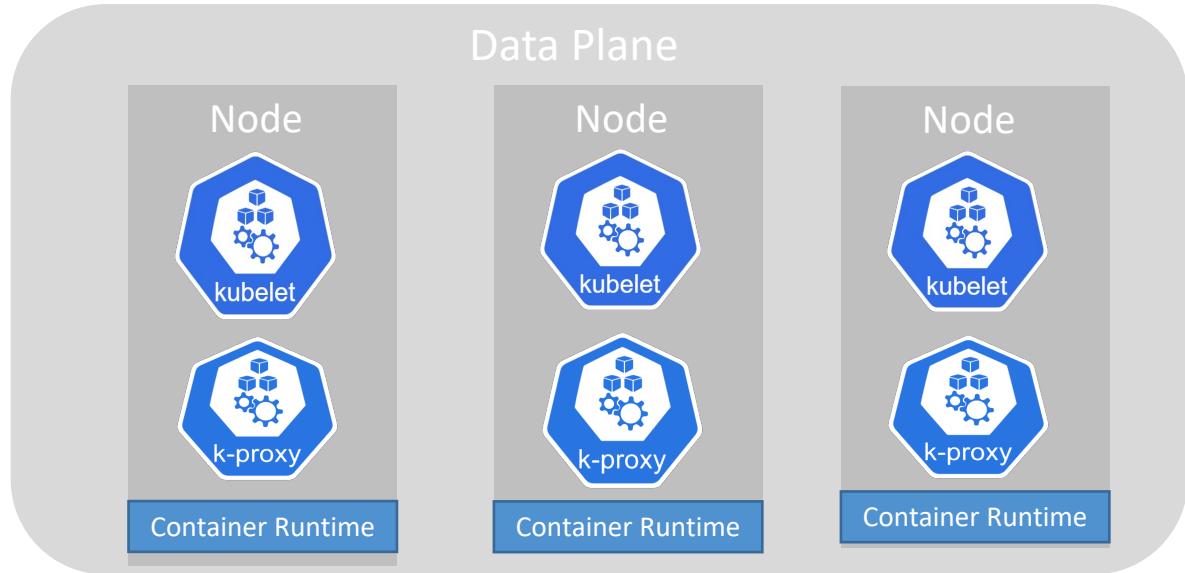


# Control Plane Components



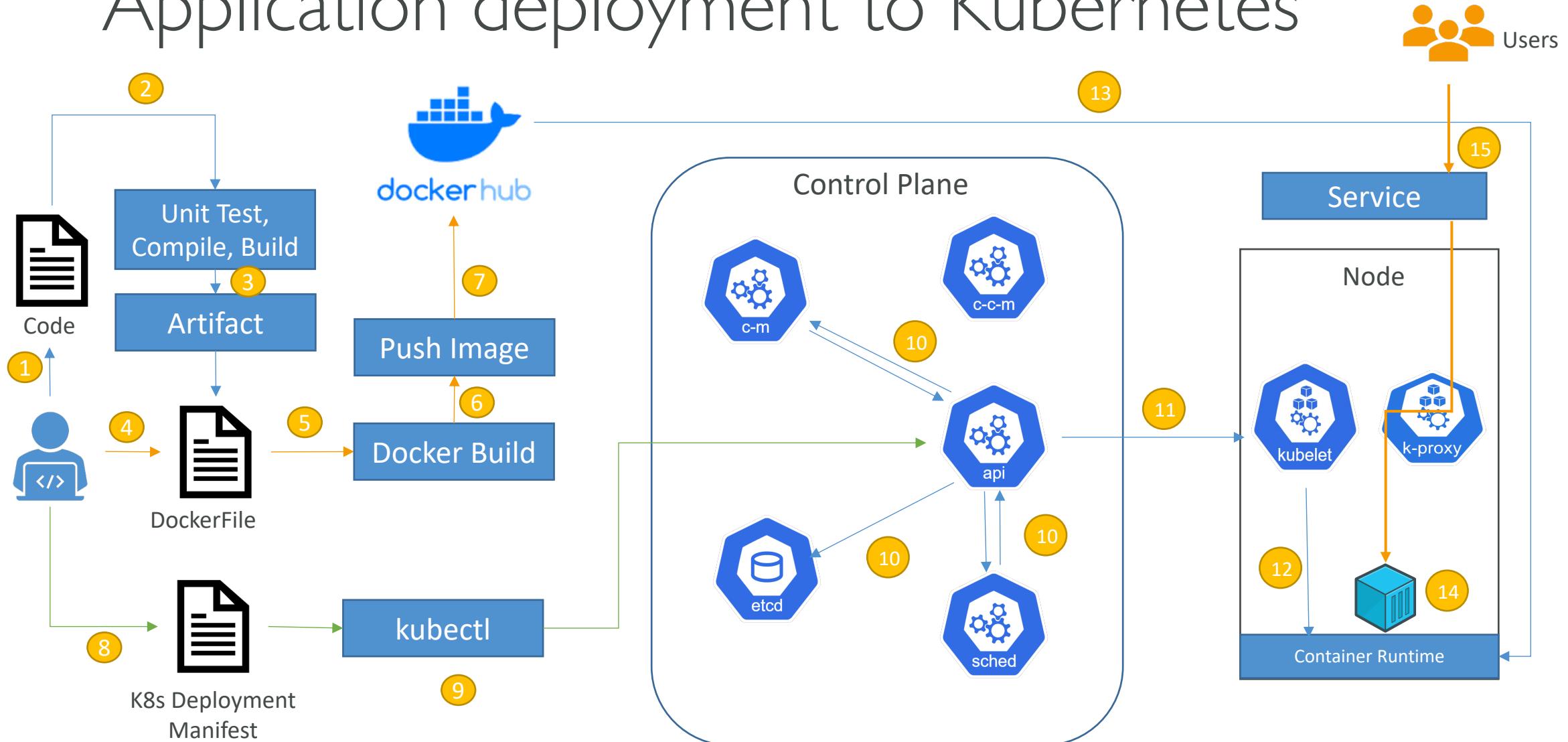
- kube-apiserver
  - Exposes Kubernetes APIs. It's a frontend for the Kubernetes control plane.
- etcd
  - Key-value store used as Kubernetes backing store for all cluster data.
- kube-scheduler
  - Watches for newly created Pods with no assigned node, and selects a node for them to run on.
- kube-controller-manager
  - Runs controller processes such as Node controller, Replication controller, Namespace controller, Job Controller, EndpointSlice controller etc.
- cloud-controller-manager
  - Links Kubernetes cluster into cloud provider's API such as Node controller for determining if node (instance) is deleted in the cloud, Service controller for cloud load balancer etc.

# Kubernetes Data plane components



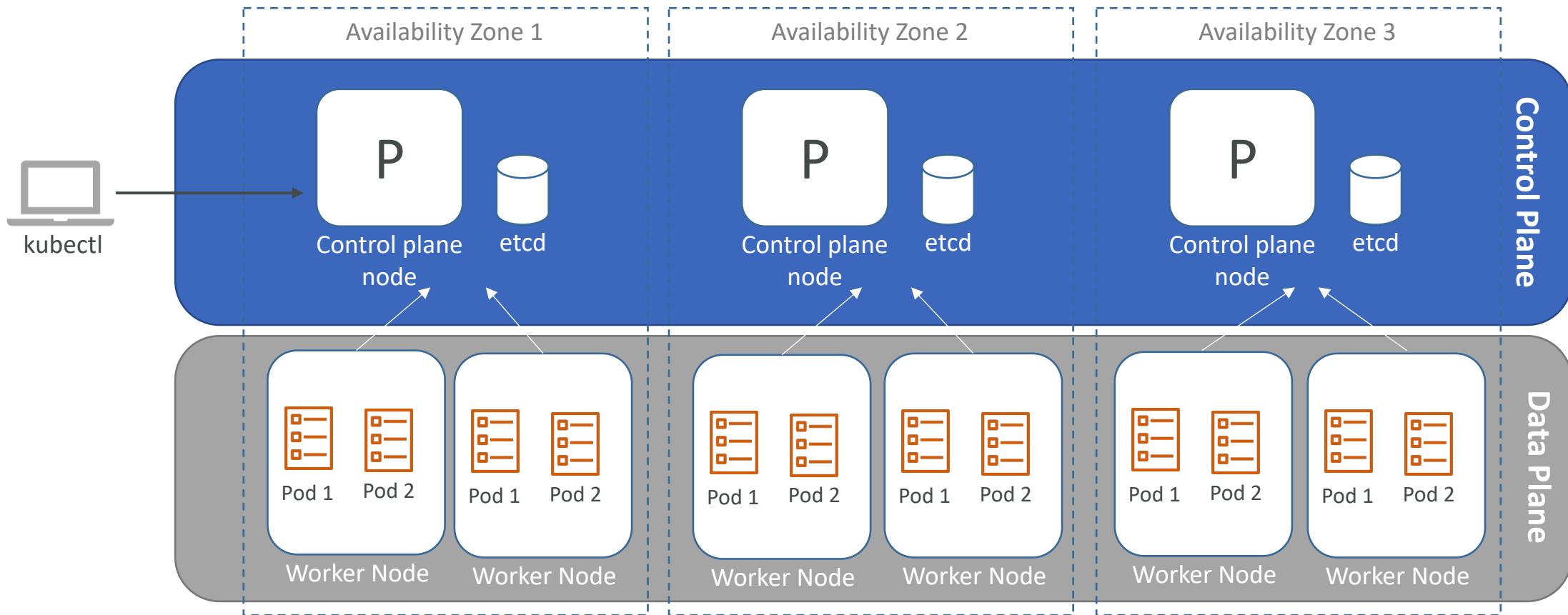
- Node
  - hosts the pods (applications)
- kubelet
  - An agent that runs on each node in the cluster. It makes sure that containers are running in a Pod.
- kube-proxy
  - Enables network communication to Pods from network sessions inside or outside of your cluster
- Container Runtime
  - Responsible for running containers. Kubernetes supports container runtimes such as containerd, CRI-O, and any other implementation of the Kubernetes CRI

# Application deployment to Kubernetes

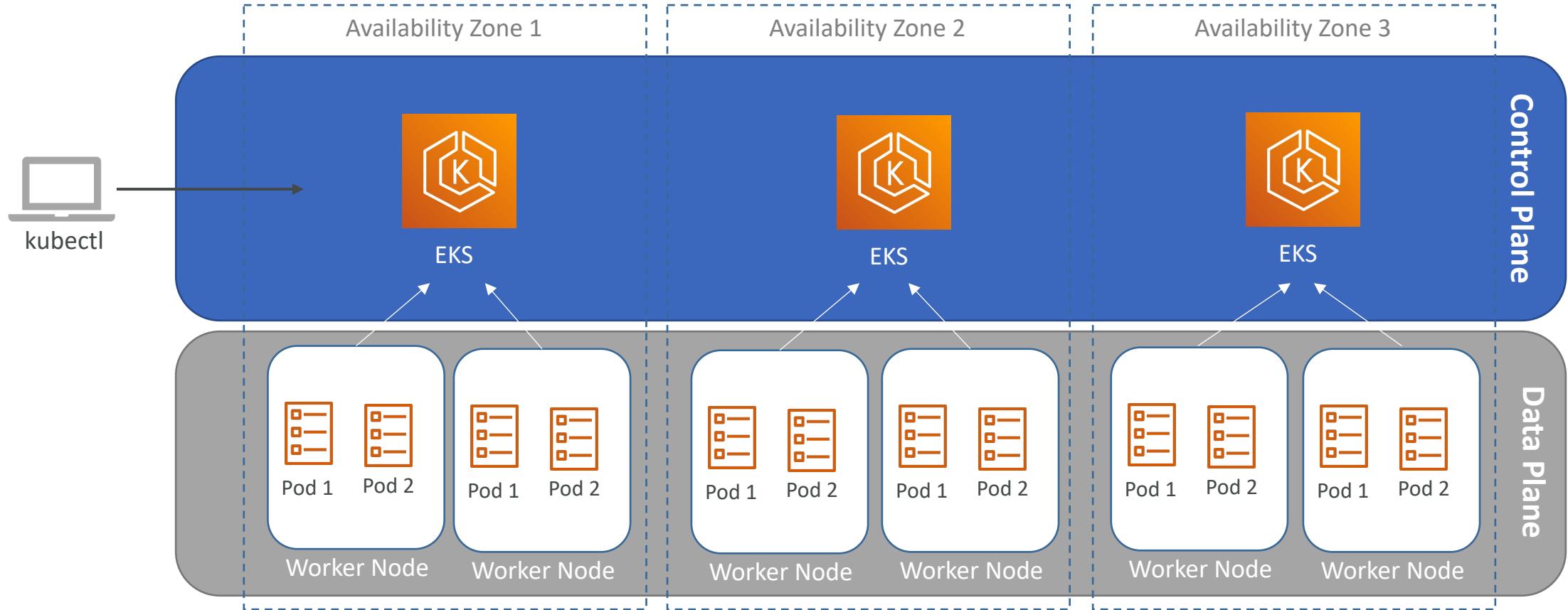


# Amazon EKS Architecture

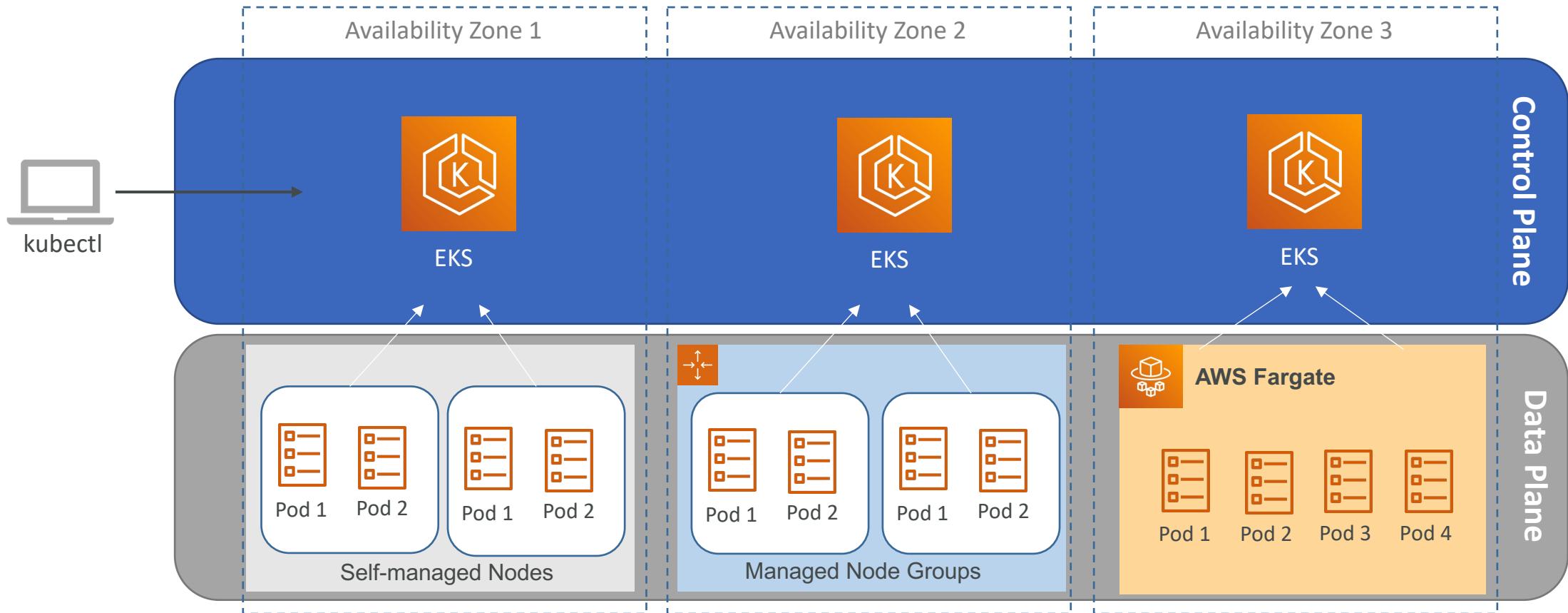
# Kubernetes Control plane & data plane in AWS



# EKS Control plane & data plane



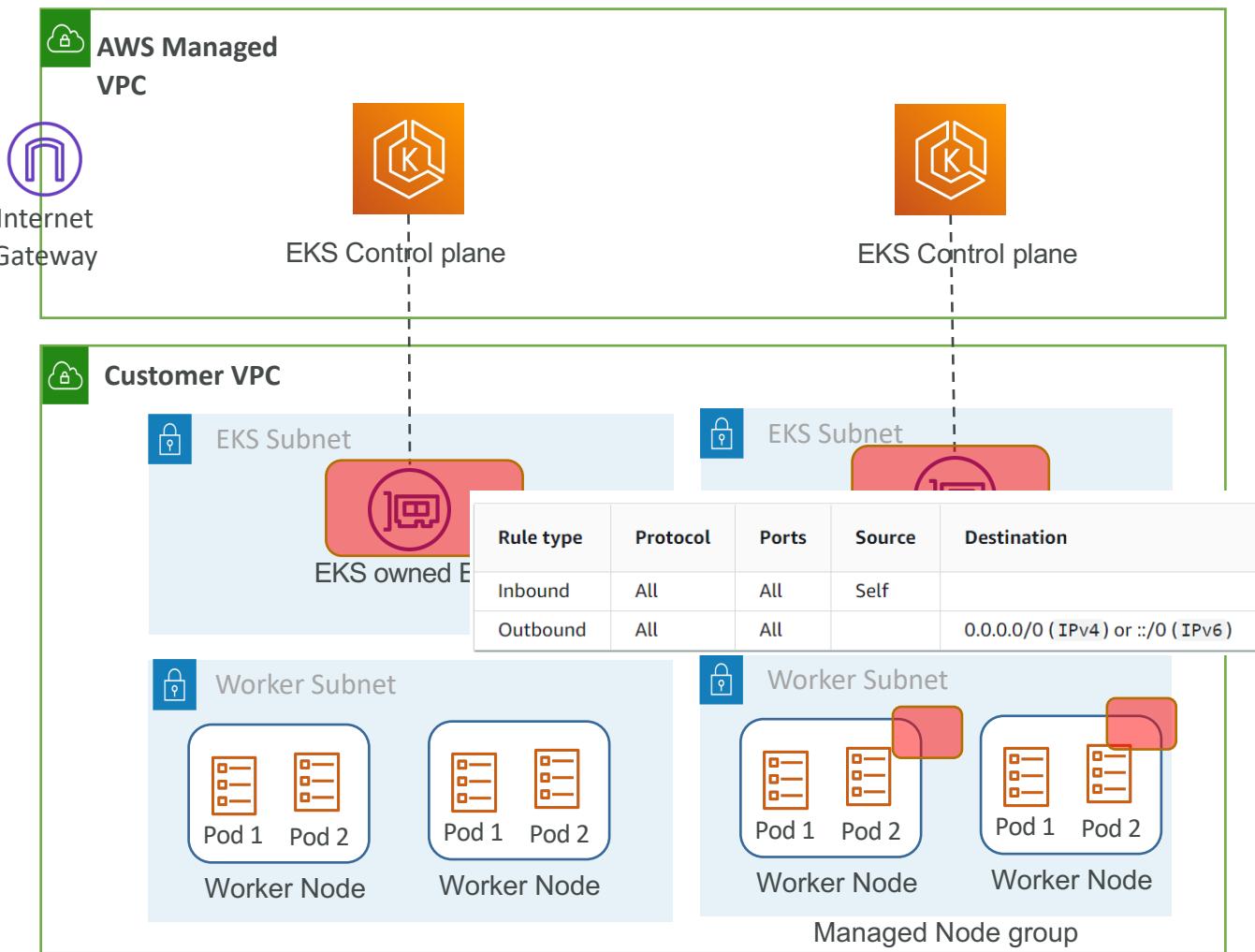
# EKS – Data plane hosting options



# Amazon EKS Cluster Networking

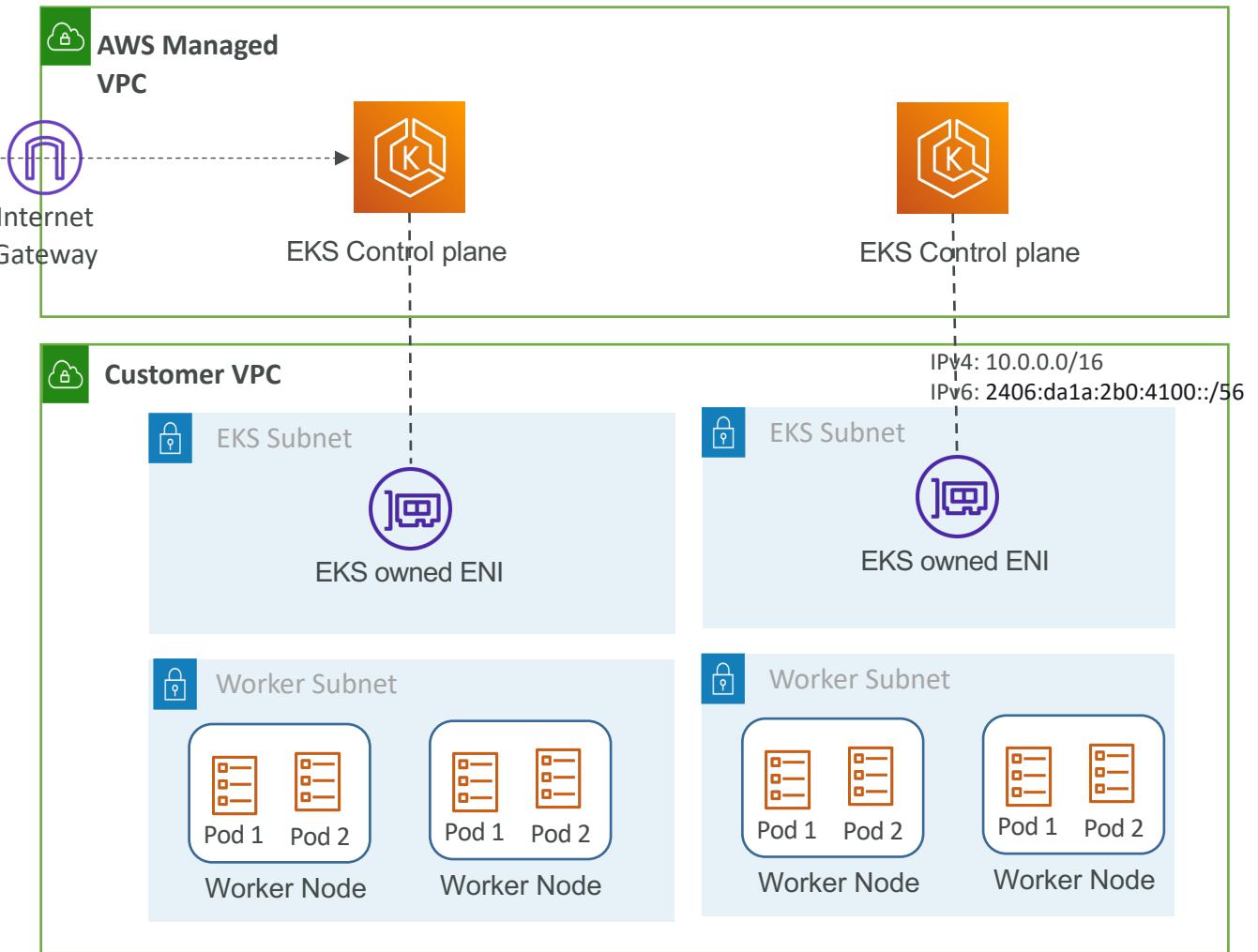
# EKS Networking

- EKS Control Plane is launched in the AWS managed account & VPC
- Data plane nodes are launched in Customer account & VPC
- EKS provisions 2-4 ENIs in the Customer VPC to enable the communication between Control plane and VPC.
- It's recommended to have separate subnets for EKS ENIs. EKS needs at least 6 IPs in each subnet (16 recommended)
- EKS creates and associates SG to these EKS owned ENIs (and also to Managed Group Nodes).

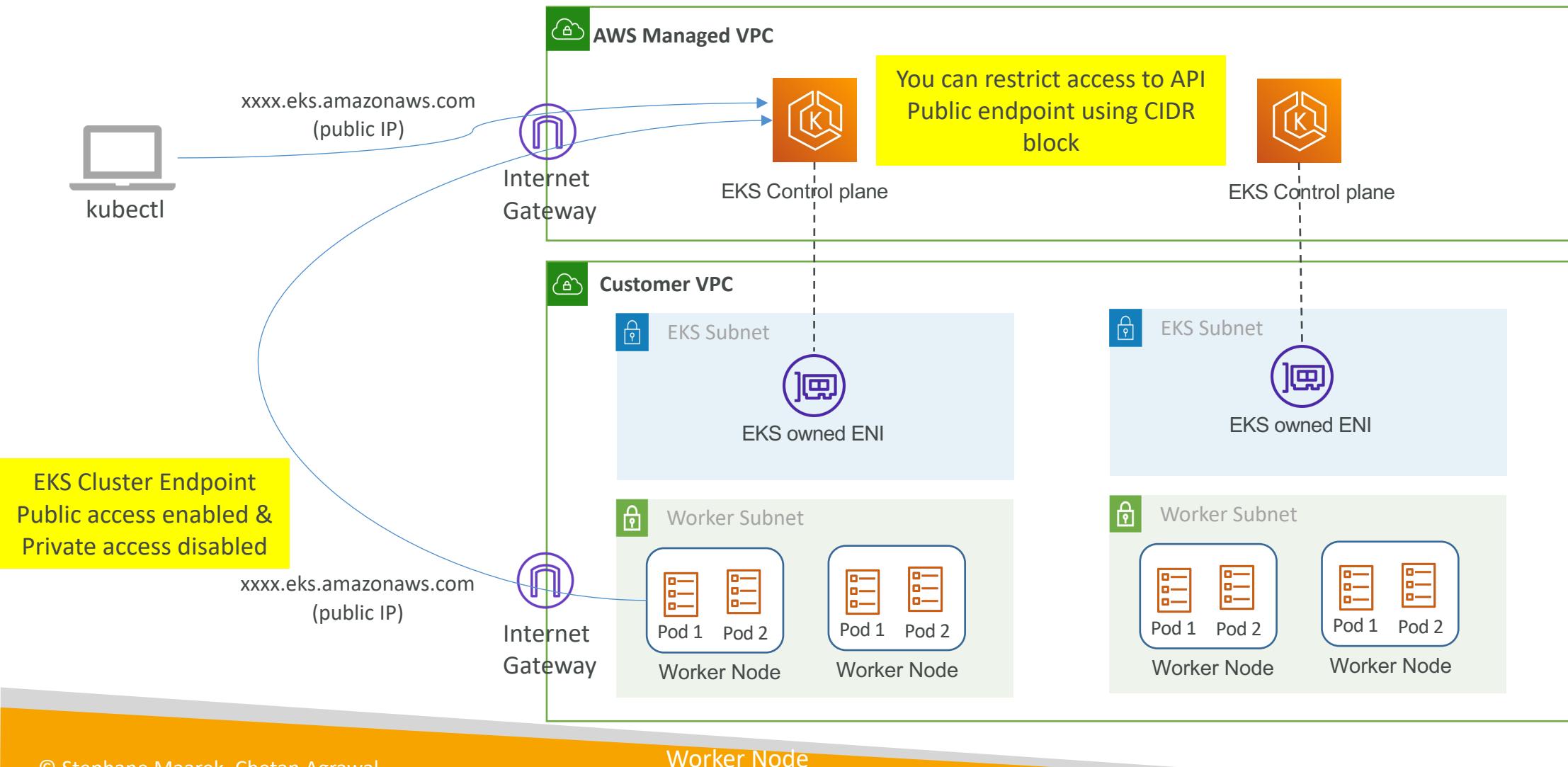


# EKS Networking

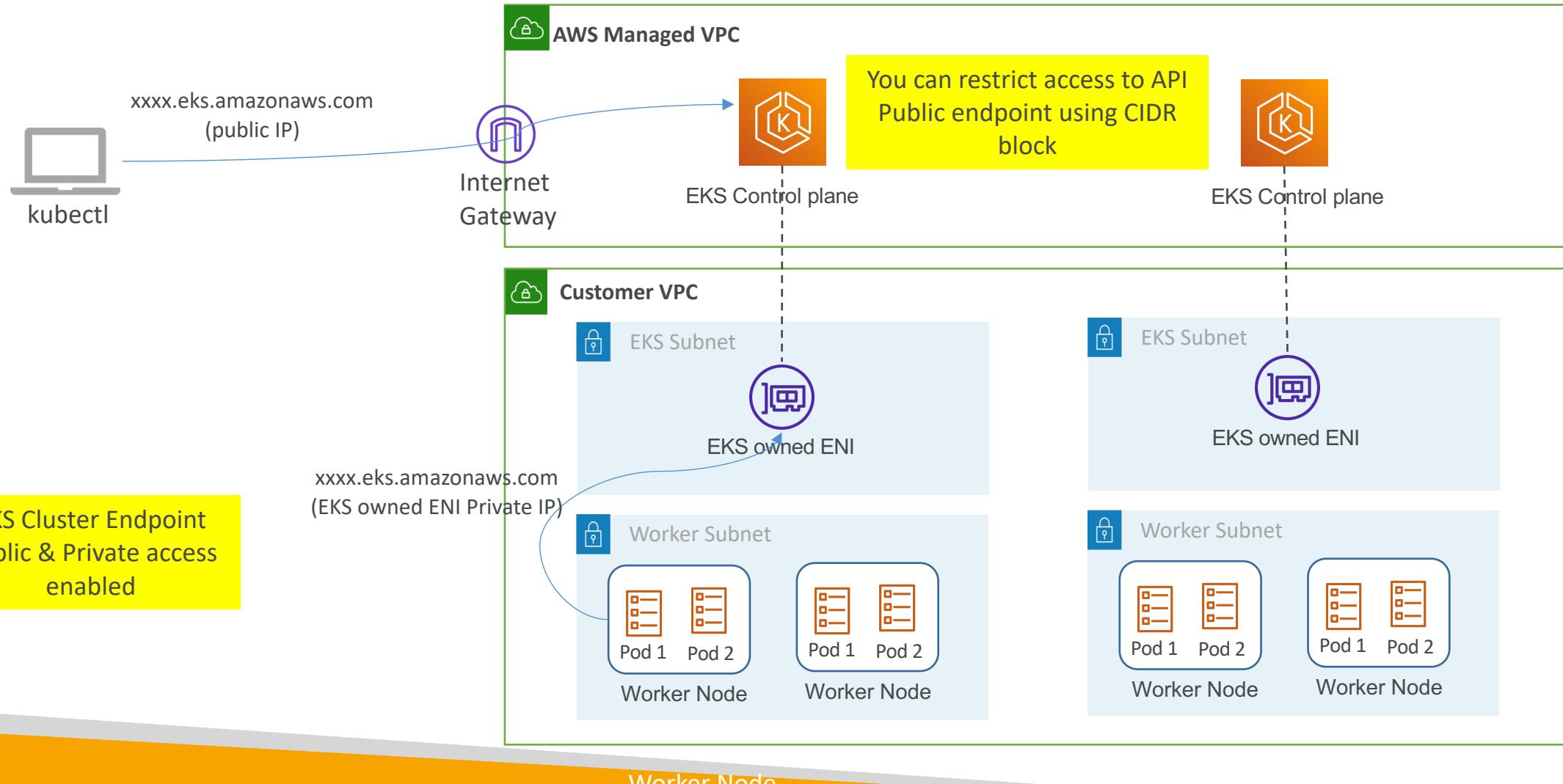
- EKS Control Plane is launched in the AWS managed account & VPC
- Data plane nodes are launched in Customer account and VPC
- EKS provisions 2-4 ENIs in the Customer VPC to enable the communication between Control plane and VPC.
- It's recommended to have separate subnets for EKS ENIs. EKS needs at least 6 IPs in each subnet (16 recommended)
- EKS creates and associates SG to these EKS owned ENIs (and also to Managed Group Nodes).
- Kubernetes API Server can be accessed over the internet (by default)
- EKS allows assigning IPv4 or IPv6 IP addresses to Pods (but not in dualstack mode)



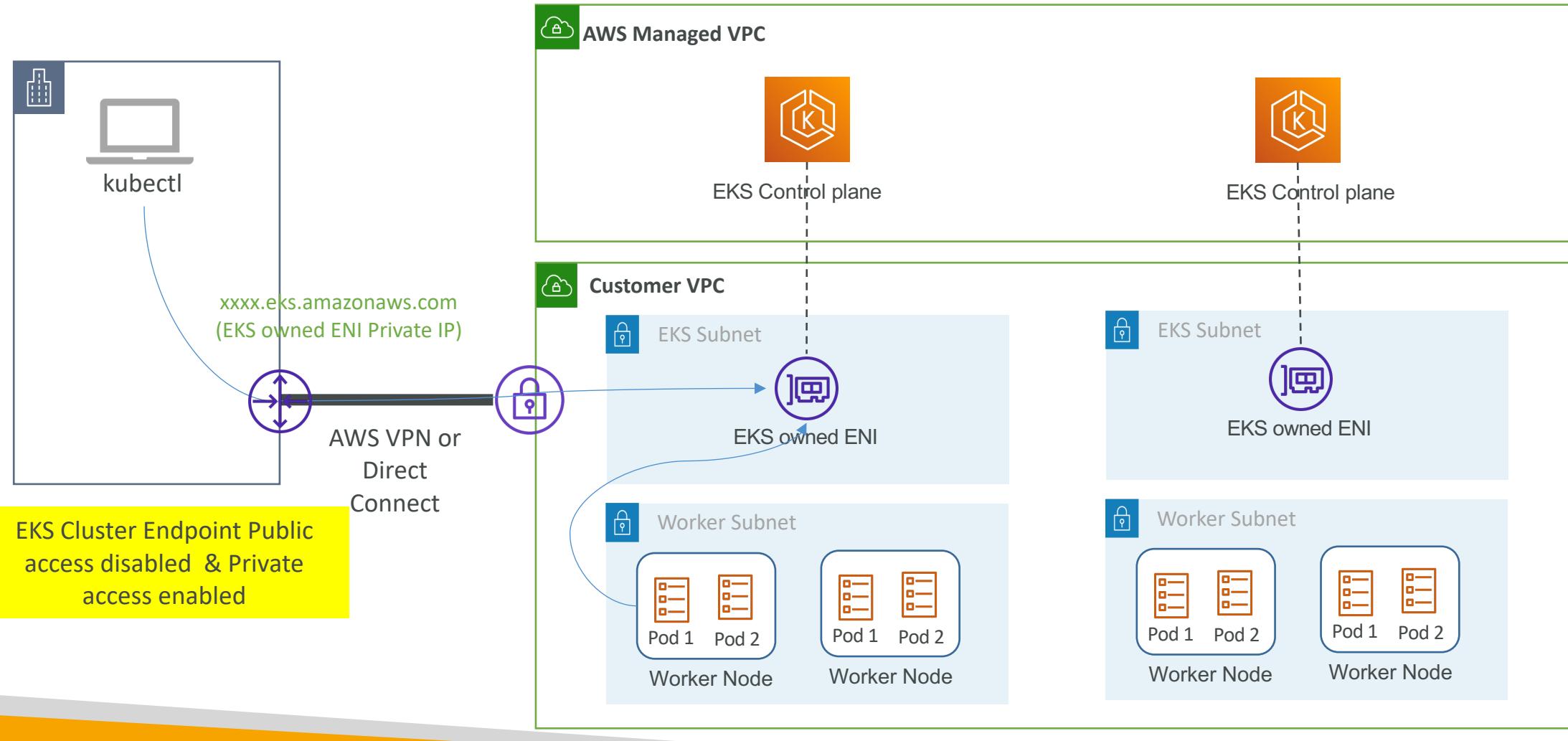
# EKS cluster endpoint access – Public (default)



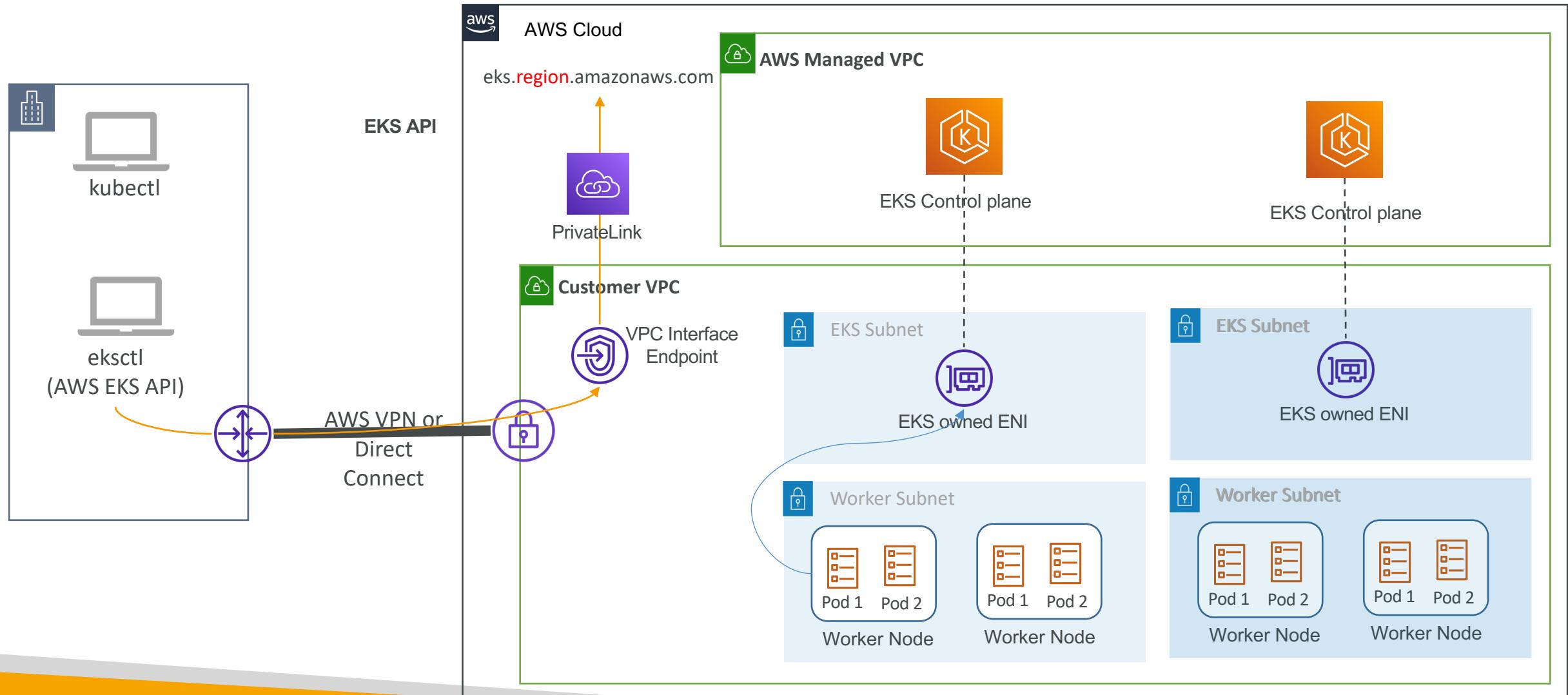
# EKS cluster endpoint access – Public & Private



# EKS cluster endpoint access – Private only

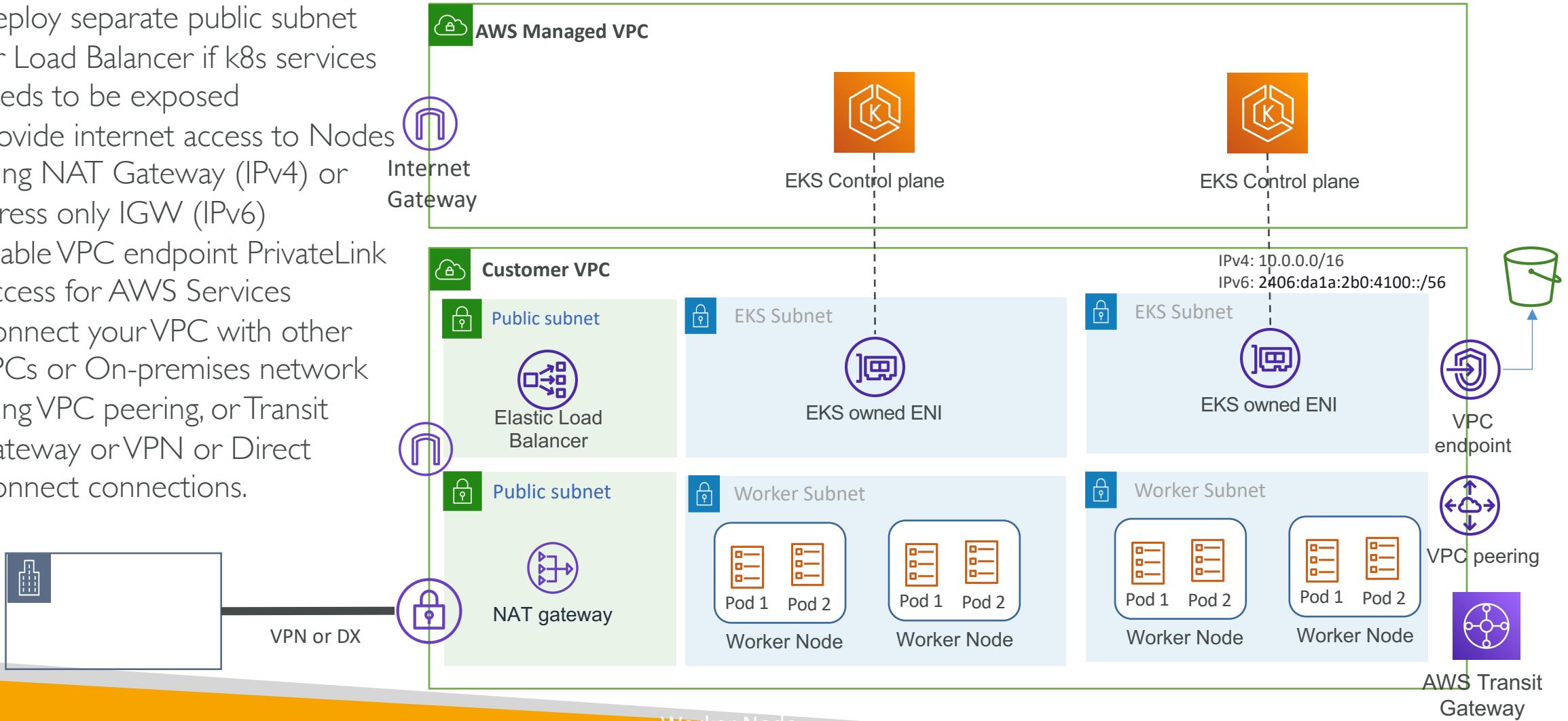


# EKS API Private access through PrivateLink



# EKS VPC extended connectivity

- Deploy separate public subnet for Load Balancer if k8s services needs to be exposed
- Provide internet access to Nodes using NAT Gateway (IPv4) or Egress only IGW (IPv6)
- Enable VPC endpoint PrivateLink Access for AWS Services
- Connect your VPC with other VPCs or On-premises network using VPC peering, or Transit Gateway or VPN or Direct Connect connections.



# Amazon EKS Pod Networking - CNI

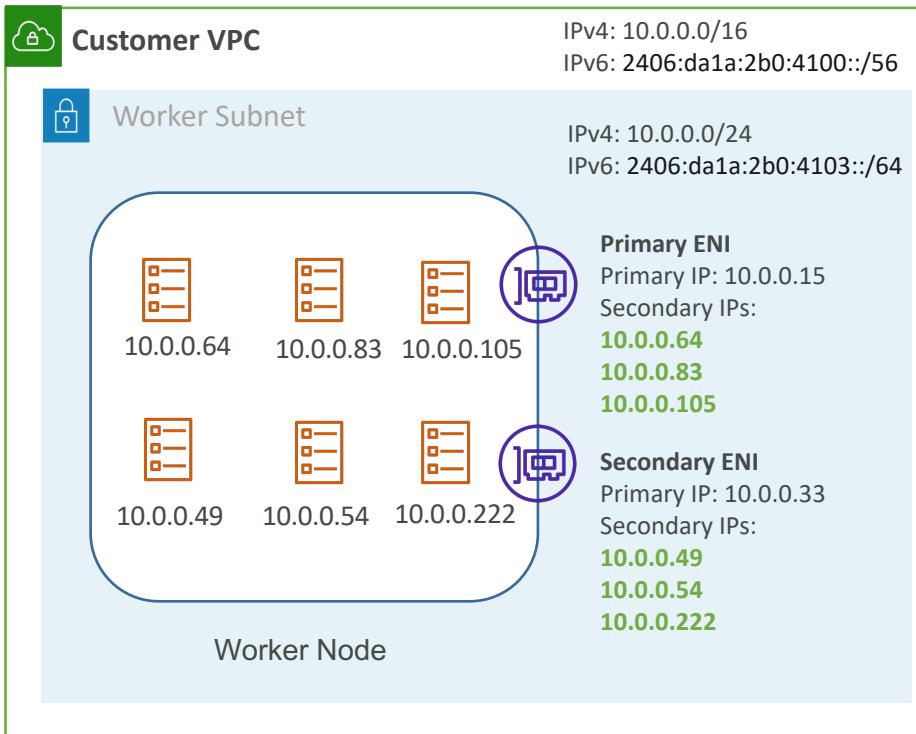
# Kubernetes Network Model

## CNCF networking specifications

- Every pod gets its own IP address
- Containers in the same Pod share the network IP address
- All pods can communicate with all other pods without using network address translation (NAT).
- All nodes can communicate with all pods without NAT.
- The IP that a pod sees itself as is the same IP that others see it as.



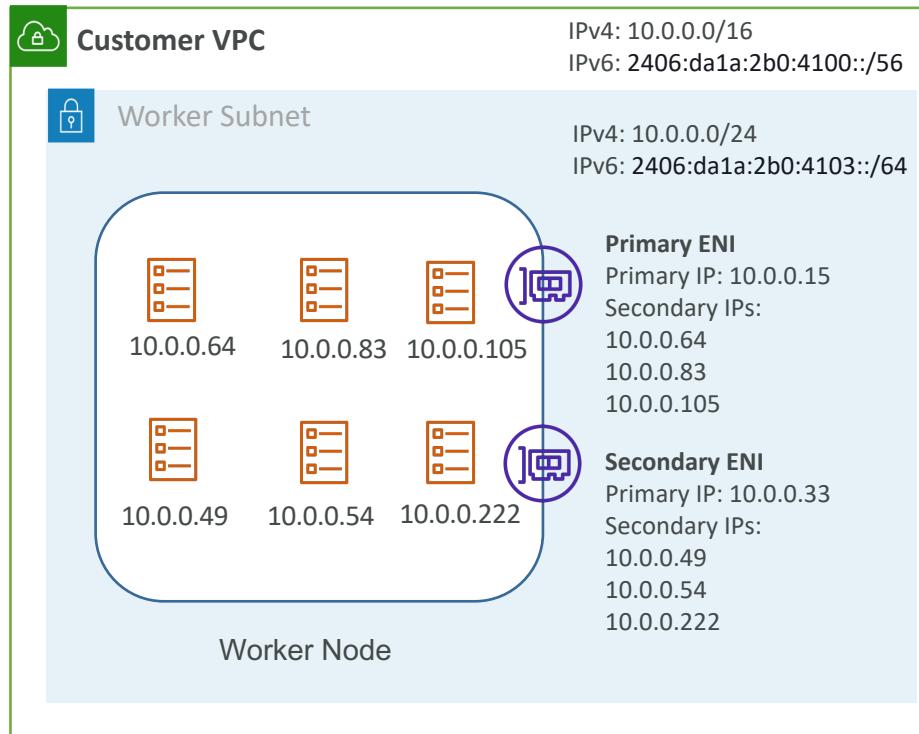
# Amazon VPC CNI plugin



- Amazon VPC Container Network Interface (CNI) plugin:
  - Creates and attaches ENIs to worker nodes
  - Assigns ENI secondary IP addresses to Pods
- Amazon EKS officially supports the Amazon VPC CNI plugin for Kubernetes
- Alternate compatible CNI plugins:

| Partner    | Product                   |
|------------|---------------------------|
| Tigera     | <a href="#">Calico</a>    |
| Isovalent  | <a href="#">Cilium</a>    |
| Weaveworks | <a href="#">Weave Net</a> |
| VMware     | <a href="#">Antrea</a>    |

# Maximum Pods per node



## IP addresses per network interface per instance type

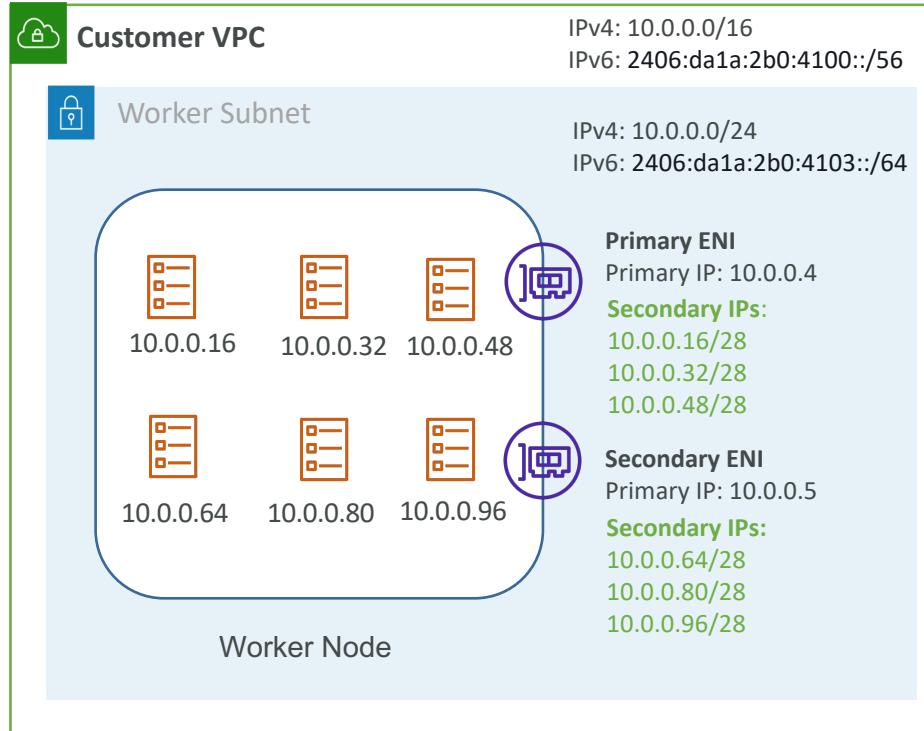
| Instance type | Maximum network interfaces | Private IPv4 addresses per interface | IPv6 addresses per interface |
|---------------|----------------------------|--------------------------------------|------------------------------|
| m5.large      | 3                          | 10                                   | 10                           |
| m5.xlarge     | 4                          | 15                                   | 15                           |
| m5.2xlarge    | 4                          | 15                                   | 15                           |
| m5.4xlarge    | 8                          | 30                                   | 30                           |

Max Pods = (Total number of network interfaces)x (Maximum IPs per network interface - 1) + 2

Example (m5.large with IPv4 address):

$$\text{Max Pods} = 3 \times (10-1) + 2 = 29$$

# Increased available IP addresses for Pods



**Only AWS Nitro-based nodes use this capability**

## Prefix delegation:

Assign a prefix to EC2 ENI

- /28 block for IPv4 (x16)
- /80 block for IPv6 (x280 trillion)

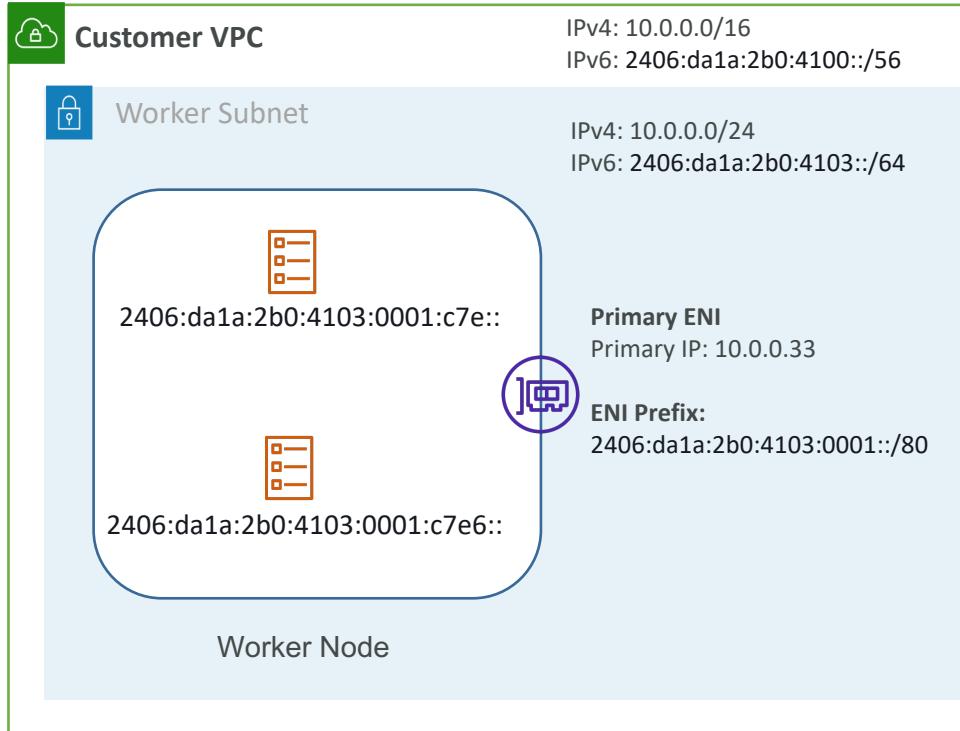
Max Pods = (Total number of network interfaces) x (Maximum IPs per network interface - 1) + 2

**Example (m5.large with IPv4 address):**

$$\text{Max Pods} = 3 \times (10-1) \times 16 + 2 = 434$$

**Max Pods (recommended) = 110**

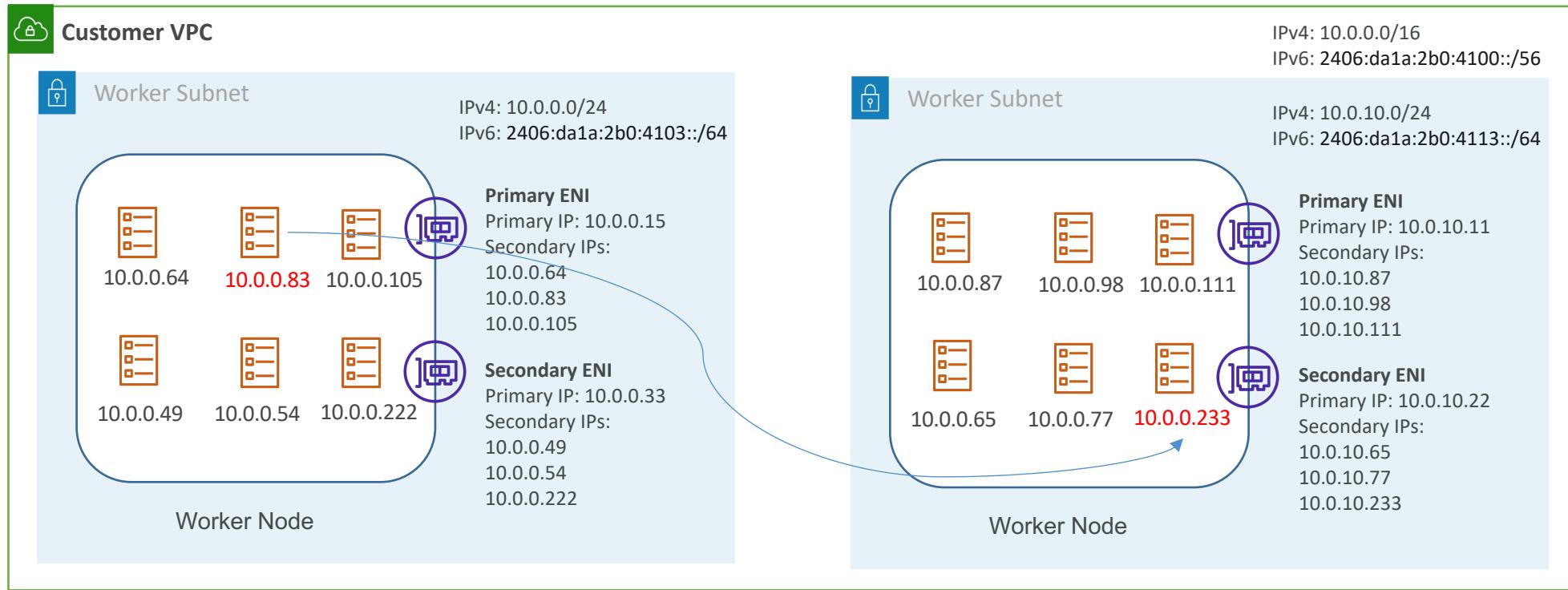
# Assigning IPv6 addresses to pods and services



## Supported with AWS Nitro-based instances & Fargate

- By default, Kubernetes assigns IPv4 addresses to pods and services but we can also configure cluster with IPv6 addresses.
- EKS doesn't support dual-stack pods or services.
- For Amazon EC2 nodes, you must configure the Amazon VPC CNI add-on with IP prefix delegation and IPv6.
- You must also assign IPv4 address to VPC and subnets as VPC does require IPv4 addresses to function.
- Subnets must have auto-assign IPv6 address enabled.
- Not supported for Windows pods and services.

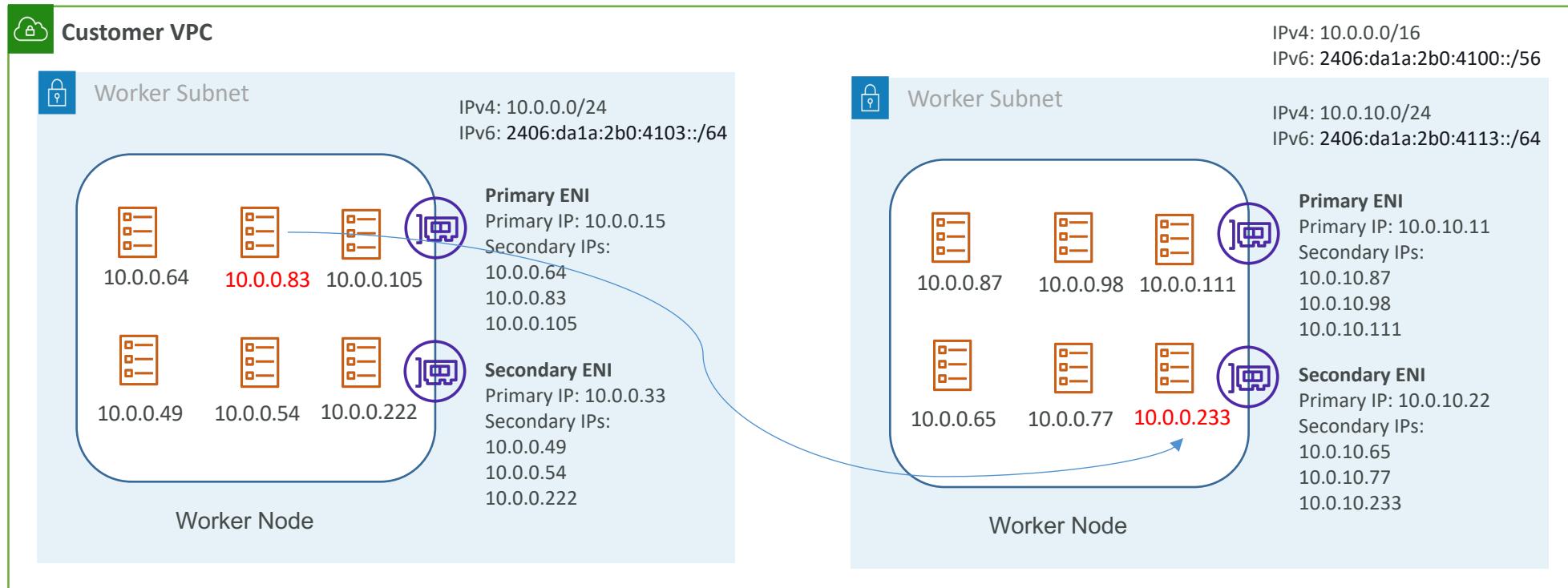
# Pod to Pod communication



Pods within the same VPC can communicate directly using Pod IP address

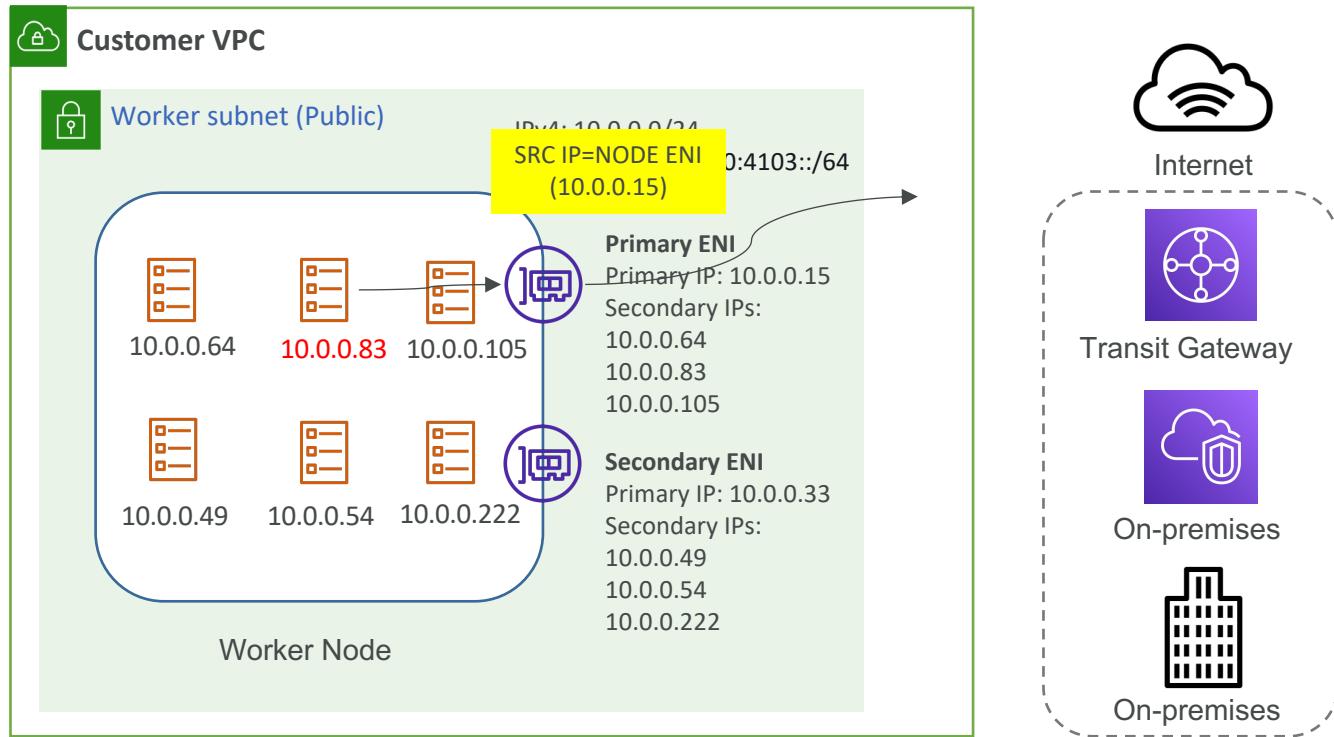
# Amazon EKS Pod Networking – traffic between Pod and external network

# Recap - Pod to Pod communication



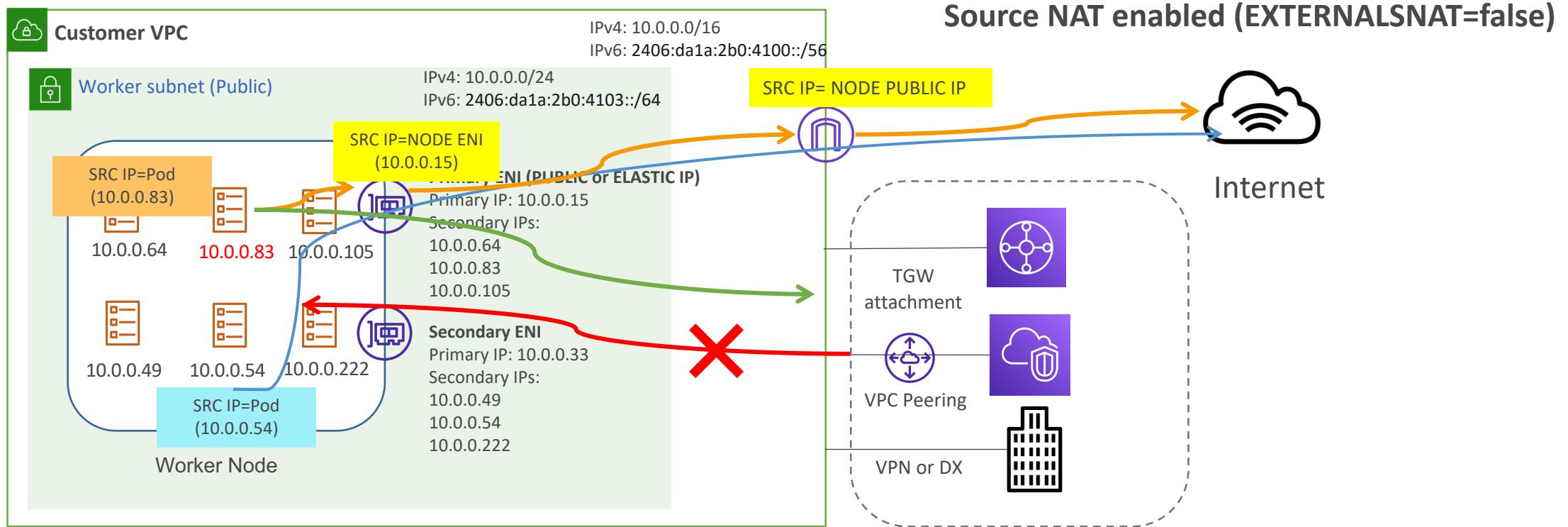
Pods within the same VPC can communicate directly using Pod IP address

# Pod to external network communication



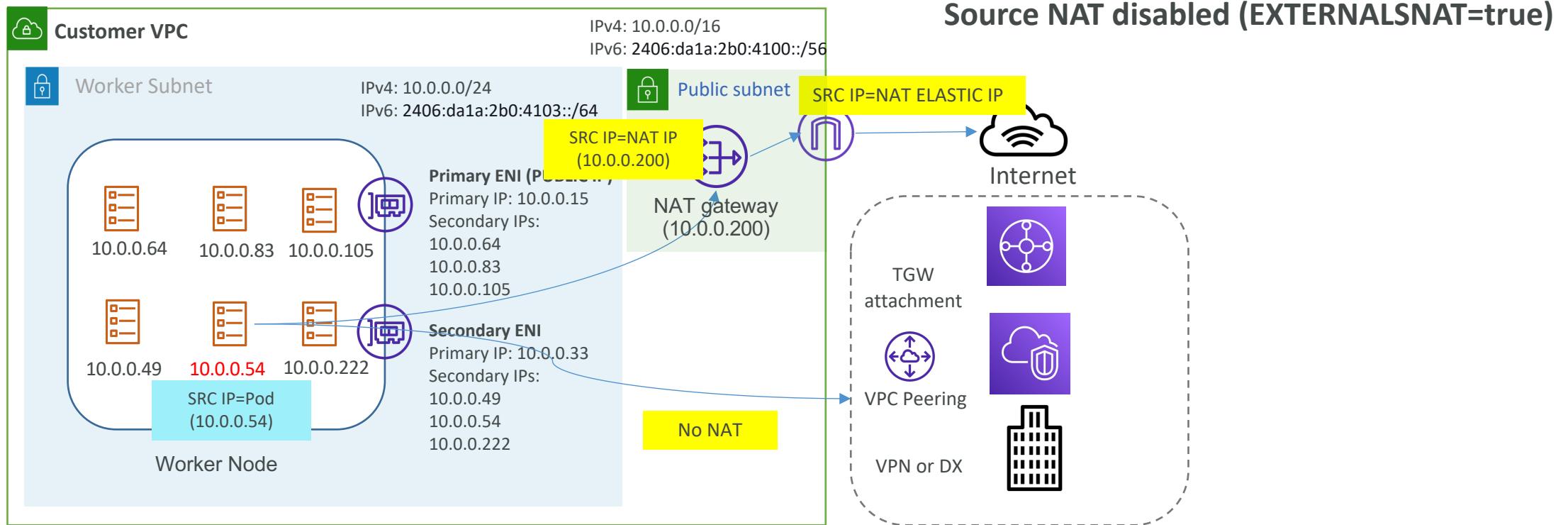
- When a pod communicates to any IPv4 address that isn't within a CIDR block of VPC, the Amazon VPC CNI plugin translates the pod's IPv4 address to the **primary private IPv4 address of the primary ENI** of the node that the pod is running on
- For IPv6 address family, this isn't applicable, because IPv6 addresses are not network translated.

# Pod to external network communication - Node in Public Subnet



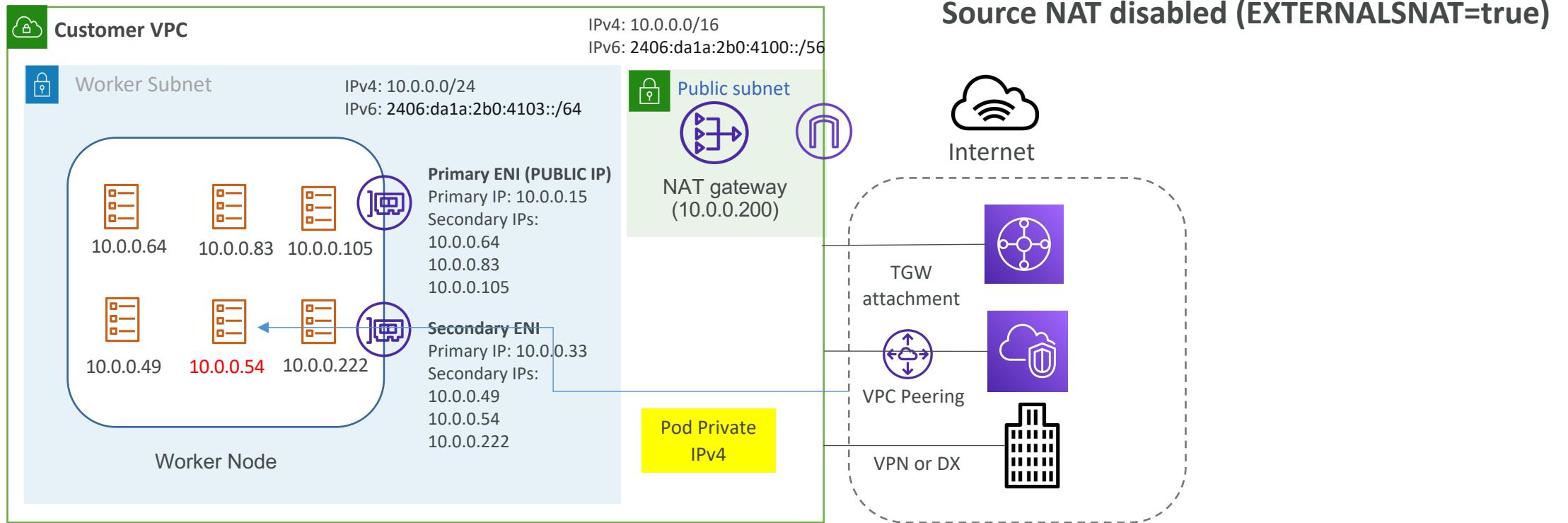
```
kubectl set env daemonset aws-node -n kube-system AWS_VPC_K8S_CNI_EXTERNALSNAT=false
```

# Pod to external network communication – Node in Private Subnet



```
kubectl set env daemonset aws-node -n kube-system AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

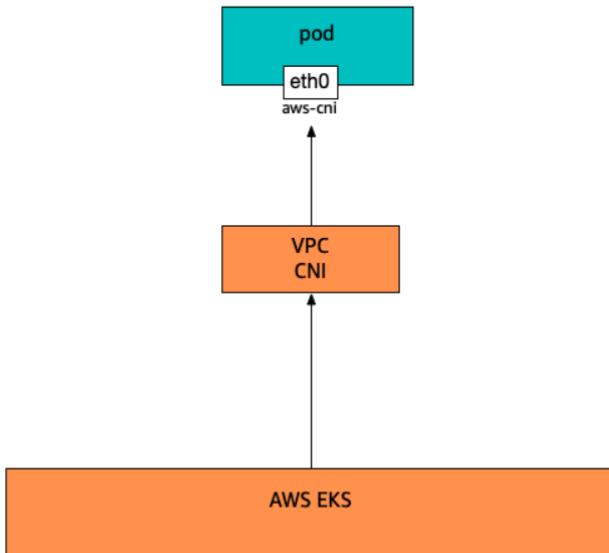
# External network to Pod communication



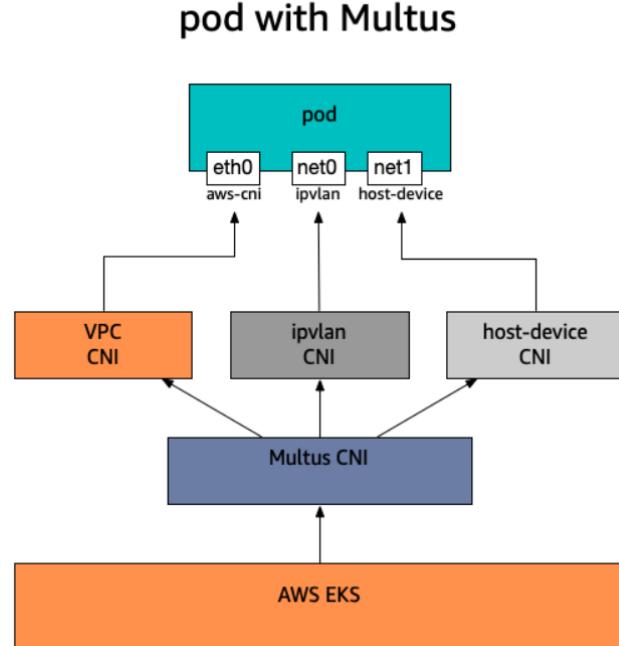
```
kubectl set env daemonset aws-node -n kube-system AWS_VPC_K8S_CNI_EXTERNALSNAT=true
```

# Multi-homed Pods with Multus CNI

pod without Multus



pod with Multus



- Enables attaching multiple interfaces to pods
- With Multus, you can create a multi-homed pod that has multiple interfaces
- AWS support for Multus comes with VPC CNI

<https://github.com/aws-samples/eks-install-guide-for-multus/blob/main/README.md>

# Security Groups in EKS

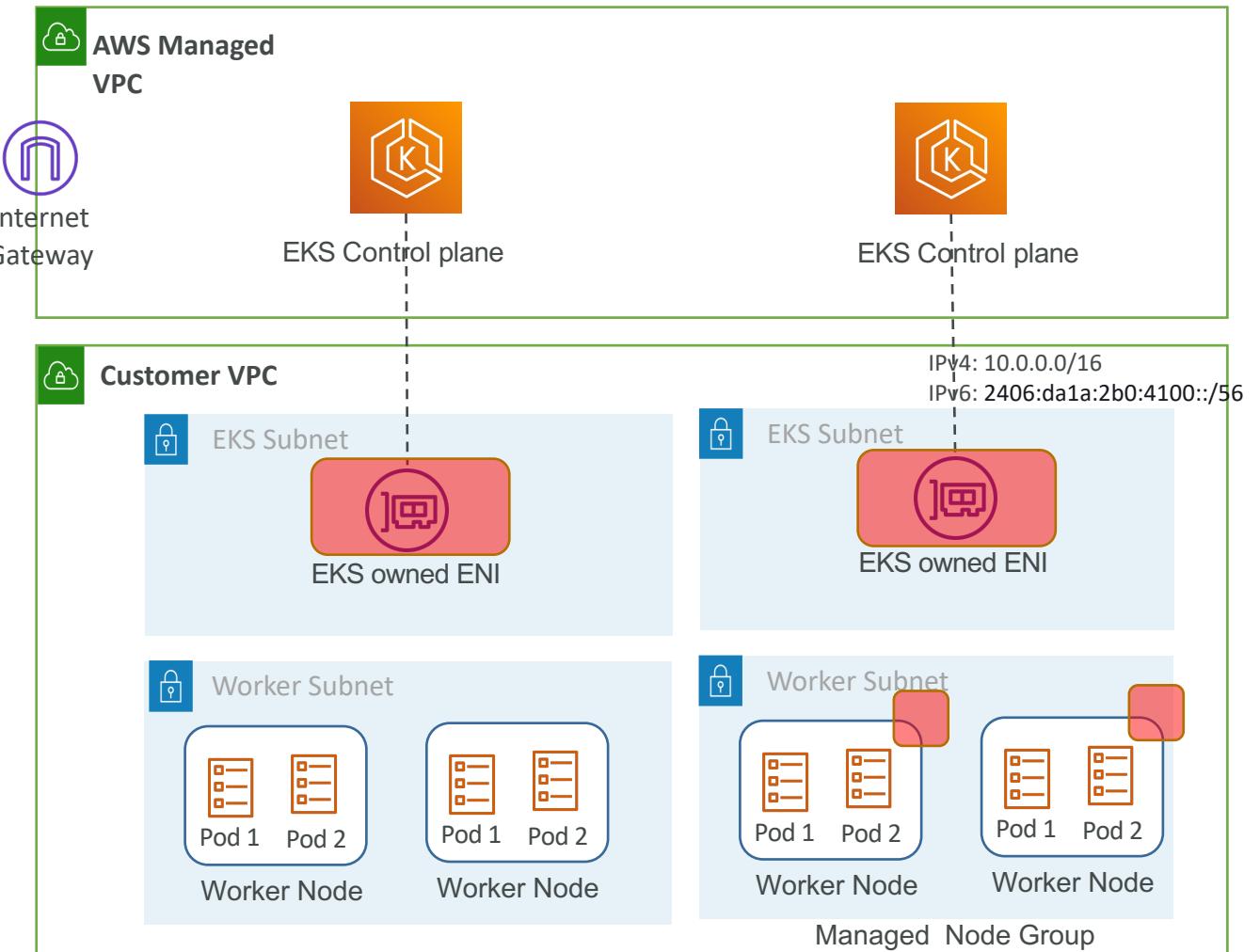
# EKS Cluster Security group

- When you create EKS Cluster, it creates and associates SG with following rules:

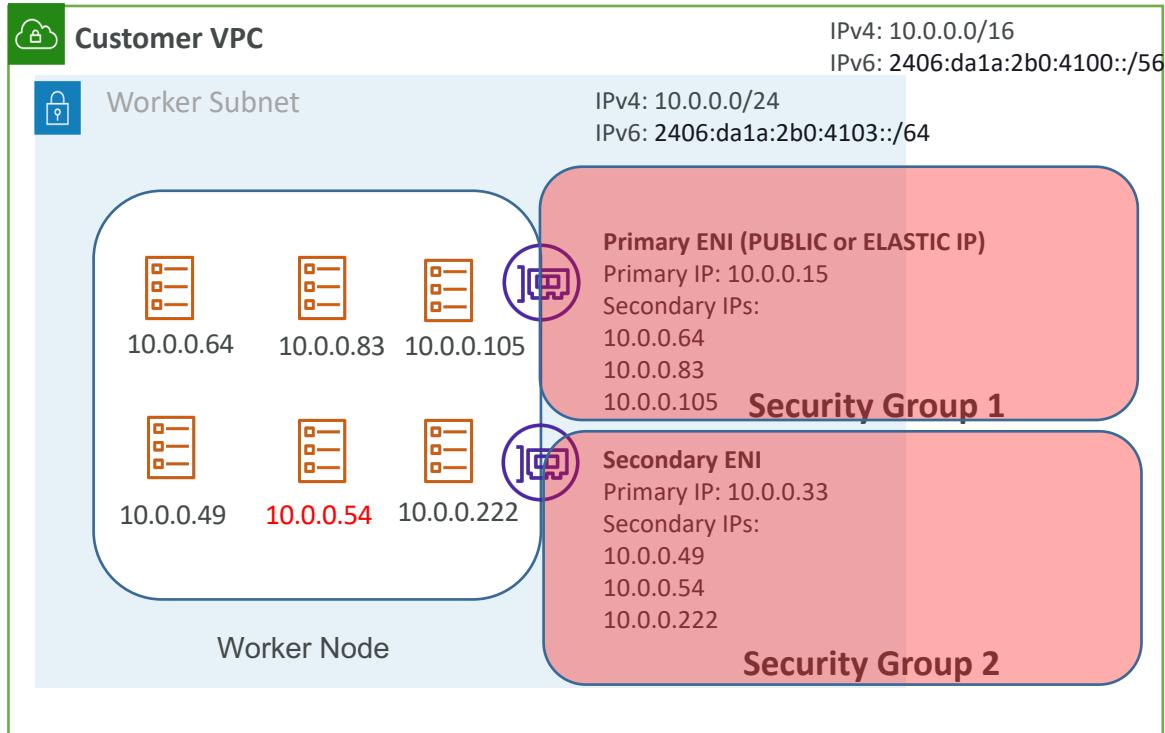
| Rule type | Protocol | Ports | Source | Destination                     |
|-----------|----------|-------|--------|---------------------------------|
| Inbound   | All      | All   | Self   |                                 |
| Outbound  | All      | All   |        | 0.0.0.0/0 (IPv4) or ::/0 (IPv6) |

- EKS associates this SG with:
  - ENIs created by EKS in CustomerVPC
  - ENIs of the nodes in Managed Node group
- At minimum following Outbound rules are required:

| Rule type      | Protocol    | Port  | Destination            |
|----------------|-------------|-------|------------------------|
| Outbound       | TCP         | 443   | Cluster security group |
| Outbound       | TCP         | 10250 | Cluster security group |
| Outbound (DNS) | TCP and UDP | 53    | Cluster security group |

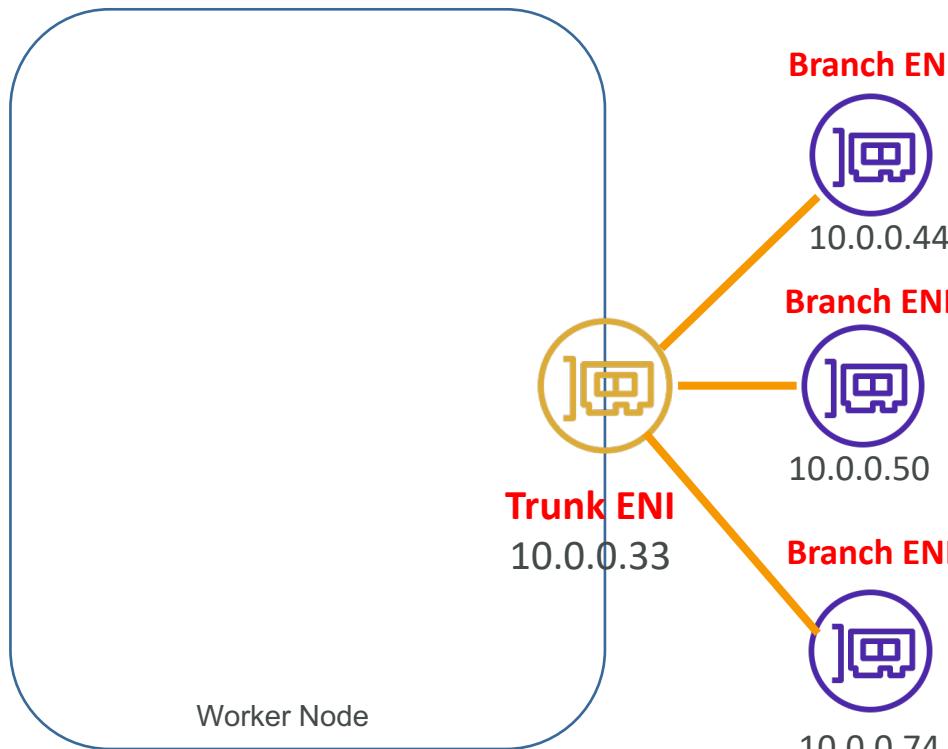


# Pod Security groups – The problem



- SG is assigned to Node ENIs and hence all Pods having secondary IPs from the same ENI will use the same SG
- This is a drawback if you need different security groups for different Pods
- One of the option is to use Network policy engine like **Calico** which provides network security policies to restrict inter pod traffic using iptables
- EKS native option is to use **Trunk and Branch ENIs**

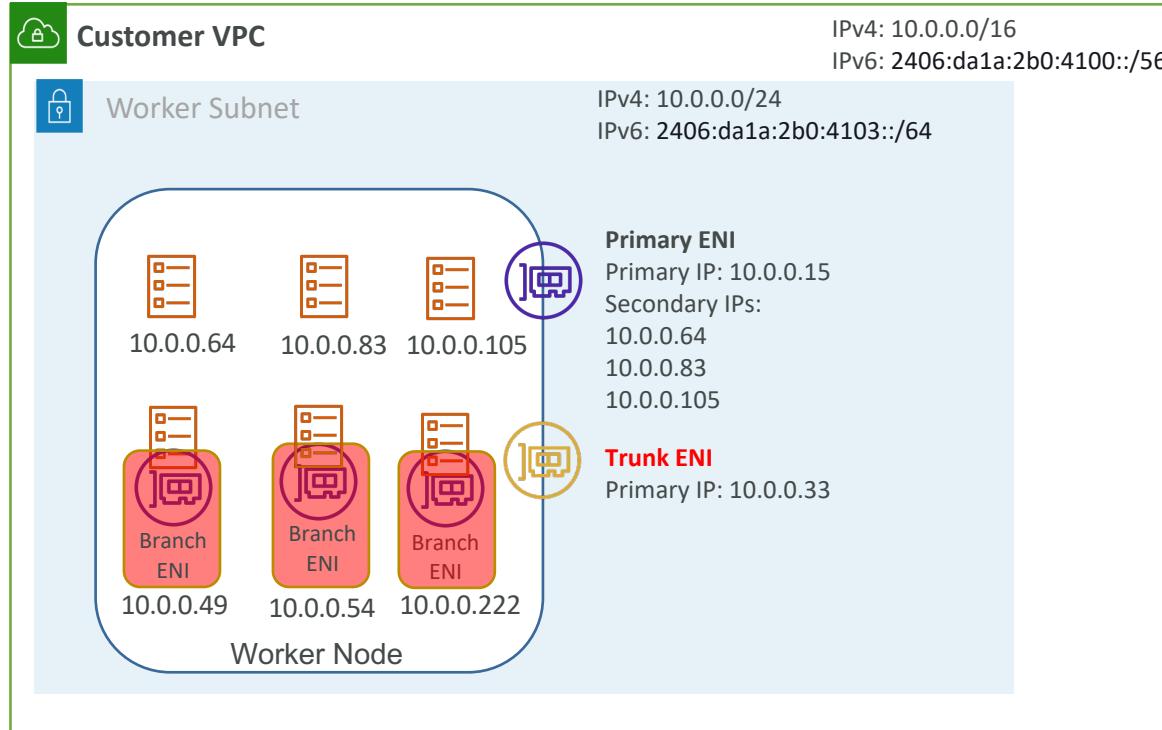
# Pod Security groups – The solution Trunk and Branch ENIs



- Amazon EKS and ECS supports Trunk & Branch ENI feature
- A VPC resource controller add-on named “amazon-vpc-resource-controller-k8s” manages Trunk & Branch Network Interfaces
- When ENABLE\_POD\_ENI=true, VPC resource controller creates special network interface called a trunk network interface with description “aws-k8s-trunk-eni” and attaches it to the node
- The controller also creates branch interfaces with description “aws-k8s-branch-eni” and associates them with the trunk interface.

```
kubectl set env daemonset aws-node -n kube-system  
ENABLE_POD_ENI=true
```

# Pod Security groups – The solution Trunk and Branch ENIs



- Each Pod gets dedicated ENI (branch ENI) mapped to trunk ENI
- Independent Security group per Pod

## Note

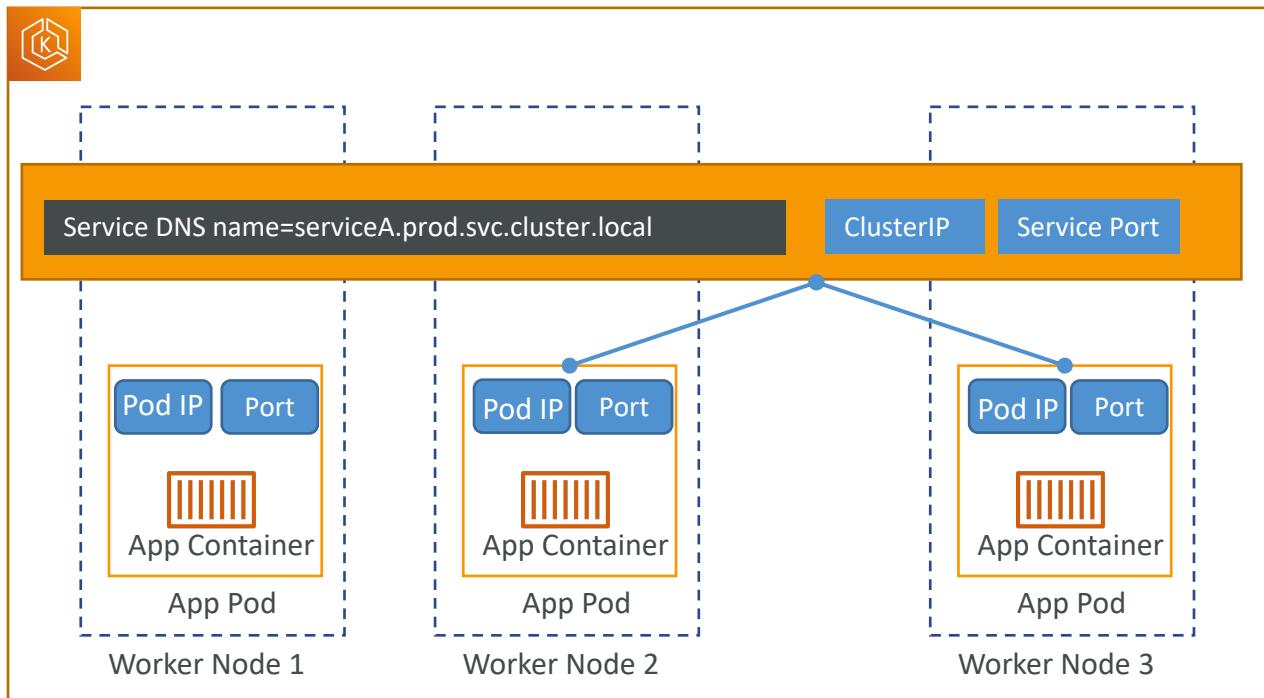
- Security groups for pods can't be used with Windows nodes
- If cluster is using IPv6 address family then this feature only works with Fargate nodes
- Supported by most Nitro based system (t instance family is not supported)
- The Node instances should be listed in `limits.go` file with `IsTrunkingCompatible: true`

# Exposing EKS services

# Kubernetes Service

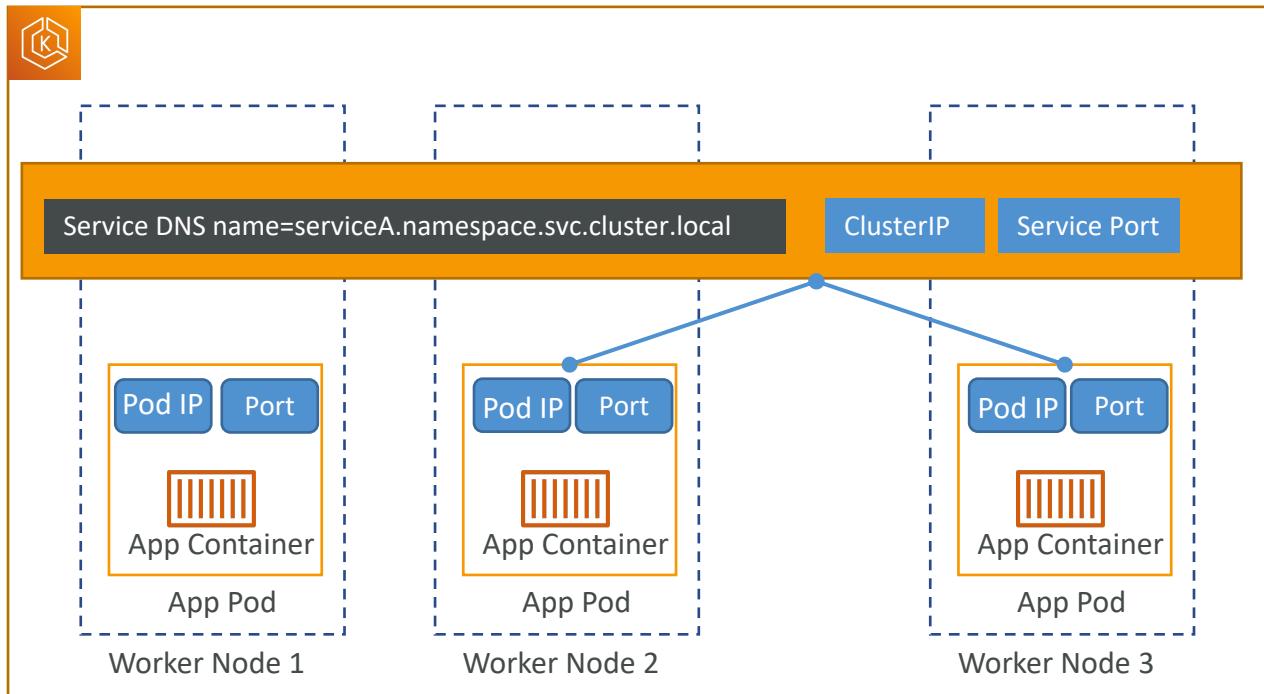
- Accessing applications by their Pod's IPs is usually an anti-pattern because
  - Pods are non-permanent objects
  - Pods may be created and destroyed
  - Pods move between the cluster's nodes due to a scaling event, a node replacement, or a configuration change.
- Kubernetes Service is a way to expose an application running on a set of Pods as a network service
- Kubernetes & EKS Supports following Service Types:
  - ClusterIP (access services from inside EKS cluster using Virtual IP)
  - NodePort (access services externally using Node static port)
  - LoadBalancer (Network load balancing, access services externally using CLB/NLB Layer4)
  - Ingress (Application load balancing, access services externally using ALB Layer7)

# ClusterIP



- ClusterIP is the default service type
- Makes the service reachable/accessible **only from within the cluster**
- Service is exposed on a virtual IP on each node. This IP is not exposed outside of a cluster.
- The service virtual IP is assigned from a pool which is configured by setting following parameter in kube-apiserver:  
`--service-cluster-ip-range`
- If not configured explicitly then Amazon EKS provisions either `10.100.0.0/16` or `172.20.0.0/16` for this Virtual IP.
- A `kube-proxy` daemon on each cluster node defines the ClusterIP to Pod IP mapping in `iptables` rules
- Service is accessible with private DNS `<service-name>.<namespace-name>.svc.cluster.local`

# ClusterIP

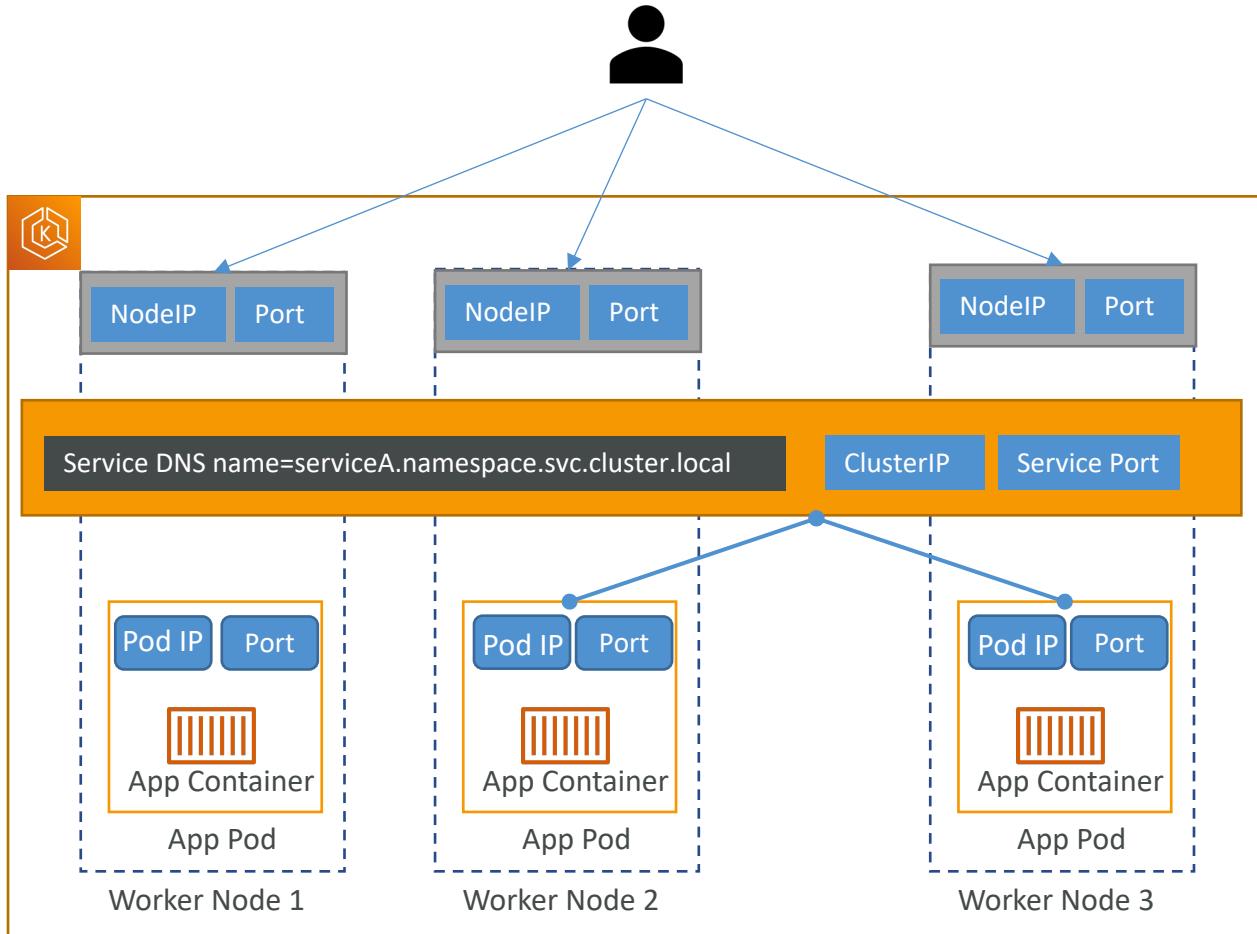


```

YAML
apiVersion: v1
kind: Service
metadata:
  name: some-service
  namespace: some-namespace
spec:
  type: ClusterIP
  selector:
    app.kubernetes.io/name: some-app
  ports:
    - name: svc-port
      port: 80
      targetPort: app-port
      protocol: TCP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: some-deployment
  namespace: some-namespace
spec:
  replicas: 1
  selector:
    matchLabels:
      app.kubernetes.io/name: some-app
  template:
    metadata:
      labels:
        app.kubernetes.io/name: some-app
    spec:
      containers:
        - name: nginx
          image: public.ecr.aws/nginx/nginx
          ports:
            - name: app-port
              containerPort: 80

```

# NodePort

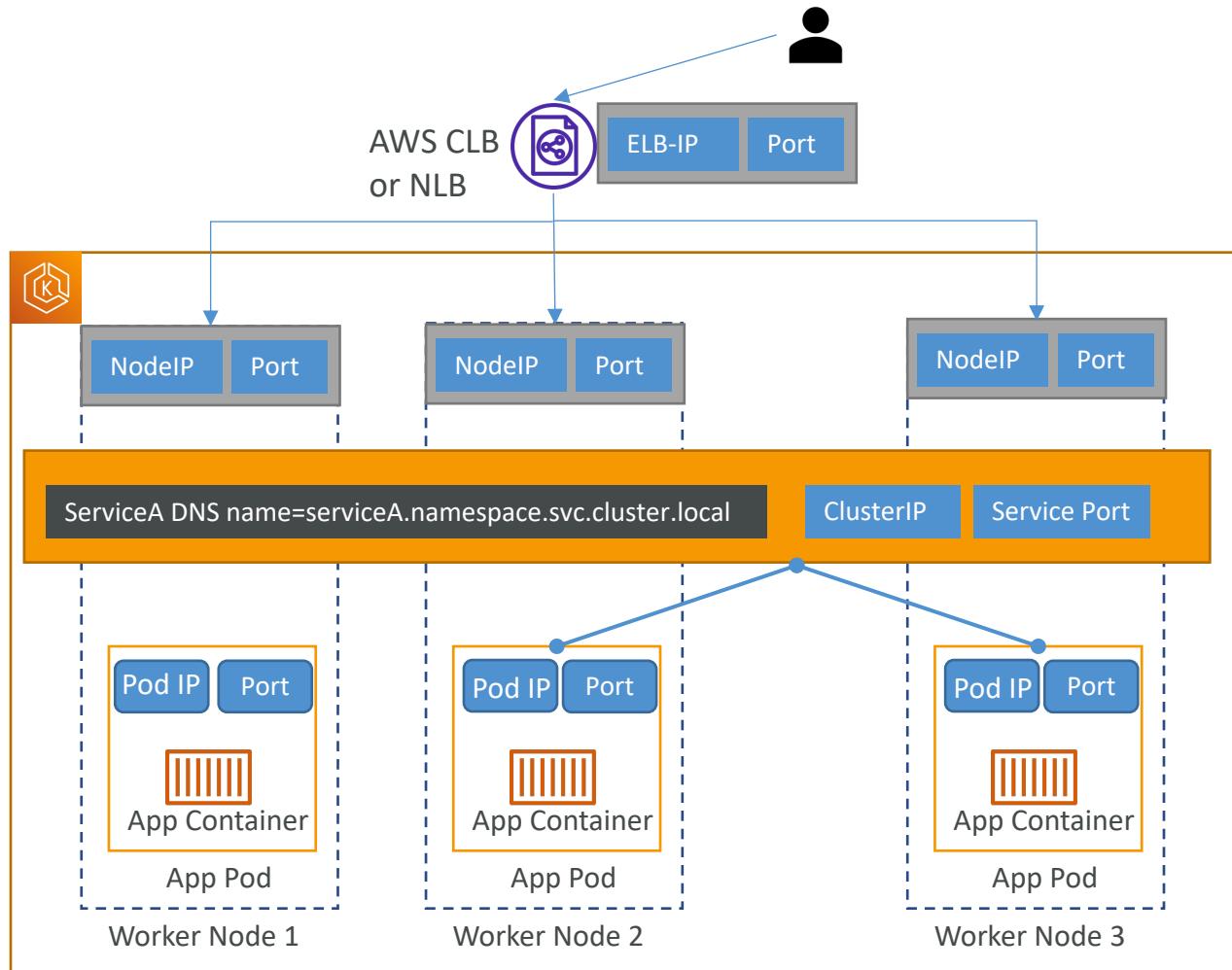


- NodePort is used to make a Kubernetes service accessible from outside the cluster
- Exposes the service on each worker node's IP at a static port, called the NodePort
- One Node port per Service
- Port range: 30000-32767
- NodePort internally uses ClusterIP to route the NodeIP/Port requests to ClusterIP service
- Client needs to keep track of Node IPs and any IP changes over the time
- **Not a feasible option to expose services to the outside world**

# EKS Network & Application Load Balancing

- ServiceType=LoadBalancer
  - Handled by Kubernetes Controller Manager (in-tree cloud controller)
  - Deploys AWS CLB (default) or NLB in instance mode
  - Layer 4 with NLB and Layer 4/7 with CLB
  - Now also supported by newer controller called AWS Load Balancer Controller
- ServiceType=Ingress
  - Handled by AWS Load Balancer Controller (formerly AWS ALB Ingress controller)
  - Deploys ALB in Instance & IP mode for ingress resource
  - Layer 7

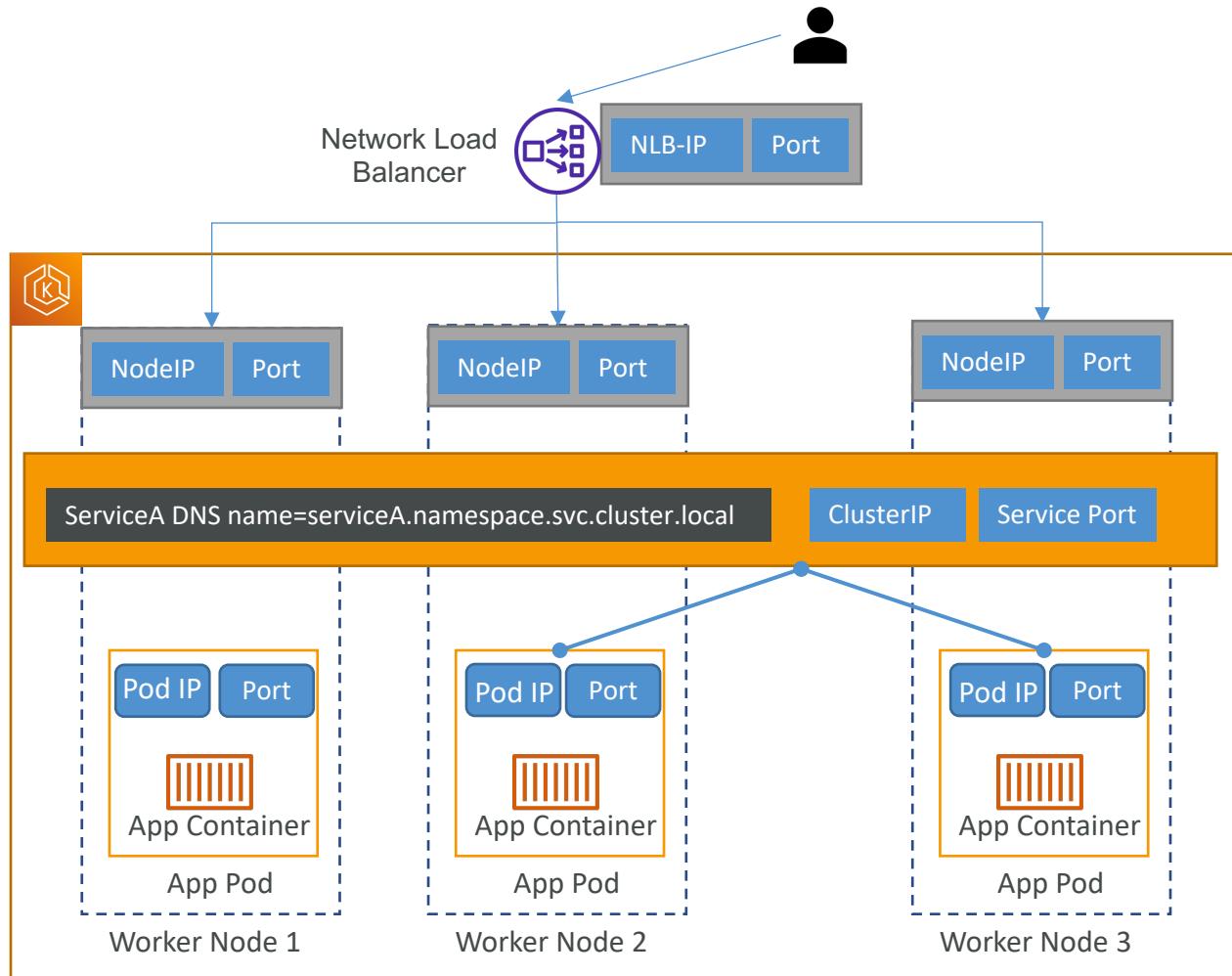
# LoadBalancer Service (with legacy controller)



- Exposes services to the client outside of the cluster
- LoadBalancer service is built on top of NodePort service
- Supports:
  - Classic Load Balancer (CLB)
    - Layer 4/Layer 7 traffic (TCP, SSL/TLS, HTTP, HTTPS)
  - Network Load Balancer (NLB)
    - Layer4 traffic (TCP, UDP, TLS)
    - Instance mode only

Legacy controller

# LoadBalancer Service (with newer controller)



- Recommended to use AWS Load Balancer Controller
- For target as IP (for EC2 or Fargate) use:

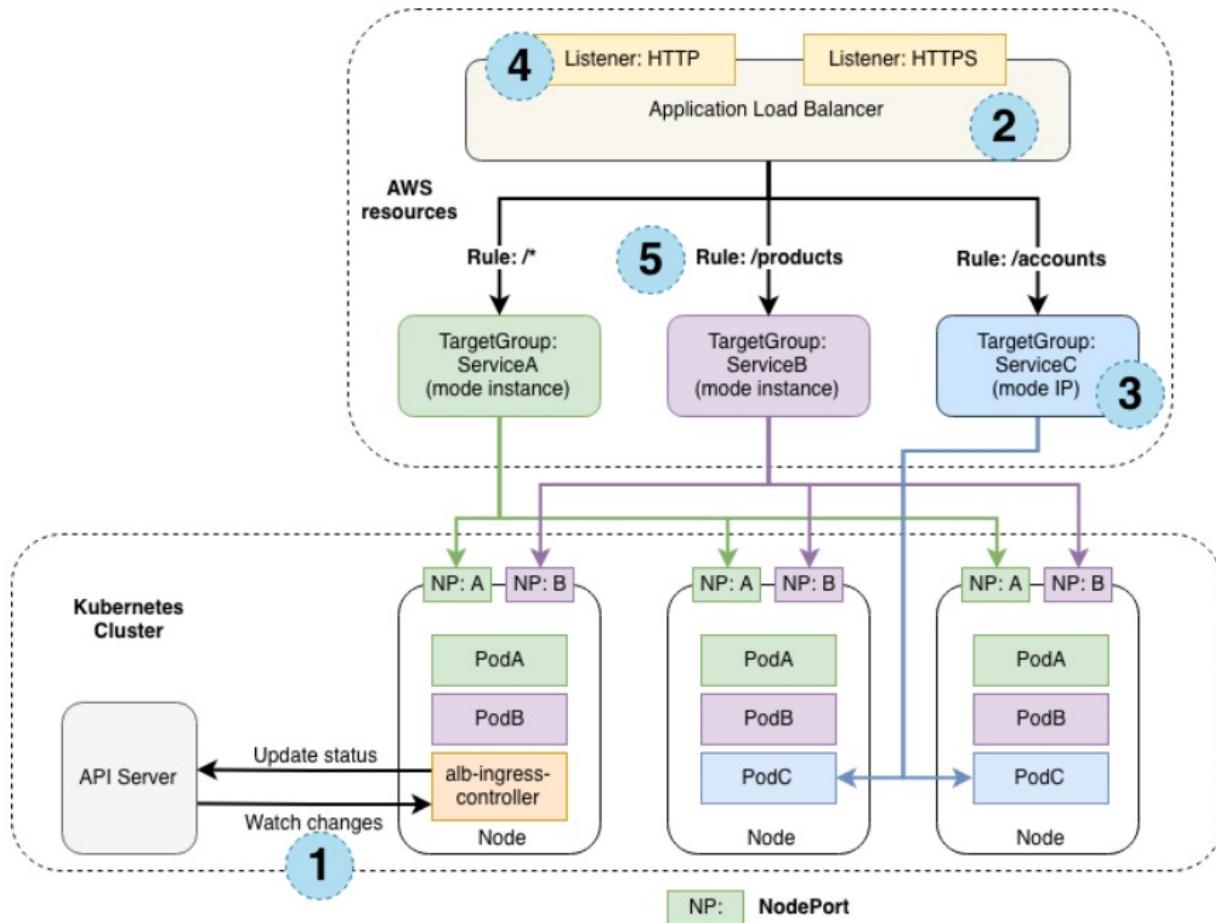
```
service.beta.kubernetes.io/aws-load-balancer-type: "external"  
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "ip"
```

- For target as Instance (for EC2) use:

```
service.beta.kubernetes.io/aws-load-balancer-type: "external"  
service.beta.kubernetes.io/aws-load-balancer-nlb-target-type: "instance"
```

- Each service needs a dedicated NLB
- Scaling & management is a challenge when number of services grows

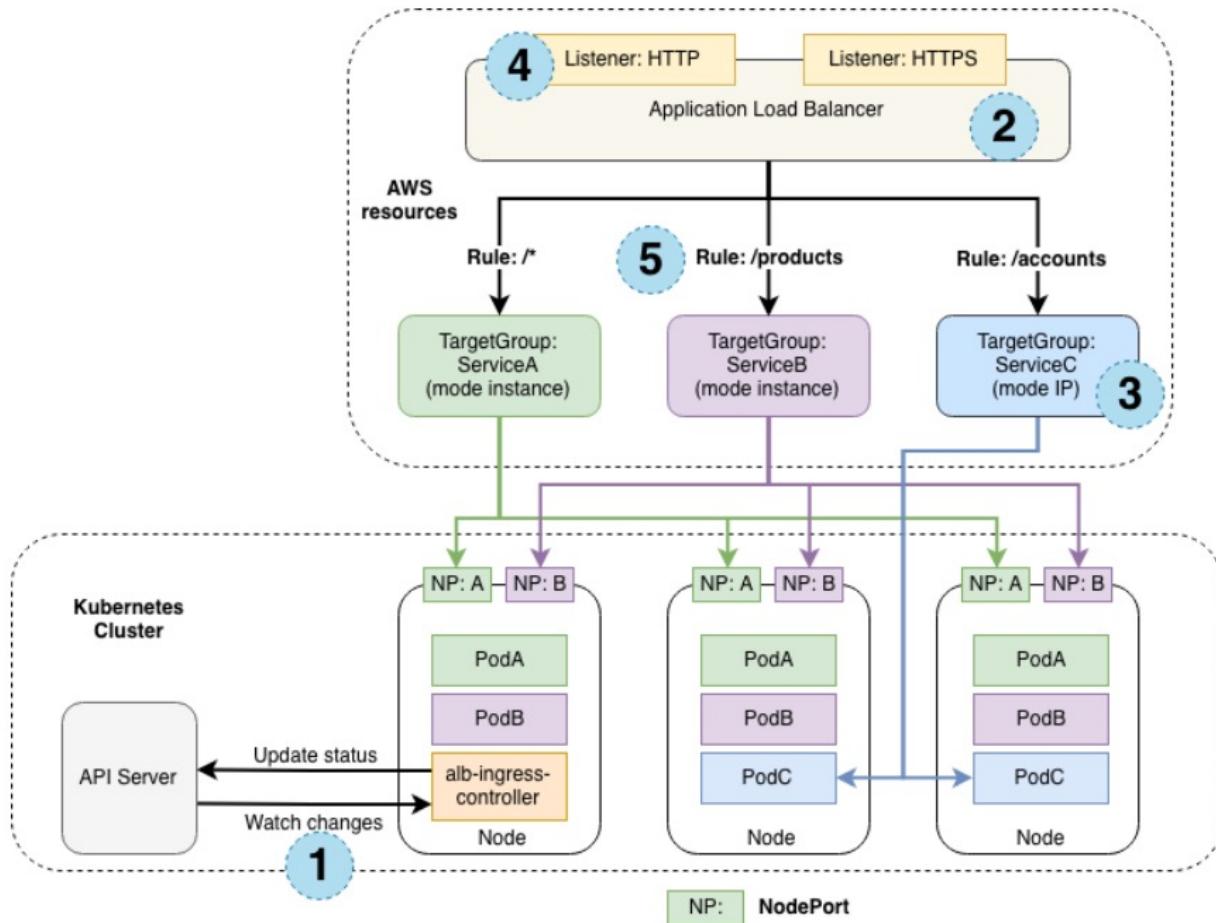
# Kubernetes Ingress



- Exposes services to the client outside of the cluster
- Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster.
- Traffic routing is controlled by rules defined on the Ingress resource.
- Saves cost and complexity as multiple services can be added behind a single ALB using ALB target groups.
- EKS uses **AWS Load Balancer Controller** for provisioning load balancer resources

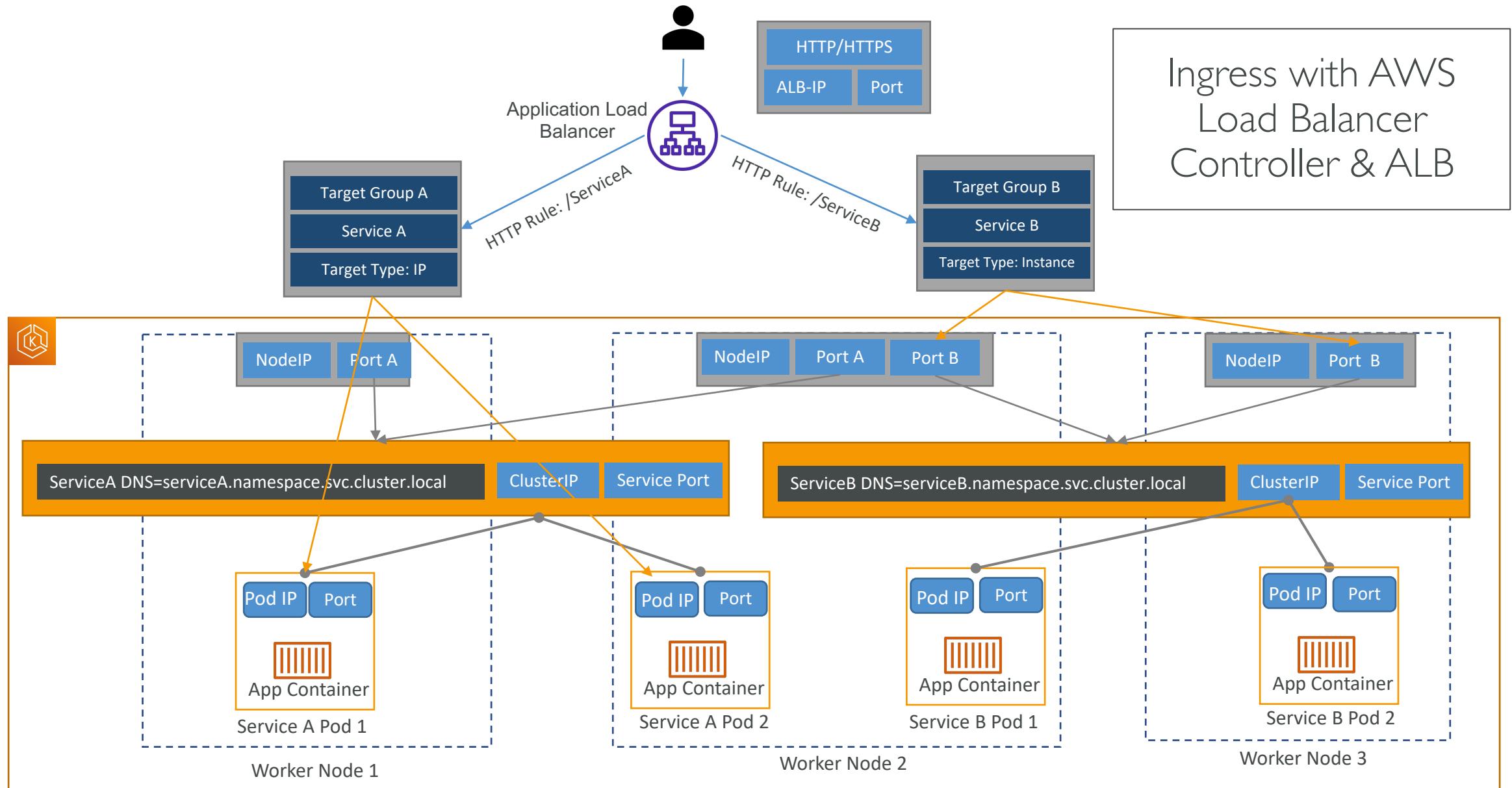
Diagram: <https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/how-it-works/>

# AWS Load Balancer Controller



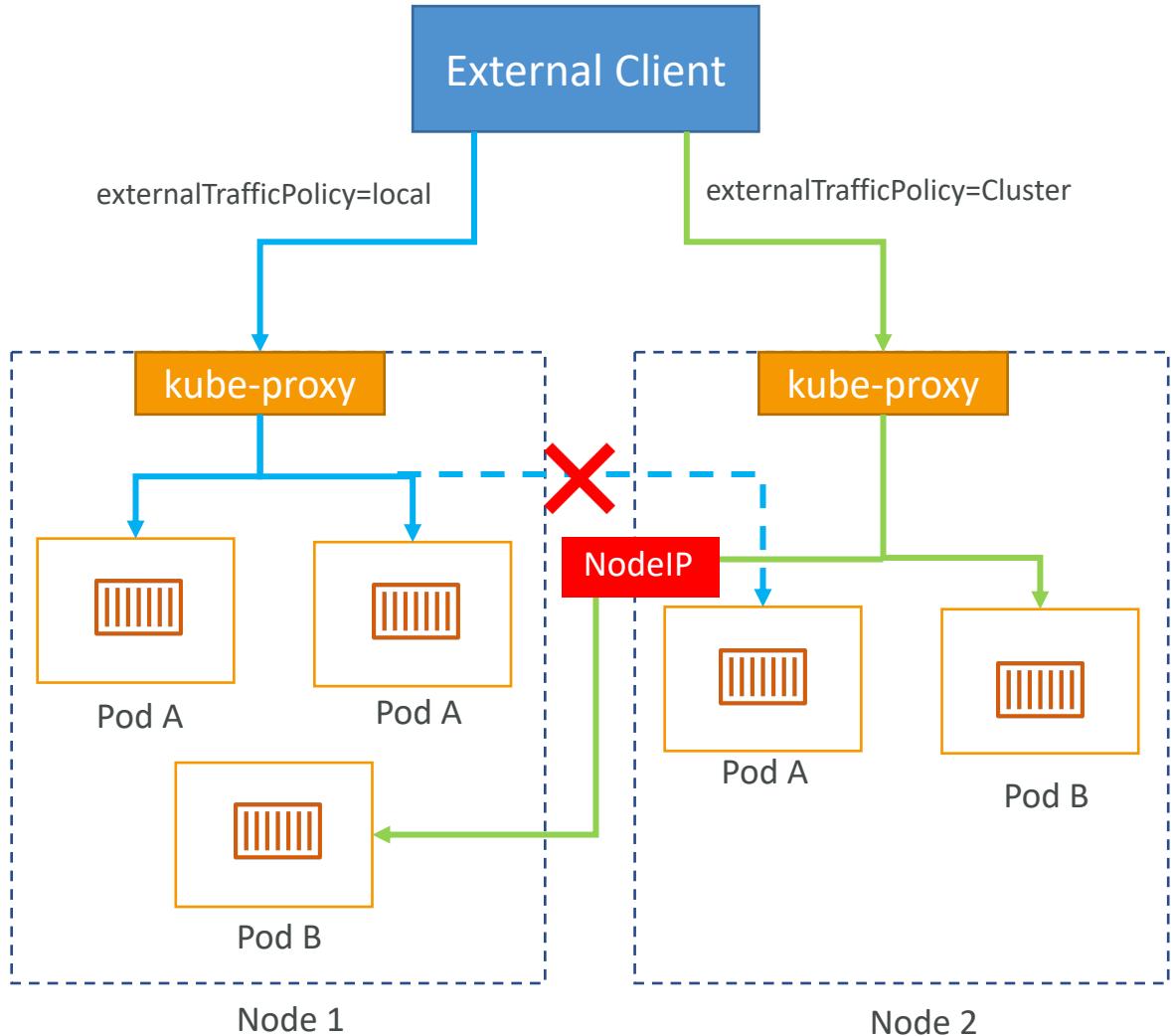
- The AWS implementation for Ingress controller
- Translates the Ingress rules, parameters, and annotations into the ALB configuration, creating listeners and target groups and connecting their targets to the backend Services.
- Supports target as Instance or Pod IP.
- Annotation used:  
**kubernetes.io/ingress.class: alb**
- Share ALB with multiple services by using annotation:  
**alb.ingress.kubernetes.io/group.name : my-group**
- Traffic for IPv6 is supported for IP targets only. Use annotation:  
**alb.ingress.kubernetes.io/ip-address-type: dualstack**

Diagram: <https://kubernetes-sigs.github.io/aws-load-balancer-controller/v2.4/how-it-works/>



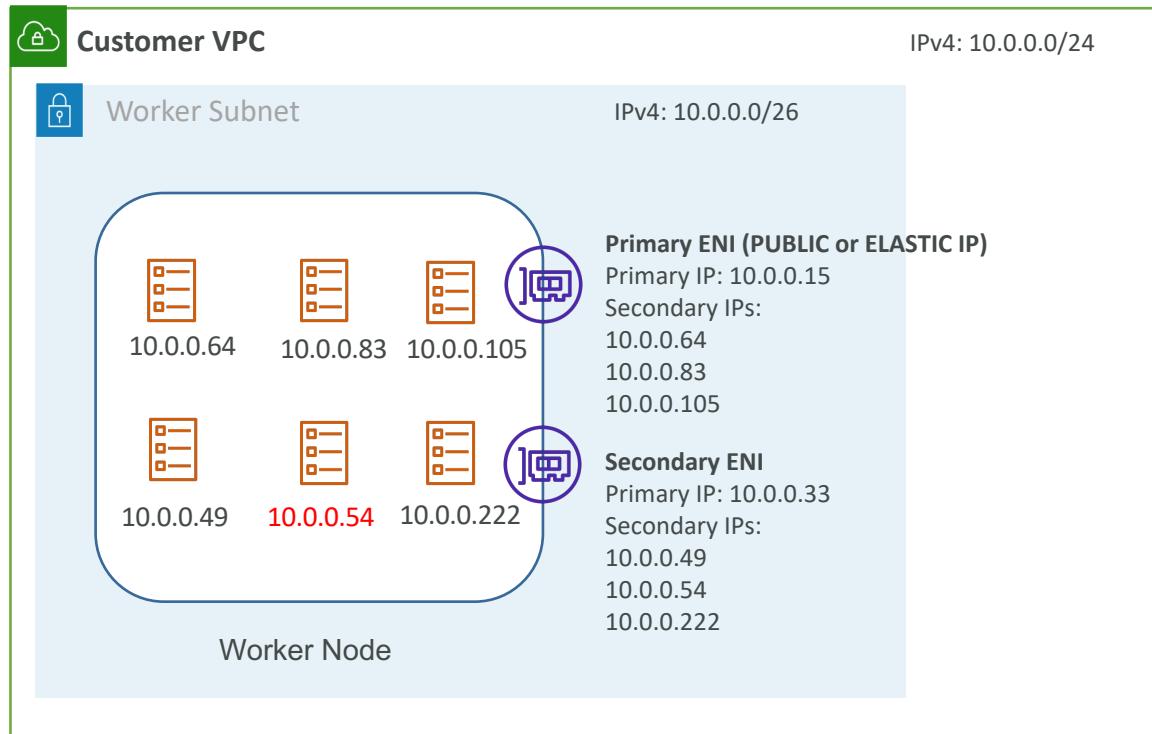
# Preserving Client IP

- For NLB with LoadBalancer service:
  - `externalTrafficPolicy` service spec defines how load-balancing happens in the cluster.
  - If `externalTrafficPolicy=Cluster`, the traffic may be sent to another node and source IP is changed to node's IP address thereby Client IP is not preserved. However load is evenly spread across the nodes.
  - By setting `externalTrafficPolicy=Local`, traffic is not routed outside of the node and client IP addresses is propagated to the end Pods. This could result in uneven distribution of traffic.
- For ALB Ingress service:
  - HTTP header `X-Forwarded-For` is used to get the Client IP.



# EKS Custom Networking

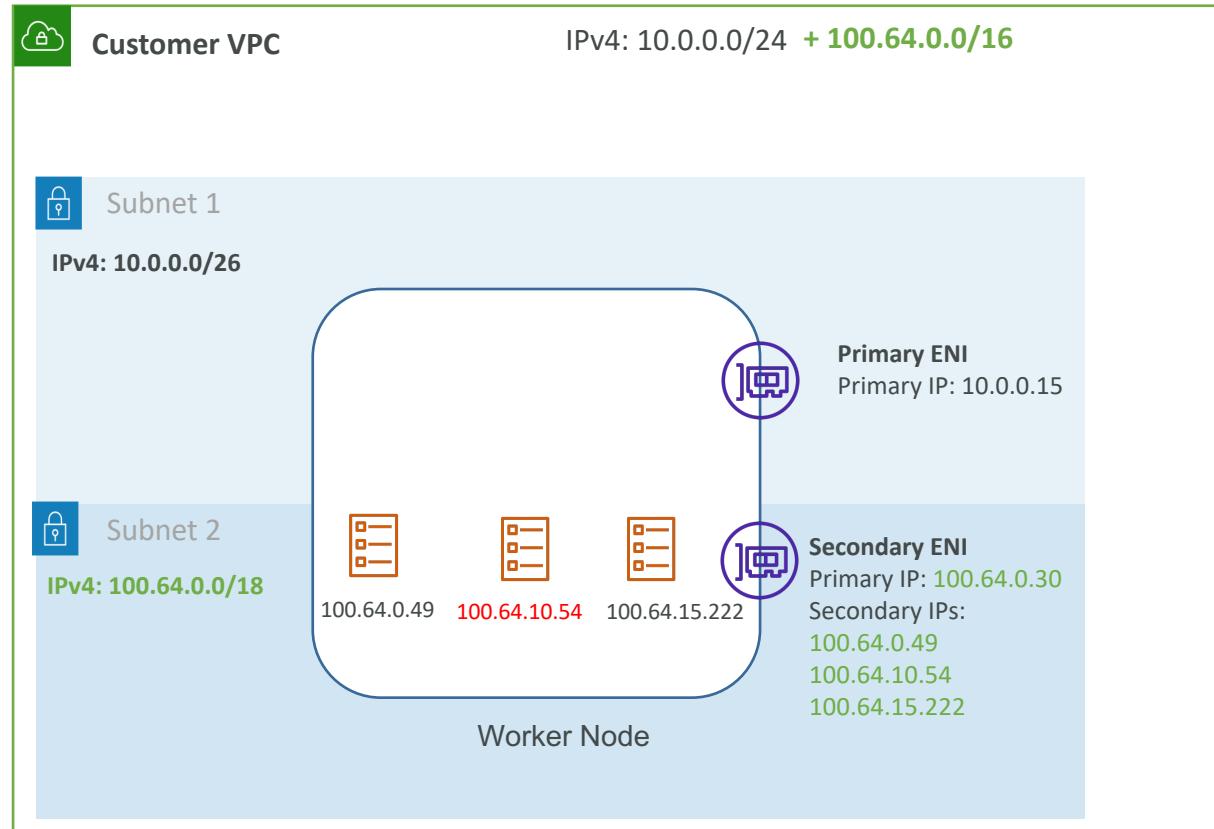
# Pod Network – Custom networking (IPv4)



## Problem

- If you have limited IP space, it will constraint the number of Pods
- /24 CIDR will have 251 unique IPv4 addresses

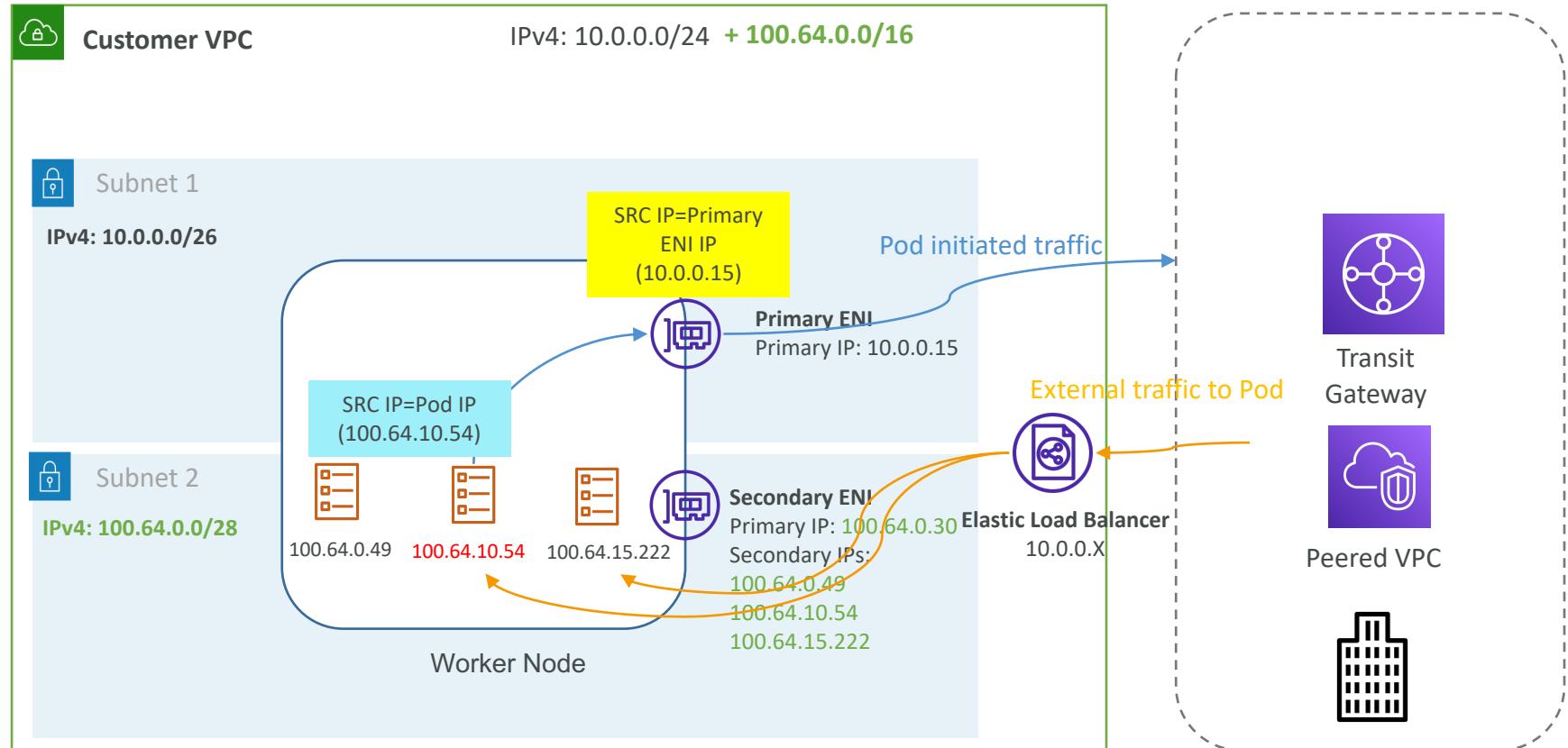
# Pod Network – Custom networking



- Add Secondary VPC CIDR in the range 100.64.0.0/16 (~65000 IPs) to the VPC
- This CIDR IP addresses are routable only within the VPC
- Enable VPC CNI Custom Networking
- VPC CNI plugin creates Secondary ENI in the separate subnet
- Only IPs from Secondary ENI are assigned to Pods
- Custom Networking can be combined with SNAT

```
kubectl set env daemonset aws-node -n kube-system AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true  
kubectl set env daemonset aws-node -n kube-system AWS_VPC_K8S_CNI_EXTERNALSNAT=false
```

# Pod Network – Custom networking



```
kubectl set env daemonset aws-node -n kube-system AWS_VPC_K8S_CNI_CUSTOM_NETWORK_CFG=true
kubectl set env daemonset aws-node -n kube-system AWS_VPC_K8S_CNI_EXTERNALSNAT=false
```

# EKS Networking Summary

# EKS Networking Summary

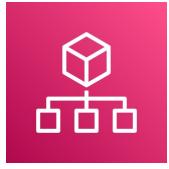
- EKS control plane is launched in AWS managed VPC and EKS data plane (worker nodes) is launched in customer VPC.
- EKS provisions ENIs into customer VPC to enable communication between EKS control plane and data plane
- EKS cluster API endpoint is publicly accessible by default but can be configured as a private in which case it can be accessed from customer VPC via the EKS owned ENI
- EKS uses Amazon VPC Container Network Interface (CNI) plugin for Pod networking.
- CNI allocates IPs to each Pod from available **Secondary IPs**
- Maximum number of Pods per node depends on number of ENIs and number of IP addresses per ENI
- For supported Nitro based instance types, Pod per node limit can be increased using Prefix delegation (/28 for IPv4 and /80 for IPv6)
- Custom Networking enables associating secondary VPC CIDR (100.64.0.0/16) and when combined with SNAT enables much larger IPv4 private IPs for Pods.
- CNI allows Nodes to enable/disable SNAT to allow outbound internet access to Pods through the Internet gateway or NAT gateway respectively.

# EKS Networking Summary

- By default, ENI security group is assigned to all the Pods which have been allocated secondary IPs for that ENI
- Pods specific security group can be assigned using Trunk & Branch ENI feature for selected Nitro system based instances.
- Pod services can be configured using ClusterIP, NodePort, LoadBalancer and Ingress resources.
- ClusterIP allows accessing services from within the cluster only.
- NodePort allows accessing services externally using Node IP and static port
- LoadBalancer service can be configured to use CLB or NLB in instance mode.
- Ingress service can be configured to use ALB in instance or IP mode.
- AWS Load Balancer Controller can be used for LoadBalancer (with NLB IP mode) and Ingress service (with ALB) configurations.
- `externalTrafficPolicy=Local` allows NLB in instance mode to preserve client IP address by disabling kube-proxy to send traffic to other nodes.

# AWS Management & Governance

# AWS Management & Governance services



AWS Organizations



AWS Control Tower



AWS Personal Health Dashboard



AWS Cost & Usage Report



AWS License Manager



AWS OpsWorks



AWS Systems Manager



Amazon CloudWatch



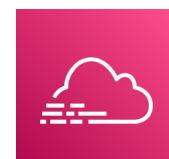
AWS CloudFormation



AWS Service Catalog



AWS Config



AWS CloudTrail

# We will focus on..

- AWS CloudFormation and CDK
- AWS Service Catalog
- AWS Config
- AWS CloudTrail

# AWS CloudFormation

# Infrastructure as a Code

- If you have been doing a lot of manual work for deployment of the infrastructure in AWS e.g. creating VPC, Subnets, EC2 instances, VPN connection etc. ....
- All this manual work will be very tough to reproduce:
  - In another region
  - in another AWS account
  - Within the same region if everything was deleted
- Wouldn't it be great, if all our infrastructure was... code?

# AWS CloudFormation



- CloudFormation is a declarative way of outlining your AWS Infrastructure, for any resources (most of them are supported).
- For example, within a CloudFormation template, you say:
  - I want a VPC and Subnets
  - I want an internet gateway and attach it to the VPC
  - I want a security group
  - I want two EC2 machines using this security group in the subnet just created
- Then CloudFormation creates those for you, in the **right order**, with the exact configuration that you specify

# Benefits of AWS CloudFormation

## Infrastructure as code

- No resources are manually created, hence no manual errors
- The code can be version controlled for example using git
- Changes to the infrastructure are reviewed through code

## Cost

- Each resources within the stack is tagged with an identifier so you can easily see how much a stack costs you
- You can estimate the costs of your resources using the CloudFormation template
- Savings strategy: In Dev, you could automate deletion of templates at 5 PM and recreated at 8 AM, safely

# Benefits of AWS CloudFormation

## Productivity

- Ability to destroy and re-create an infrastructure on the fly
- Automated generation of Diagram for your templates!
- Declarative programming (no need to figure out ordering and orchestration)

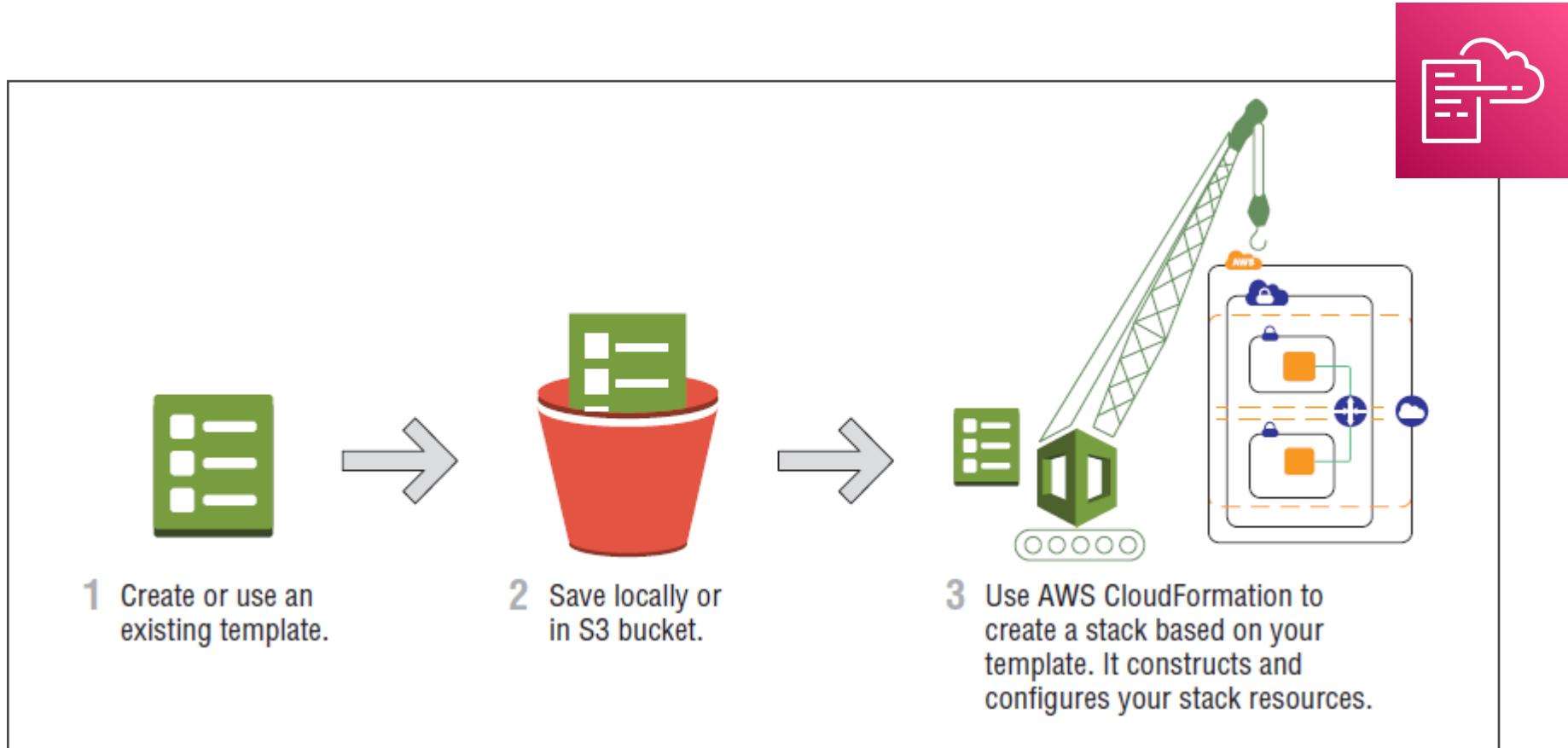
Separation of concern: create many stacks for many apps, and many layers. Ex:

- Network stacks
- App stacks

## Don't re-invent the wheel

- Leverage existing templates on the web!
- Leverage the documentation

# AWS CloudFormation



# CloudFormation template examples

```
{  
  "Type": "AWS::EC2::VPCPeeringConnection",  
  "Properties": {  
    "VpcId": String,  
    "PeerVpcId": String  
  }  
}
```

*VPC peering in the same account, same region*

```
{  
  "Type": "AWS::EC2::VPCPeeringConnection",  
  "Properties": {  
    "PeerOwnerId": String,  
    "PeerRegion": String,  
    "PeerRoleArn": String,  
    "PeerVpcId": String,  
    "Tags": [ Tag, ... ],  
    "VpcId": String  
  }  
}
```

*VPC peering cross account cross region*

# CloudFormation – Feature & Components

- **CloudFormation Designer**
  - A graphical tool for creating, viewing, and modifying AWS CloudFormation templates
- **ChangeSets**
  - Generate & Preview the CloudFormation changes before they get applied
- **StackSets**
  - Deploy a CloudFormation stack across multiple accounts and regions
- **Stack Policies**
  - Prevent accidental updates / deletes to stack resources

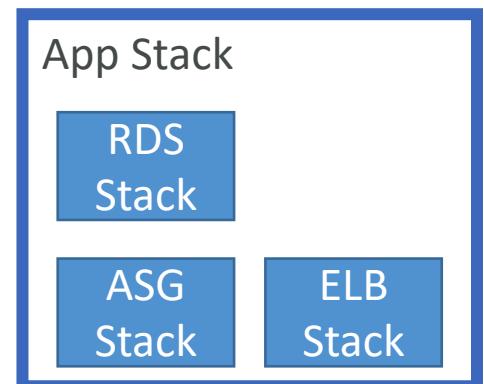
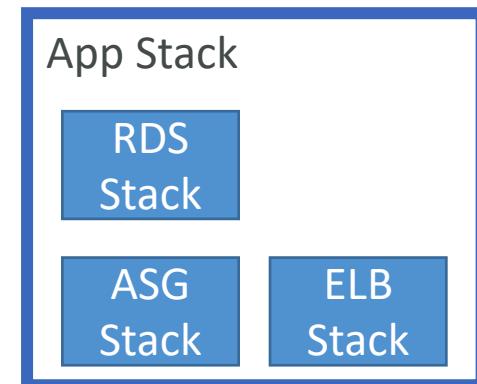
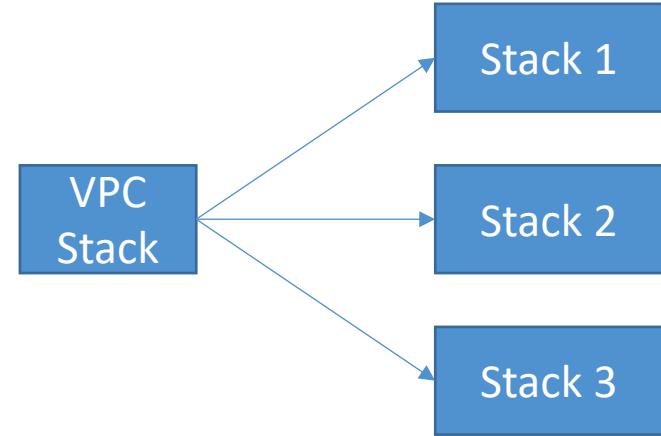
# CloudFormation – Feature & Components

- **Cross Stacks**

- Helpful when stacks have different lifecycles
- Use Outputs Export and Fn::ImportValue
- When you need to pass export values to many stacks (VPC Id, etc...)

- **Nested Stacks**

- Nested stacks are stacks created as part of other stacks using the AWS::CloudFormation::Stack resource.
- Helpful when components must be re-used
- Ex: re-use how to properly configure an Application Load Balancer
- The nested stacks are not shared.



# CloudFormation – manage resource dependencies

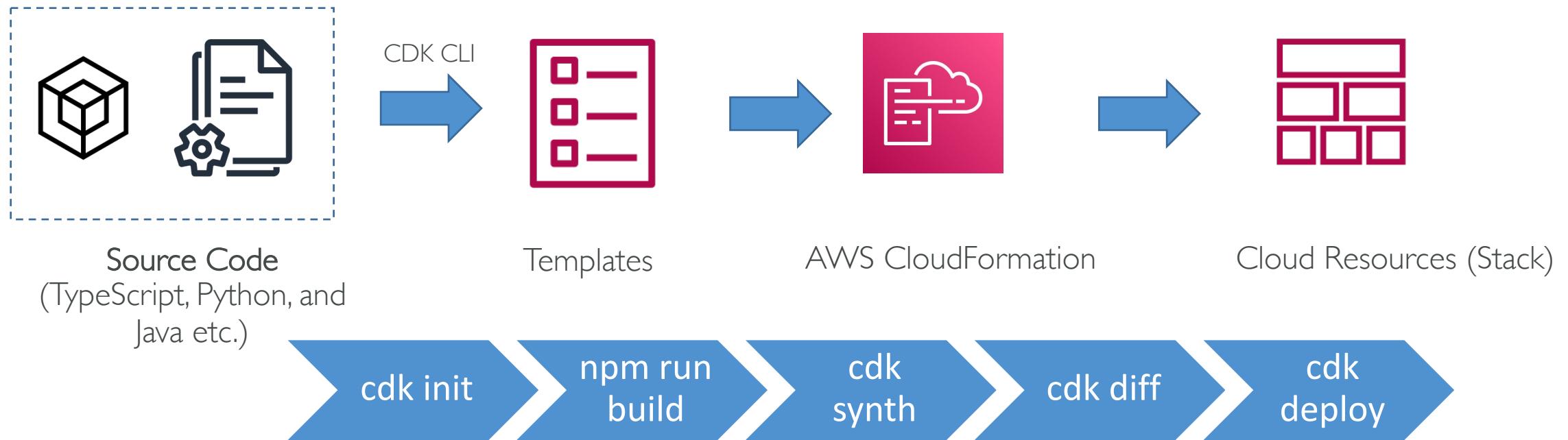
- DependsOn
  - Control the order of the resource creation. DependsOn will force the resource creation wait until the resource specified by “DependsOn” is created successfully.
  - Example:
    - If you reference a VPN gateway that is in the same template as your VPN gateway route propagation, you must explicitly declare a dependency on the VPN gateway attachment.
    - The AWS::EC2::VPNGatewayRoutePropagation resource cannot use the VPN gateway until it has successfully attached to the VPC.
    - Add a DependsOn Attribute in the AWS::EC2::VPNGatewayRoutePropagation resource to explicitly declare a dependency on the VPN gateway attachment.

# CloudFormation – manage resource dependencies

- WaitCondition
  - To coordinate stack resource creation with configuration actions that are external to the stack creation.
  - Waits until the success/failure signal received or timeout occurs
  - For the resources to respond to the wait condition, they must have an access to the cloudformation specific S3 bucket presigned URL where they can send the response signal
  - WaitCondition itself can DependsOn the underlying resource
  - Examples:
    - Create a WaitCondition which DependsOn EC2 instance.
    - The WaitHandler will wait for the signal to be received.
    - EC2 bootstrap action will send the Success signal to the wait condition.

# AWS Cloud Development Kit (CDK)

- An open-source software development framework to define your cloud application resources using familiar programming languages.

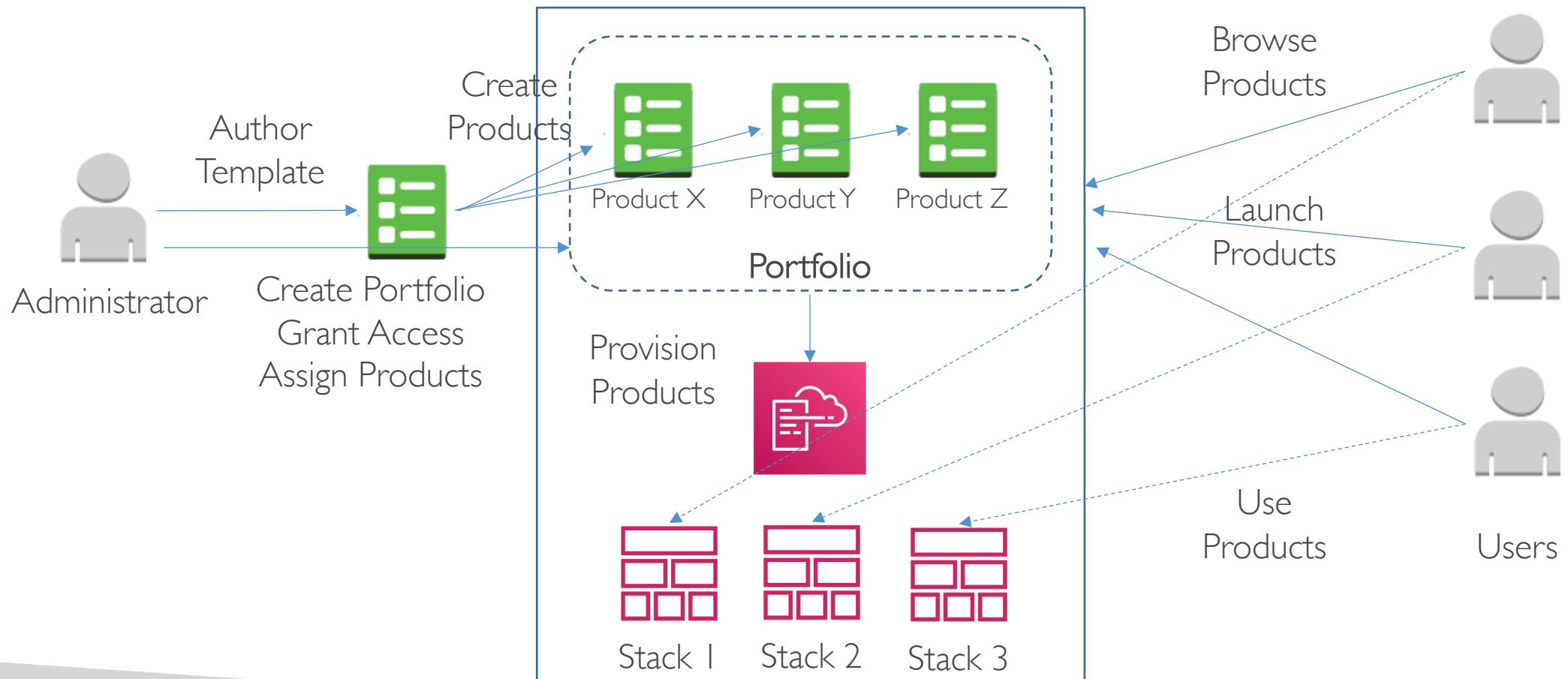


# AWS Service Catalog

# AWS Service Catalog

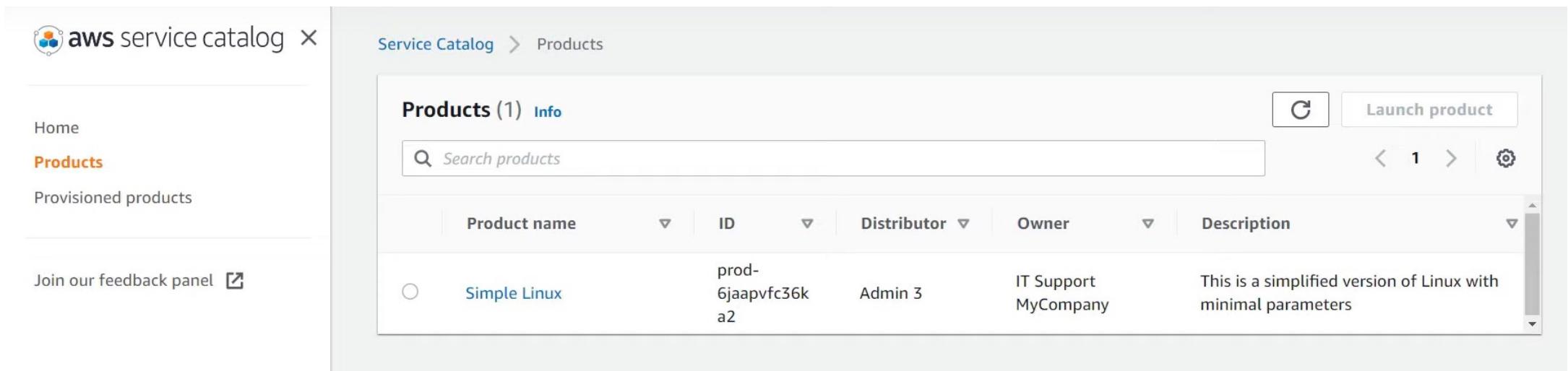
- Allow users to launch group of approved IT resources as a Product in a self-service manner
- It uses CloudFormation templates to launch the related resources / architecture / software / servers.
- Products can be versioned and can be shared across AWS organization, Organization Units (OU) or AWS accounts
- It uses user's IAM permissions or a launch constraints for launching the products
- Users can optionally provide the parameters.
- Administrator can also provide details like support email
- Output of the CloudFormation can also be part of the launch output e.g. Website URL etc.

# AWS Service Catalog



# AWS Service Catalog

- User browse the products listed in AWS Service catalog
- User selects the product
- User launches the product



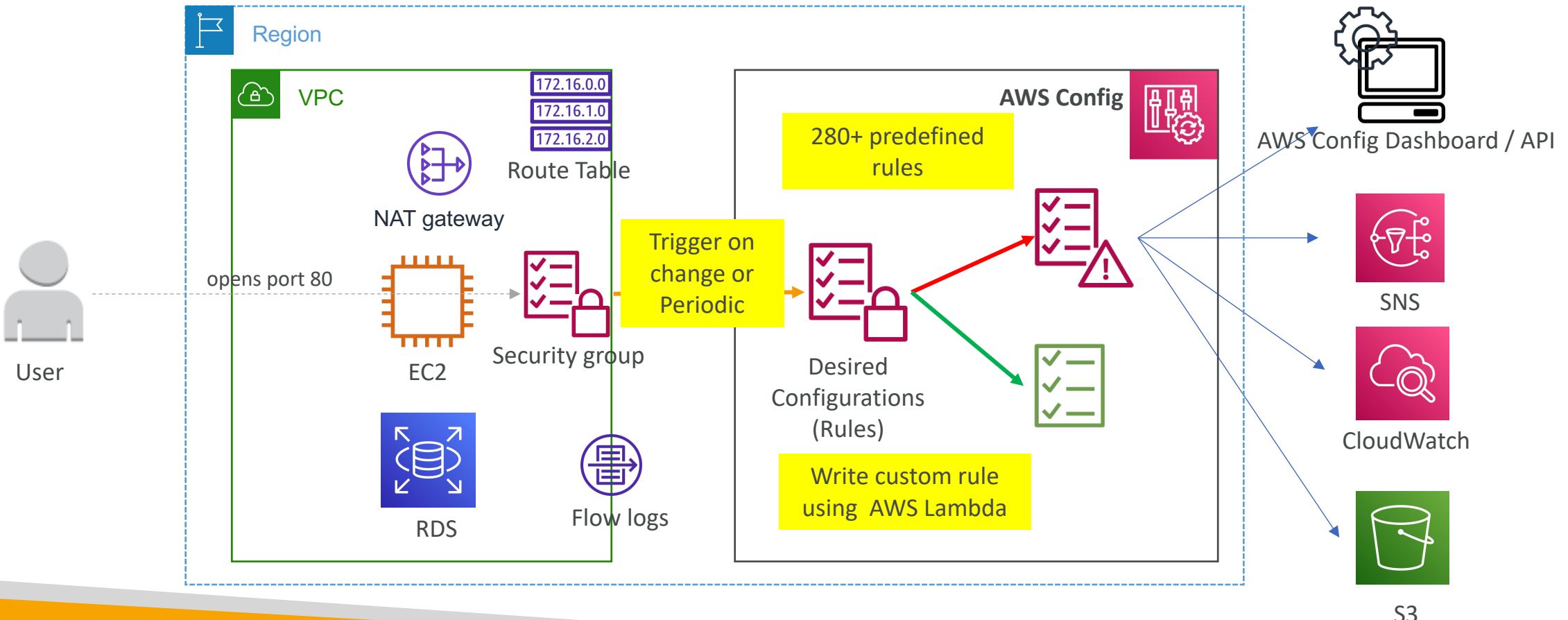
The screenshot shows the AWS Service Catalog interface. On the left, there's a sidebar with links: Home, Products (which is currently selected and highlighted in orange), Provisioned products, and Join our feedback panel. The main area is titled "Service Catalog > Products". It displays a table titled "Products (1)" with one item. The table has columns: Product name, ID, Distributor, Owner, and Description. The product listed is "Simple Linux" with ID "prod-6jaapvfc36ka2", owned by "Admin 3" and "IT Support MyCompany", and a description stating "This is a simplified version of Linux with minimal parameters". There are also "Launch product" and "Edit" buttons at the top right of the table.

| Product name | ID                 | Distributor | Owner                   | Description   |
|--------------|--------------------|-------------|-------------------------|---|
| Simple Linux | prod-6jaapvfc36ka2 | Admin 3     | IT Support<br>MyCompany | This is a simplified version of Linux with minimal parameters |

# AWS Config

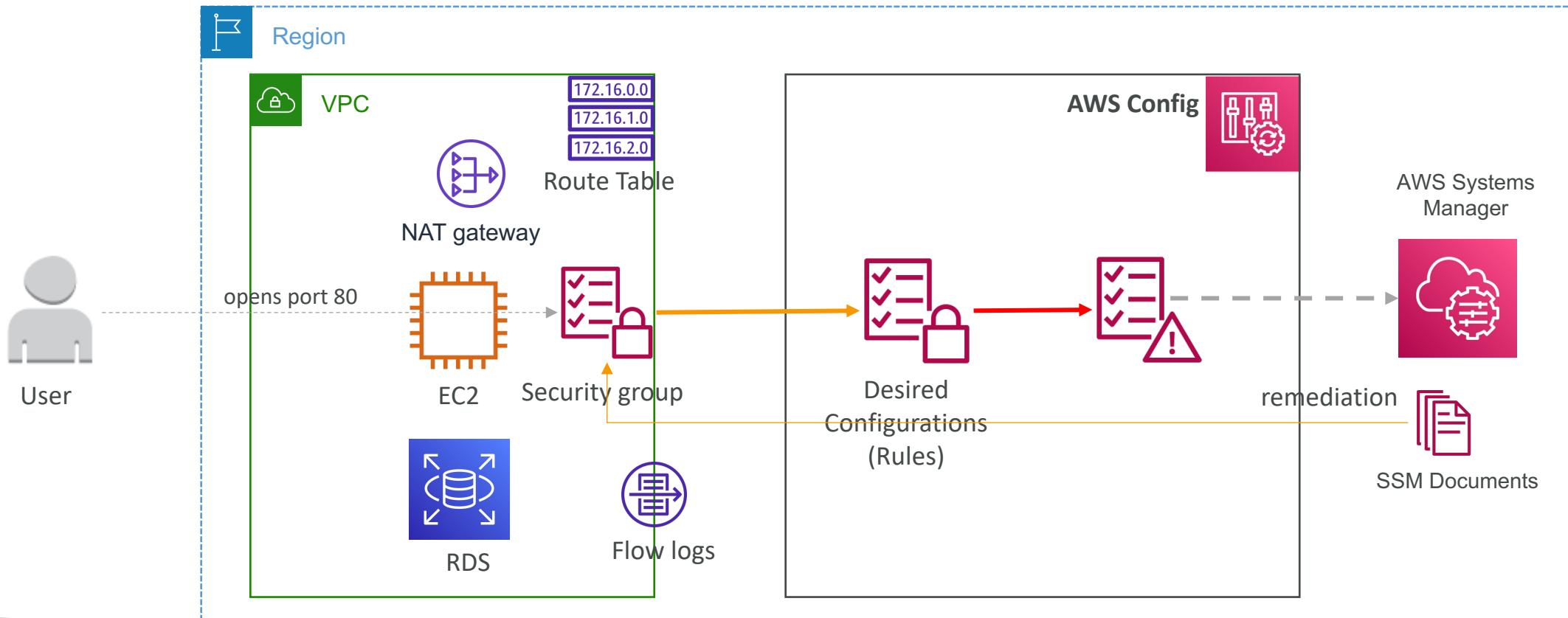
# AWS Config

Assess, Audit and Evaluate the configuration of your AWS resources



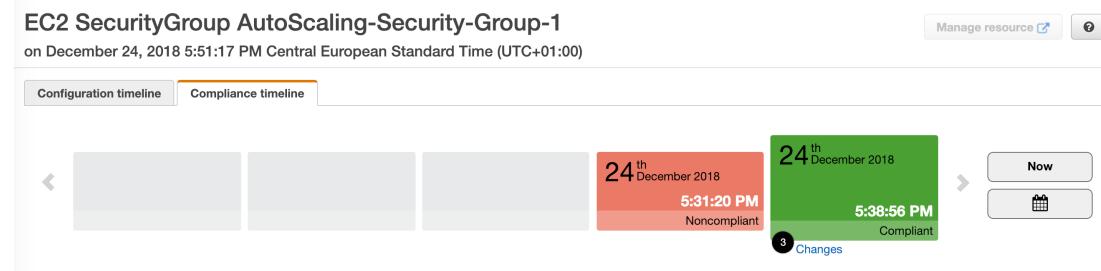
# AWS Config - Remediation

Auto remediation using Systems Manager SSM documents

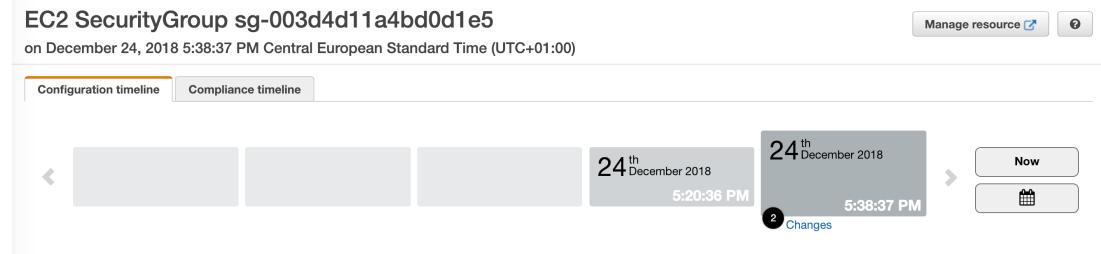


# AWS Config Resource

- View compliance of a resource over time



- View configuration of a resource over time



- View CloudTrail API calls if enabled



# AWS Config summary

- Helps record configurations and changes over time
- **AWS Config Rule does not prevent actions from happening (no deny)**
- Questions that can be solved by AWS Config:
  - Is there unrestricted SSH access to my security groups?
  - Do my buckets have any public access?
  - How my ALB configuration has changed over time?
- Can make custom config rules (must be defined in AWS Lambda)
  - Example 1: Evaluate if each EBS disk is of type gp2
  - Example 2: Evaluate if each EC2 instance is t2.micro
- AWS Config is a regional service however can aggregate data across multiple regions and accounts

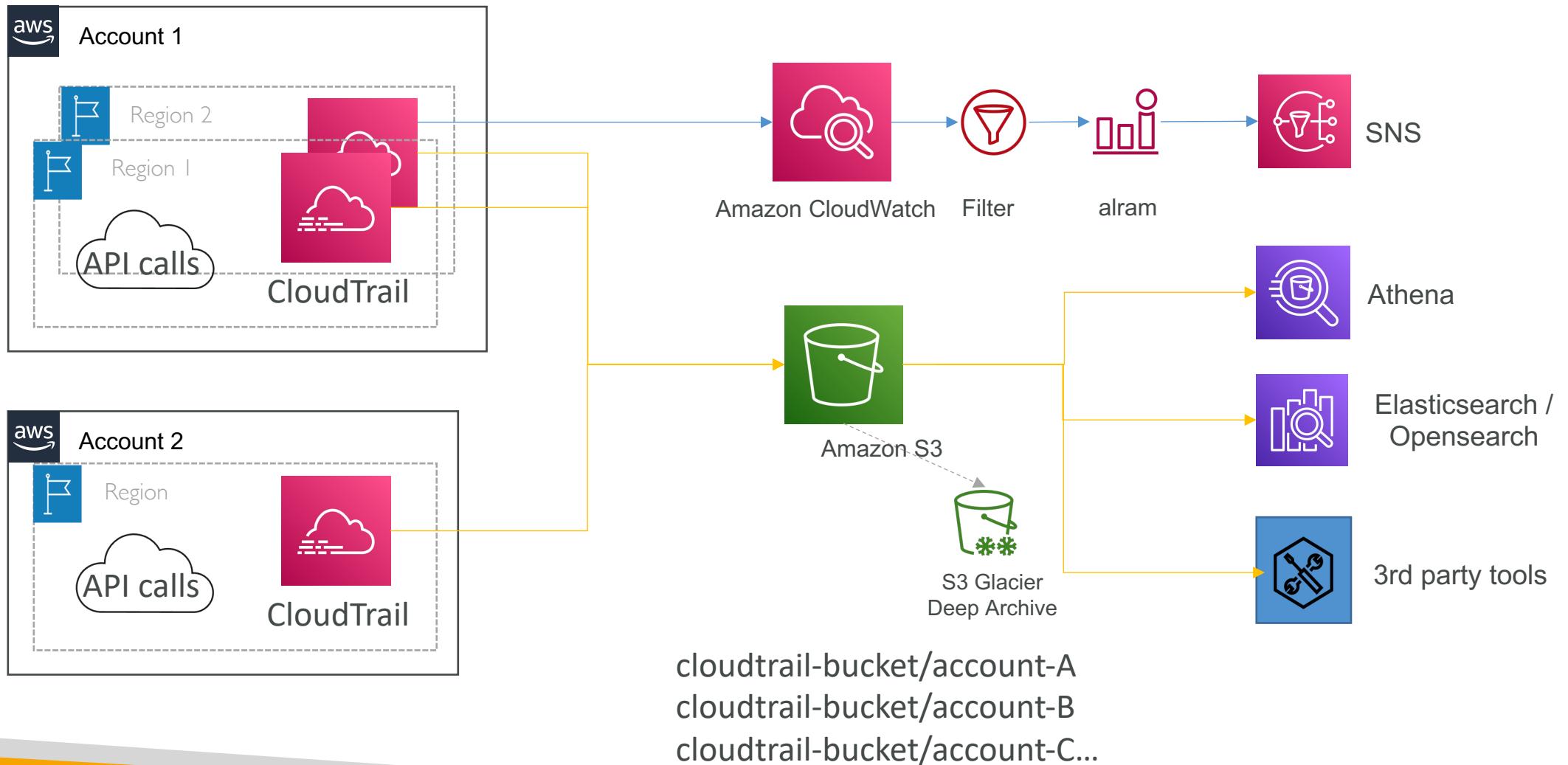
# AWS CloudTrail

# AWS CloudTrail



- Provides audit for your AWS Account activities by logging all the API calls
- CloudTrail is enabled by default!
- Get an history of events / API calls made within your AWS Account by:
  - Console
  - SDK
  - CLI
  - AWS Services
- If a resource is deleted in AWS, look into CloudTrail first!
- CloudTrail console shows the past 90 days of activity. Optionally, you can persist the CloudTrail logs into CloudWatch or S3
- Can be region specific or global & include global events (e.g. IAM)

# CloudTrail Trail

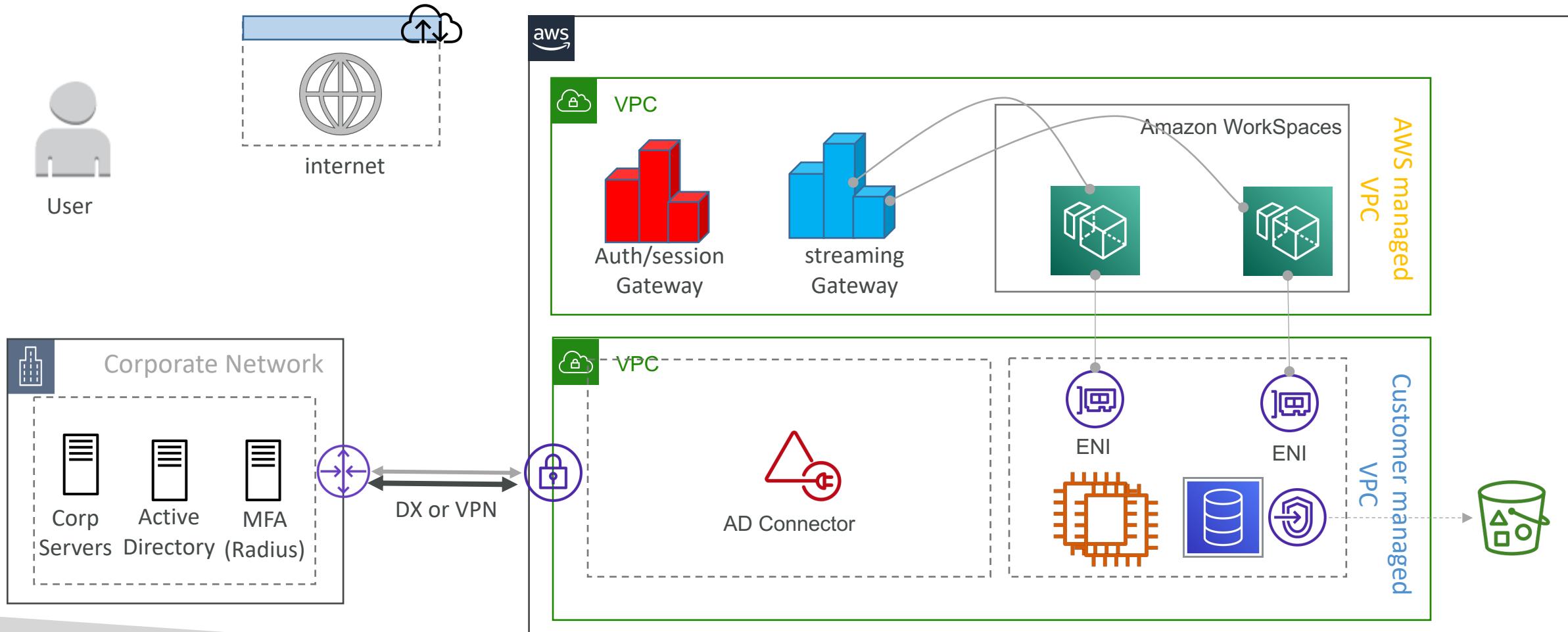


# Sample event

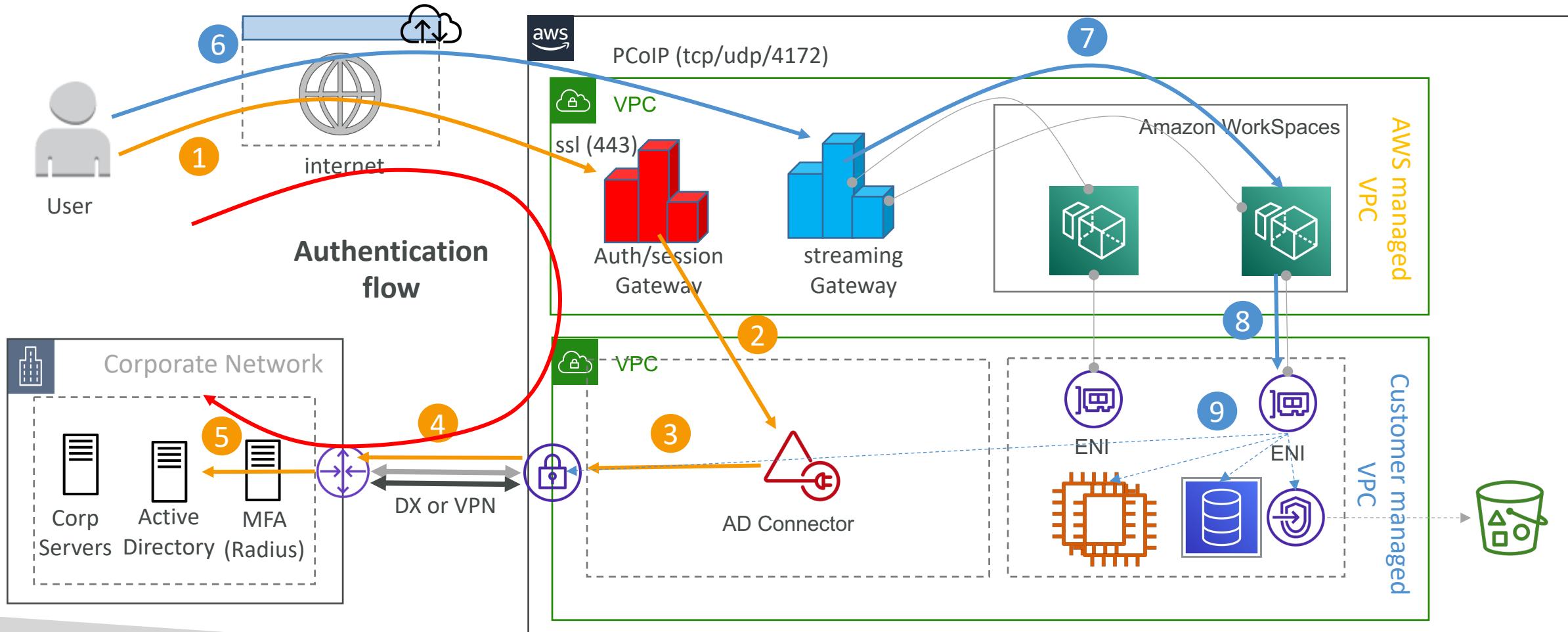
```
{  
  "eventVersion": "1.08",  
  "userIdentity": {  
    "type": "IAMUser",  
    "principalId": "xxxxxxxxxxxxxxxxxxxx",  
    "arn": "arn:aws:iam::xxxxxxxxxx:user/admin",  
    "accountId": "xxxxxxxxxxxx",  
    "accessKeyId": "xxxxxxxxxxxxxxxxxxxx",  
    "userName": "admin" ← User  
    ...  
  },  
  "eventTime": "2022-07-22T02:02:13Z", ← Event time  
  "eventSource": "ec2.amazonaws.com",  
  "eventName": "TerminateInstances", ← Action  
  "awsRegion": "ap-south-1",  
  "sourceIPAddress": "AWS Internal",  
  "userAgent": "AWS Internal",  
  "requestParameters": {  
    "instancesSet": {  
      "items": [  
        {  
          ...  
          "instanceId": "i-xxxxxxxxxxxxx" ← Resource  
        }  
      ]  
    }  
  }  
}
```

# VPC resource access for Amazon Workspaces & Appstream2.0

# Amazon Workspaces/Appstream networking



# Amazon Workspaces/Appstream networking



# Private NAT Gateway

# Problems with Private IPs

- Limited by Private IP ranges defined by RFC1918
- Need to have separate Private IP range for different business units
- Microservices architecture require services to have their own Private IPs thereby requiring more and more Private IPs
- This causes..

Difficulty to establish connectivity between the business units with overlapping CIDRs

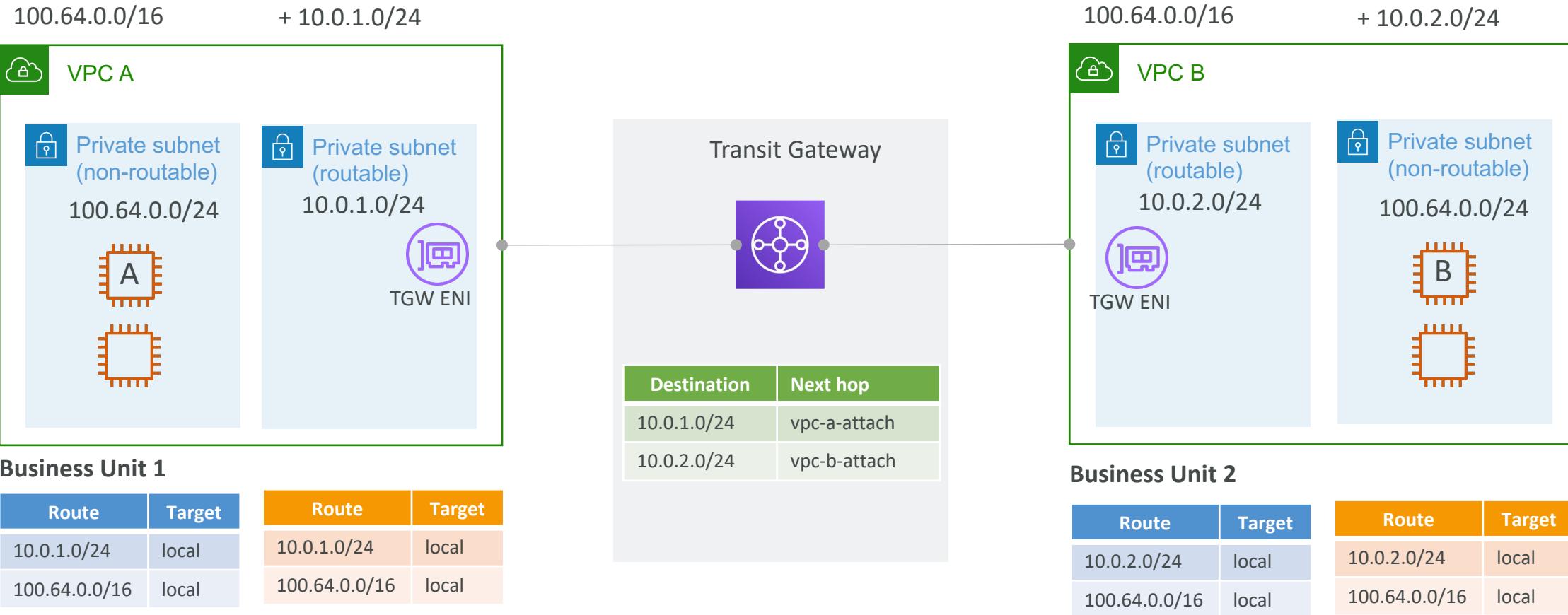
# Possible solutions for overlapping IP ranges

- Using AWS PrivateLink
- Using IPv6 addresses
- Using self managed NAT'ing appliances
  - Additional appliances to manage
  - Operational overhead

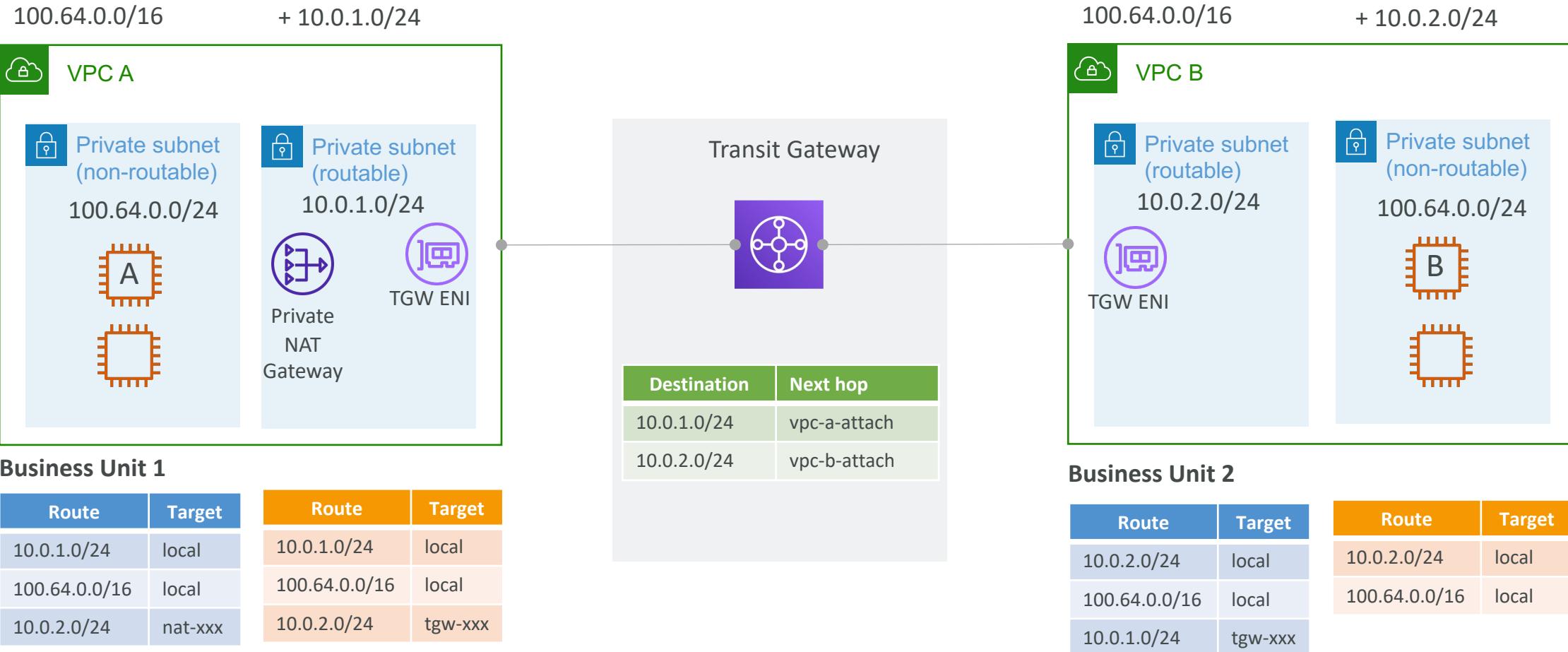
# Private NAT Gateway

- Allows communication between VPCs or VPC and on-premises network having the overlapping CIDRs
- Networks could be connected over Virtual Private Gateway (VGW) or Transit Gateway
- Private NAT performs the Private IP network address translation
- For deploying Private NAT you would often need to divide your address space in routable and non-routable (overlapping) address space and then configure the routing such that there is a communication between non-routable address range via the routable IP addresses

# Solution Architecture :VPC A -> VPC B

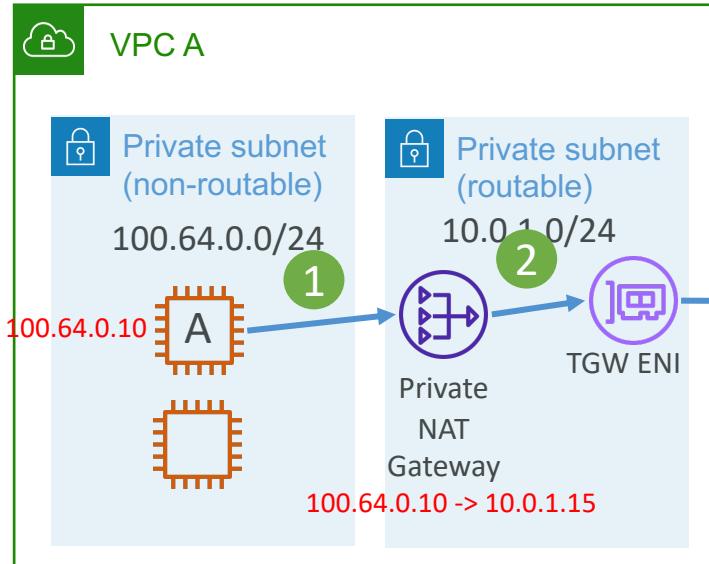


# Solution Architecture :VPC A -> VPC B



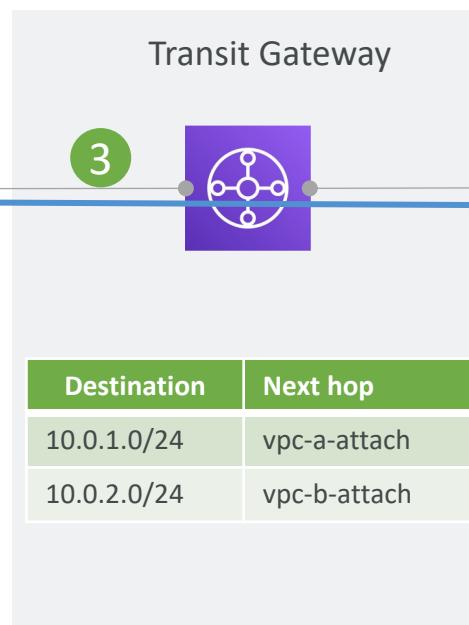
# Solution Architecture: VPC A -> VPC B

100.64.0.0/16 + 10.0.1.0/24



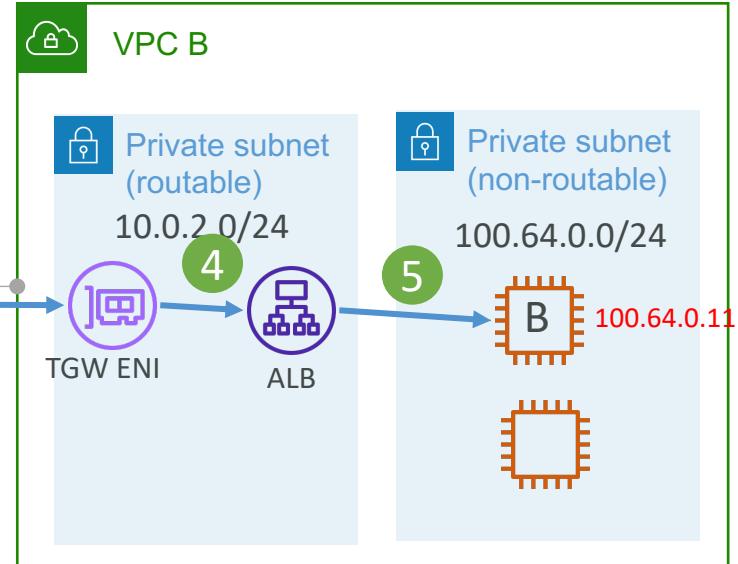
## Business Unit 1

| Route         | Target  | Route         | Target  |
|---------------|---------|---------------|---------|
| 10.0.1.0/24   | local   | 10.0.1.0/24   | local   |
| 100.64.0.0/16 | local   | 100.64.0.0/16 | local   |
| 10.0.2.0/24   | nat-xxx | 10.0.2.0/24   | tgw-xxx |



Instance A -> ALB DNS  
100.64.0.10 -> 10.0.2.10

100.64.0.0/16 + 10.0.2.0/24

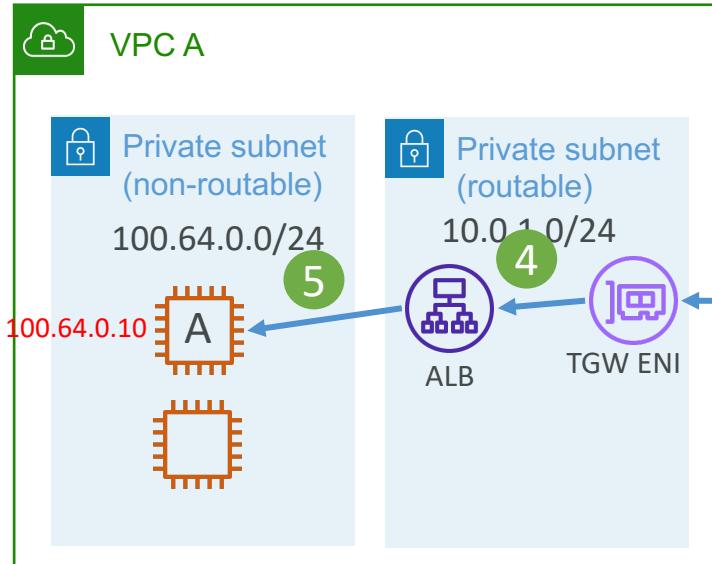


## Business Unit 2

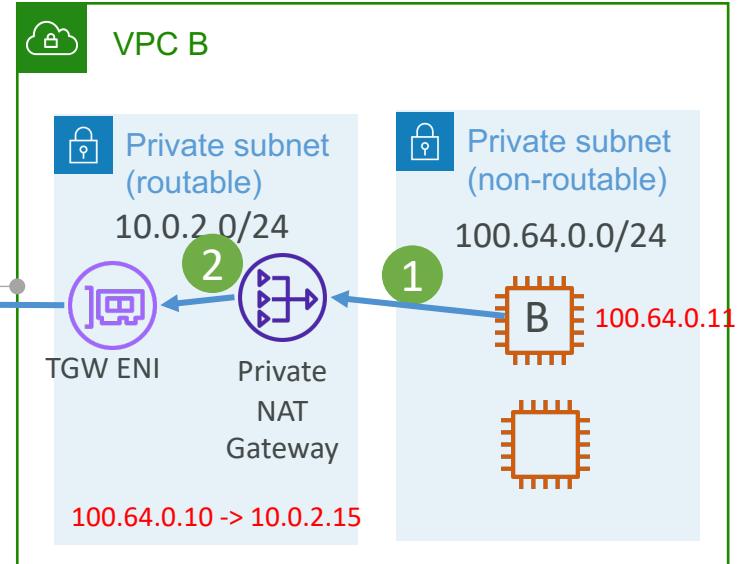
| Route         | Target  | Route         | Target |
|---------------|---------|---------------|--------|
| 10.0.2.0/24   | local   | 10.0.2.0/24   | local  |
| 100.64.0.0/16 | local   | 100.64.0.0/16 | local  |
| 10.0.1.0/24   | tgw-xxx |               |        |

# Solution Architecture –VPC B -> VPCA

100.64.0.0/16 + 10.0.1.0/24



100.64.0.0/16 + 10.0.2.0/24

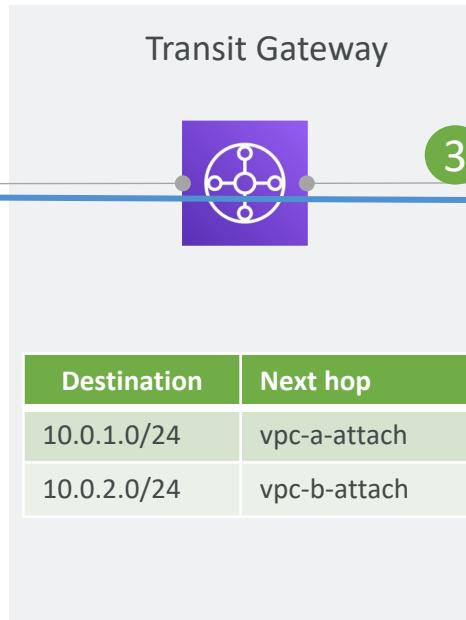


## Business Unit 1

| Route         | Target | Route         | Target |
|---------------|--------|---------------|--------|
| 10.0.1.0/24   | local  | 10.0.1.0/24   | local  |
| 100.64.0.0/16 | local  | 100.64.0.0/16 | local  |

| Route       | Target  |
|-------------|---------|
| 10.0.2.0/24 | tgw-xxx |



Instance B -> ALB DNS  
100.64.0.11 -> 10.0.1.10

## Business Unit 2

| Route         | Target  | Route         | Target  |
|---------------|---------|---------------|---------|
| 10.0.2.0/24   | local   | 10.0.2.0/24   | local   |
| 100.64.0.0/16 | local   | 100.64.0.0/16 | local   |
| 10.0.1.0/24   | tgw-xxx | 10.0.1.0/24   | nat-xxx |