AWS

VPC and Networking

Document Revisions

Document version: v2.1
Last updated: 08-Aug-2024

Changes:

1. Reordered the slides as per the published sections
2. Completely refreshed all the lectures with latest AWS Console experience and latest AWS services features

Why this course?

Download course slides from the resources section of this lecture

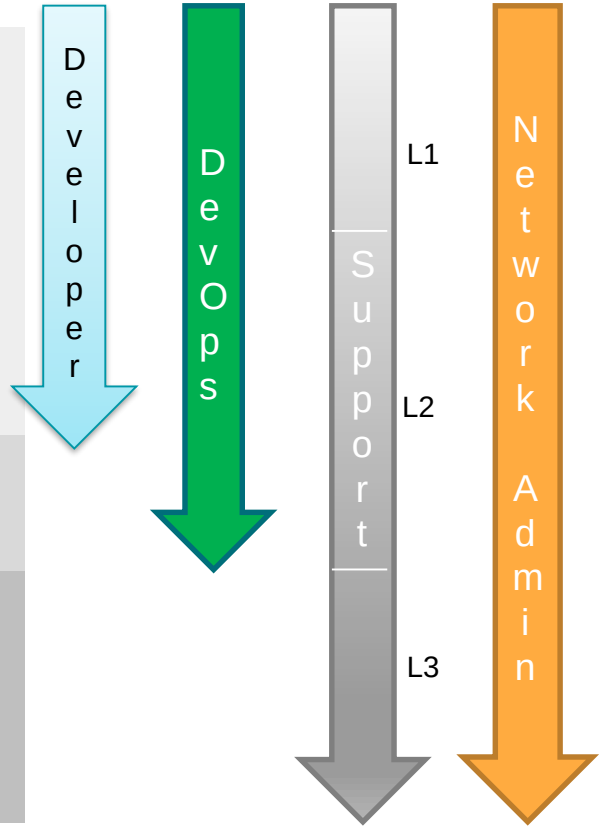
Which all AWS networking services you should know?

- VPC Fundamentals
- VPC Endpoint & PrivateLink
- VPC Peering
- Transit Gateway
- Route53 (basics)
- CloudFront (basics)
- Elastic Load Balancer
- API Gateway, AppSync
- VPC Lattice, AppMesh
- CloudFormation, Terraform
- AWS Network Manager
- VPC flow logs
- Site to Site VPN & Client VPN
- Direct Connect
- CloudWAN
- VPC (advanced)
- Route 53 (advanced)
- Network Security services

- ✓ AWS Region and AZs
- ✓ Network Addressing (CIDR)
- ✓ Internet Gateway
- ✓ Subnets
- ✓ Route Tables
- ✓ NAT
- ✓ IP addresses (Public, Private, Elastic),
- ✓ Security Groups
- ✓ Network ACL

- ✓ Network Automation
- ✓ Network logging
- ✓ Network Monitoring

- ✓ Enhanced Networking – high bandwidth & performance
- ✓ Hybrid Connectivity with Direct Connect / BGP routing
- ✓ Network Security
- ✓ IPS / IDS systems



What’s the right way to learn AWS Networking?

- ✓ See the big picture - Overview of AWS Networking Services
- ✓ Build the strong foundation - Understand the VPC & Networking Basics
- ✓ Experience it – By performing a lots of hands-on exercises

What are we going to learn in this course?

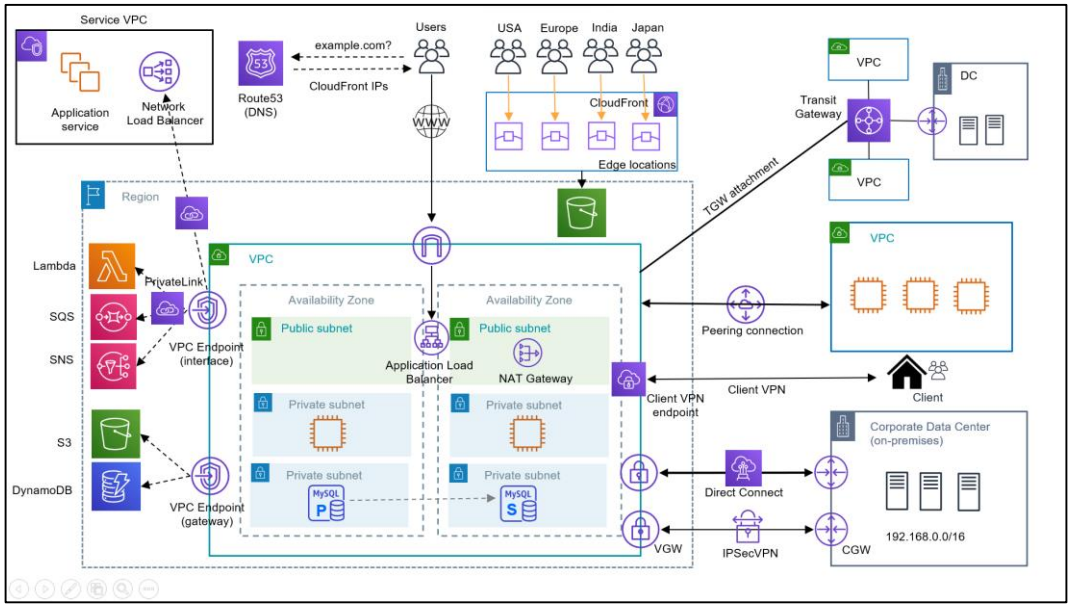
Build the strong foundation

VPC Fundamentals

- Understanding VPC CIDR
- Subnets & Route tables
- Internet gateway
- IP addresses – Private, Pubic, Elastic
- Network Address Translation (NAT)
- Security Groups & Network ACL

VPC Private Connectivity Options

- VPC Endpoints & PrivateLink
- VPC Peering
- Transit Gateway
- Site-to-Site VPN
- Client VPN
- Direct Connect



What are we going to learn in this course?

Hands-on exercises

1. Create VPC and launch EC2 instance in a Public Subnet and connect over SSH
2. Launch EC2 instance in a Private subnet and connect over SSH
3. Access outbound internet using NAT Gateway
4. Access outbound internet using NAT EC2 instance
5. VPC Gateway endpoint to access S3 privately
6. VPC interface endpoint to access SQS privately
7. VPC PrivateLink to access customer services privately
8. VPC peering across AWS regions
9. Transit Gateway with inter VPC connectivity
10. Transit Gateway with custom routing
11. Hybrid connectivity - AWS Site-to-Site VPN to access on-premises network
12. Custom domain name for your website using Route53 Public hosted zone
13. AWS region level failover with Route53
14. Application Load Balancer to distribute traffic across multiple webservers
15. Application Load Balancer Sticky session, path based and host based routing
16. Application Load Balancer with custom domain name and HTTPs (TLS termination)
17. Host a static website on Amazon S3
18. Secure the static website over HTTPS using Amazon S3 and CloudFront
19. Automate VPC deployment with AWS Cloudformation

and more…

+ Assignments
+ Project

What’s the right way to learn AWS Networking?

- ✓ See the big picture - Overview of AWS Networking Services
- ✓ Build the foundation - understand the basics
- ✓ Experience it - get hands-on experience

Exercise Pre-requisites

Create and setup your AWS Account

By Chetan Agrawal

Create and setup your AWS Account

1) Create an AWS account
2) Login to AWS account using root user
3) Verify account activation
4) Verify EC2 limits
5) Create a Billing Alarm
6) Create an IAM user
7) Create SSH key pairs

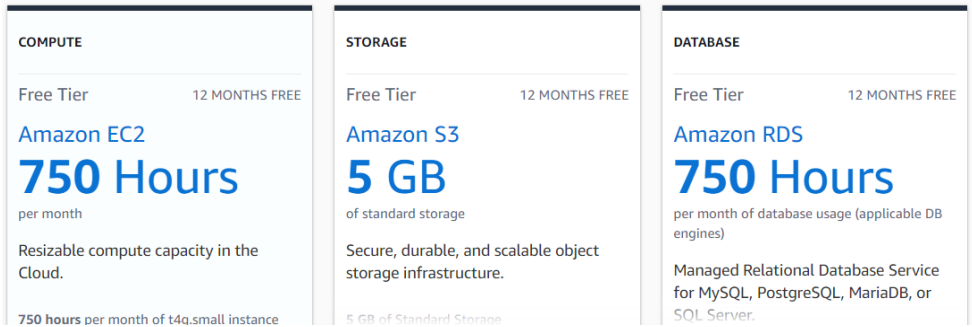
Create an AWS account

For creating AWS account, you will need:

1. Email Address
2. Mobile Number
3. Billing Address (no proofs required)
4. Valid Debit Card or Credit Card

For new AWS accounts you get 12 months Free tier for limited AWS services.

Visit https://aws.amazon.com/free/

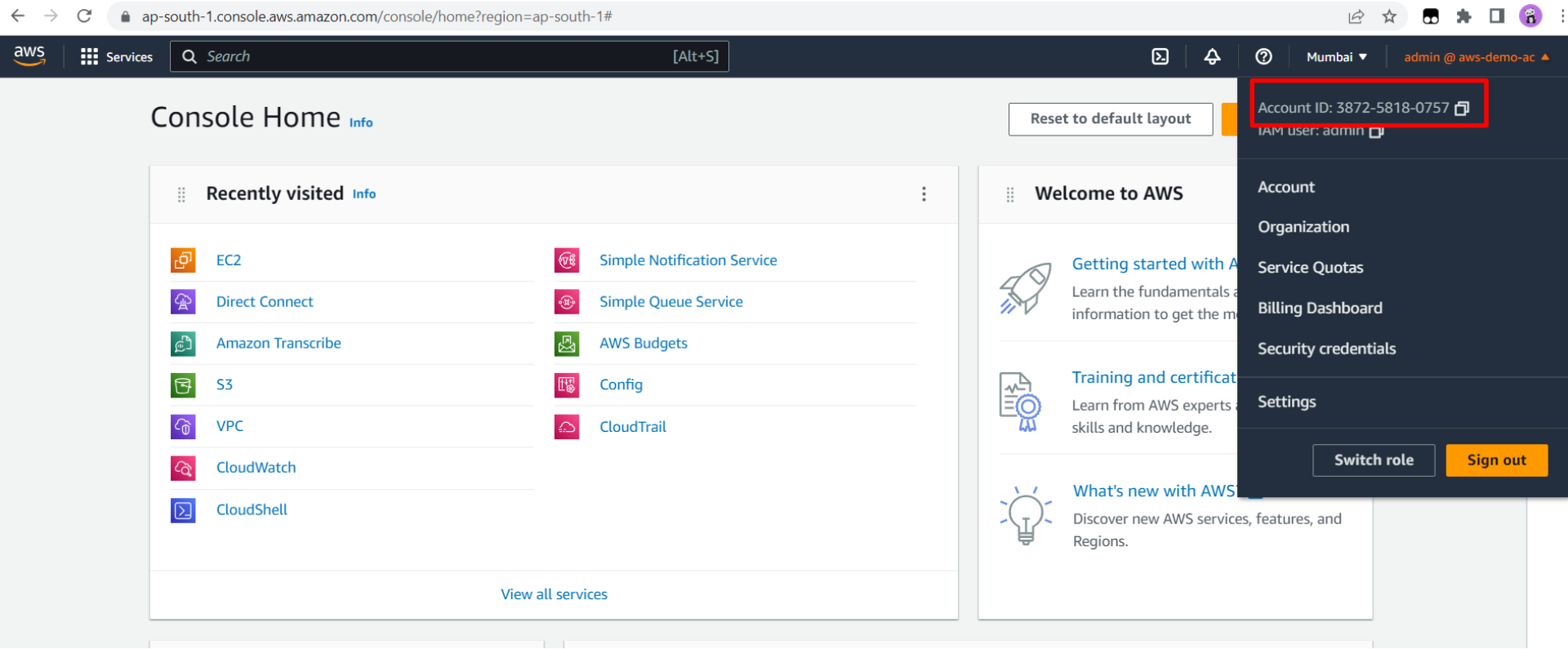


Create an AWS account

1. Go to https://aws.amazon.com -> Create an AWS Account
2. Provide all the details like Email Address, aws account password and unique account name
3. You should receive an email from AWS. Verify AWS Account by clicking the link in the email
4. Provide Phone, Billing Address details
5. Provide Debit/Credit card details
6. AWS authorizes small payment of $1 on your card. After successful authorization, this charge is reverted to your card
7. AWS also performs a phone number verification by calling you on your number where you have to enter the code shown on the screen
8. Account Type – Personal & Individual user and complete the registration process
9. Go to https://aws.amazon.com -> Sign In to the Console -> “Sign in using root user” and login to your account using your email id and aws account password
10. Note down your AWS account id by expanding top right dropdown

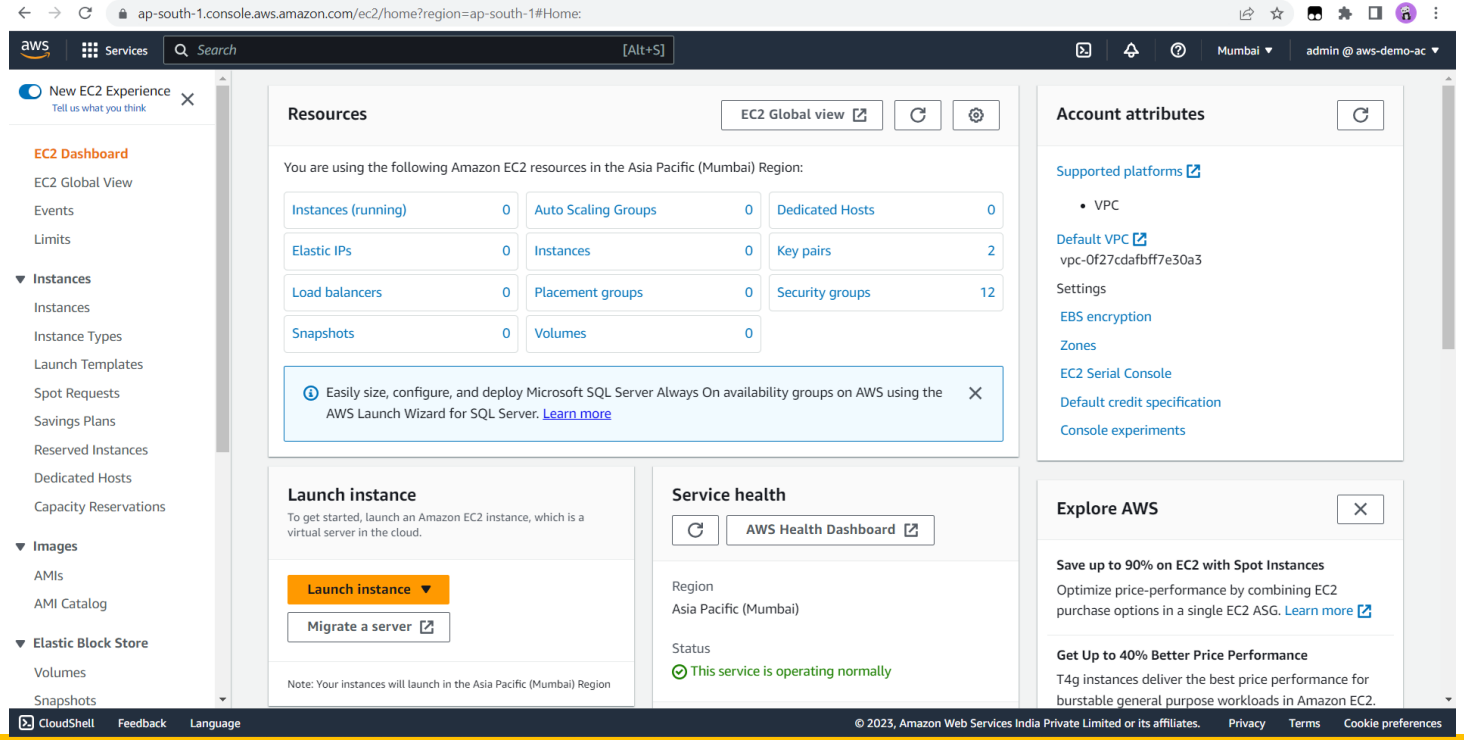
Note: For different reasons, sometimes your account may not be activated e.g. payment failed or you did not provide all the required information. Follow the instructions on the console to fix the problem. It may take up to 24-48 hrs to be fully activated.

Create an AWS account



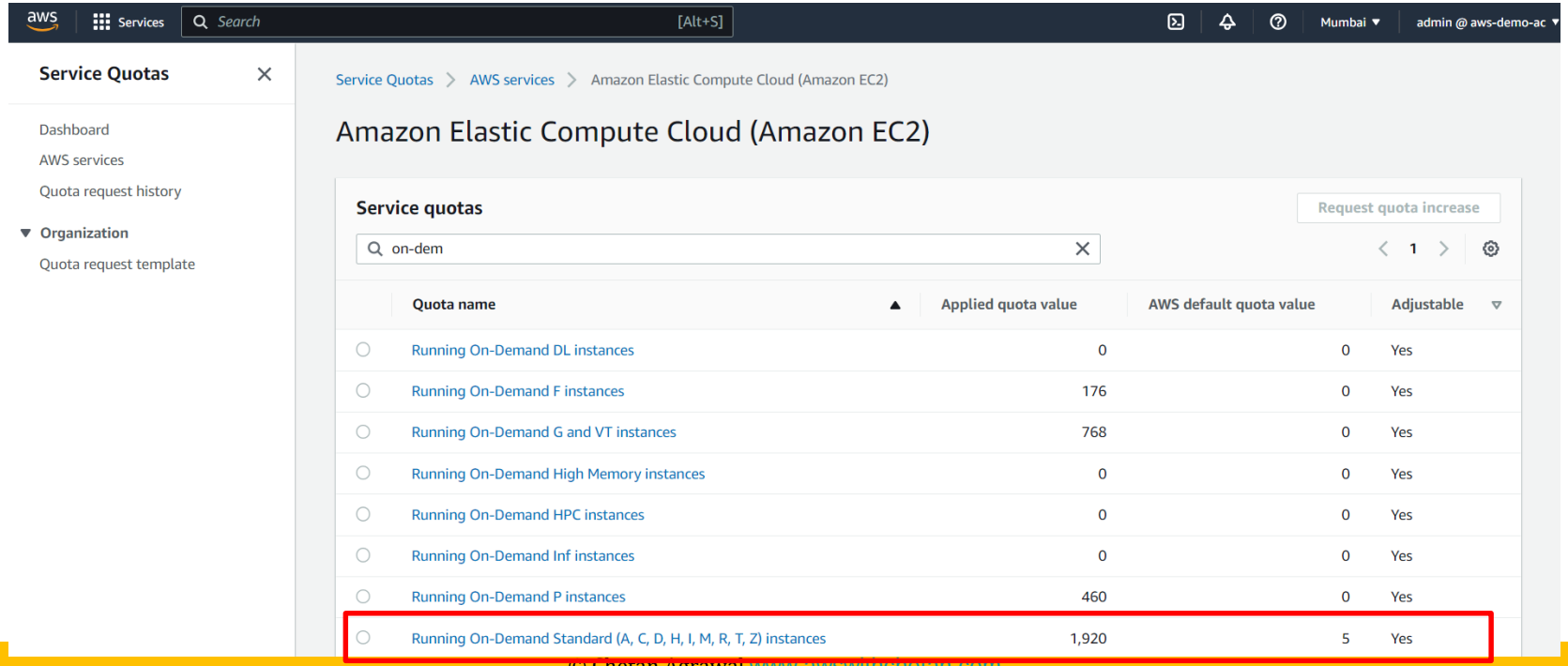
Verify account activation

1. Search for ec2 in the top search bar and go to ec2 console to verify that your account is fully active.



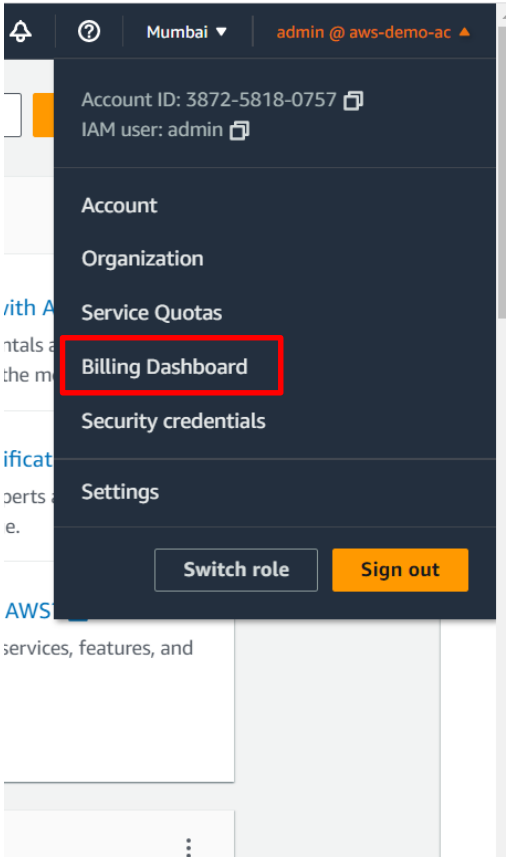
Verify EC2 limits

1. Go to EC2 console -> Limits -> Service Quota and verify that the Running On-Demand Standard (A, C, D, H, I, M, R, T, Z) instances quota is set to 5 or more.



Set a Billing alarm

1. Login to your AWS account using root user (email id/password)
2. Set a Billing alert so that you get notified when your AWS usage bill exceeds some threshold (say > $5/month)
    i. Billing Dashboard -> Billing Preferences -> Alert preferences -> Edit -> Receive CloudWatch Billing Alerts -> Save
    ii. Go to SNS -> Make sure you are in N.Virginia region (us-east-1) -> Create a topic -> **Standard** -> Name: BillingAlertTopic -> Create topic
    iii. Select Topic you just created -> Create subscription -> Email protocol -> Add your email id -> Create subscription
    iv. Go to CloudWatch -> Make sure you are in N.Virginia region (us-east-1) -> Alarms -> Billing
    v. Create alarm -> Metric Name:EstimatedCharges, Currency: USD, Statistic: Maximum, Period: 6 hours, Threshold Type: Static, Whenever EstimatedCharges is: **Greater**, than -> 5 (or whatever threshold you want to set. This is USD amount for your estimated usage) -> Next
    vi. Alarm State trigger: In alarm -> Send a notification to the following SNS topic -> Select an existing SNS topic -> Select the topic you created in Step ii above -> next
    vii. Alarm name: BillingAlarm5USD -> next -> Create alarm



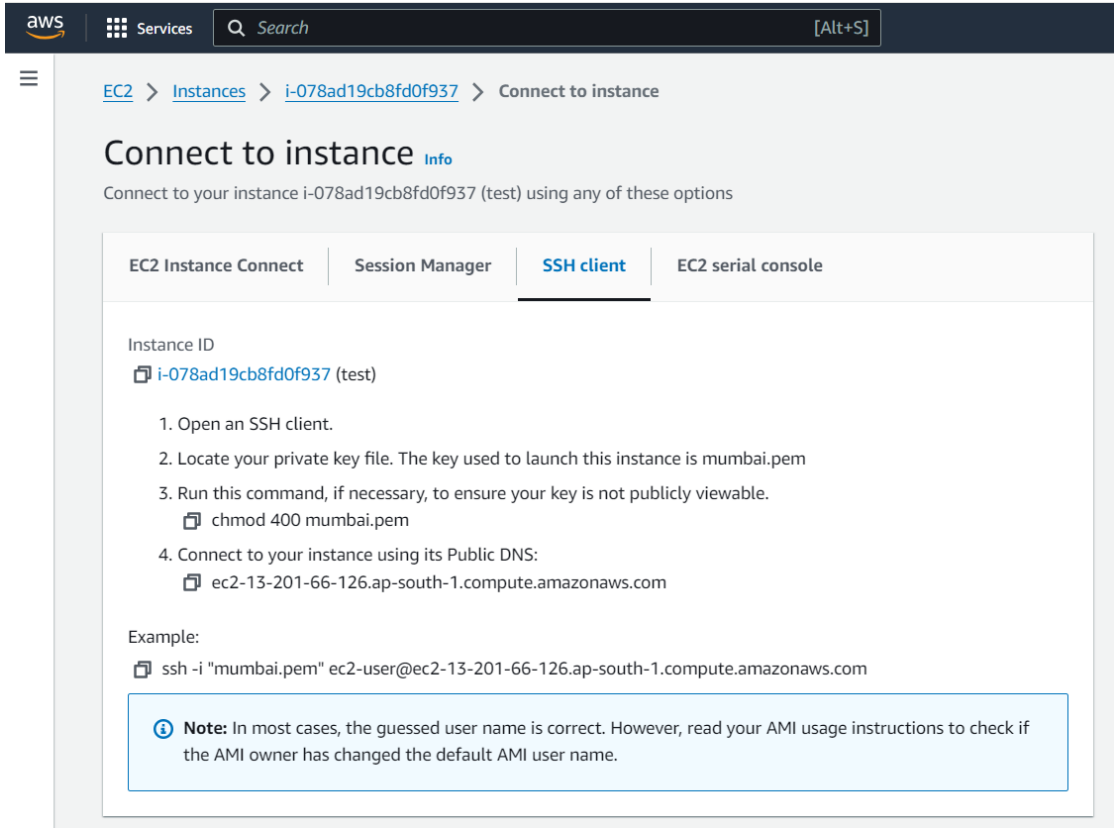
Create an IAM user

1. Login to your AWS account using Root user (email id/password)
2. Create IAM admin user. This user should be used for all the exercises. Do not use Root user.
    i. IAM console -> Users -> Add user
    ii. User name: admin (or whatever you prefer)
    iii. Check box: Provide user access to AWS Management console
    iv. Select: I want to create an IAM user
    v. Console Password: Custom password -> Use strong password
    vi. Uncheck: Users must create a new password at next sign-in -> Next
    vii. Permissions Options -> Attach Policies directly -> Permissions policies -> Select "AdministratorAccess" -> Next
    viii. Create User
3. Logout of your current session and log back in using IAM user
    i. Visit Go to https://aws.amazon.com -> Sign In to the Console
    ii. Enter IAM user name
    iii. Enter IAM user password

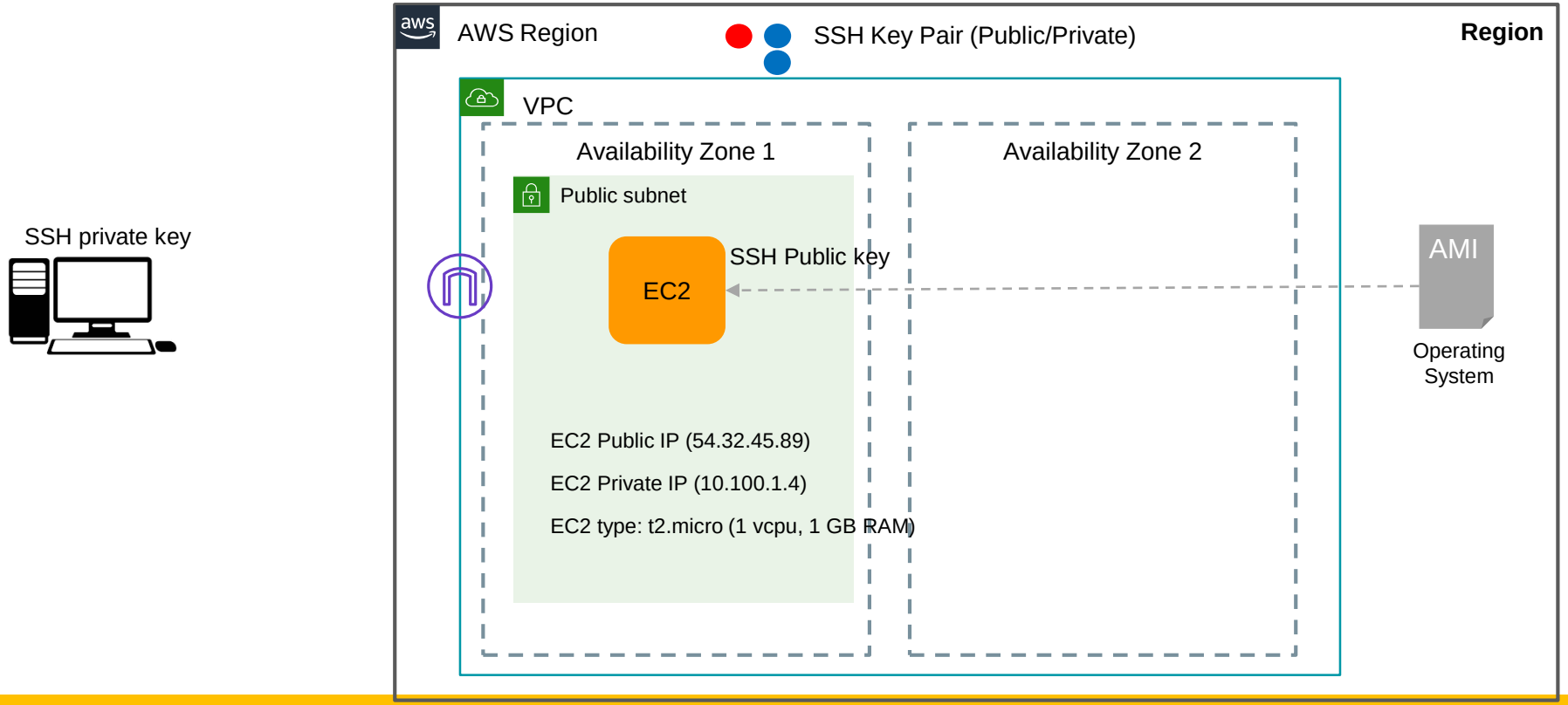
Create SSH key-pair to login to EC2 instances

By Chetan Agrawal

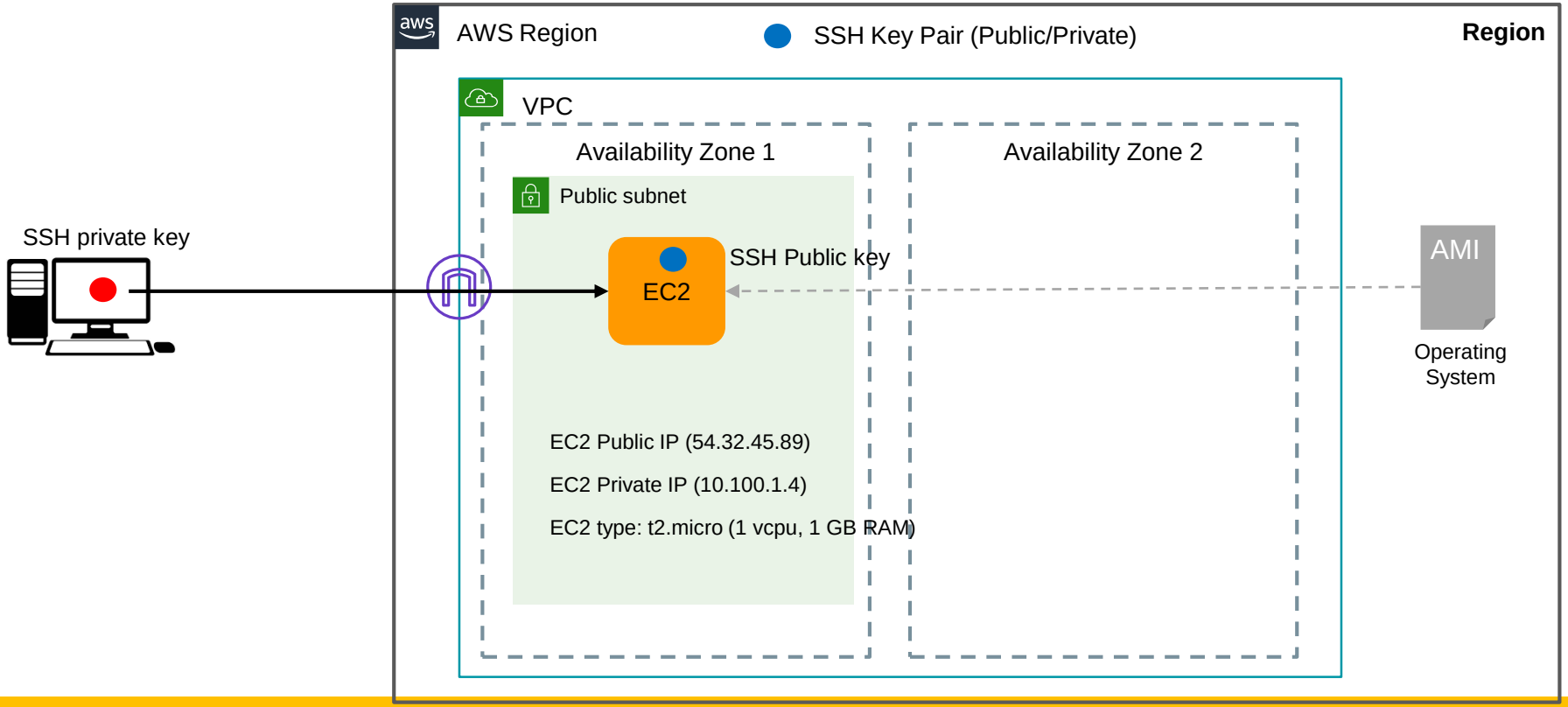
Connecting to EC2 instance



Process to launch EC2 instance

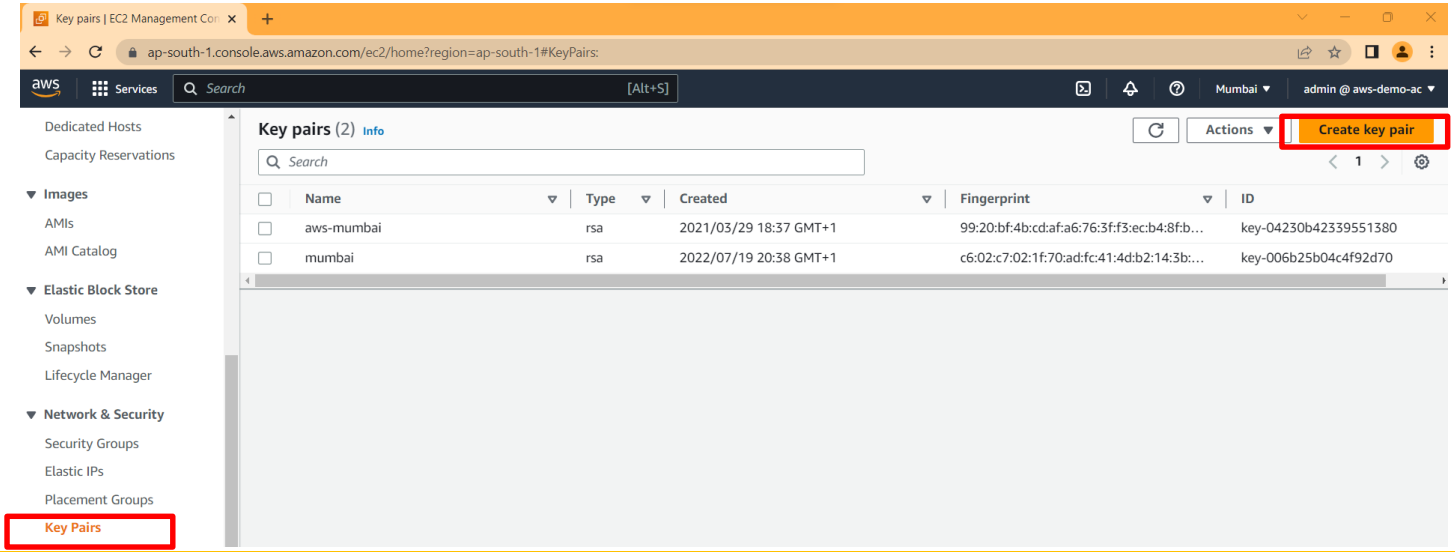


Process to launch EC2 instance



Create SSH key-pair

1. SSH key pairs have regional scope. Create SSH key pairs for the regions that you will be using for your exercises.
2. Go to **EC2 console** -> Left panel -> Network & Security -> Key pairs
3. Create key pair -> Name: mumbai-key, Key pair type: RSA, Private key file format: .pem -> Create key pair
4. Download and save .pem file in a secure location in your local workstation
5. Repeat the same process for your second AWS region.



Using SSH key

Windows Users:

1. Download Putty and Puttygen software from https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html
2. If you are having Windows OS in your local workstation then you would need to convert your .pem key format to .ppk
3. Open PuTTygen -> Load -> browse to .pem file (you may have to select All Files *.*) -> Save private key (RSA) -> Save (without passphrase)
4. Now you should have both .pem and .ppk key files saved in your local workstation
5. You can now use Putty client with .ppk file for logging into the Linux EC2 instance

Linux Users:

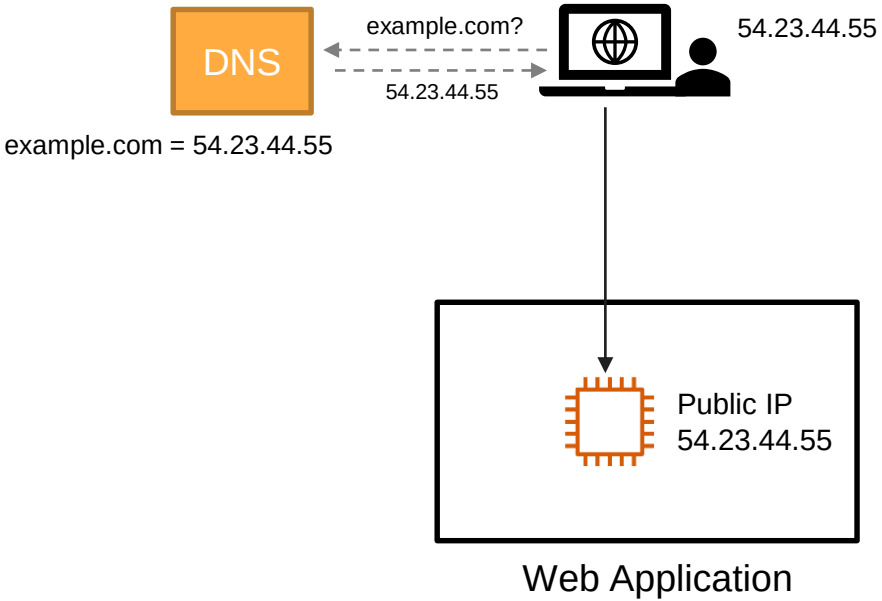
1. If you are having Linux/Ubuntu/Mac OS in your local workstation then you can use .pem key directly
2. Make sure you change the key permissions using command: *$chmod 400 <key.pem>*
3. You can now use native terminal and .pem file for logging into the Linux EC2 instance

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/connect.html

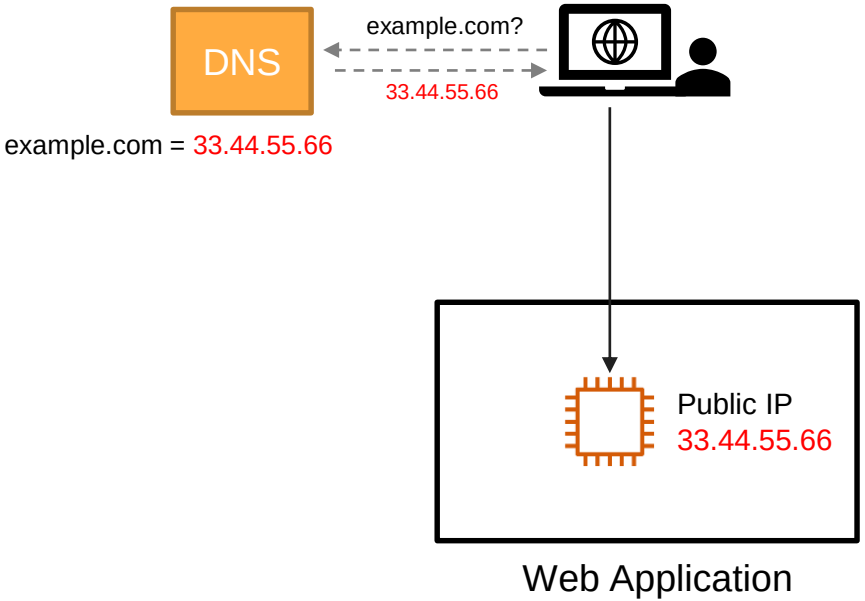
Get a Public Domain name and configure DNS

By Chetan Agrawal

Why we need Public domain name?

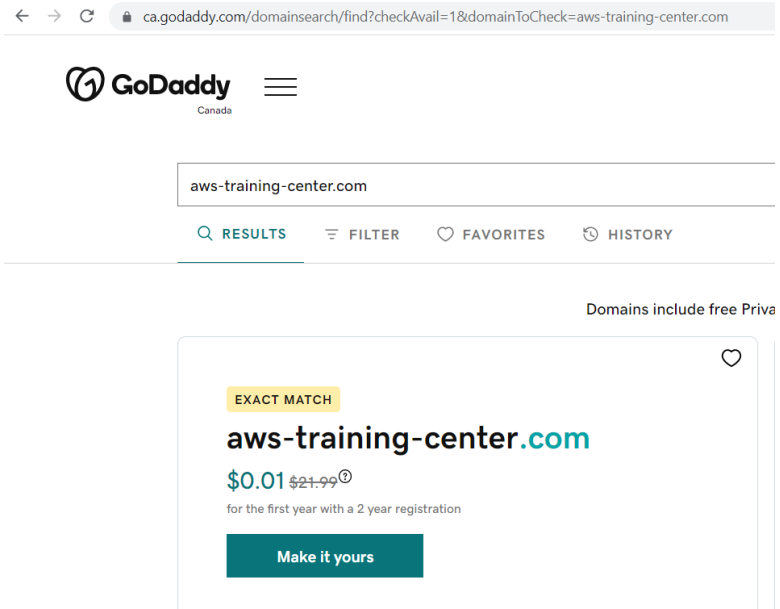


Why we need Public domain name?



Buy a public domain name of your choice

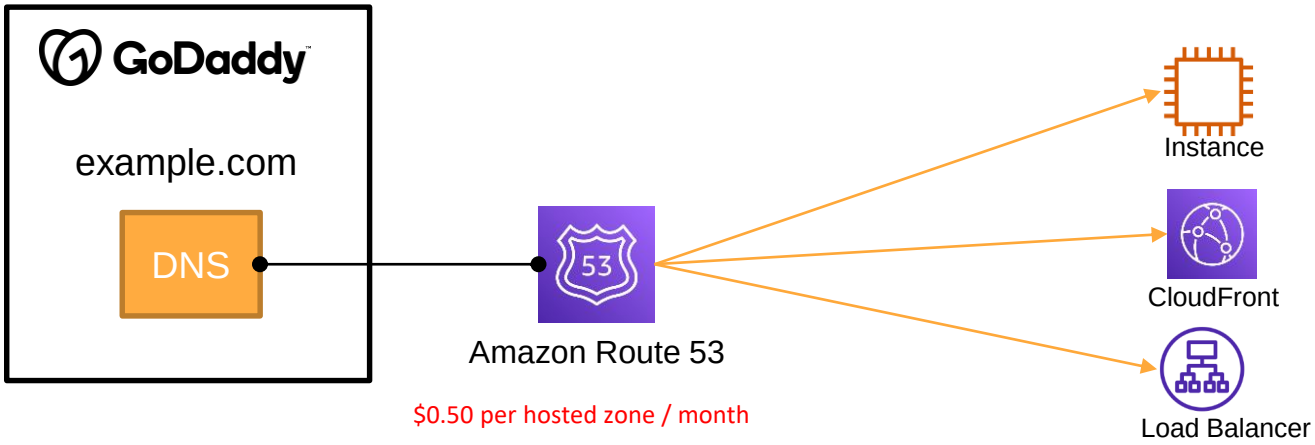
1. Some of the exercises need a public domain name e.g. awstrainingcenter.com
2. It's recommended that you buy any public domain name of your choice and use it for those exercises. e.g your_name.com
3. It may cost anywhere between $5 to $20 (or INR 300 to 1500) for 1 year depending on the domain name you choose.
4. You may want to purchase it from godaddy.com or any other public domain name service provider.



Note: www is a subdomain. Once you buy your domain name e.g. awstrainingcenter.com and then you can use any subdomain under it e.g. www.awstrainingcenter.com or web.awstrainingcenter.com likewise. Also, the Top Level Domain (TLD) can be anything e.g. .in or .live whatever. It need **not be** .com. Generally .com domains are little costlier than other top level domains.

Let’s setup a Route 53 DNS for your domain

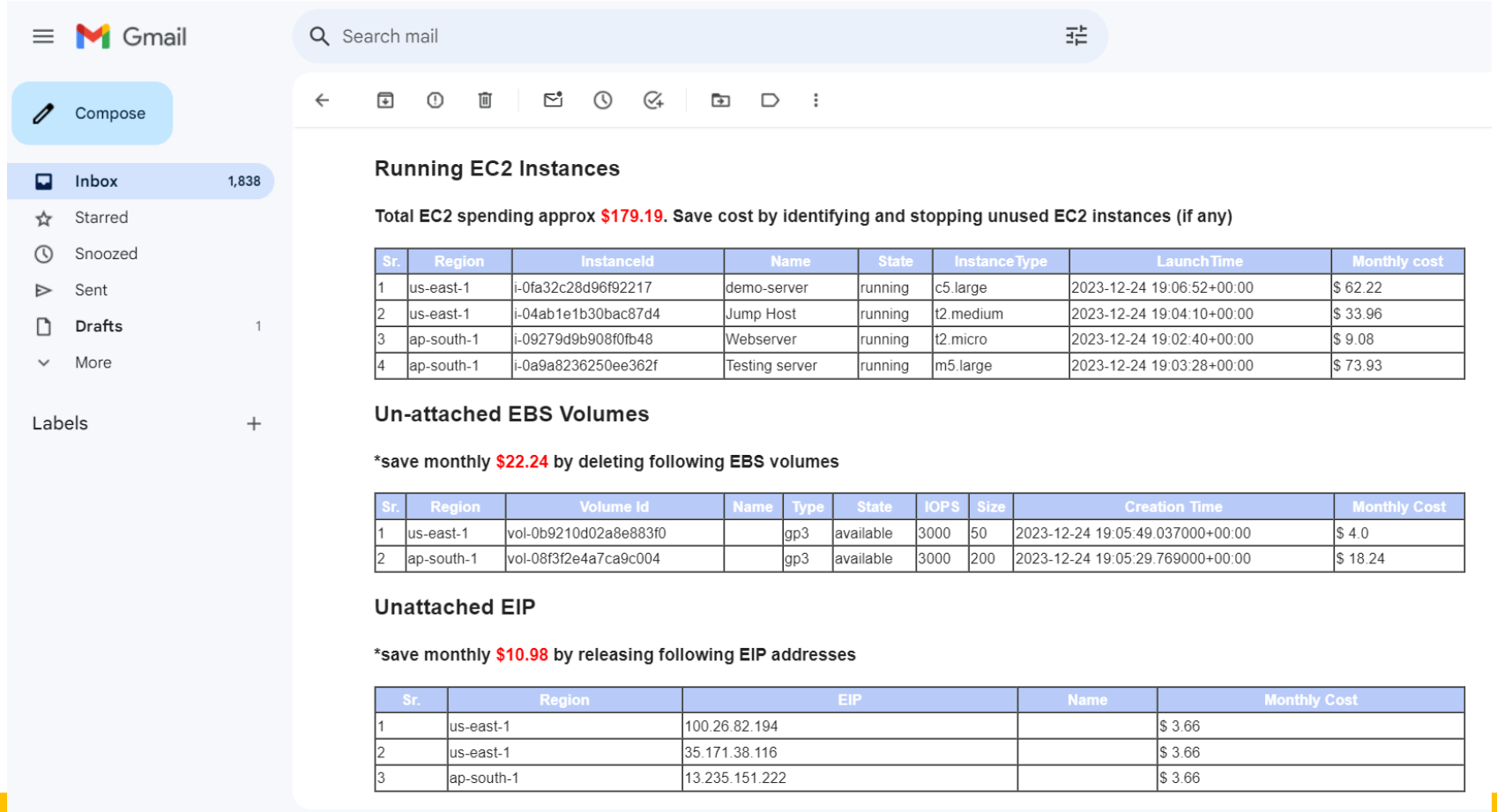
1. Go to AWS Route 53 console
2. Create a Public Hosted Zone (PHZ) with the same domain name that you purchased
3. From PHZ, note down the names of the 4 nameservers (NS records)
4. Go to your domain portal (e.g. godaddy dashboard) and go to DNS management
5. Replace the preconfigured nameservers with Route53 nameservers and save the changes.



(Optional) Simple automation to get notified about running EC2 instances in your account

By Chetan Agrawal

How about getting this email everyday?



Running EC2 Instances

Total EC2 spending approx $179.19. Save cost by identifying and stopping unused EC2 instances (if any)

Sr.	Region	InstanceId	Name	State	InstanceType	LaunchTime	Monthly cost
1	us-east-1	i-0fa32c28d96f92217	demo-server	running	c5.large	2023-12-24 19:06:52+00:00	$ 62.22
2	us-east-1	i-04ab1e1b30bac87d4	Jump Host	running	t2.medium	2023-12-24 19:04:10+00:00	$ 33.96
3	ap-south-1	i-09279d9b908f0fb48	Webserver	running	t2.micro	2023-12-24 19:02:40+00:00	$ 9.08
4	ap-south-1	i-0a9a8236250ee362f	Testing server	running	m5.large	2023-12-24 19:03:28+00:00	$ 73.93

Un-attached EBS Volumes

***save monthly $22.24 by deleting following EBS volumes**

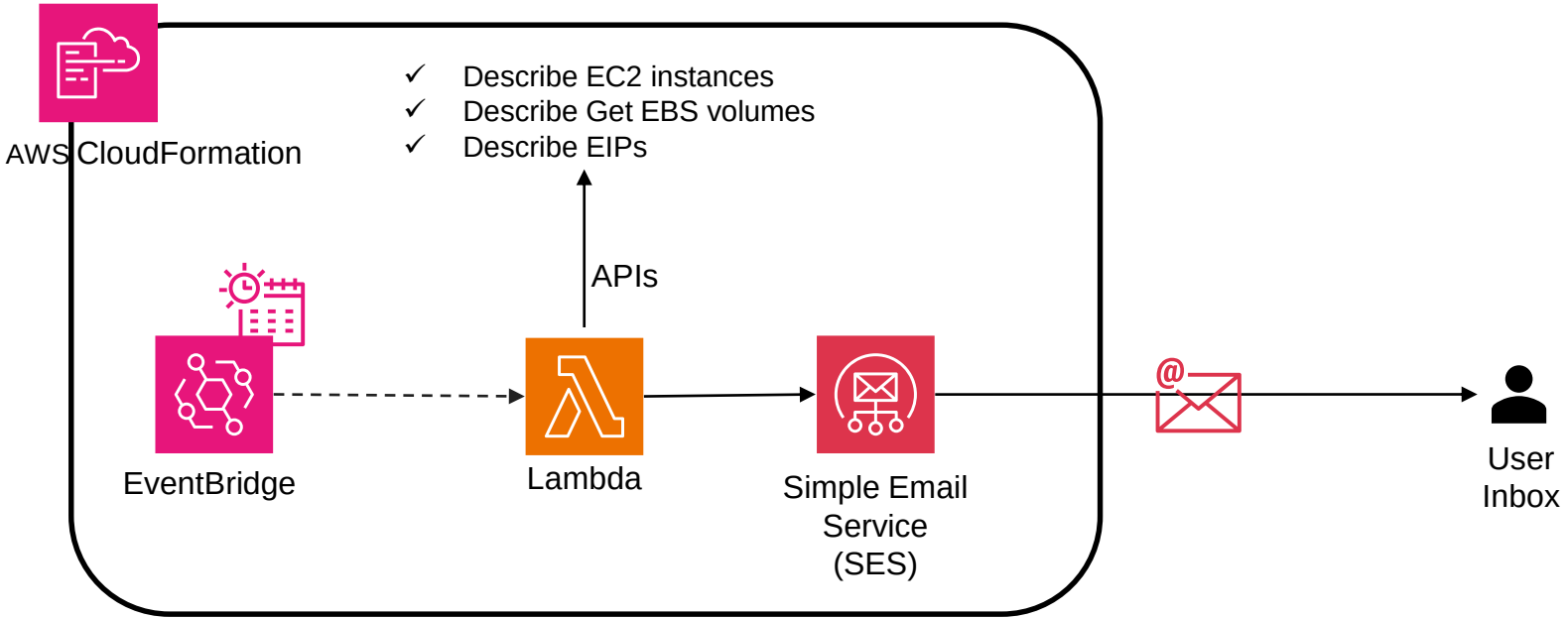
Sr.	Region	Volume Id	Name	Type	State	IOPS	Size	Creation Time	Monthly Cost
1	us-east-1	vol-0b9210d02a8e883f0		gp3	available	3000	50	2023-12-24 19:05:49.037000+00:00	$ 4.0
2	ap-south-1	vol-08f3f2e4a7ca9c004		gp3	available	3000	200	2023-12-24 19:05:29.769000+00:00	$ 18.24

Unattached EIP

***save monthly $10.98 by releasing following EIP addresses**

Sr.	Region	EIP	Name	Monthly Cost
1	us-east-1	100.26.82.194		$ 3.66
2	us-east-1	35.171.38.116		$ 3.66
3	ap-south-1	13.235.151.222		$ 3.66

Build a simple automation



No cost for deploying and running this automation in your AWS account ☺

How?

- Just deploy a CloudFormation stack in your AWS account using the link given in the description
- (Optionally) You can download the CloudFormation template and launch your stack by uploading the template in AWS CloudFormation.
- Make sure you use N. Virginia (us-east-1) region for this deployment
- Provide values for all the parameters:
- **AwsRegions**: Provide the comma separated list of AWS region codes for the regions which you normally use (e.g. us-east-1,ap-south-1)
- **FromEmail**: From email address.
- **ToEmail**: To which email should be sent. You can provide comma separated list of email ids.
- **Schedule**: Hour of the day when this report should be sent in your local timezone.
- **SesRegion**: This region is used to send the email using Amazon SES service (default to us-east-1).

Quick launch link:

https://us-east-1.console.aws.amazon.com/cloudformation/home?region=us-east-1#/stacks/quickcreate?templateURL=https://s3.amazonaws.com/awswithchetan.com/assets/cost-optimization/daily-usage-report-cloudformation.template&stackName=DailyUsageReport&Schedule=19

Tips

- When creating CloudFormation stack you have to provide values for AWS regions. You can add as many regions as you want with comma separated list of regions e.g. us-east-1,ap-south-1,eu-west-1
- If you want to change any parameters after you create the stack, then you can simply change the Lambda Environment variable from the Lambda function configurations.
- If you want to change the schedule of triggering Lambda function, then you can modify it in the AWS EventBridge scheduler. Optionally, you can anytime delete the CloudFormation stack and recreate it by providing the appropriate parameter values during the launch.
- You can modify Lamba function to add more functionality. For example, in development environment you may want to also automatically stop running EC2 instances. For this you need to just add one more API call in the code. But while doing so, make sure you also implement some mechanism e.g. ec2 tags to skip stopping some of the ec2 instances because may be those EC2 instances are in-use by someone and automation shouldn't stop those.
- You can also copy lambda function code to .py file and run it locally. Instructions for how to run it locally are provided in the initial section of the code.

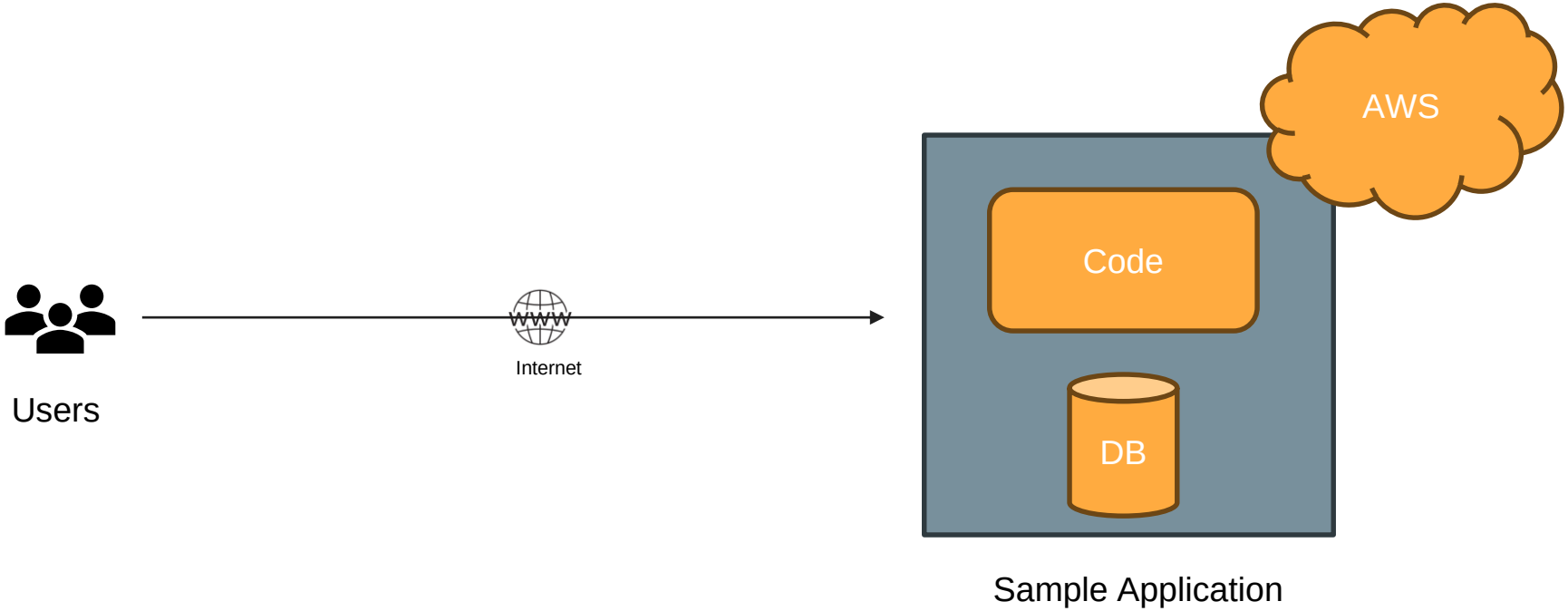
Remember

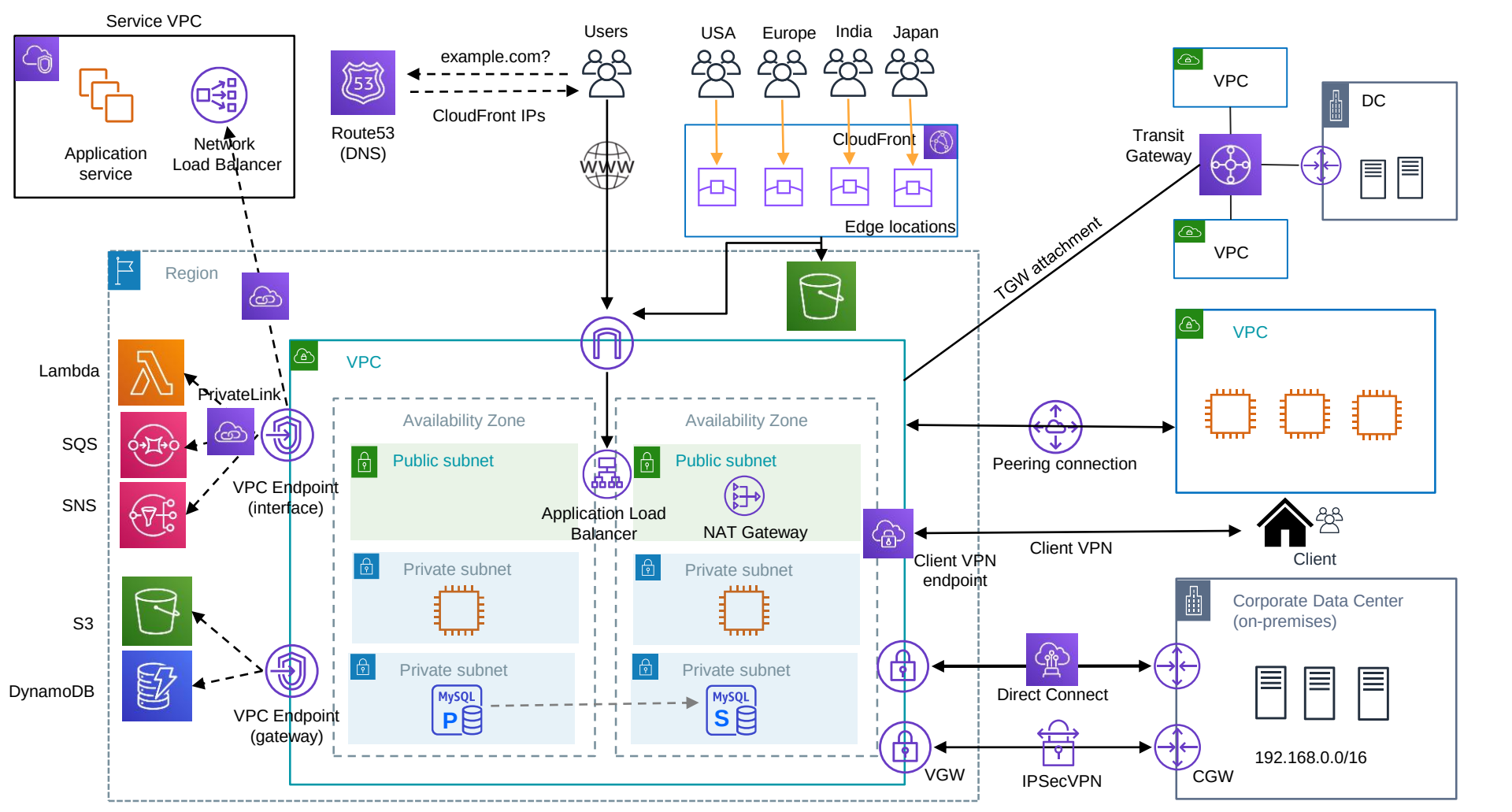
Before logging out of your AWS console, always check if you have any EC2 instances running or any other resources provisioned which may attract cost

1. EC2 console -> EC2 Global view -> Make sure there are no EC2 instances running in any region (unless you are running intentionally for some other workload)
2. VPC console -> VPC Dashboard -> Make sure there are no
    a) NAT gateways
    b) Elastic IPs
    c) Endpoints
    d) Site-to-Site VPN connections
    e) Transit Gateway
3. Repeat this for the regions which you are using for your exercises.

Overview of
VPC and AWS Networking

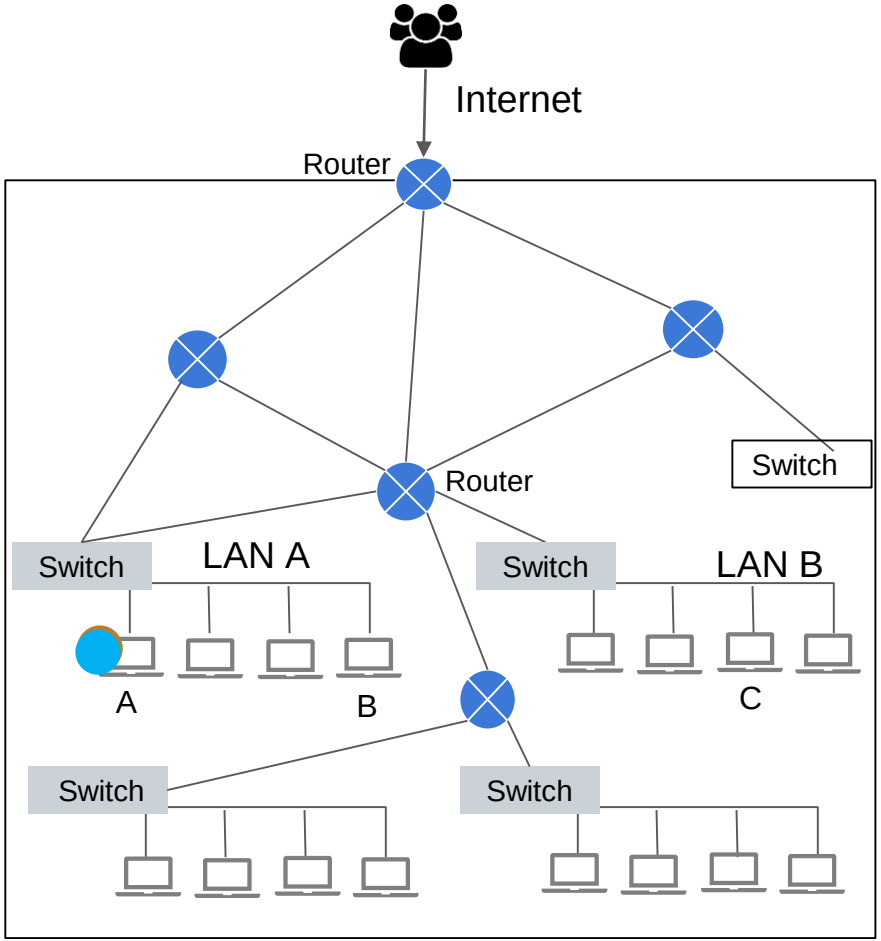
The big picture..

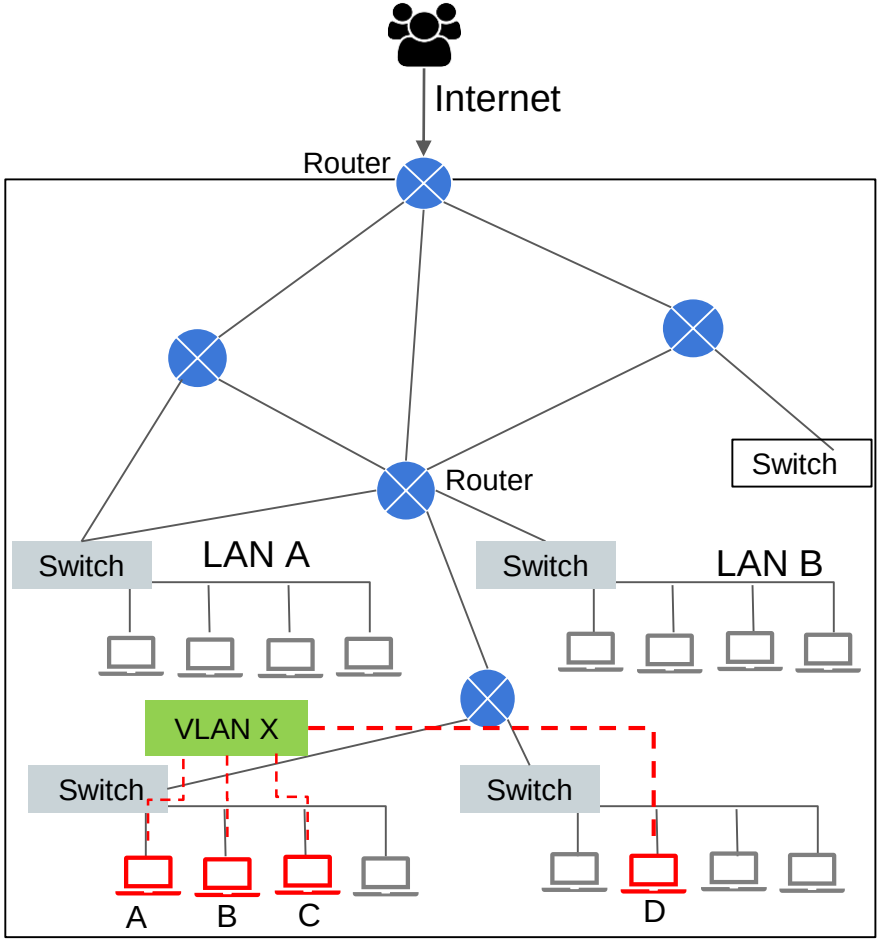




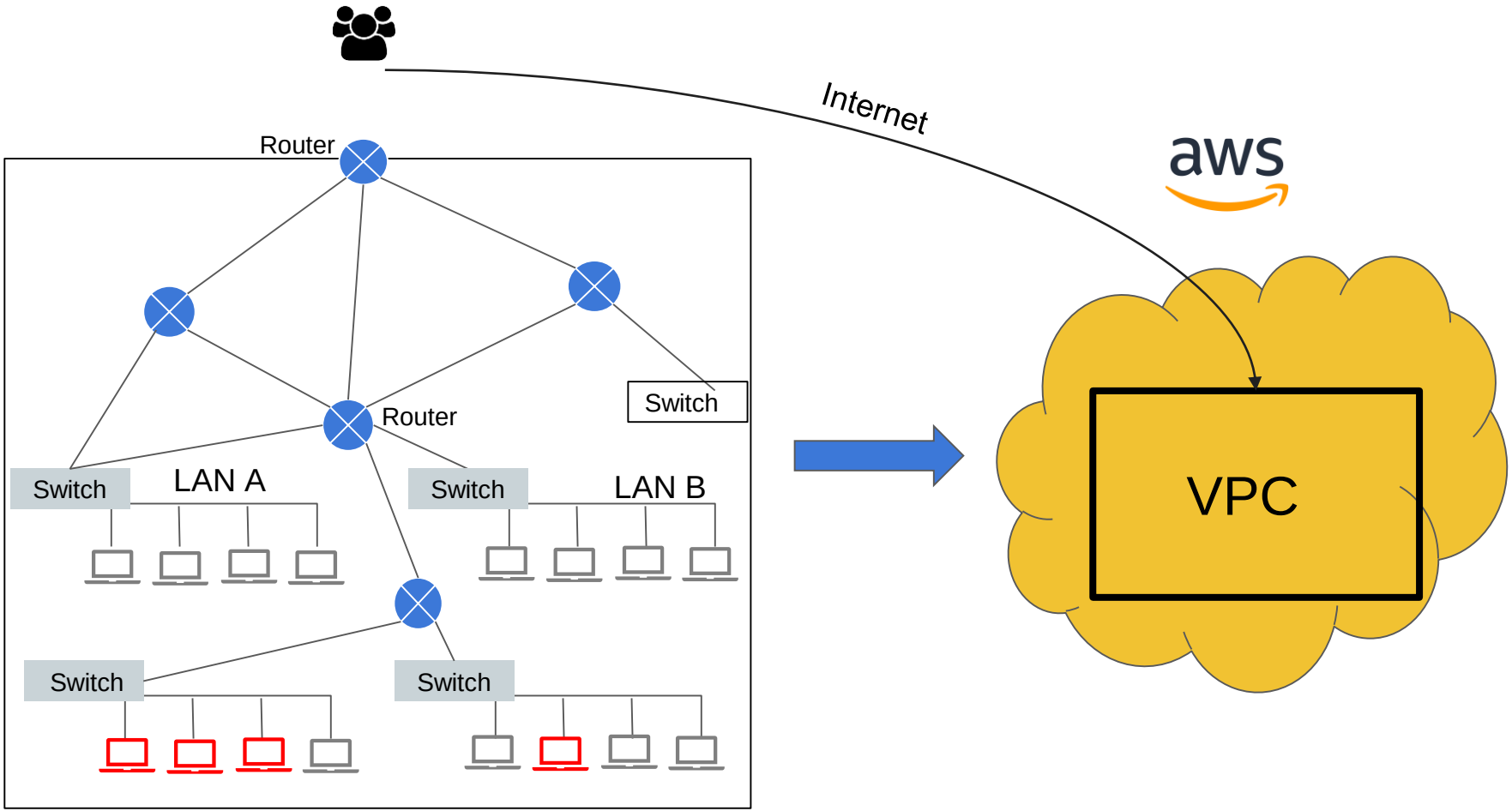
Amazon VPC

Moving from traditional on-premises network to virtual network in the Cloud

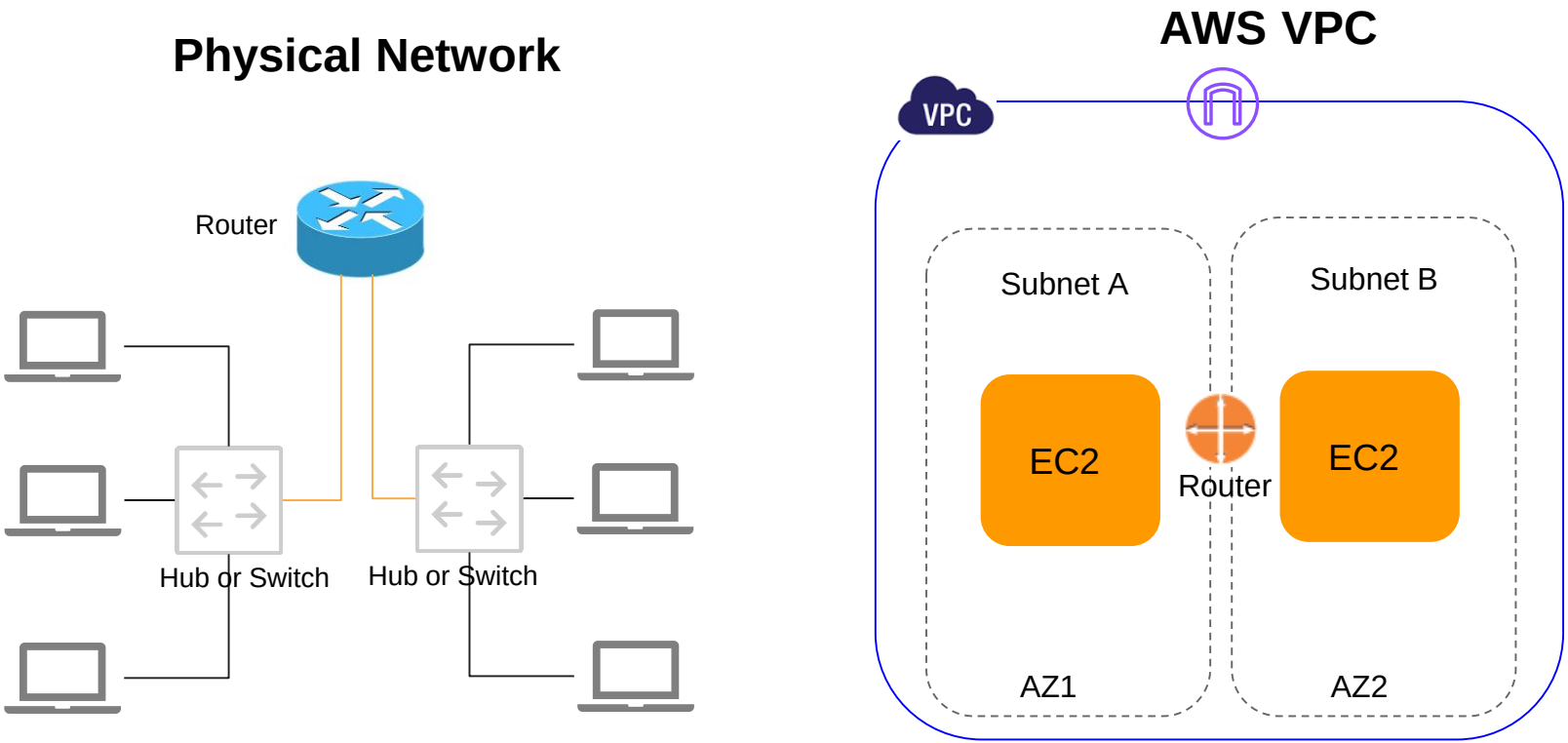




https://www.youtube.com/@PracticalNetworking

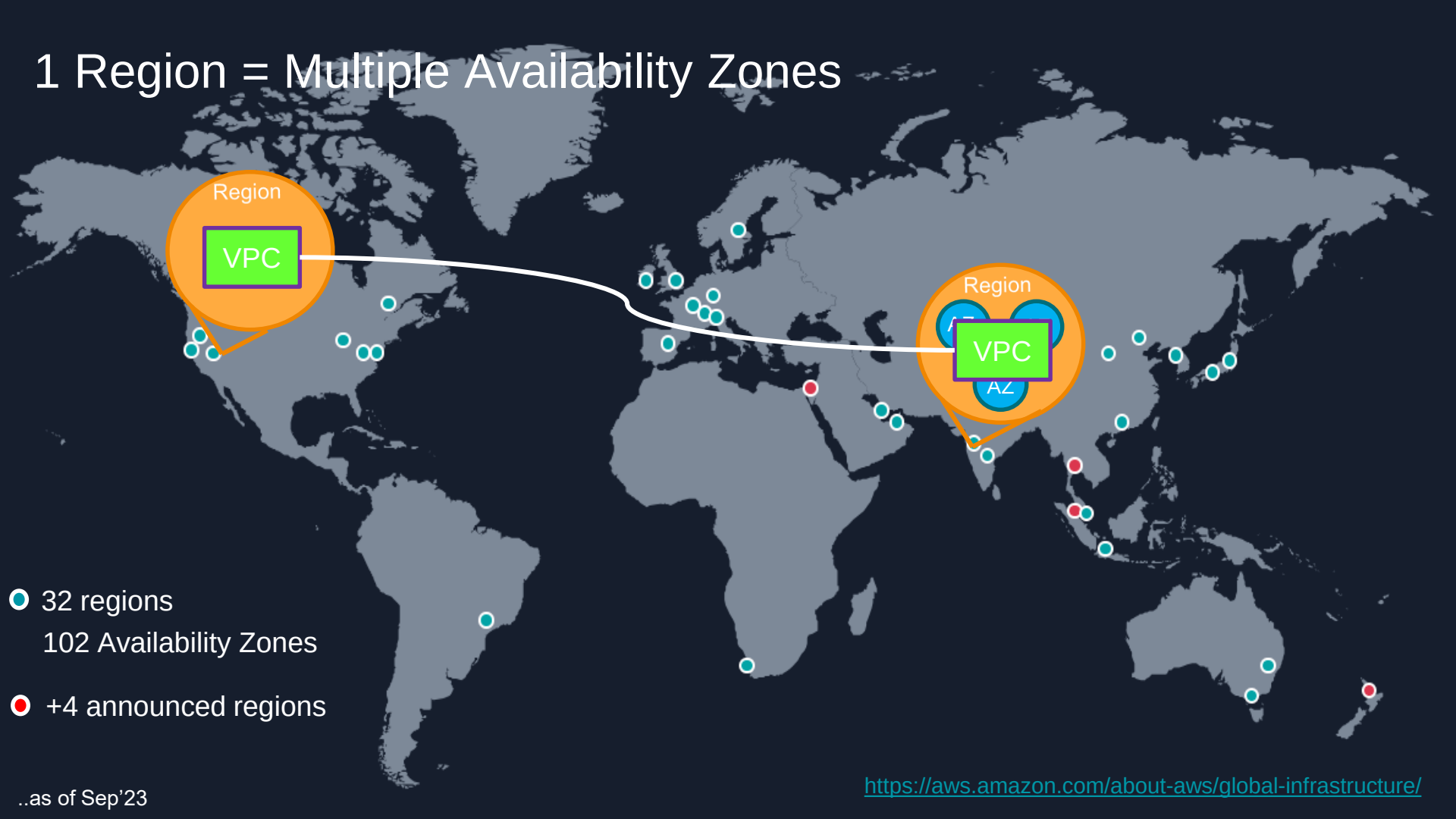


Traditional IT network vs VPC



AWS Account -> Region & AZ -> VPC

1 Region = Multiple Availability Zones

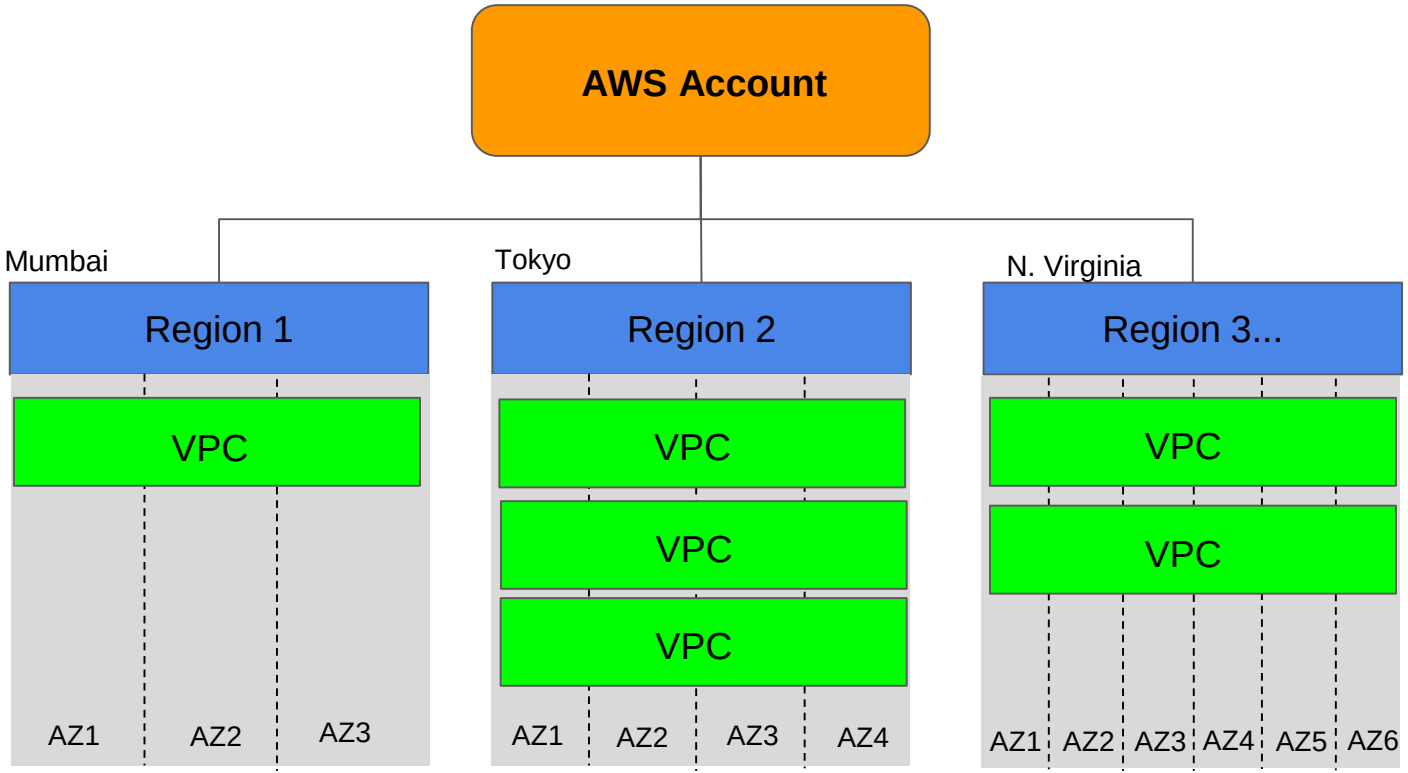


- 32 regions
  102 Availability Zones
- +4 announced regions

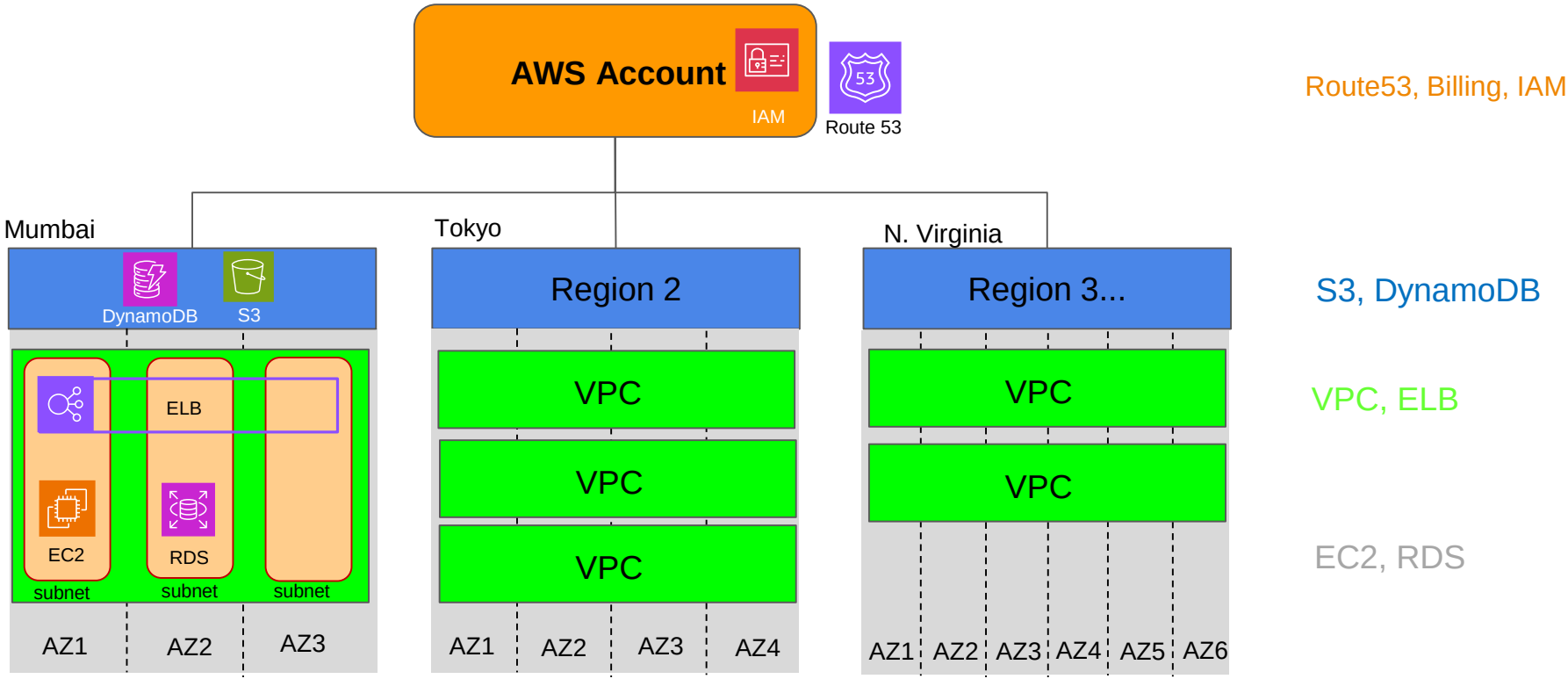
..as of Sep'23

https://aws.amazon.com/about-aws/global-infrastructure/

VPC Scope with respect to AWS account, region and AZs



AWS services scope with respect to VPC



Route53, Billing, IAM

S3, DynamoDB

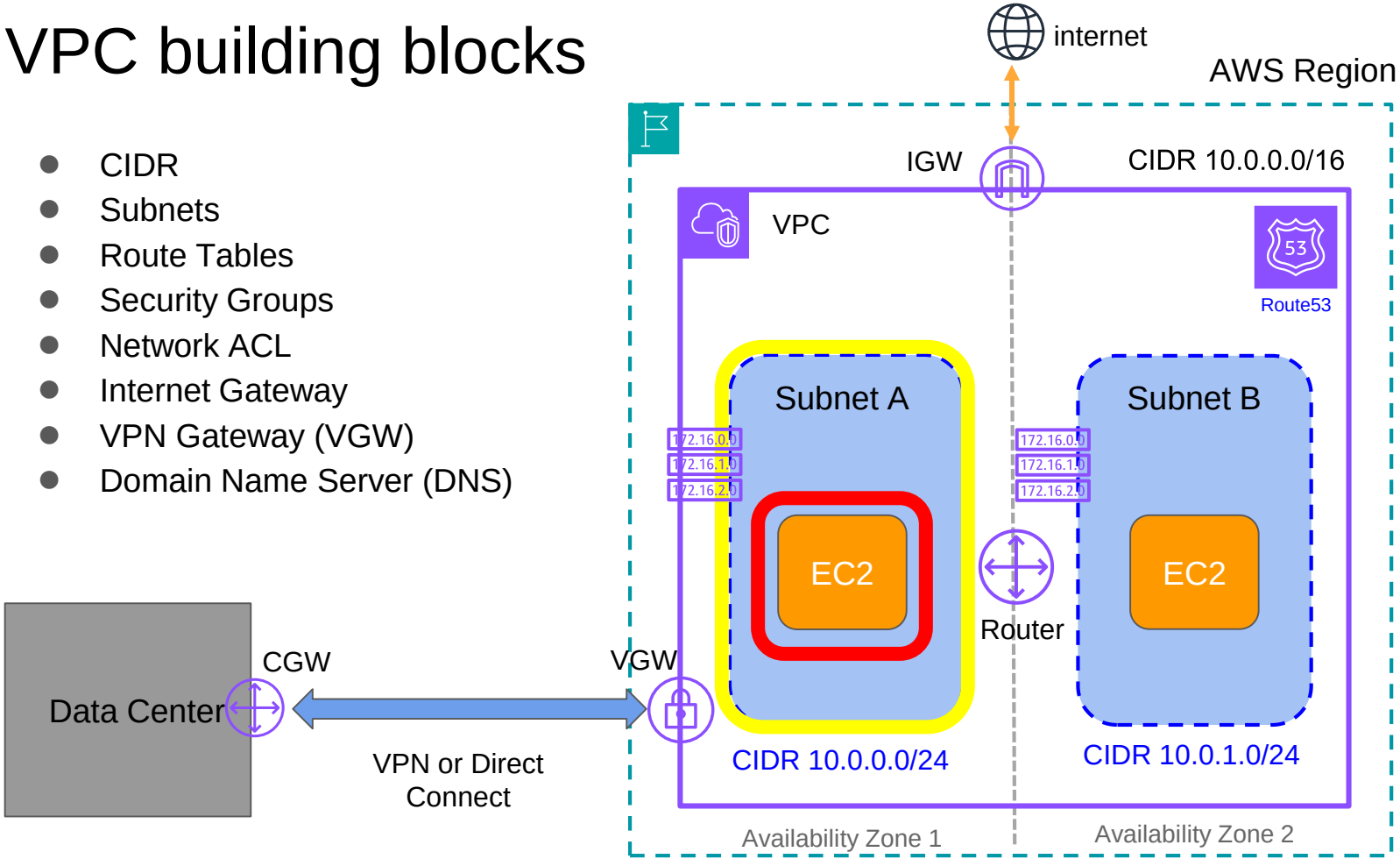
VPC, ELB

EC2, RDS

VPC building blocks

VPC building blocks

- CIDR
- Subnets
- Route Tables
- Security Groups
- Network ACL
- Internet Gateway
- VPN Gateway (VGW)
- Domain Name Server (DNS)



VPC CIDR

IPv4 and IPv6 addresses

12.34.56.67

fe80:cd00:0000:0cde:1257:0000:211e:729c

- IP address is the identity of the each host in the network
- There are 2 types of IP addresses
  - IPv4 (32 bit)
  - IPv6 (128 bit)
- IPv4 address
  - Represented as four octets (4 x 8 bits)
  - Each octet represented in decimal value 0-255. Example: 192.168.56.212

192.168.56.212

192 = 128 + 64 = 2^7 + 2^6

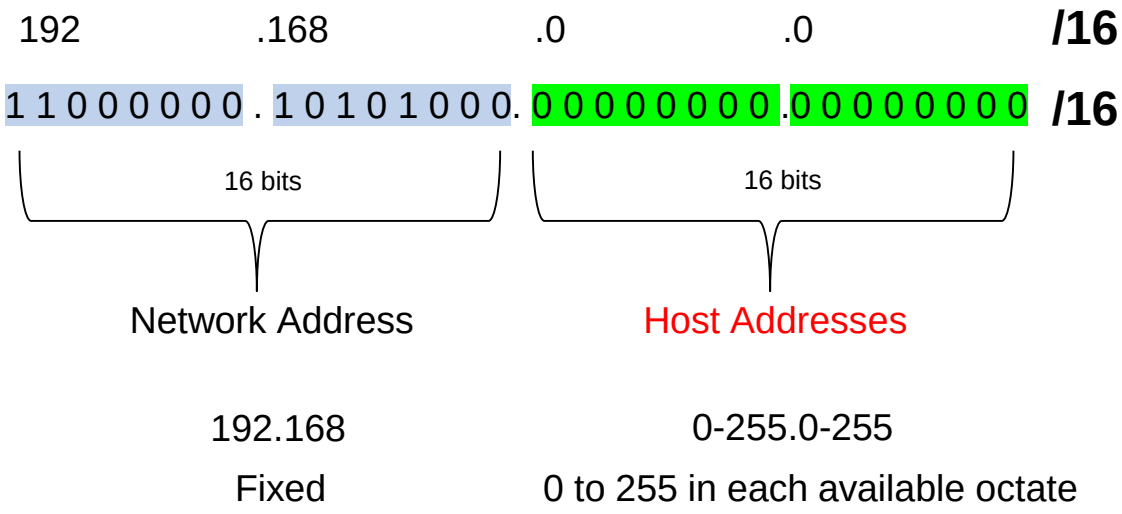
Bits	7	6	5	4	3	2	1	0

1 1 0 0 0 0 0 0 . 1 0 1 0 1 0 0 0 . 0 0 1 1 1 0 0 0 . 1 1 0 1 0 1 0 0

CIDR – Classless Inter Domain Routing

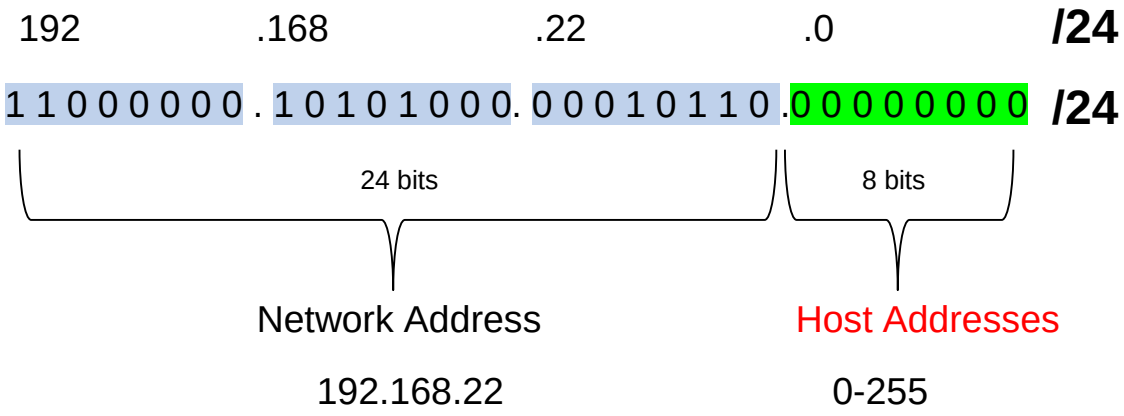
- IP addressing scheme that replaces old address style of Class A, B, C
- Represented as a IP address and prefix

  Example: IPv4 CIDR 192.168.0.0/16



CIDR – Classless Inter Domain Routing

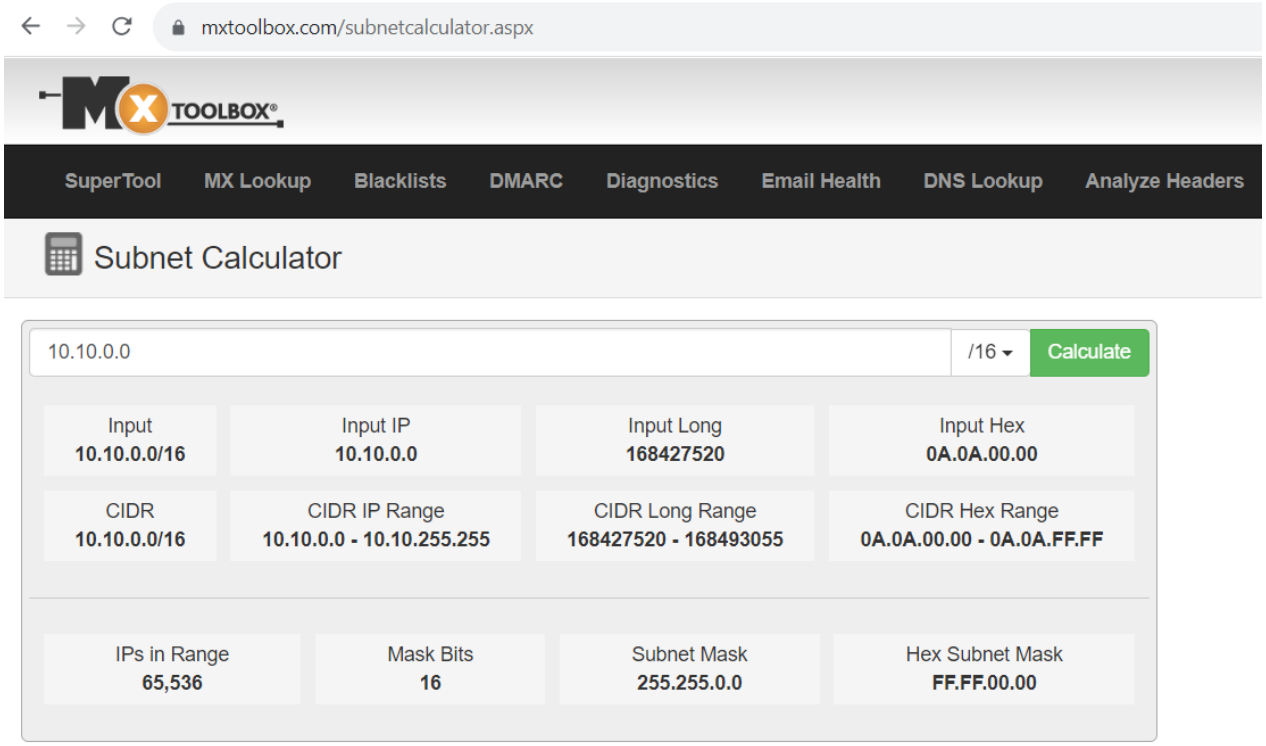
Example: IPv4 CIDR 192.168.22.0/24



VPC Addressing

- **AWS VPC CIDR (IPv4)**
  - VPC prefix between /16 (65536 IPs) and /28 (16 IPs)
  - RFC 1918 IP ranges for Private network and corresponding AWS recommended ranges
    - 10.0.0.0/8 => 10.0.0.0 – 10.255.255.255 **=>** **AWS CIDR 10.X.0.0/16**
    - 172.16.0.0/12 => 172.16.0.0 - 172.31.255.255 **=>** **AWS CIDR 172.16.0.0/16 to 172.31.0.0/16**
    - 192.168.0.0/16 => 192.168.0.0 - 192.168.255.255 **=>** **AWS CIDR 192.168.0.0/16**
  - Subnet CIDR prefix between /16 to /28 (same as VPC CIDR)
- **AWS VPC CIDR (IPv6)**
  - VPC CIDR with prefix /56 (2^72 IPs)
  - IPv6 CIDR is allocated by AWS
  - Subnet CIDR prefix /64
  - IPv6 IP addresses are globally unique and publicly routable

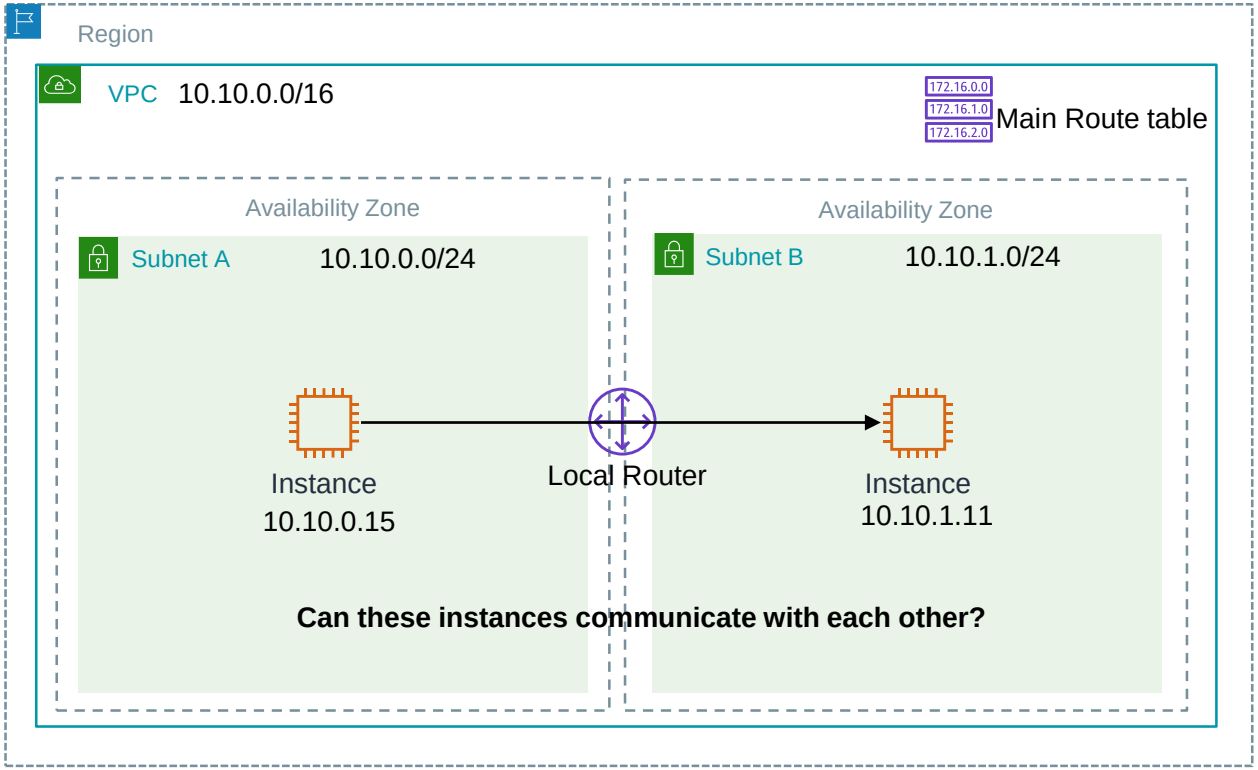
Tip – Use online tools



https://mxtoolbox.com/subnetcalculator.aspx

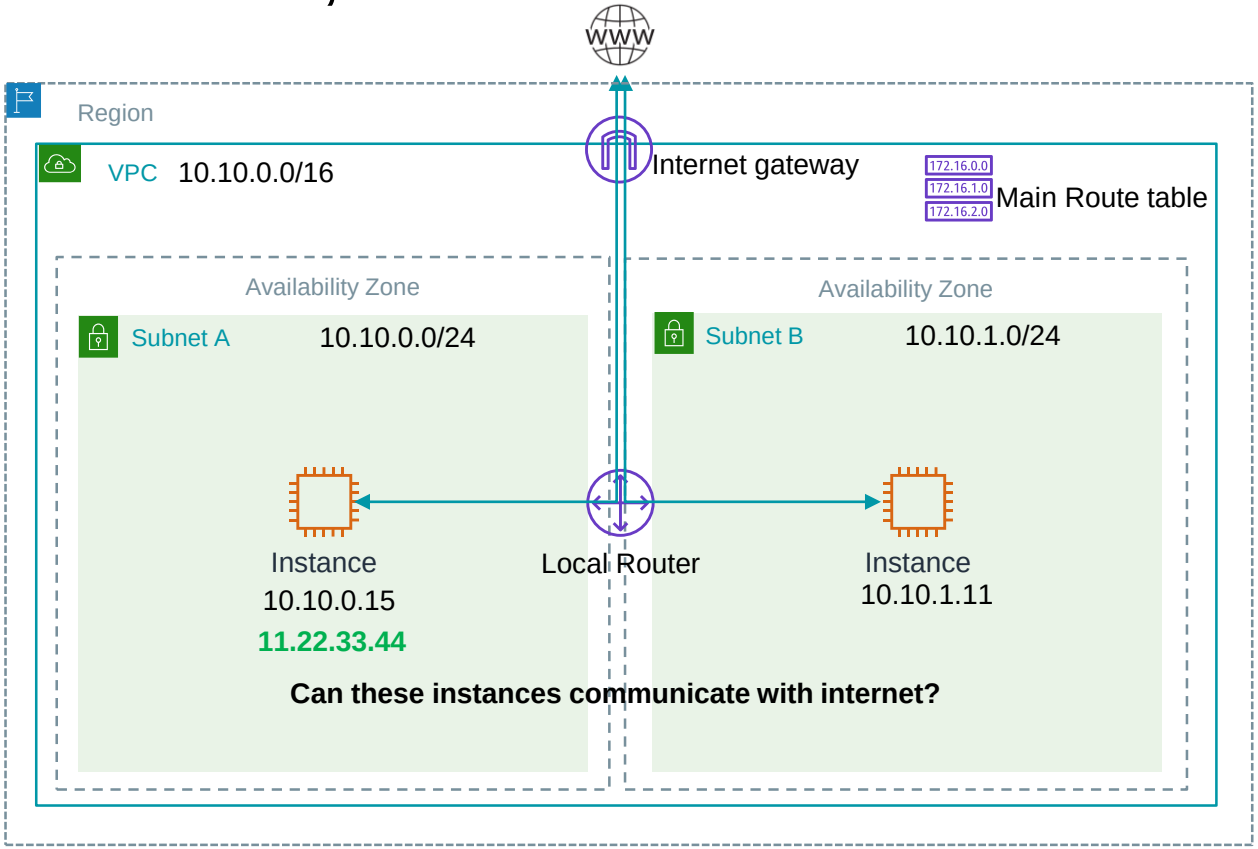
Subnets, Route tables & Internet Gateway

Subnets, Route Tables and Internet Gateway



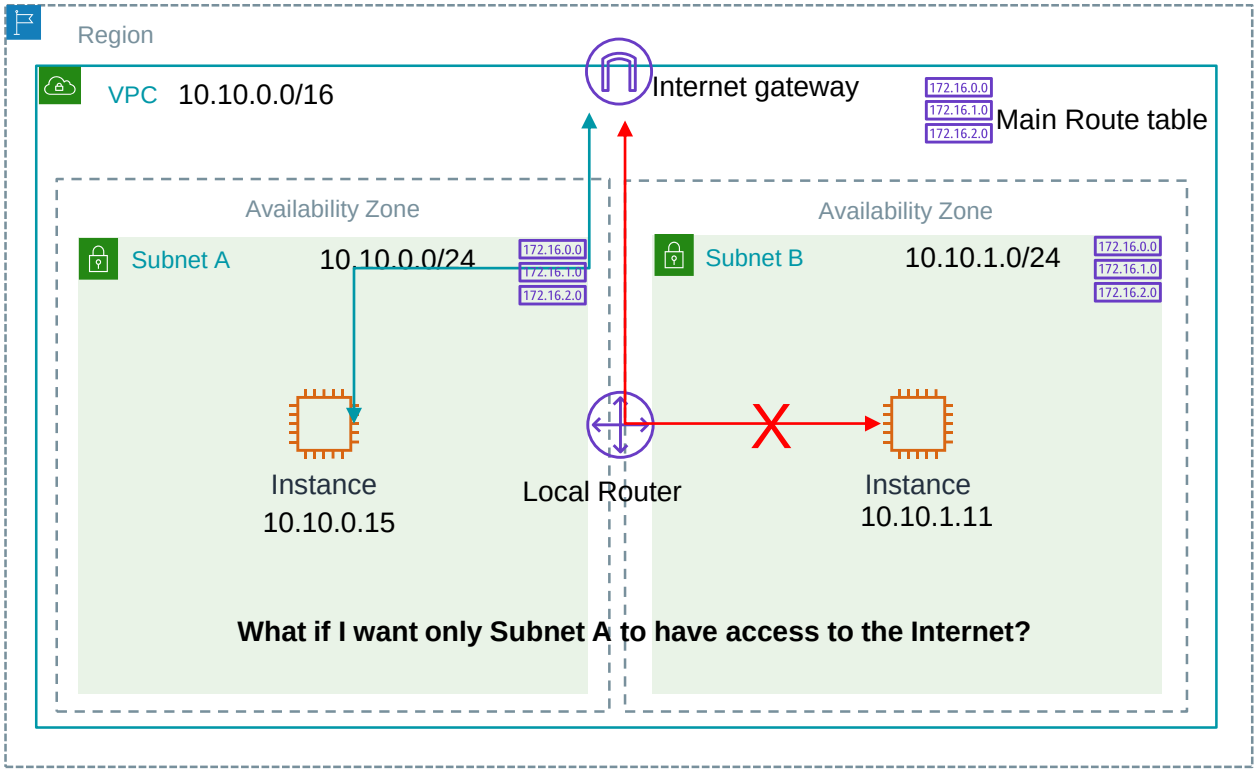
Destination	Target
10.10.0.0/16	Local

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
10.10.0.0/16	Local
0.0.0.0/0	Internet Gateway

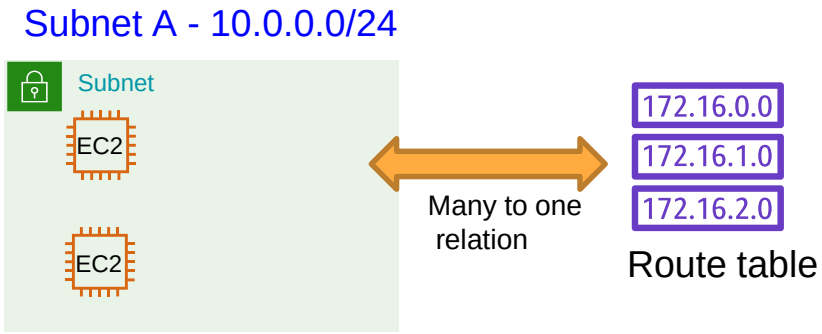
Subnet B Route Table

Destination	Target
10.10.0.0/16	Local

Now subnets are **not** following the Main Route Table

Route Table

- Contains rules to route the traffic in/out of Subnets
- 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 also modify the main route table



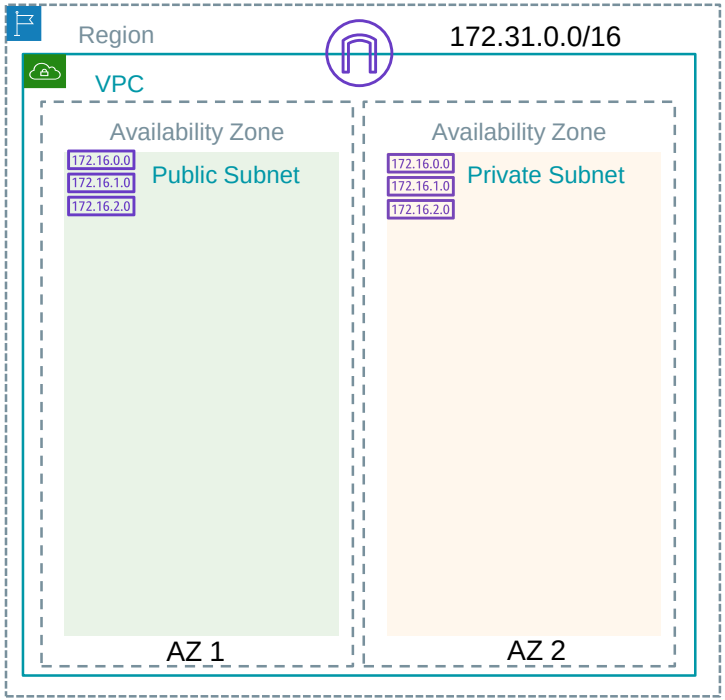
Destination	Target
10.0.0.0/16	local
0.0.0.0/0	igw-xxxxx

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

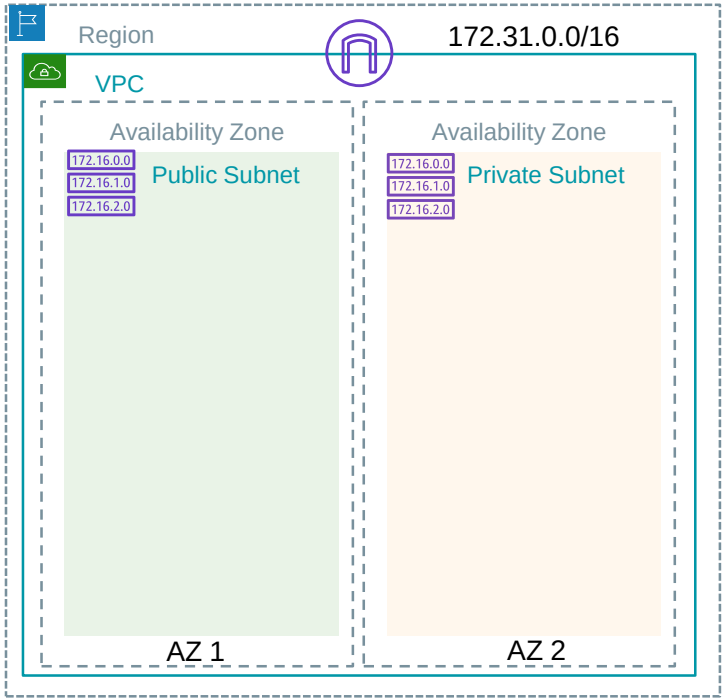


Subnets

- 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

1 Subnet => 1 AZ

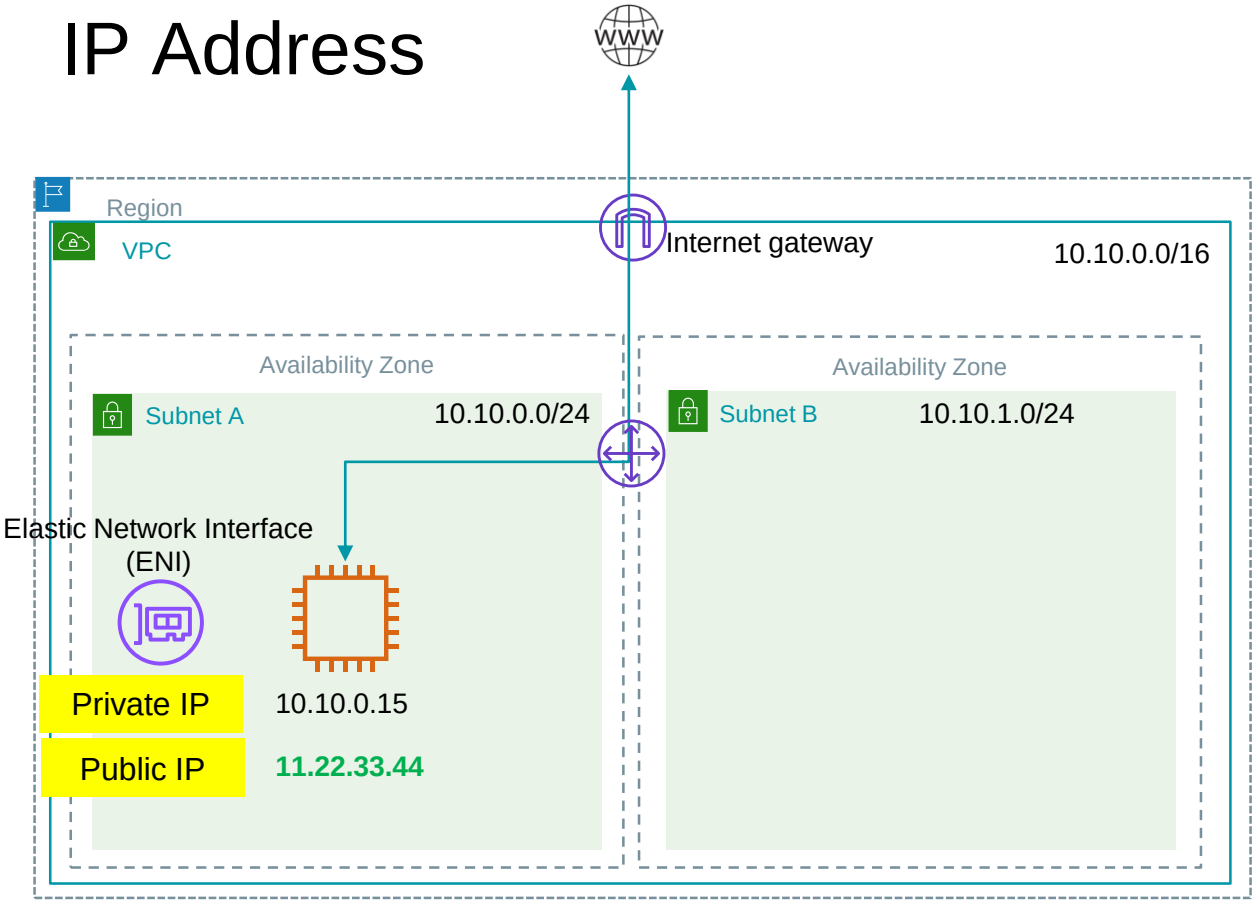


Subnets – Good to know

- 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
- Example: if CIDR block 10.0.0.0/24, reserved IPs 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 this address is reserved

IP Addresses in VPC

IP Address



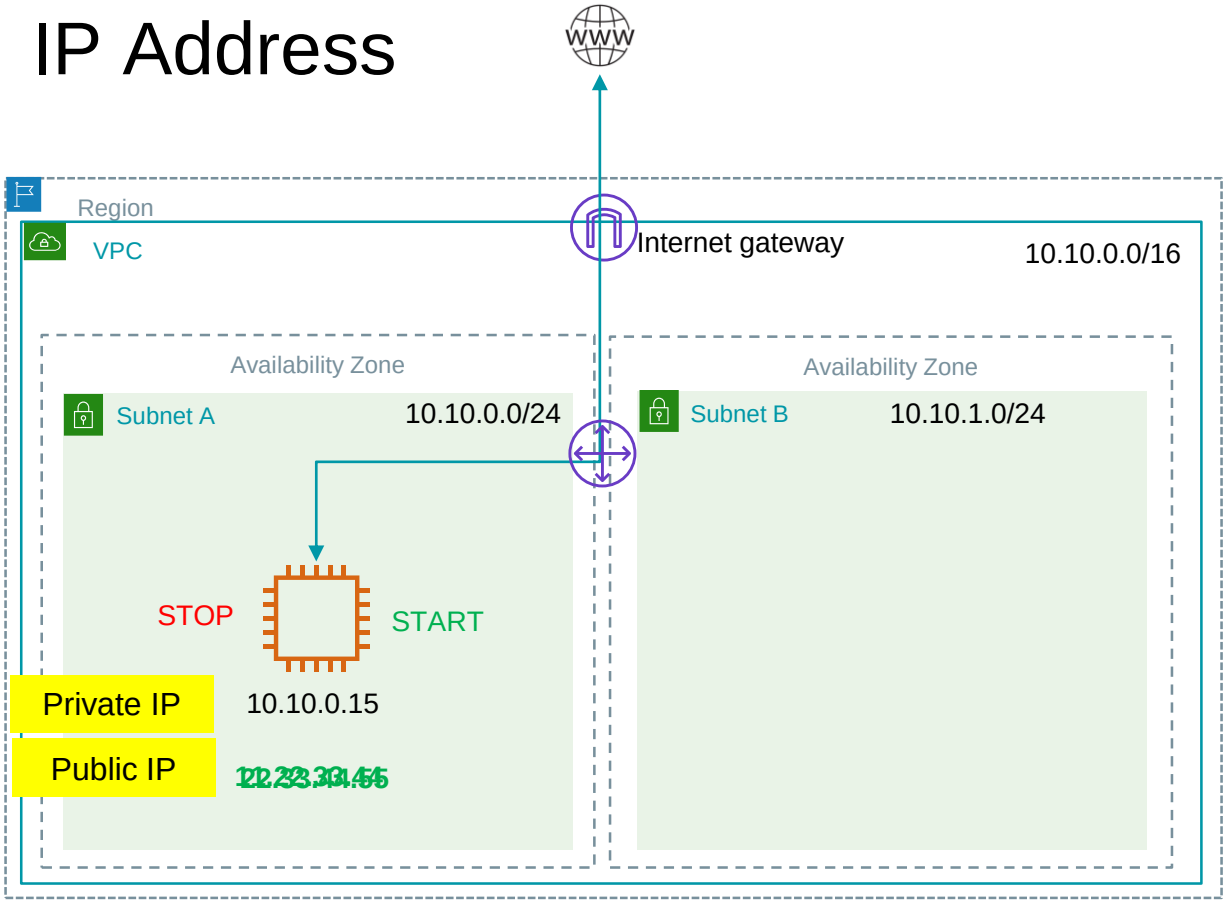
Private IP

- Private IP is assigned from the subnet range

Public IP

- Public IP is assigned from the Amazon's pool of Public IPs

IP Address



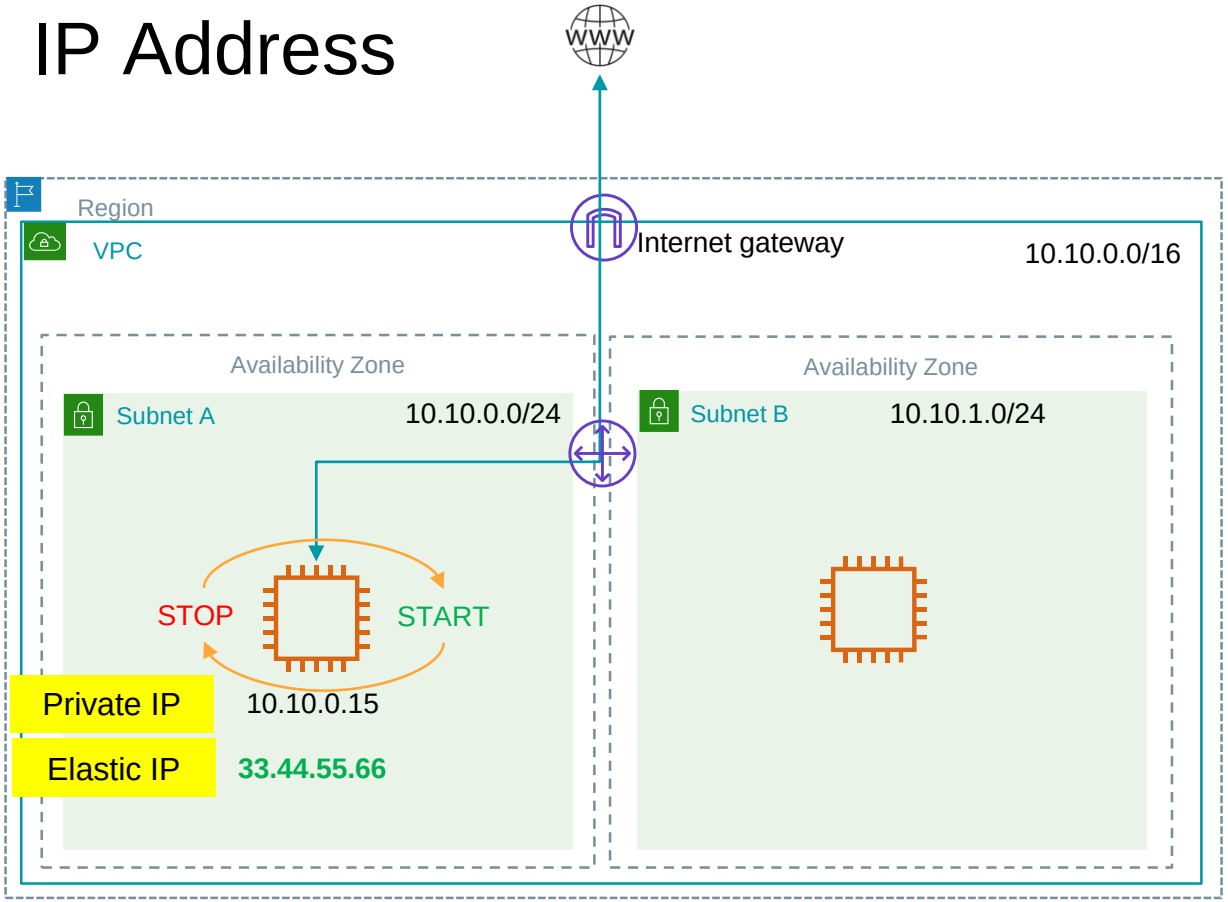
Private IP

- Private IP is assigned from the subnet range

Public IP

- Public IP is assigned from the Amazon's pool of Public IPs
- Public IP will change when you stop and start the instance

IP Address



Elastic IP

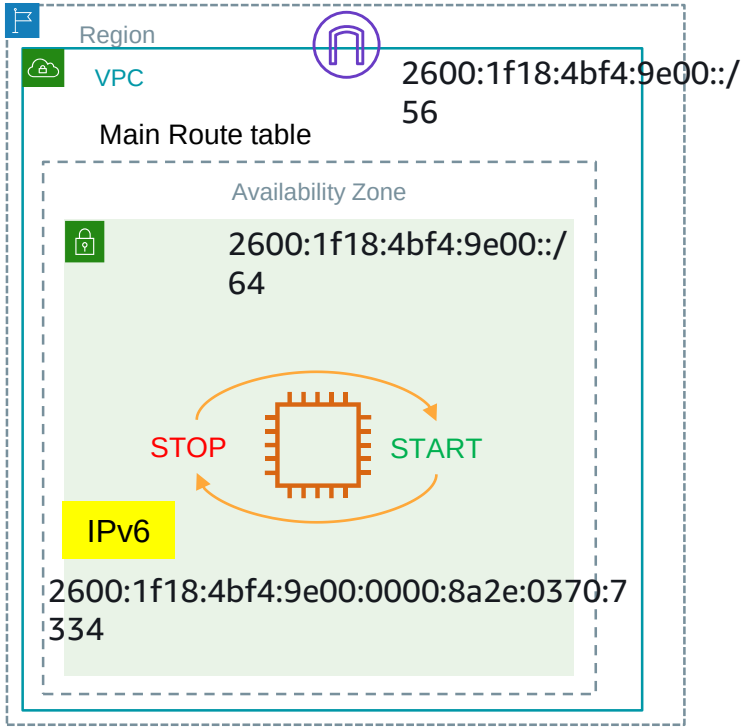
- Elastic IP is allocated to your AWS account
- Remains allocated until released
- Does not change on instance stop/start
- Can be remapped to another instance

Private, Public and Elastic IP (EIP)

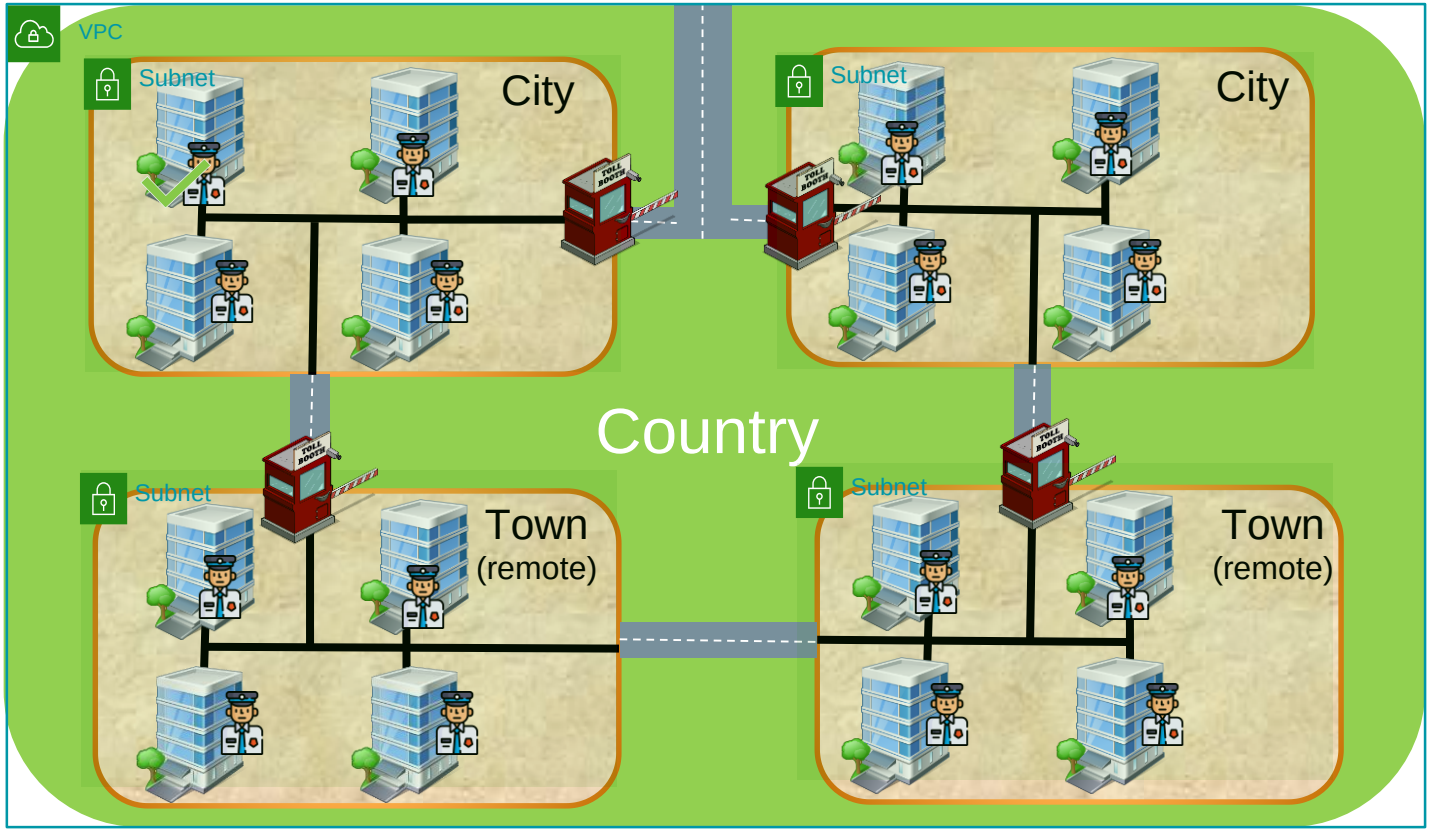
Feature	Private	Public	Elastic
Communication	Communication within VPC	Can communicate over internet	Can communicate over internet
Address range	Gets IP address from subnet range. Ex: 10.200.0.1	Gets IP address from Amazon Pool within region	Gets IP address from Amazon Pool within region
Instance stop/start behaviour	Once assigned cannot be changed	Changes over instance stop and start (not on instance OS reboot)	Do not change over instance stop and start.
Releasing IP	Released when instance is terminated	Released to pool when instance is stopped or terminated.	Not released. Remains in your account. (Billed)
Automatic Assignment	Receives private IP on launch of EC2 instance	Receives public IP on launch of EC2 instance if “Public IP addressing attribute” is set to true for subnet.	Have to explicitly allocate and attach EIP to EC2 instance. Can be reattached to other EC2.
Examples	Application servers, databases	Web servers, Load Balancers,	Web servers, Load Balancers, Websites

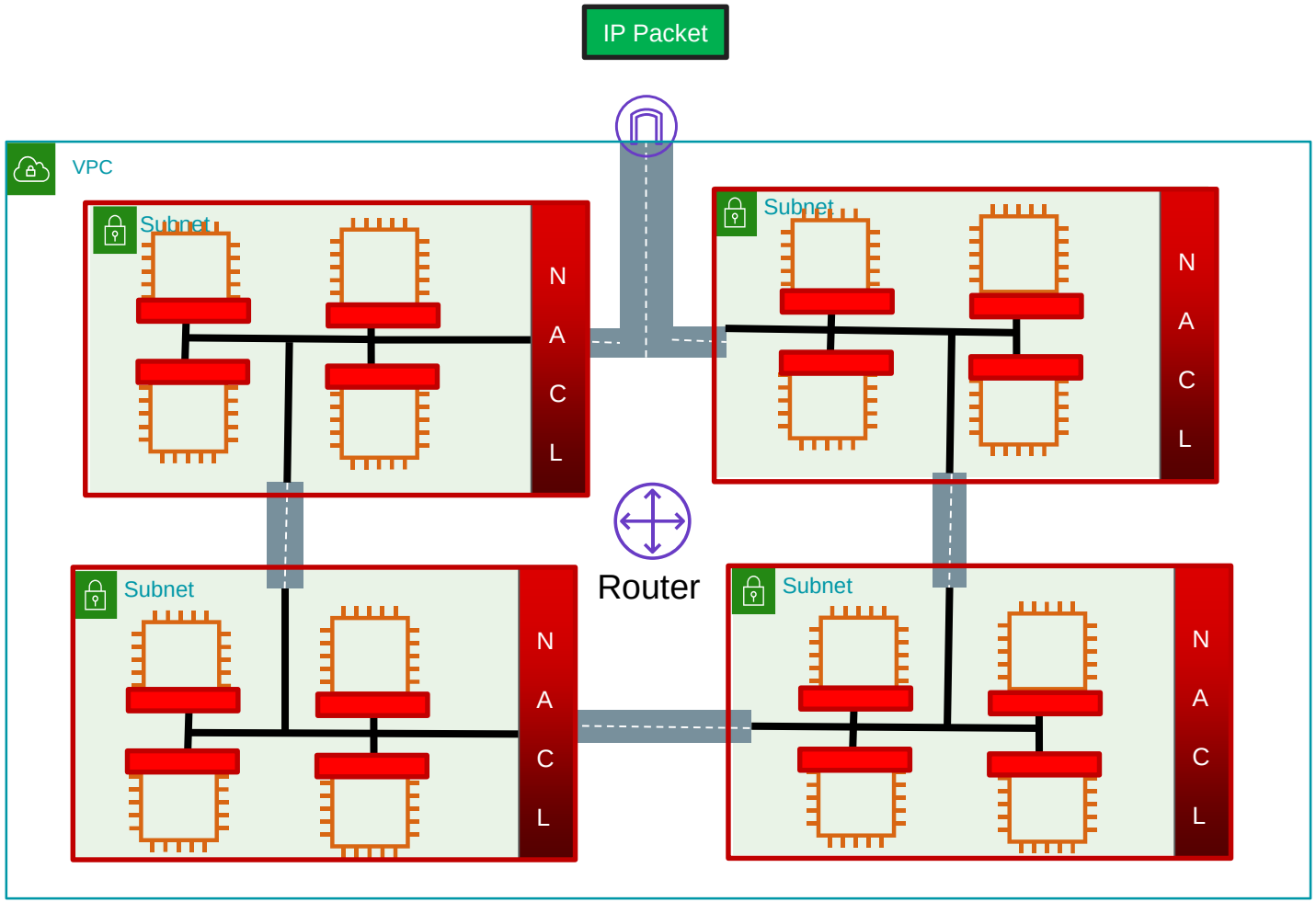
IPv6 Addresses

- 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
- VPC CIDR with prefix /56 (2^72 IPs) and Subnet CIDR prefix /64
- 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



Routing vs Firewall

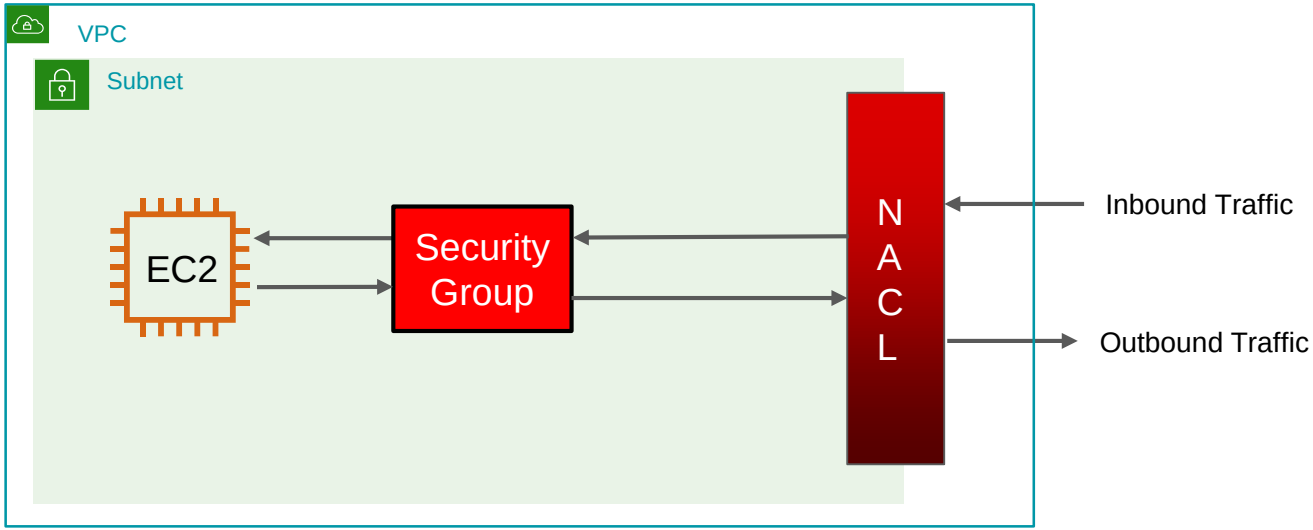




VPC Firewall - Security group

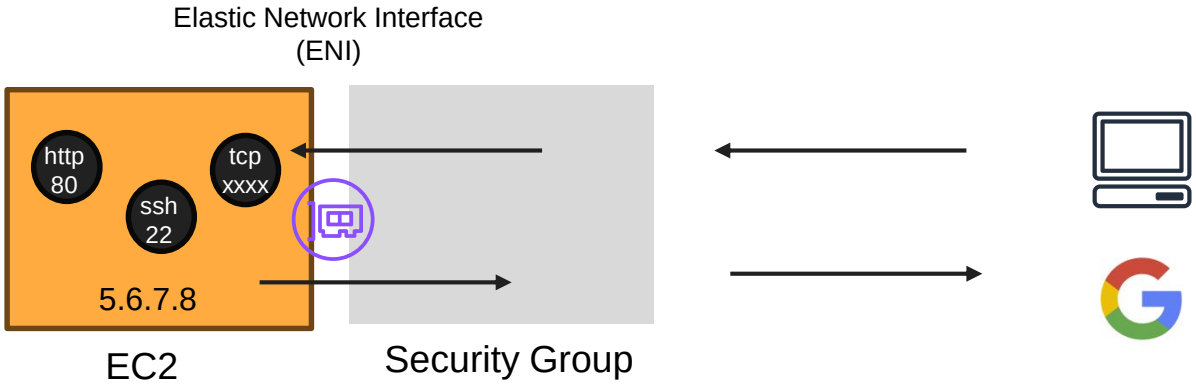
Firewalls inside VPC

- Security Groups
- Network Access Control List (NACL)

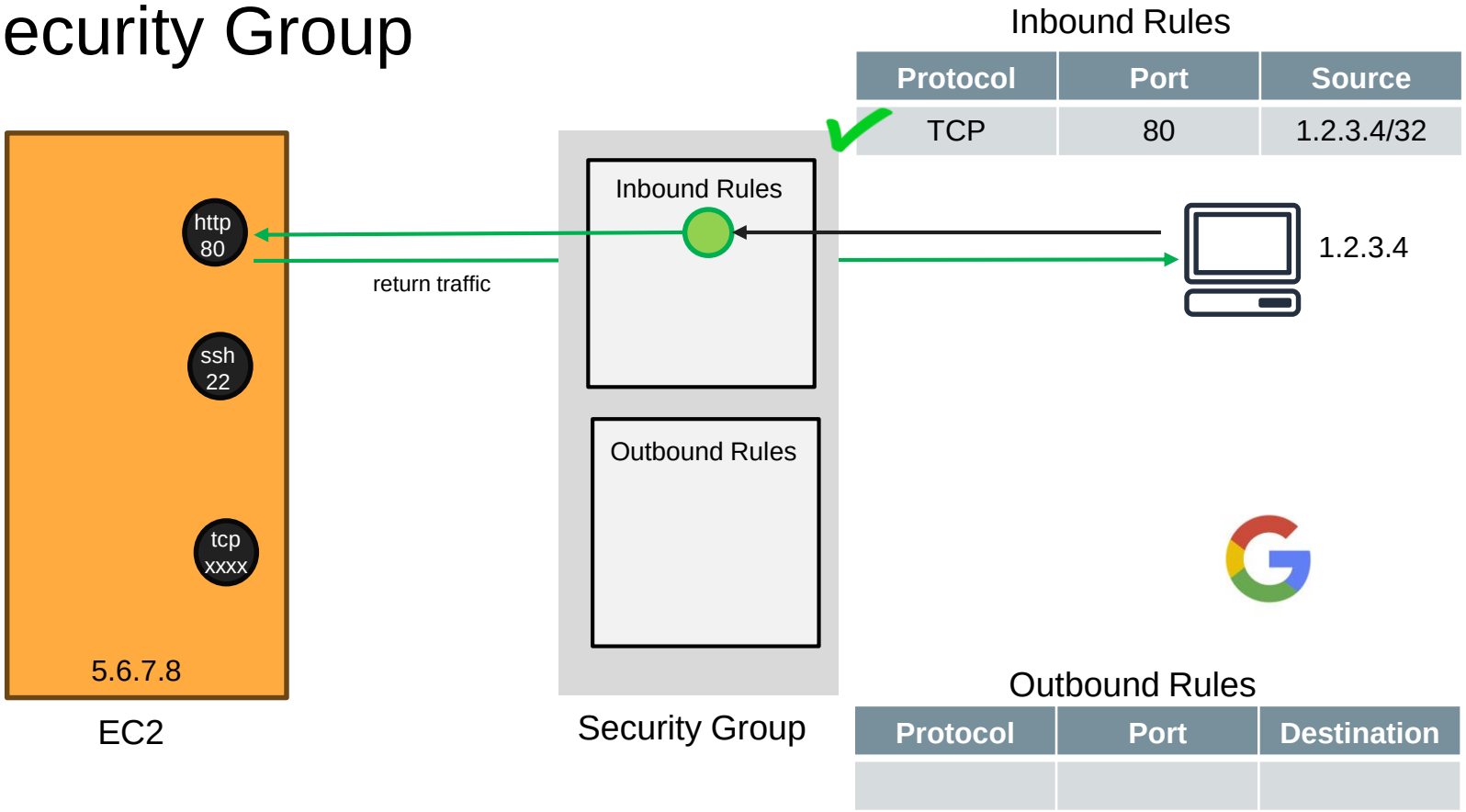


Security Group

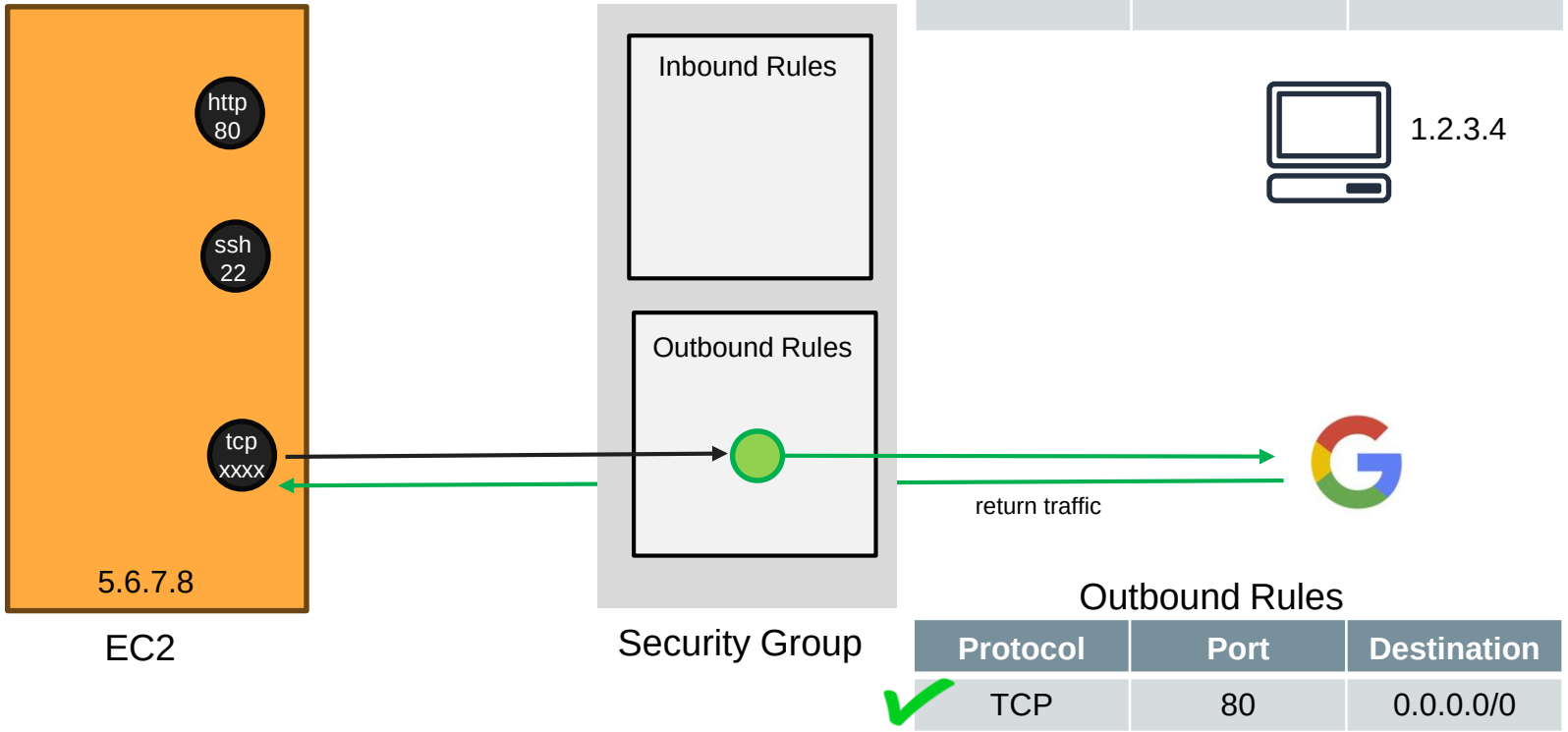
- Security Groups are most basic, native and important firewall for EC2 instances (ENIs)
- Security group has Inbound and Outbound rules
- Security group has only ALLOW rules. Does not support DENY/Block rules.
- Default Security group in each VPC
- Authorises traffic for both IPv4 and IPv6 traffic
- Security groups are stateful
- You can reference another Security group as a source



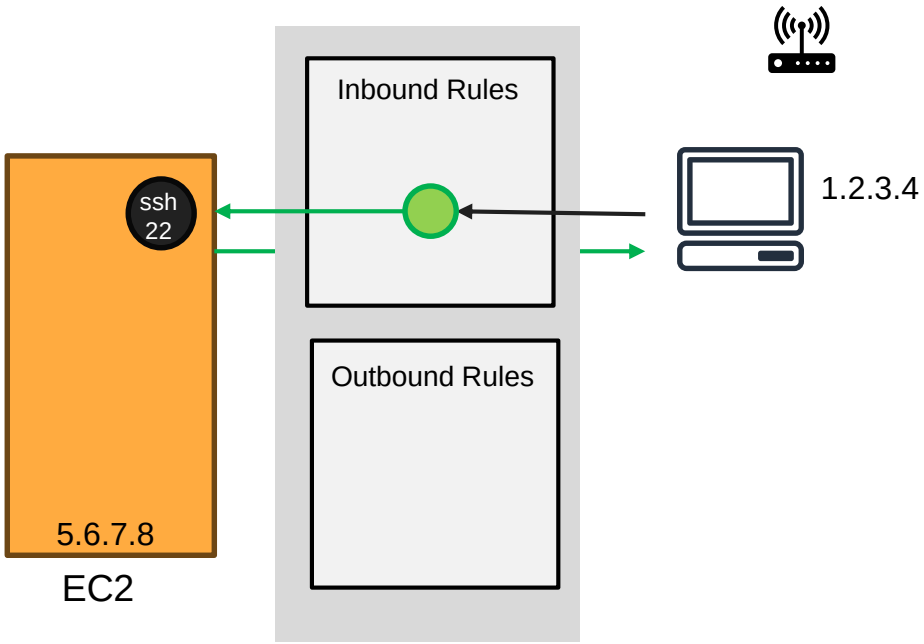
Security Group



Security Group



Security Group scenarios



Allow SSH from only your home IP

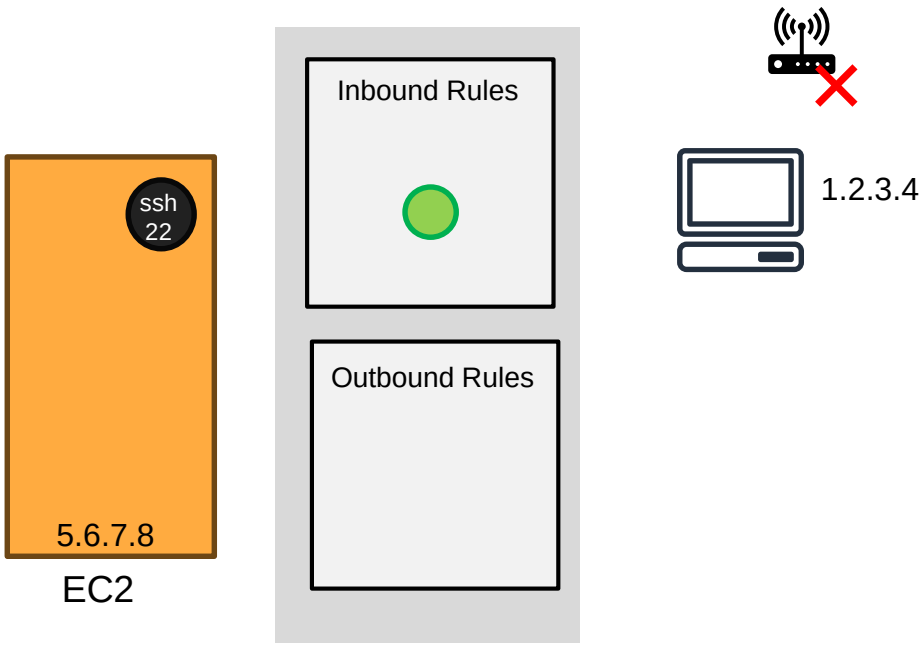
Inbound Rules

Protocol	Port	Source
TCP	22	1.2.3.4/32

Outbound Rules

Protocol	Port	Destination

Security Group scenarios



Allow SSH from only your home IP

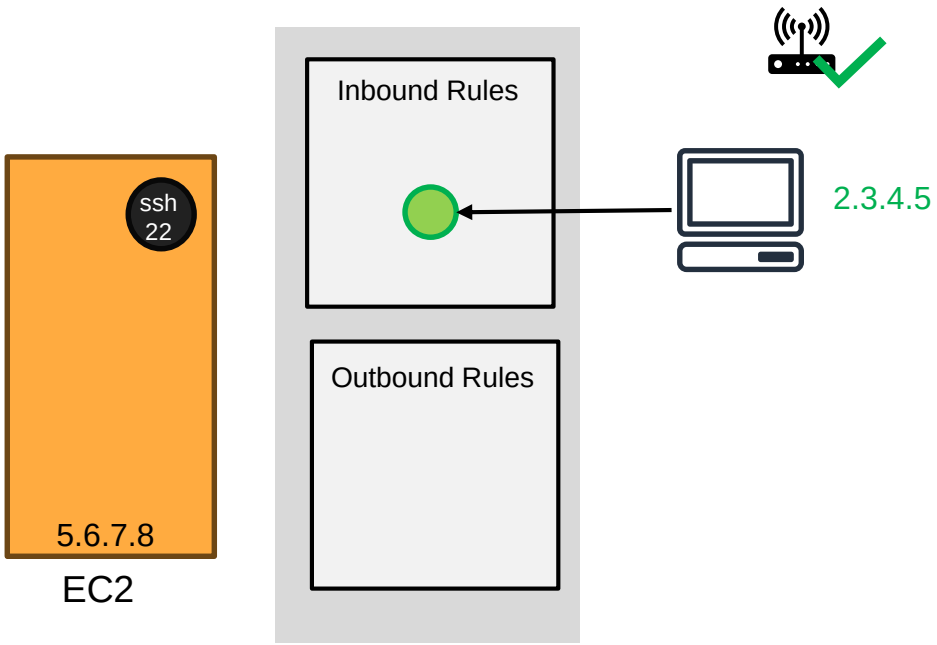
Inbound Rules

Protocol	Port	Source
TCP	22	1.2.3.4/32

Outbound Rules

Protocol	Port	Destination

Security Group scenarios



Allow SSH from only your home IP

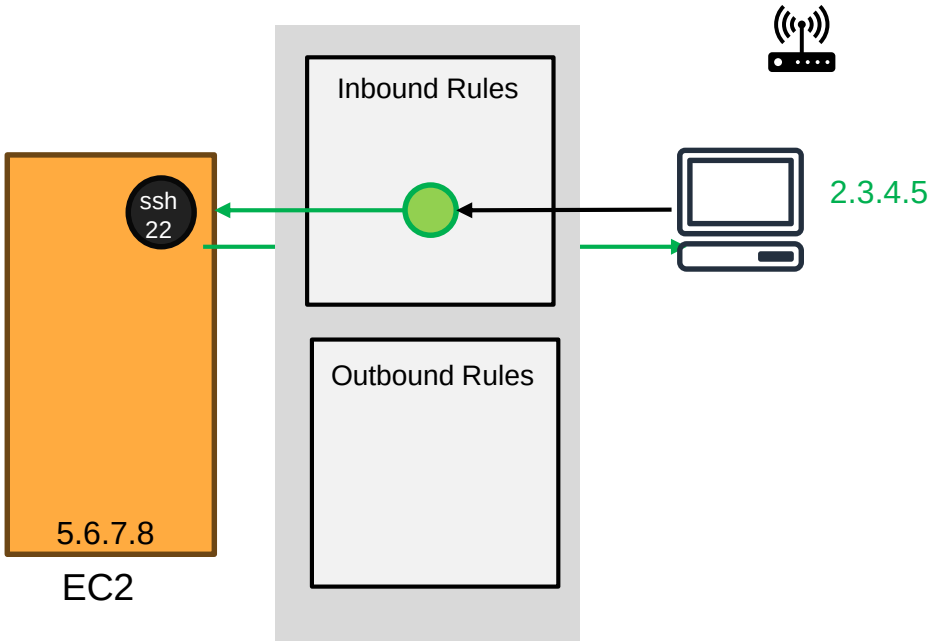
Inbound Rules

Protocol	Port	Source
TCP	22	1.2.3.4/32

Outbound Rules

Protocol	Port	Destination

Security Group scenarios



Allow SSH from only your home IP

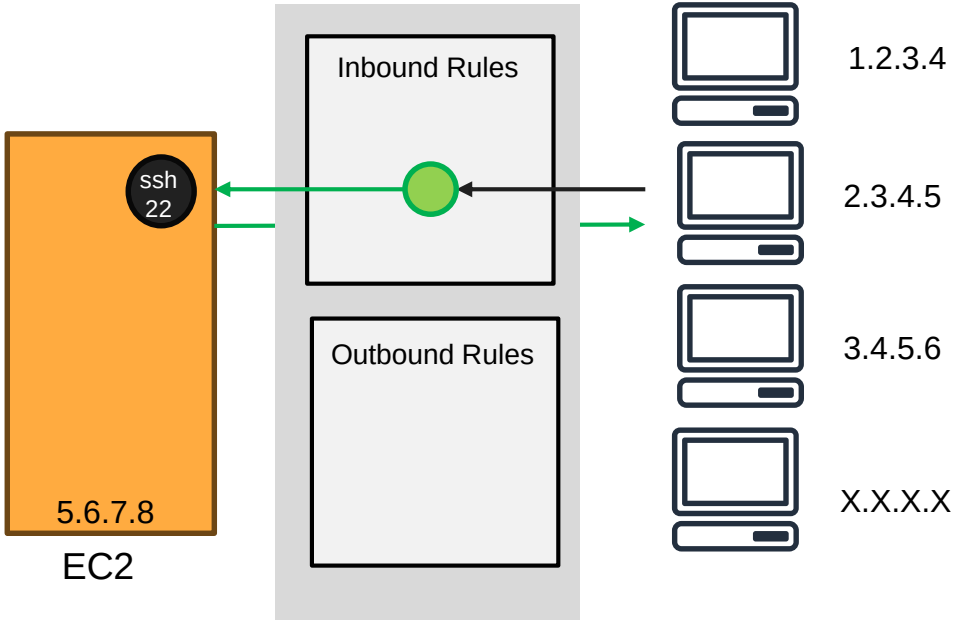
Inbound Rules

Protocol	Port	Source
TCP	22	2.3.4.5/32

Outbound Rules

Protocol	Port	Destination

Security Group scenarios



Allow SSH from anywhere

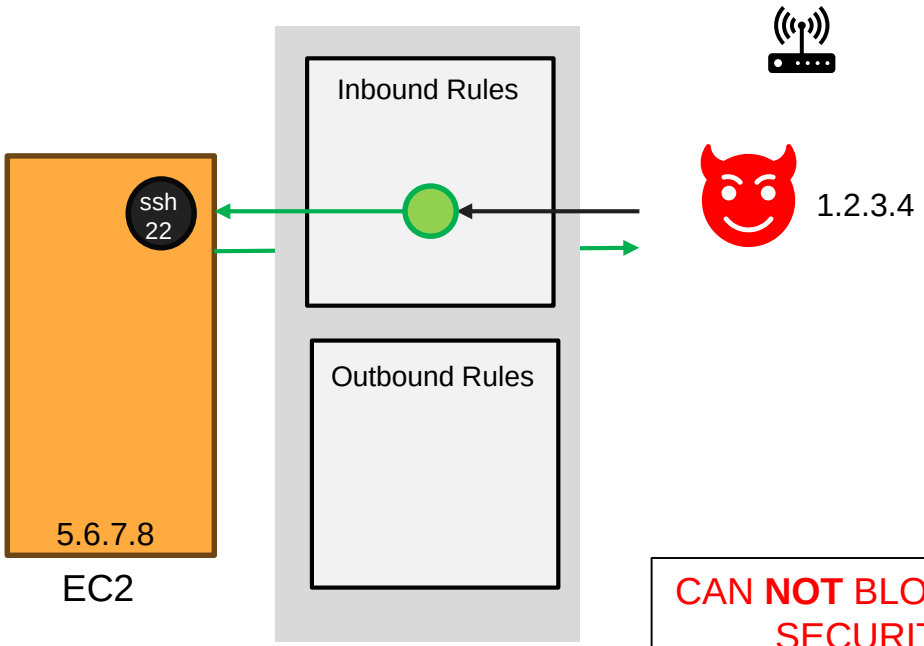
Inbound Rules

Protocol	Port	Source
TCP	22	0.0.0.0/0

Outbound Rules

Protocol	Port	Destination

Security Group scenarios



Deny or Block inbound traffic from a specific malicious IP (1.2.3.4)

Inbound Rules

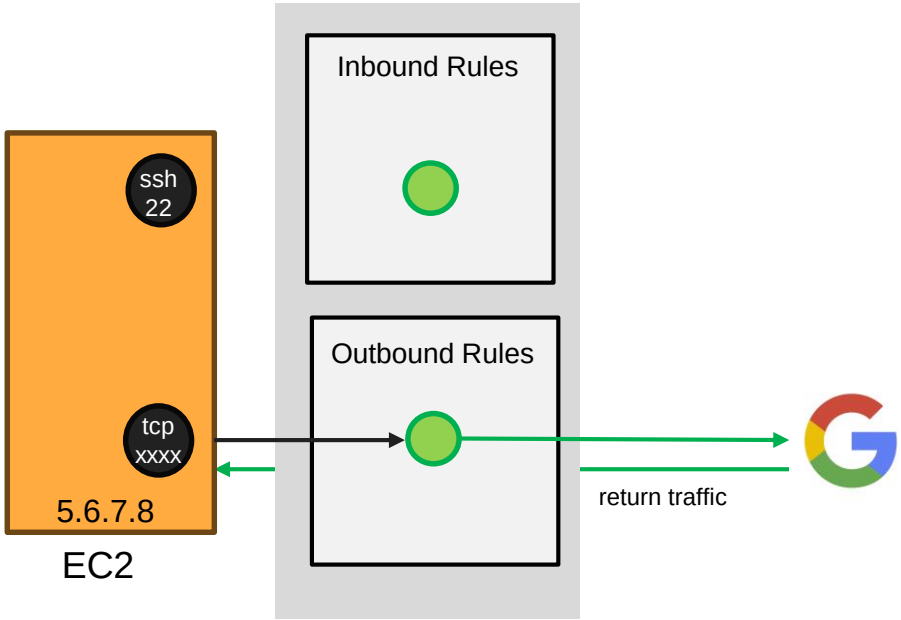
Protocol	Port	Source
TCP	22	0.0.0.0/0

Outbound Rules

Protocol	Port	Destination

CAN **NOT** BLOCK IP ADDRESS USING SECURITY GROUP RULES

Security Group scenarios



Allow outbound HTTP/HTTPs

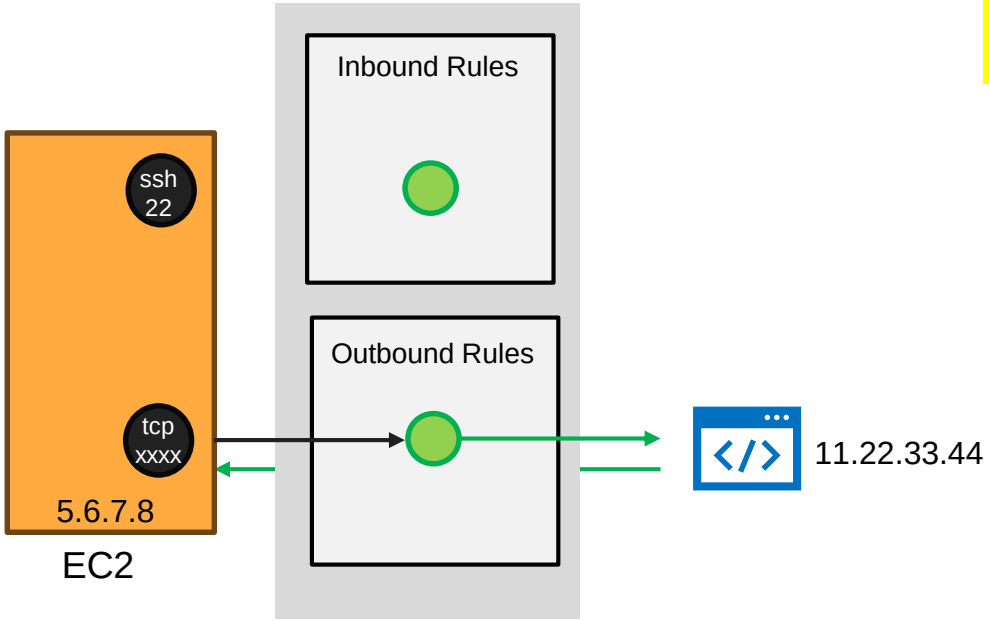
Inbound Rules

Protocol	Port	Source

Outbound Rules

Protocol	Port	Destination
TCP	80	0.0.0.0/0
TCP	443	0.0.0.0/0

Security Group scenarios



Allow outbound HTTP/HTTPS traffic to a specific destination

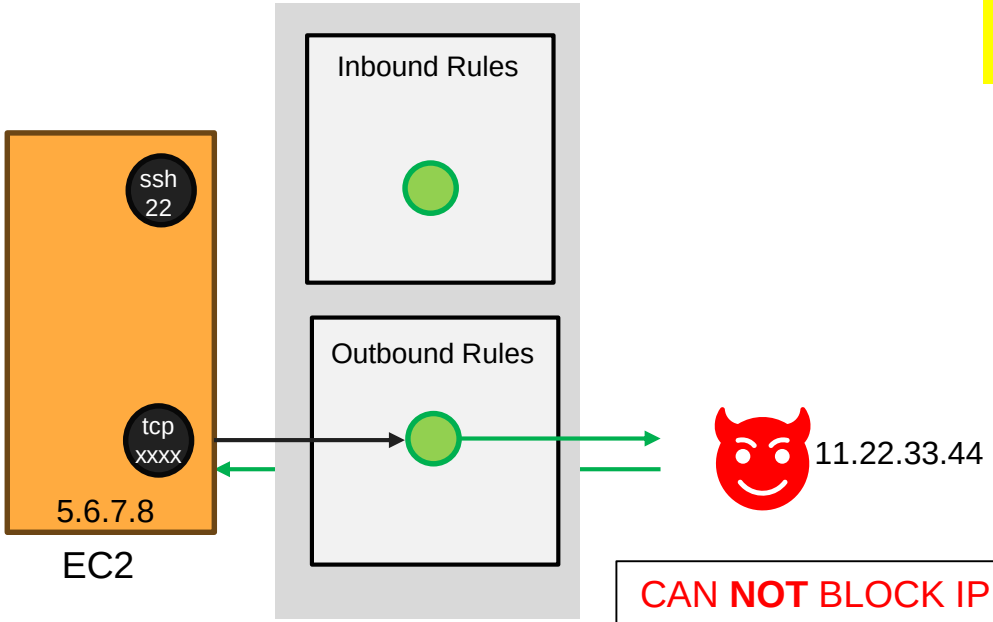
Inbound Rules

Protocol	Port	Source

Outbound Rules

Protocol	Port	Destination
TCP	80	11.22.33.44/32
TCP	443	11.22.33.44/32

Security Group scenarios



Deny or Block inbound traffic from a specific malicious IP

Inbound Rules

Protocol	Port	Source

Outbound Rules

Protocol	Port	Destination
TCP	80	0.0.0.0/0
TCP	443	0.0.0.0/0

CAN **NOT** BLOCK IP ADDRESS USING SECURITY GROUP RULES

Security Group scenarios

EC2 to EC2 communication

EC2-A Inbound Rules

Protocol	Port	Source

EC2-A Outbound Rules

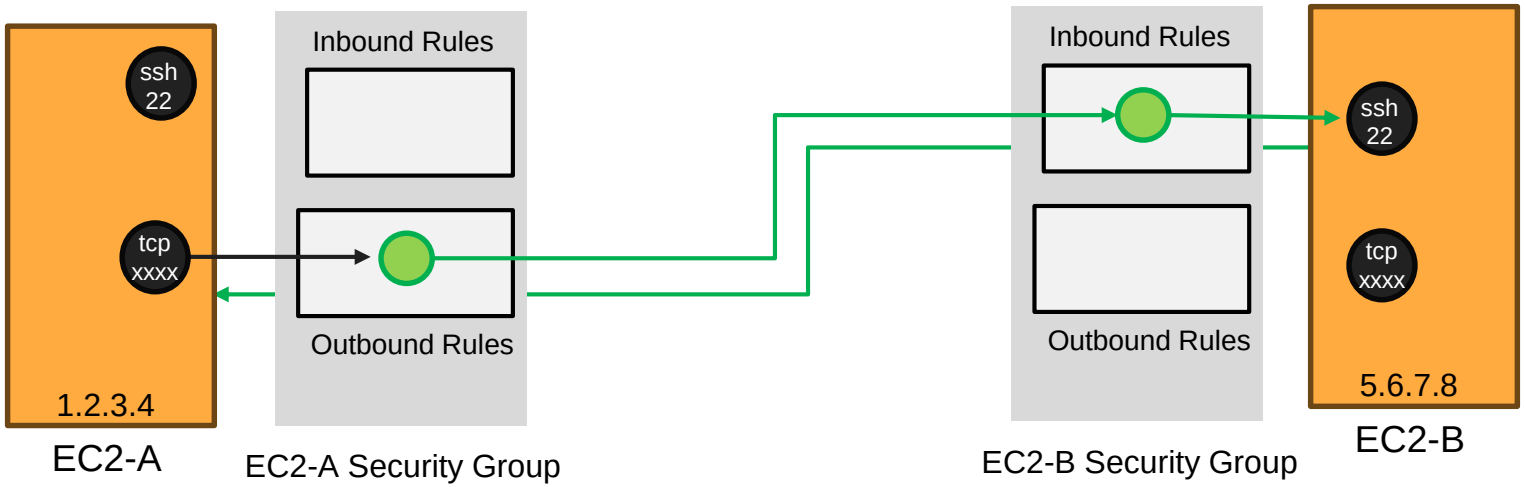
Protocol	Port	Destination
TCP	22	5.6.7.8/32

EC2-B Inbound Rules

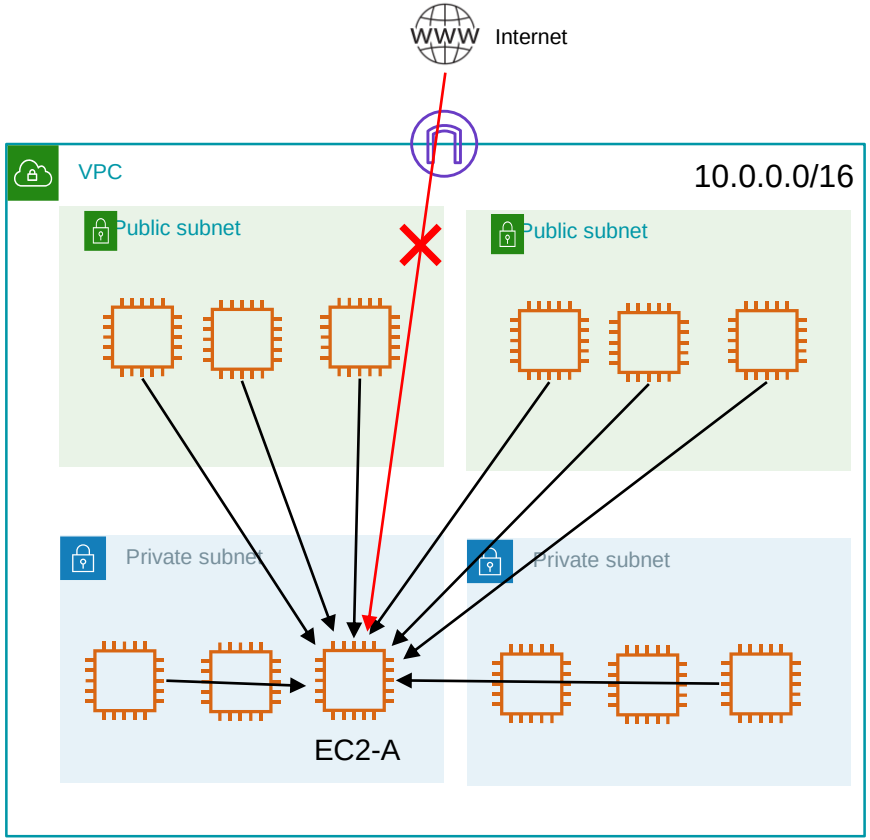
Protocol	Port	Source
TCP	22	1.2.3.4/32

EC2-B Outbound Rules

Protocol	Port	Destination



Security Group scenarios



Allow all traffic from within the VPC

EC2-A
Inbound Rules

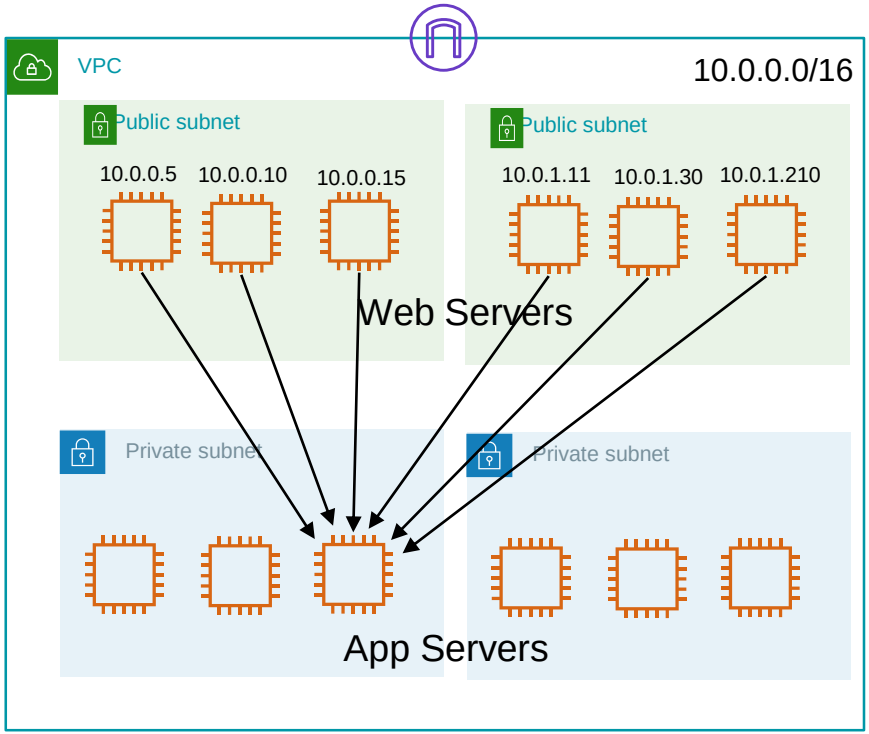
Protocol	Port	Source
All traffic	All	10.0.0.0/16

EC2-A
Outbound Rules

Protocol	Port	Destination
All traffic	All	10.0.0.0/16

Security Group scenarios

Allow traffic from only webservers



Webserver Security Group

Inbound rules

Protocol	Port	Source
HTTP	80	0.0.0.0/0

Outbound rules

Protocol	Port	Destination
All traffic	All	0.0.0.0/0

Appserver Security Group

Inbound rules

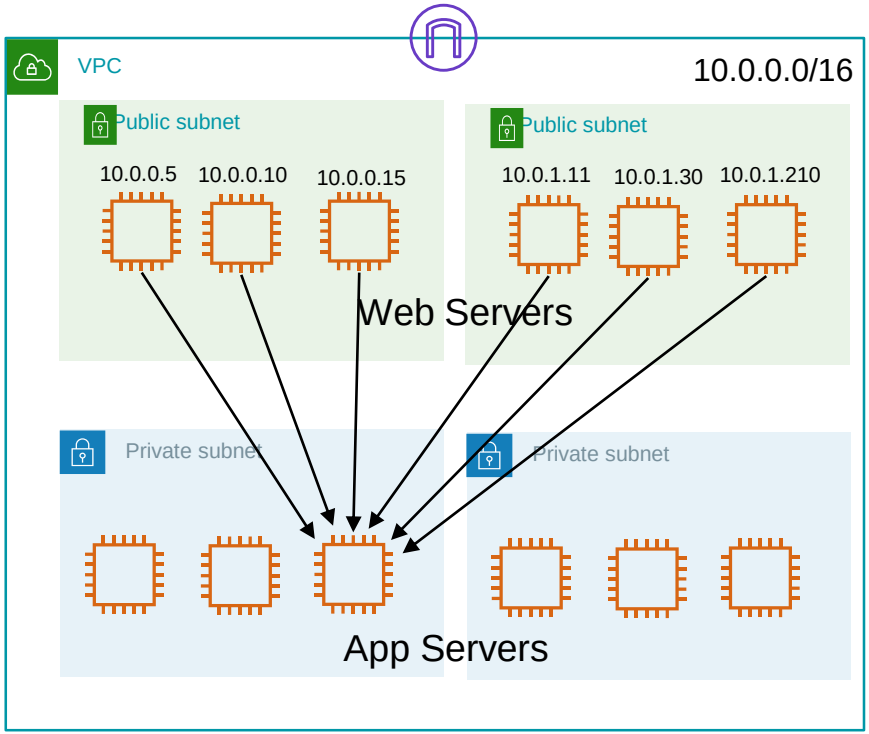
Protocol	Port	Source
HTTP	80	10.0.0.5/32
HTTP	80	10.0.0.10/32
HTTP	80	10.0.0.15/32
HTTP	80	10.0.1.11/32
HTTP	80	10.0.1.30/32
HTTP	80	10.0.1.210/32

Outbound rules

Protocol	Port	Destination
All traffic	All	0.0.0.0/0

Security Group scenarios

Allow traffic from only webservers – **Better way**



Webserver Security Group sg-xxxxxxxx

Inbound rules

Protocol	Port	Source
HTTP	80	0.0.0.0/0

Outbound rules

Protocol	Port	Destination
All traffic	All	0.0.0.0/0

Appserver Security Group

Inbound rules

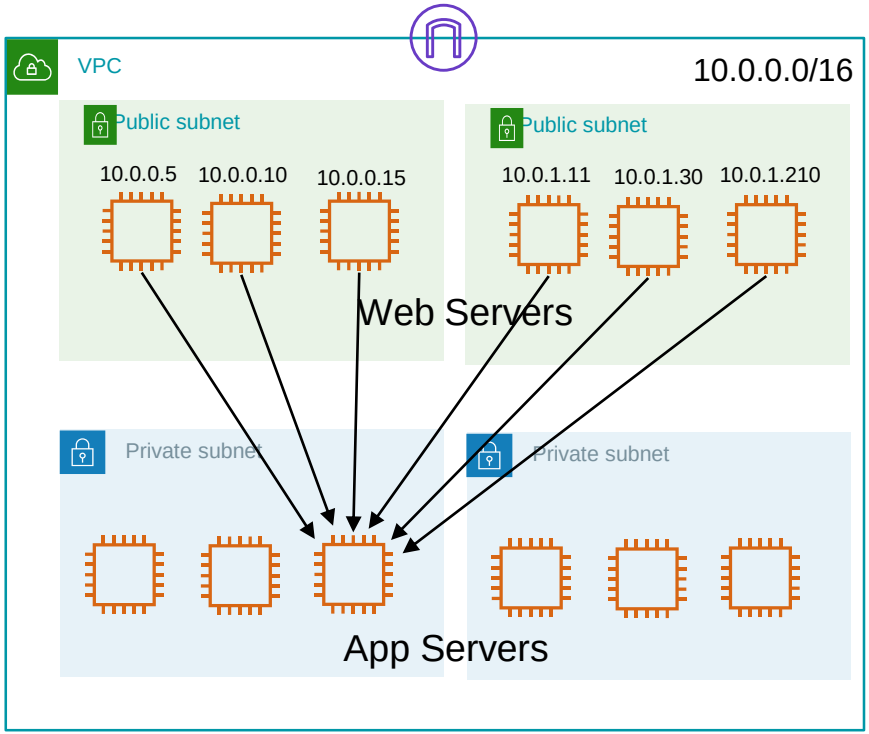
Protocol	Port	Source
HTTP	80	10.0.0.5/32
HTTP	80	10.0.0.10/32
HTTP	80	10.0.0.15/32
HTTP	80	10.0.1.11/32
HTTP	80	10.0.1.30/32
HTTP	80	10.0.1.210/32

Outbound rules

Protocol	Port	Destination
All traffic	All	0.0.0.0/0

Security Group scenarios

Allow traffic from only webservers – **Better way**



Webserver Security Group sg-xxxxxxxx

Inbound rules

Protocol	Port	Source
HTTP	80	0.0.0.0/0

Outbound rules

Protocol	Port	Destination
All traffic	All	0.0.0.0/0

Appserver Security Group

Inbound rules

Protocol	Port	Source
HTTP	80	**sg-xxxxxxxx**

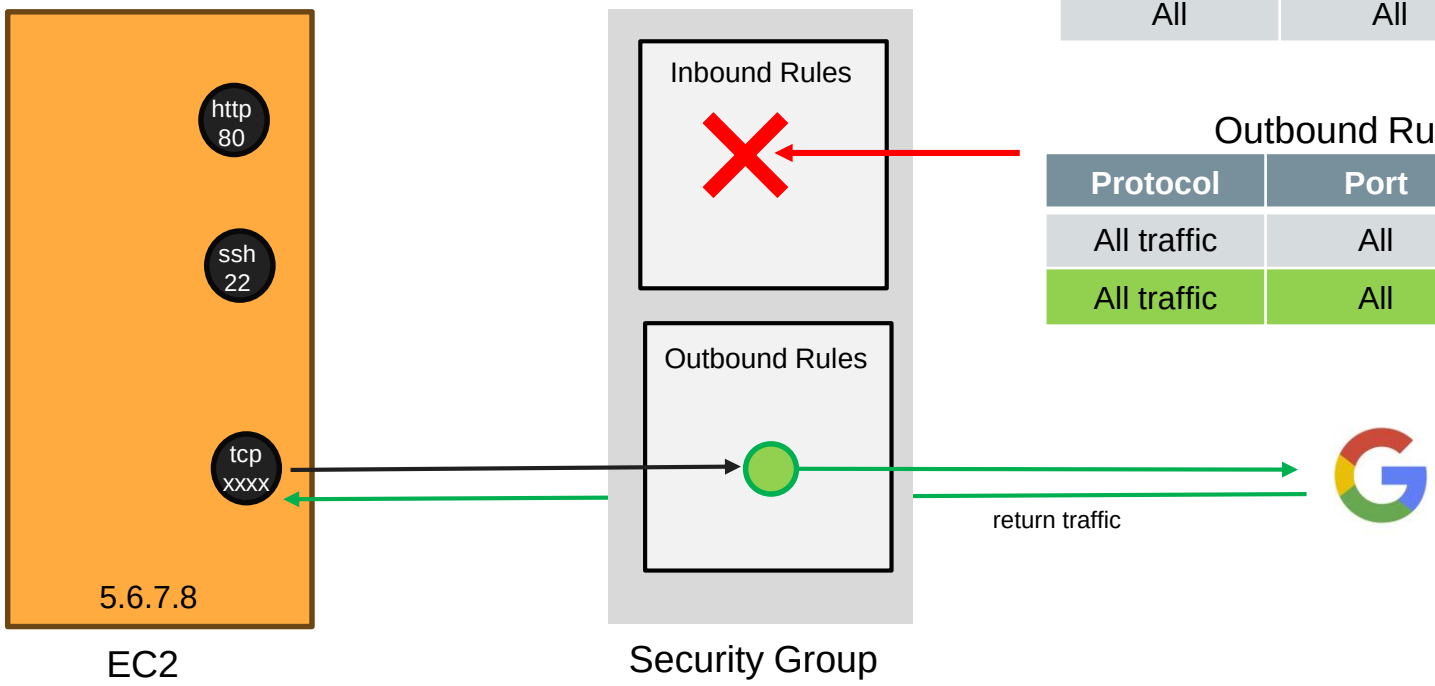
Outbound rules

Protocol	Port	Destination
All traffic	All	0.0.0.0/0

Default Security Group

Inbound Rules

Protocol	Port	Source
All	All	sg-xxxxx



Outbound Rules

Protocol	Port	Destination
All traffic	All	0.0.0.0/0
All traffic	All	::/0

Default Security Group

Inbound

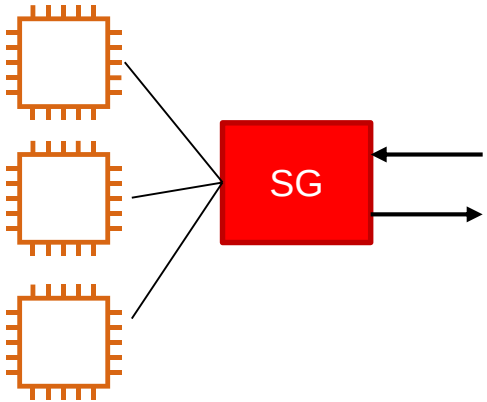
Source	Protocol	Port range	Description
sg-1234567890abcdef0	All	All	Allows inbound traffic from all resources that are assigned to this security group. The source is the ID of this security group.

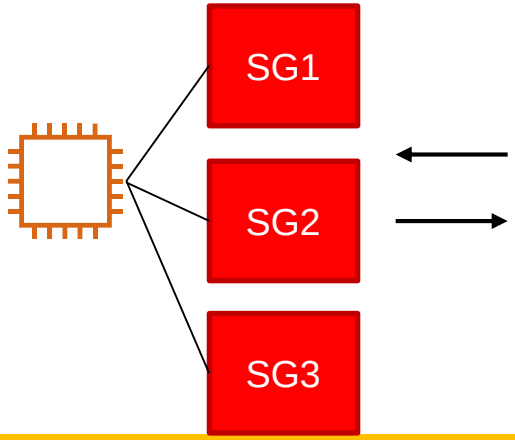
Outbound

Destination	Protocol	Port range	Description
0.0.0.0/0	All	All	Allows all outbound IPv4 traffic.
::/0	All	All	Allows all outbound IPv6 traffic. This rule is added only if your VPC has an associated IPv6 CIDR block.

Security Groups - Summary

- Single Security Group can be attached to multiple instances
- Single Instance can have multiple Security groups
- All inbound traffic is blocked by default
- All outbound traffic is authorised by default
- Authorises traffic for both IPv4 and IPv6 traffic
- Security groups are stateful
- You can reference another Security group as a source

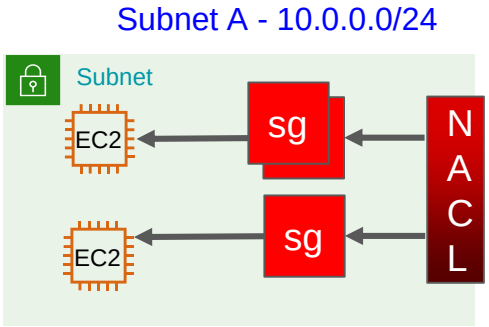




VPC Firewall - Network ACLs

Network Access Control List (NACL)

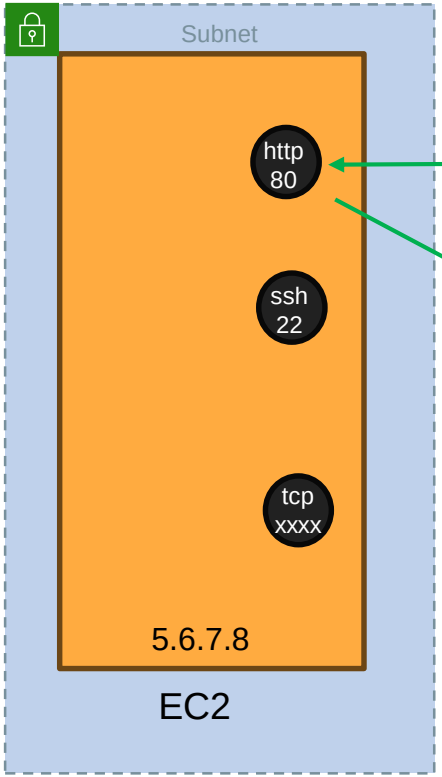
- Works at Subnet level – Hence automatically applied to all instances
- Contains both Allow and Deny rules. Rules are numbered.
- Rules are evaluated in the order of rule number (1 to 32766)
- Stateless – We need to explicitly open ports for return traffic
- Default NACL allows all inbound and outbound traffic
- **NACL are a great way of blocking a specific IP at the subnet level**

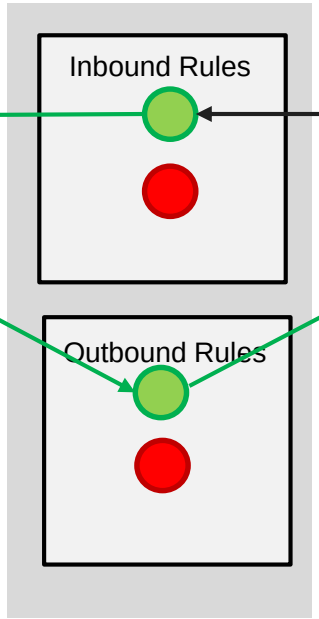


Network ACL inbound rules

#Rule	Type	Protocol	Port	Source	Allow/Deny
100	All IPv4 traffic	All	All	180.151.138.43/32	DENY
101	HTTPS	TCP	443	0.0.0.0/0	ALLOW
*	All IPv4 traffic	All	All	0.0.0.0/0	DENY

Network ACL





1.2.3.4
(port: XXXX)

return traffic

return traffic

9000
5000

Inbound Rules

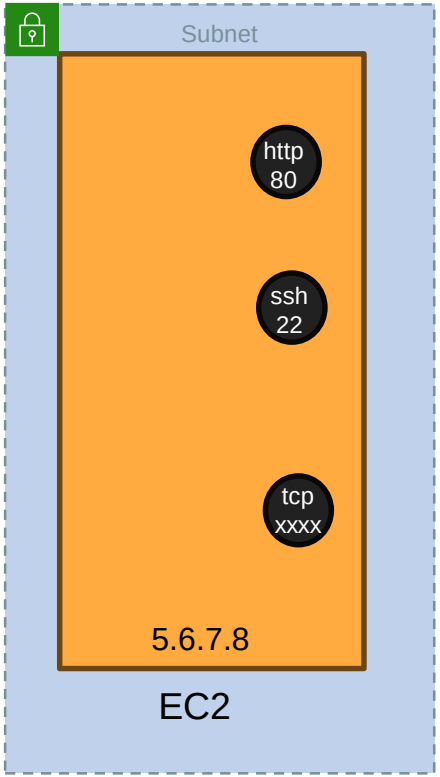
Rule number	Protocol	Port	Source	Allow/ Deny
100	TCP	80	1.2.3.4/32	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

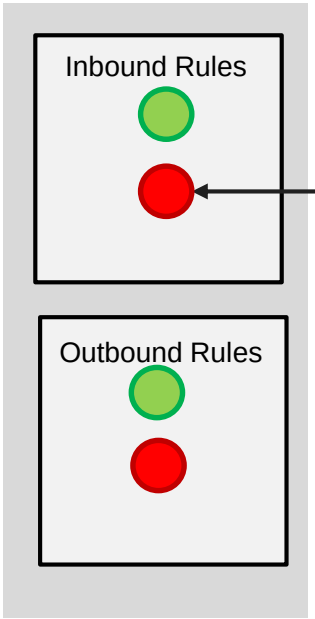
32768-
65535

Outbound Rules

Rule number	Protocol	Port	Destination	Allow/ Deny
100	TCP	XXXX	1.2.3.4/32	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Network ACL – Implicit Deny





9.10.11.12
(port: XXXX)

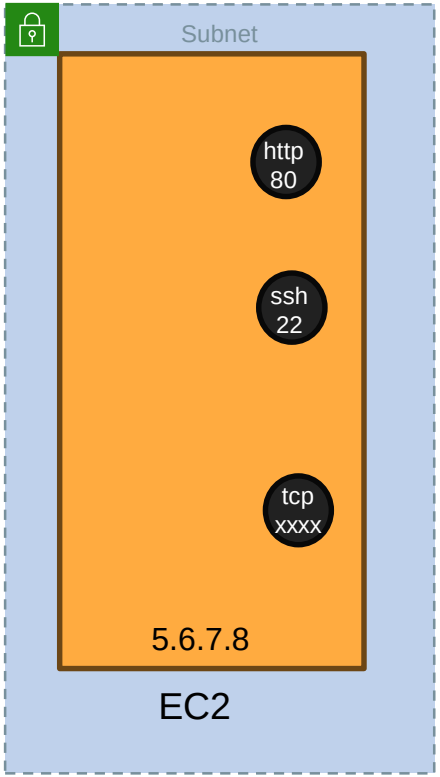
Inbound Rules

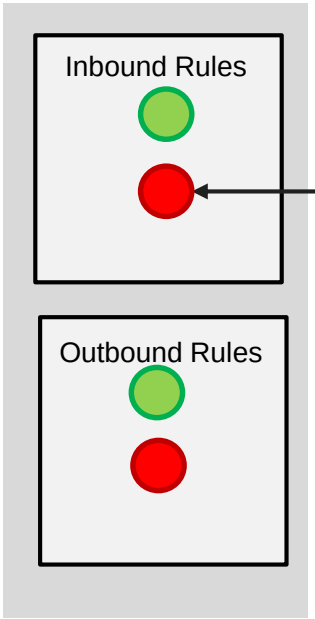
Rule number	Protocol	Port	Source	Allow/ Deny
100	TCP	80	1.2.3.4/32	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Outbound Rules

Rule number	Protocol	Port	Destination	Allow/ Deny
100	TCP	XXXX	1.2.3.4/32	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Network ACL – Explicit Deny





Inbound Rules

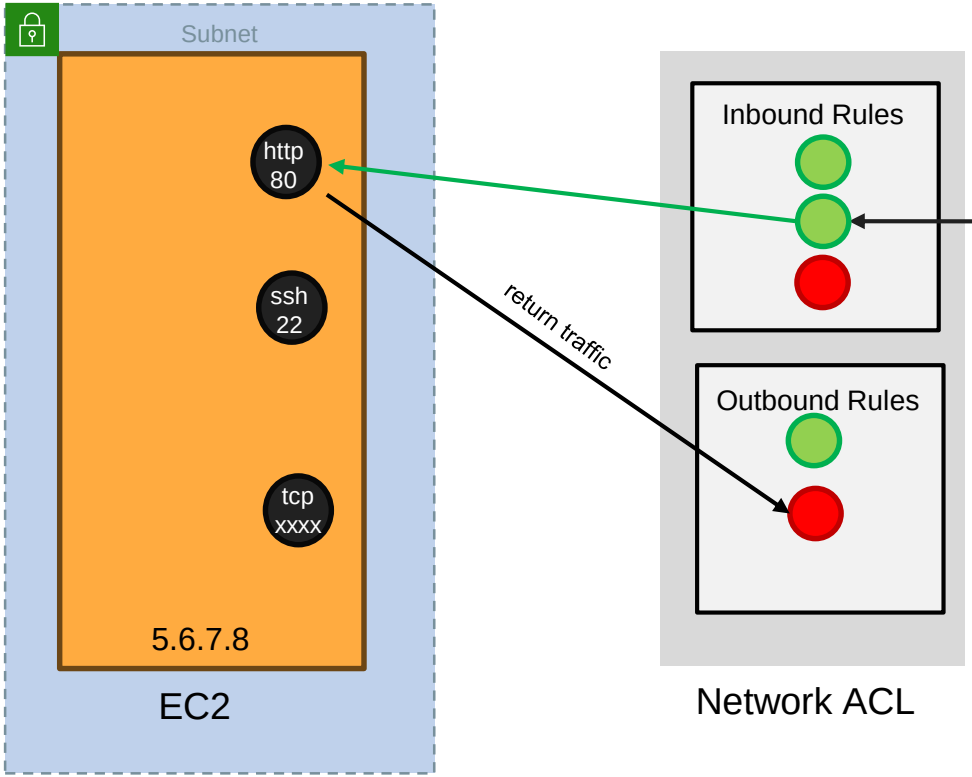
Rule number	Protocol	Port	Source	Allow/ Deny
100	TCP	80	1.2.3.4/32	Allow
101	TCP	80	9.10.11.12/32	Deny
*	All IPv4 traffic	All	0.0.0.0/0	Deny

9.10.11.12
(port: XXXX)

Outbound Rules

Rule number	Protocol	Port	Destination	Allow/ Deny
100	TCP	XXXX	1.2.3.4/32	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Network ACL – Return traffic now allowed



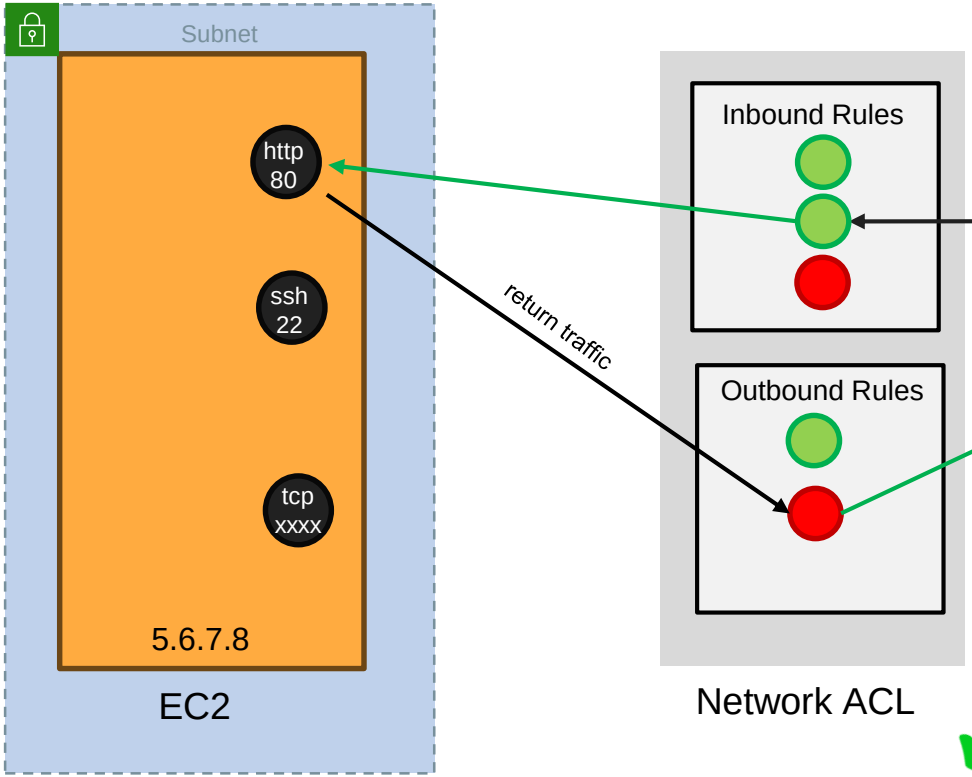
Inbound Rules

Rule number	Protocol	Port	Source	Allow/ Deny
100	TCP	80	1.2.3.4/32	Allow
200	TCP	80	9.10.11.12/32	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Outbound Rules

Rule number	Protocol	Port	Destination	Allow/ Deny
100	TCP	XXXX	1.2.3.4/32	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Network ACL – Allow return traffic



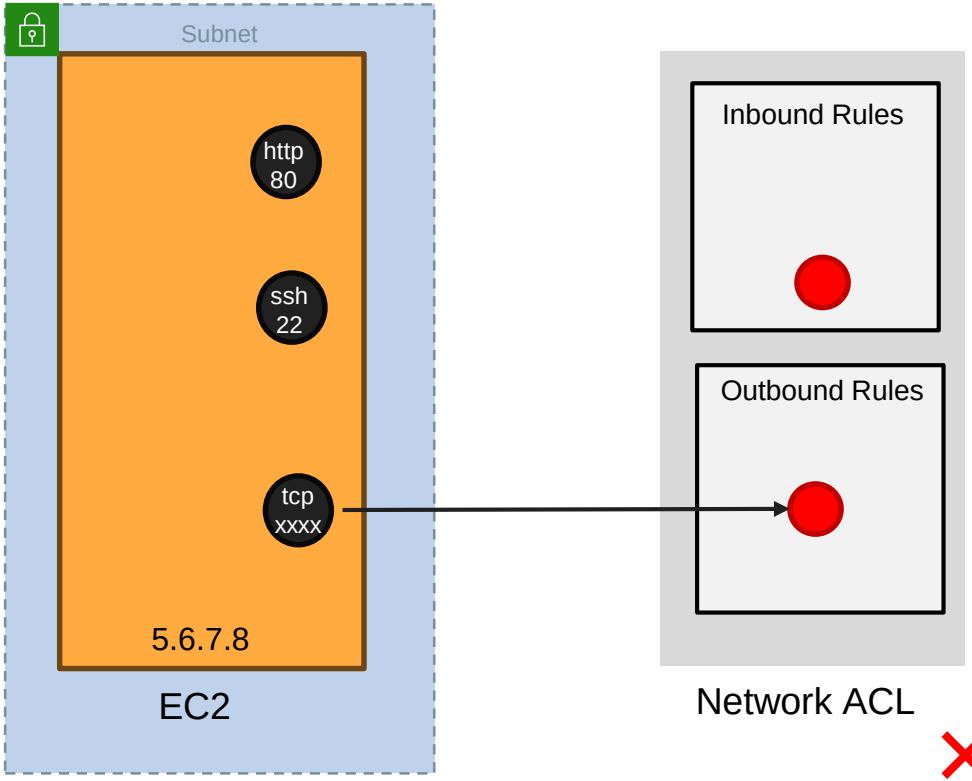
Inbound Rules

Rule number	Protocol	Port	Source	Allow/ Deny
100	TCP	80	1.2.3.4/32	Allow
200	TCP	80	9.10.11.12/32	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Outbound Rules

Rule number	Protocol	Port	Destination	Allow/ Deny
100	TCP	XXXX	1.2.3.4/32	Allow
200	TCP	XXXX	9.10.11.12/32	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Network ACL – Outbound traffic not allowed



Inbound Rules

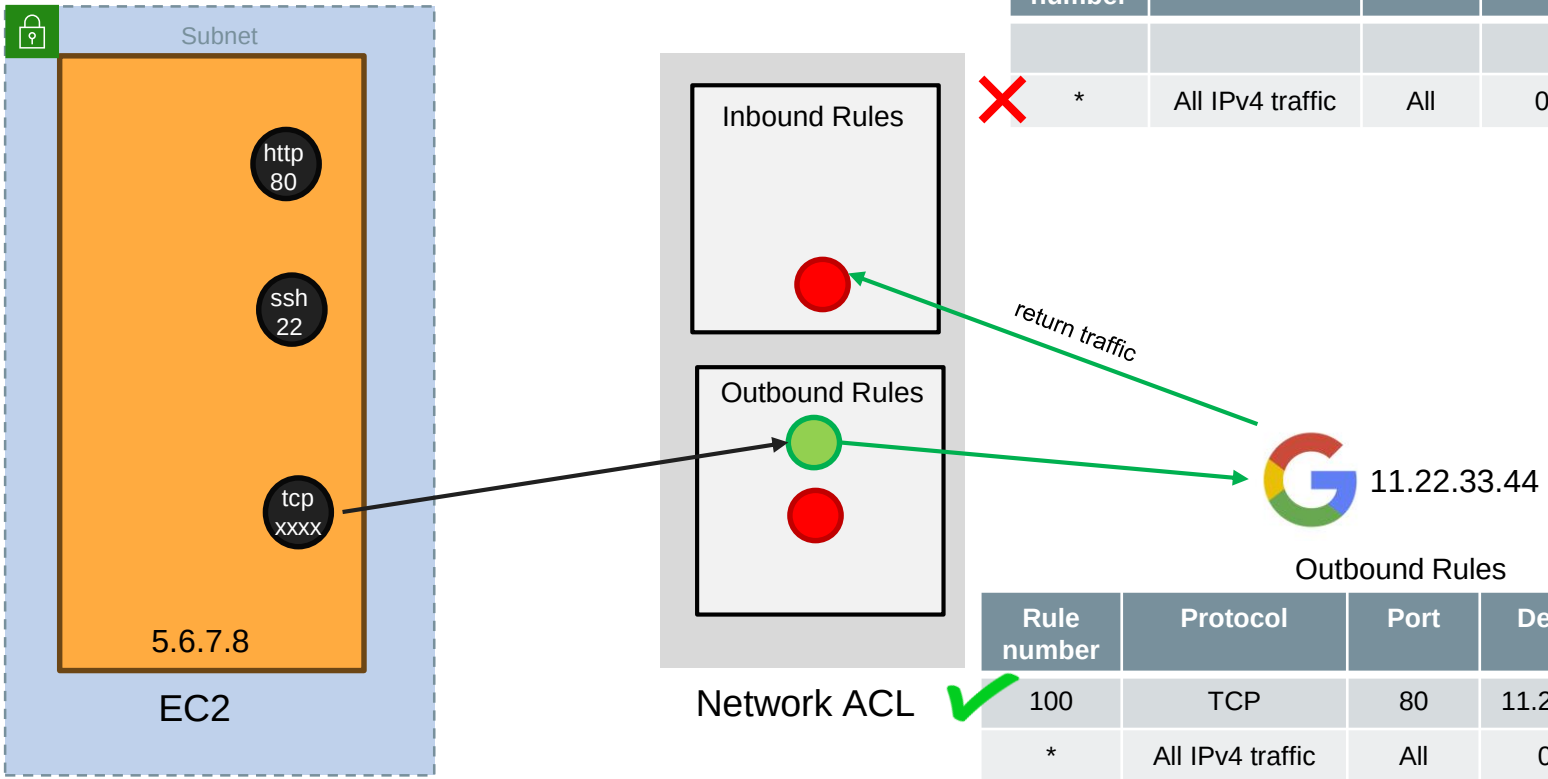
Rule number	Protocol	Port	Source	Allow/ Deny
*	All IPv4 traffic	All	0.0.0.0/0	Deny

11.22.33.44

Outbound Rules

Rule number	Protocol	Port	Destination	Allow/ Deny
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Network ACL – Return traffic not allowed



Inbound Rules

Rule number	Protocol	Port	Source	Allow/ Deny
*	All IPv4 traffic	All	0.0.0.0/0	Deny

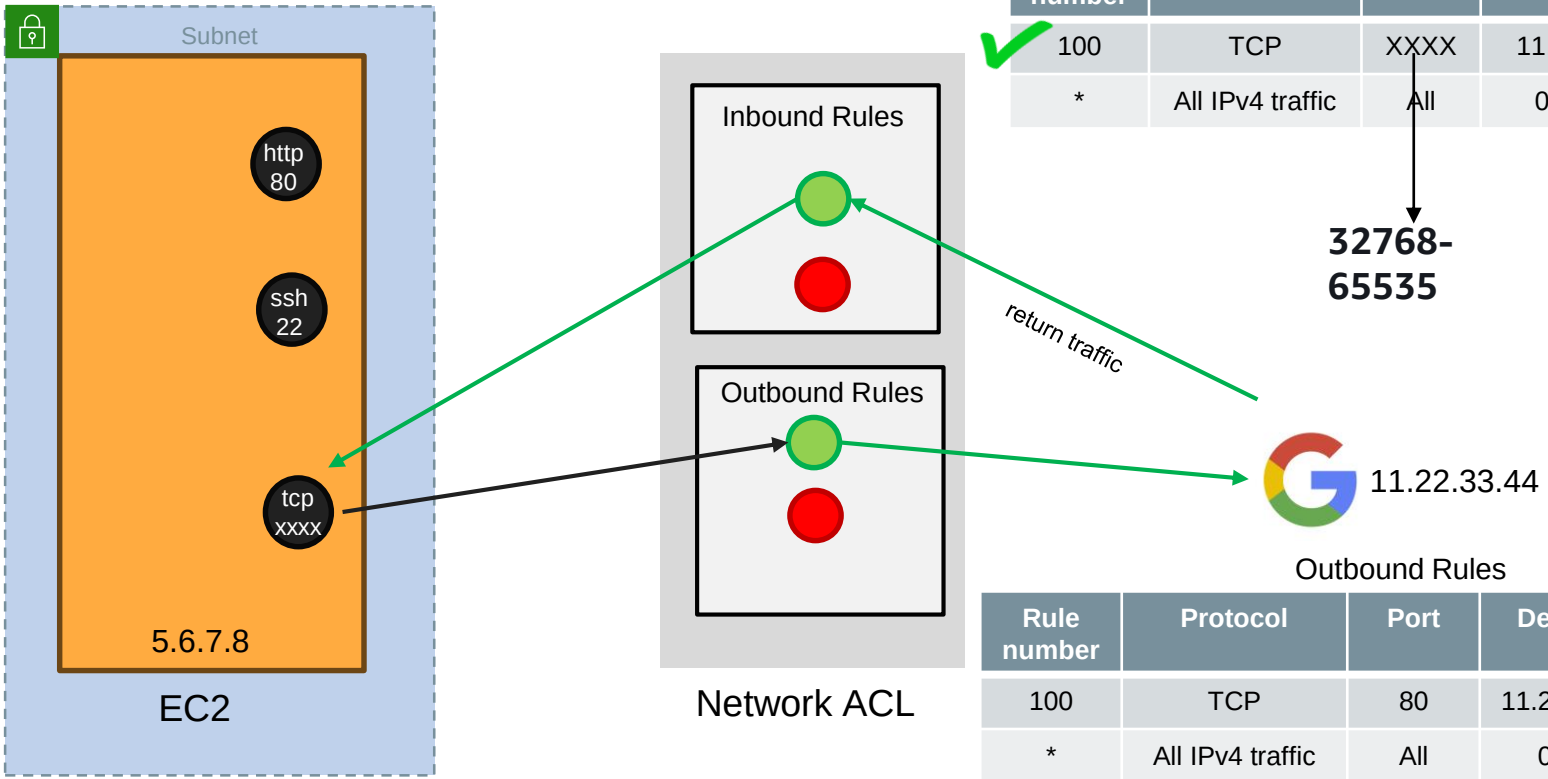
Outbound Rules

Rule number	Protocol	Port	Destination	Allow/ Deny
100	TCP	80	11.22.33.44/32	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Network ACL – Return traffic allowed

Inbound Rules

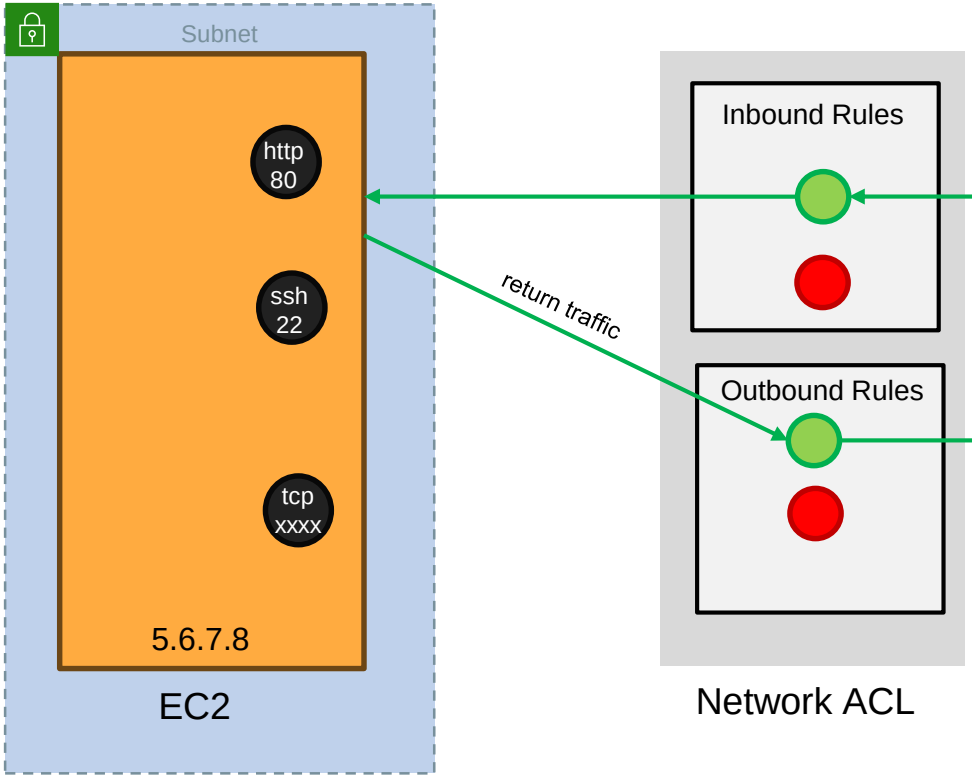
Rule number	Protocol	Port	Source	Allow/ Deny
100	TCP	XXXX	11.22.33.44	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny



Outbound Rules

Rule number	Protocol	Port	Destination	Allow/ Deny
100	TCP	80	11.22.33.44/32	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Default Network ACL - Inbound



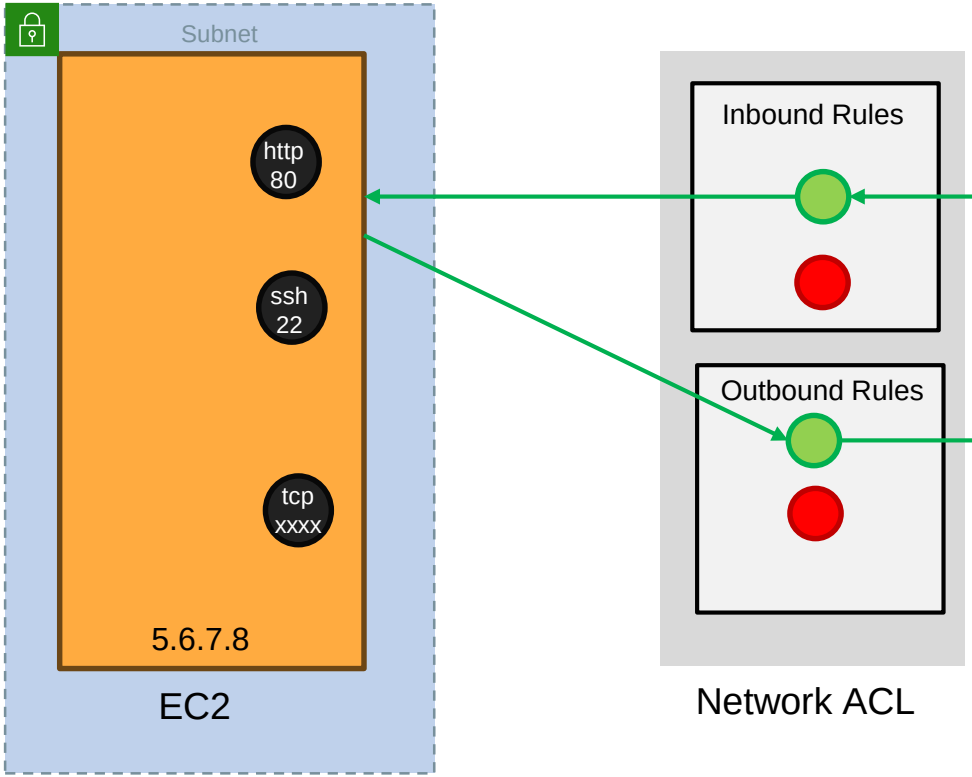
Inbound Rules

Rule number	Protocol	Port	Source	Allow/ Deny
100	All IPv4 traffic	All	0.0.0.0/0	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Outbound Rules

Rule number	Protocol	Port	Destination	Allow/ Deny
100	All IPv4 traffic	All	0.0.0.0/0	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Default Network ACL - Outbound



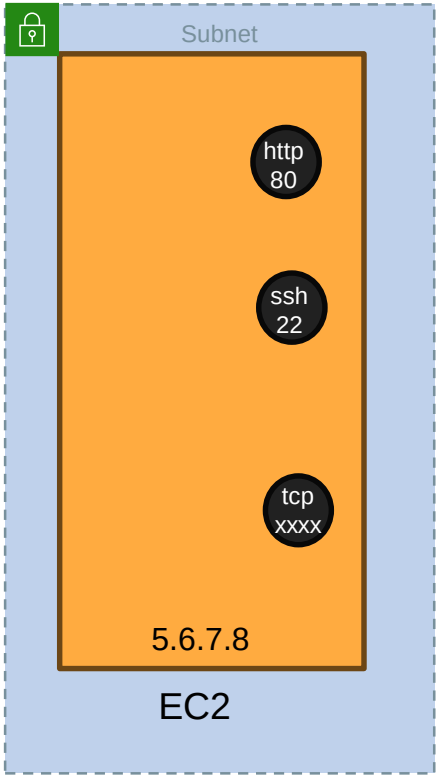
Inbound Rules

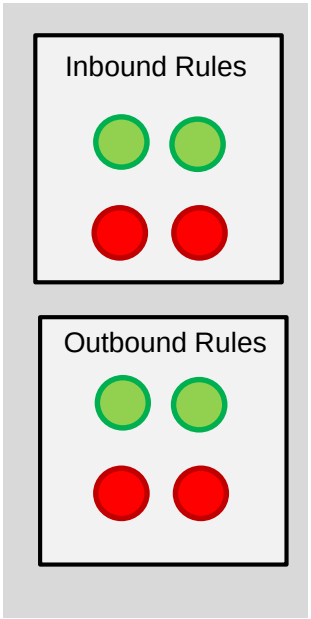
Rule number	Protocol	Port	Source	Allow/ Deny
100	All IPv4 traffic	All	0.0.0.0/0	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Outbound Rules

Rule number	Protocol	Port	Destination	Allow/ Deny
100	All IPv4 traffic	All	0.0.0.0/0	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny

Default Network ACL (IPv4 & IPv6)





Inbound Rules

Rule number	Protocol	Port	Source	Allow/ Deny
100	All IPv4 traffic	All	0.0.0.0/0	Allow
101	All IPv6 traffic	All	::/0	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny
*	All IPv6 traffic	All	::/0	Deny

Outbound Rules

Rule number	Protocol	Port	Destination	Allow/ Deny
100	All IPv4 traffic	All	0.0.0.0/0	Allow
101	All IPv6 traffic	All	::/0	Allow
*	All IPv4 traffic	All	0.0.0.0/0	Deny
*	All IPv6 traffic	All	::/0	Deny

Security Groups vs Network ACL

Security Group	Network ACL
Operates at EC2 instance or ENI level	Operates at Subnet level
Supports only Allow rules	Supports both Allow and Deny rules
Stateful – Return traffic is allowed	Stateless – Return traffic needs to be authorized in Outbound rules
All rules are evaluated before making a decision	Rules are evaluated in the order (lower to higher) and first matching rule is applied

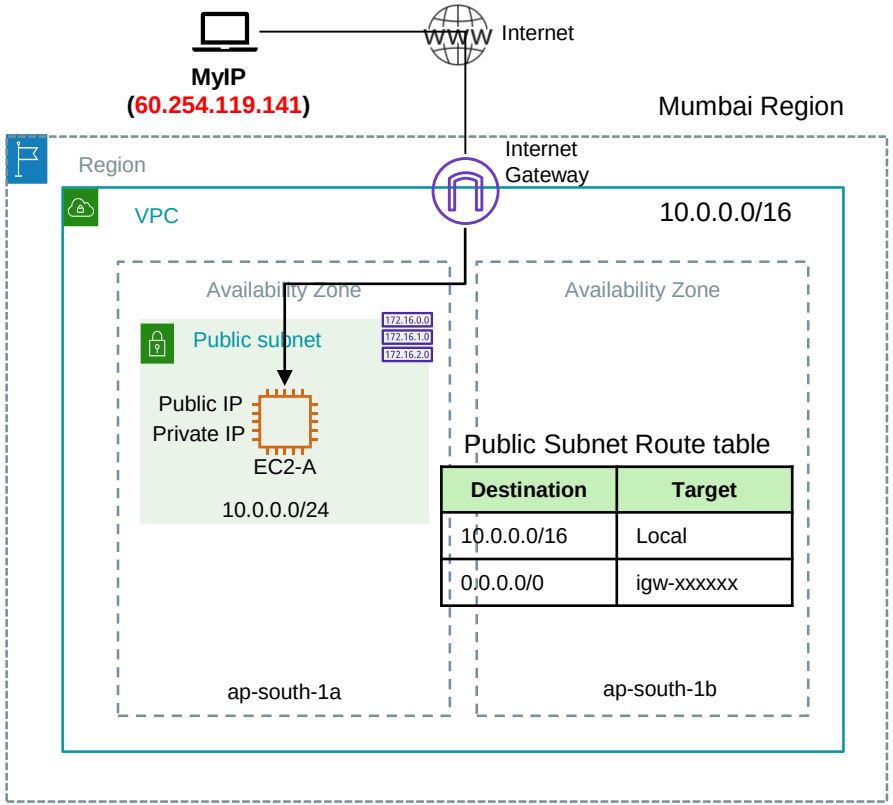
Refer VPC quota: https://docs.aws.amazon.com/vpc/latest/userguide/amazon-vpc-limits.html

Assignment - Network ACL

- Complete the Network ACL assignment as per the instructions given in the PDF with this lecture (in the resources section)
- Make sure you terminate EC2 instance after this assignment

Demo – Network ACL

Demo – How NACL works by trying out different NACL rules



High level steps

1. Create a VPC with Public Subnet, configure route table accordingly
2. Launch EC2 instance and assign Public IP. Allow SSH and HTTP in security group.
3. Connect to instance over SSH and install a httpd web server.
4. Verify that you can access web page over the internet using EC2 Elastic IP
5. Modify the Subnet NACL rules as given in the following slide and observe the behavior

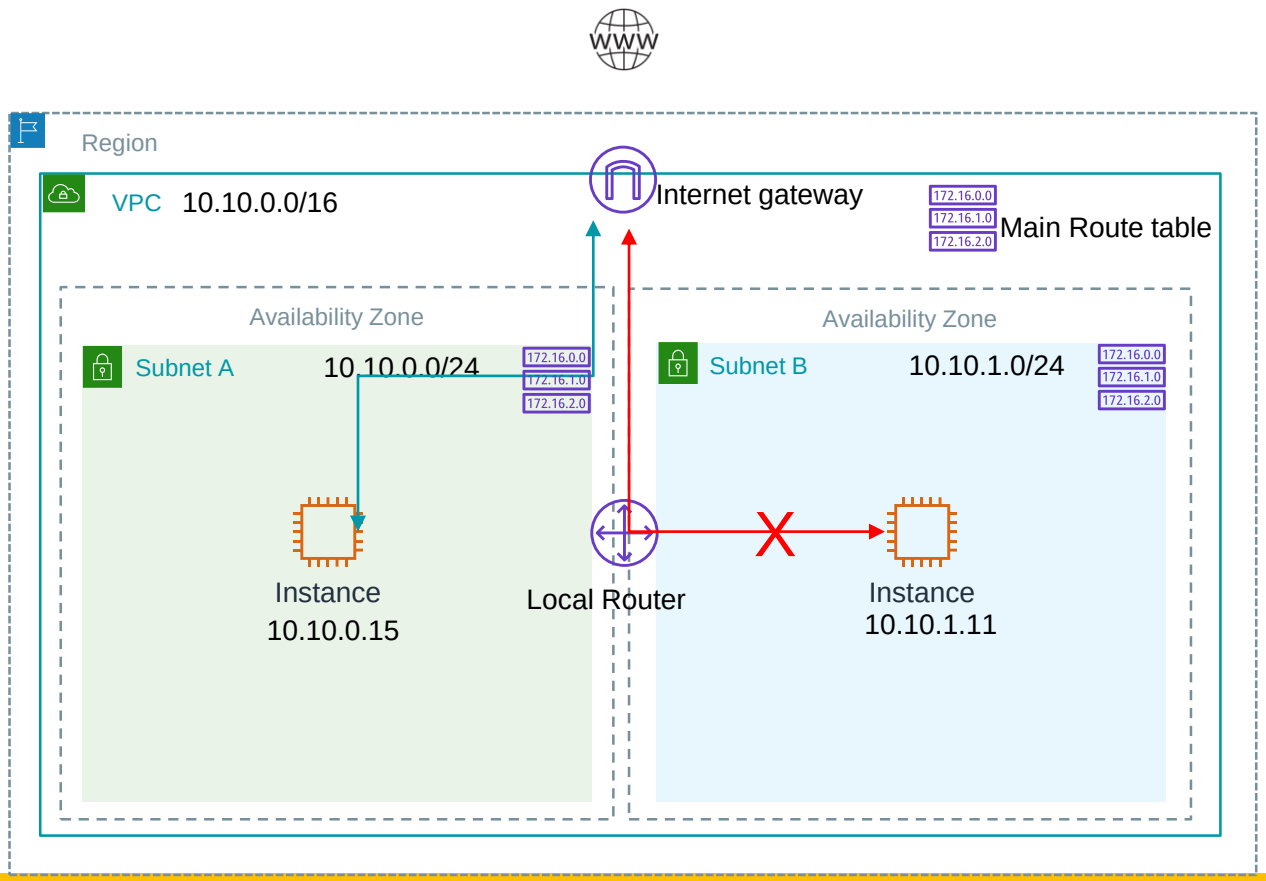
Let’s try following scenarios for Subnet NACL:

Let’s modify NACL rules in the following order and then try to connect to the webserver over HTTP (port 80) from the browser..

1. Remove the default inbound Allow rule -> Does not connect
2. Add default inbound Allow rule back and remove the default outbound Allow rule -> Does not connect
3. Add default outbound Allow rule back -> Connects
4. Remove default inbound Allow rule and instead add a new Inbound rule to allow only MyIP (your local IP/32) -> Connects
5. Remove default outbound Allow rule and instead add a new Outbound rule to allow destination as MyIP (your local IP/32) on port 80 -> Does not connect
6. Update the above rule port range from 80 to 1024-65535 -> Connects
7. Set NACL back to original i.e. default inbound and default outbound rules
8. In the inbound rules add a new rule (#200) after default Allow rule to block the traffic on port 80 from MyIP (your local IP/32) -> Connects but why?
9. Now change the Inbound rule numbers where above rule number (#) is lower than default Allow rule (#100) -> Does not connect

NAT Gateway

Public and Private subnets

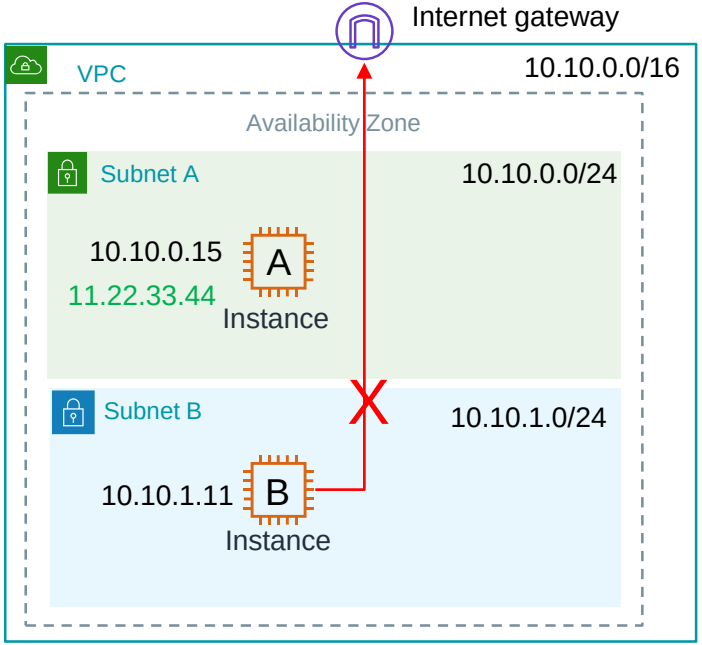


Subnet A Route Table

Destination	Target
10.10.0.0/16	Local
0.0.0.0/0	Internet Gateway

Subnet B Route Table

Destination	Target
10.10.0.0/16	Local



Can Instance B access an internet?

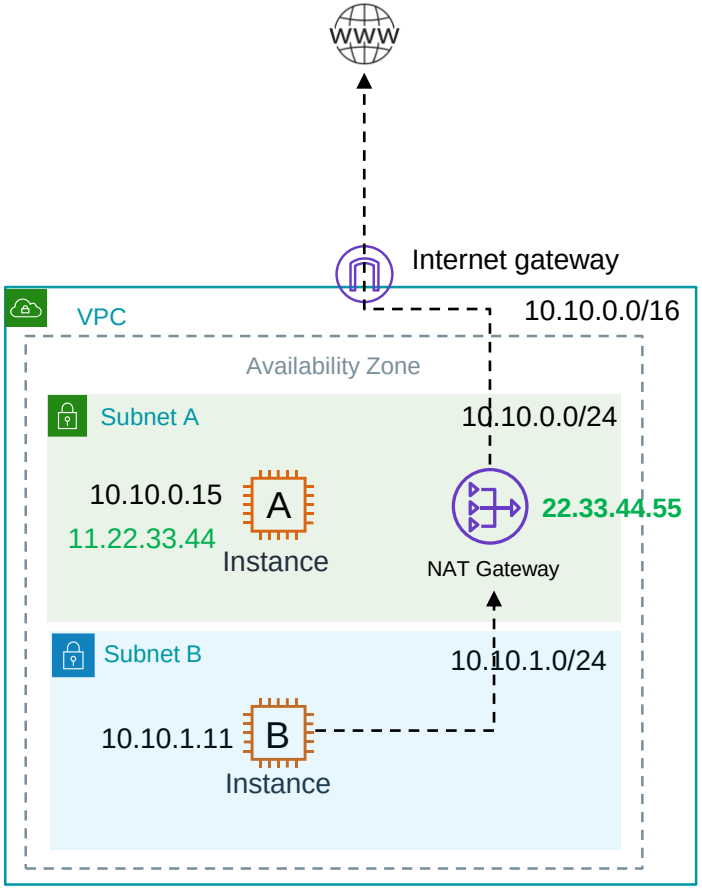
1. **Subnet is not a Public subnet**
2. **Instance does not have a Public IP**

Subnet A & Subnet C Route Table

Destination	Target
10.10.0.0/16	Local
0.0.0.0/0	Internet Gateway

Subnet B Route Table

Destination	Target
10.10.0.0/16	Local



Subnet A & Subnet C Route Table

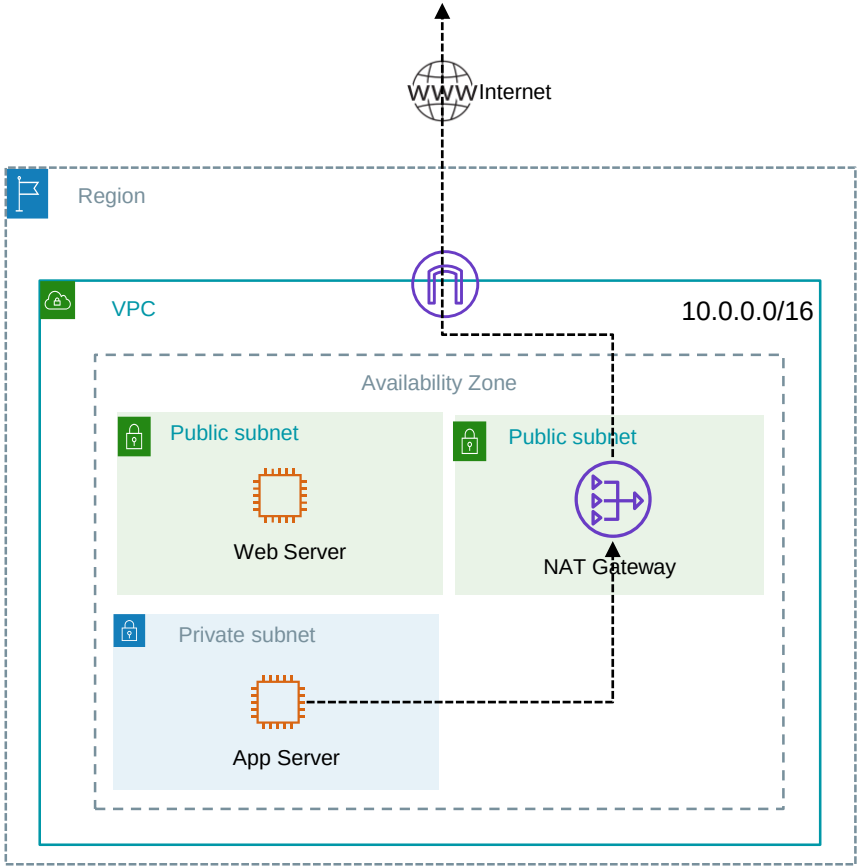
Destination	Target
10.10.0.0/16	Local
0.0.0.0/0	Internet Gateway

Subnet B Route Table

Destination	Target
10.10.0.0/16	Local
0.0.0.0/0	Nat gateway

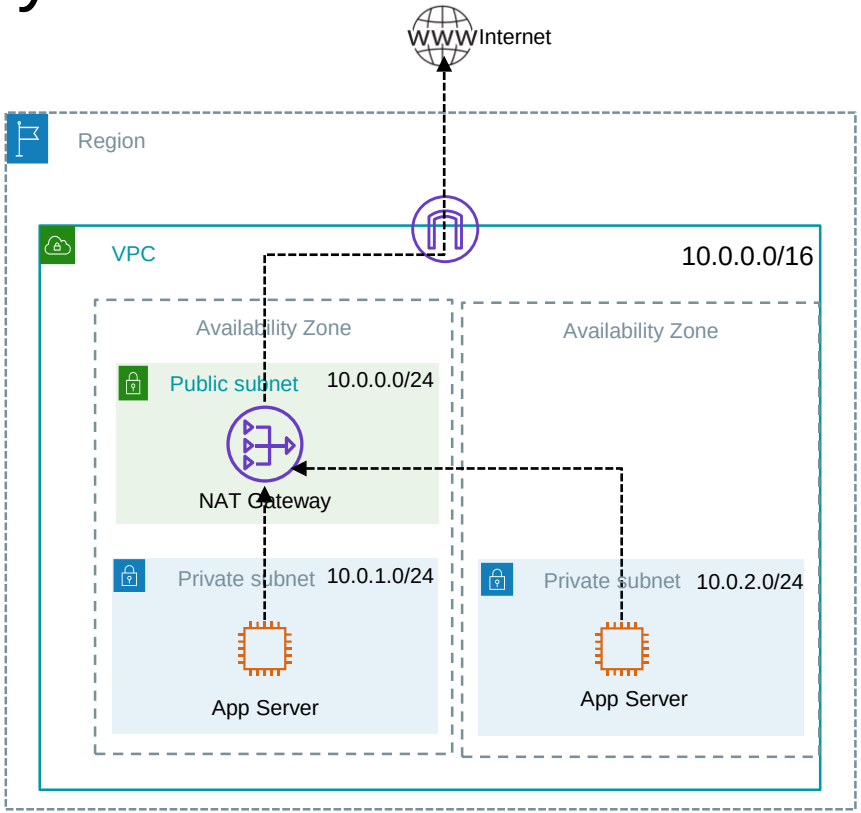
NAT Gateway

- AWS managed NAT, higher bandwidth, better availability, no admin
- Pay by the hour for usage and bandwidth
- 5 Gbps of bandwidth with automatic scaling up to 100 Gbps
- No security groups
- NACL at subnet level applies to NAT Gateway.
- Supported protocols: TCP, UDP, and ICMP
- Uses ports 1024–65535 for outbound connection
- For outbound internet access, NAT Gateway should be created in Public Subnet so that it can communicate with the Internet
- NAT Gateway should be allocated Elastic IP



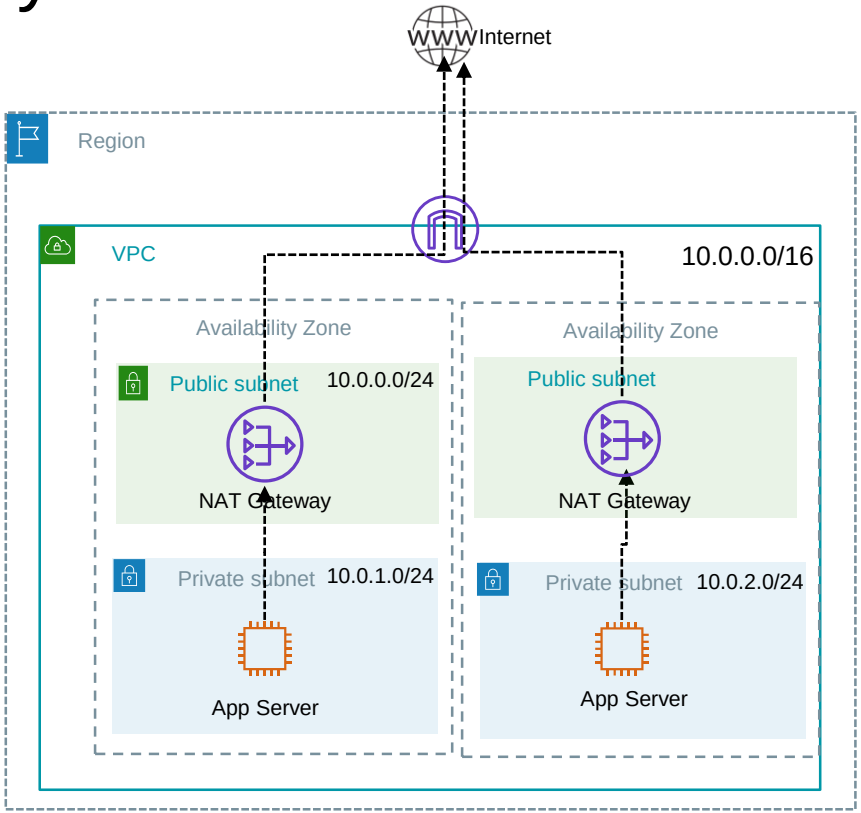
NAT Gateway High availability

- NAT Gateways are highly available within a single AZ
- For HA across multiple AZs, multiple NAT gateways can be launched



NAT Gateway High availability

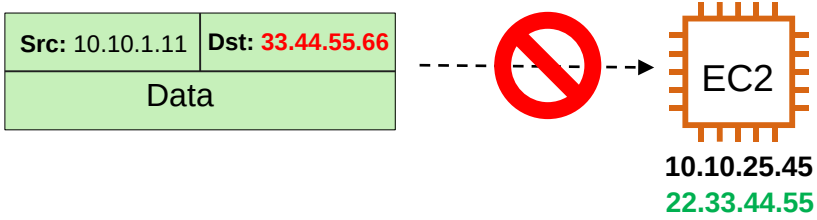
- NAT Gateways are highly available within a single AZ
- For HA across multiple AZs, multiple NAT gateways can be launched

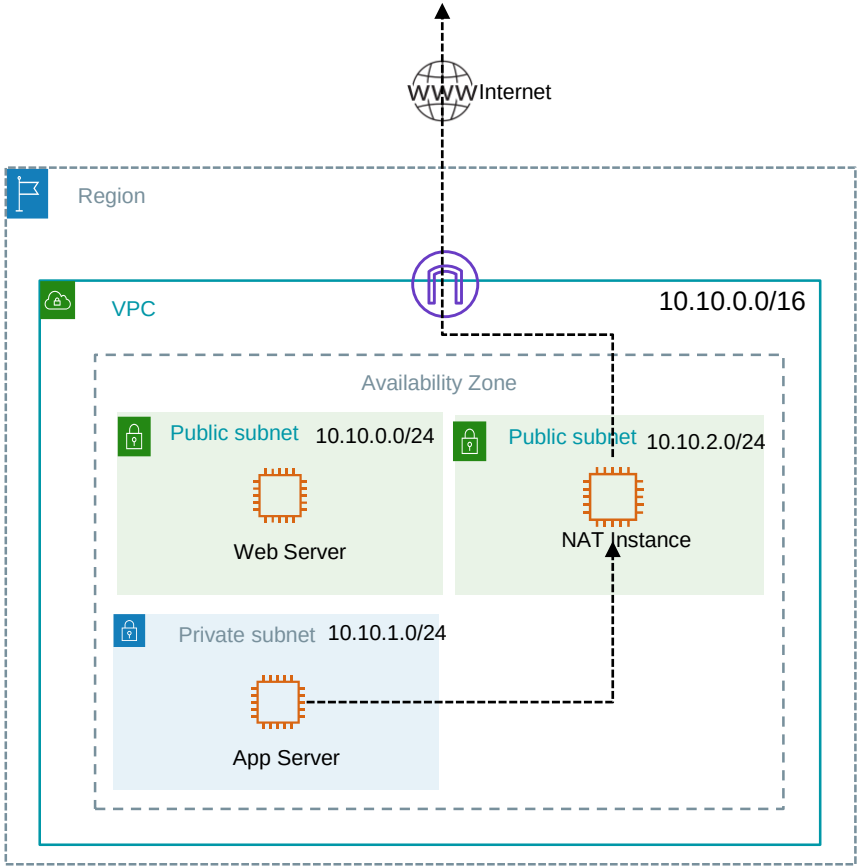


NAT Instance

NAT Instance

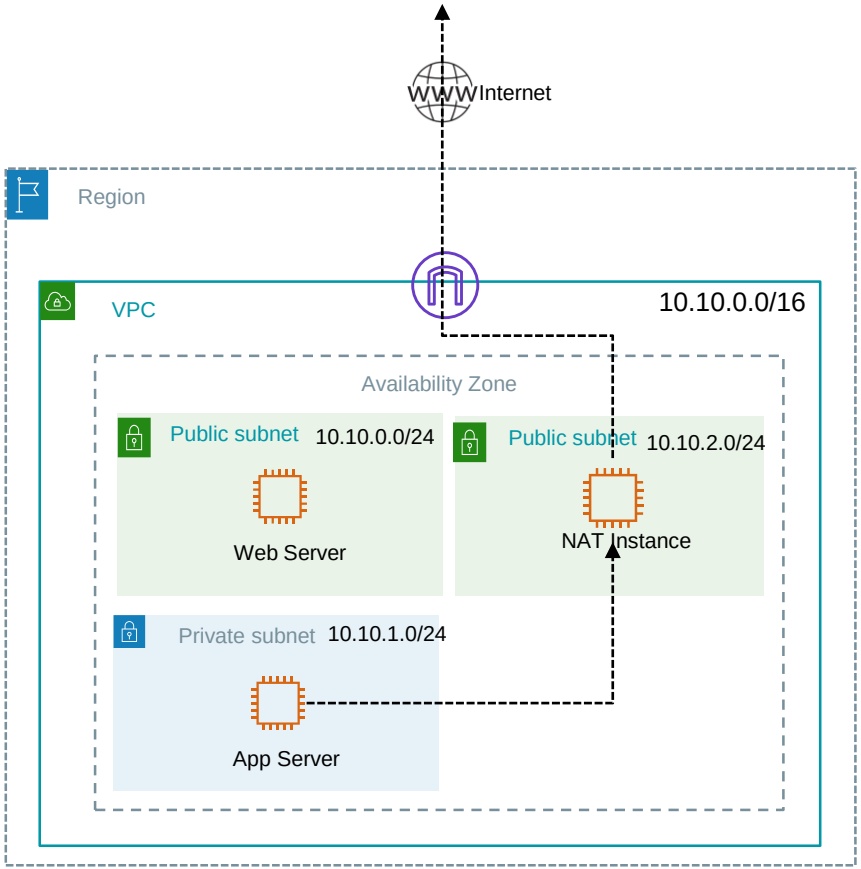
- Must be in a Public Subnet
- Must have Public or Elastic IP
- Should be launched using AWS provided NAT AMIs
- **Disable Source/Destination Check**

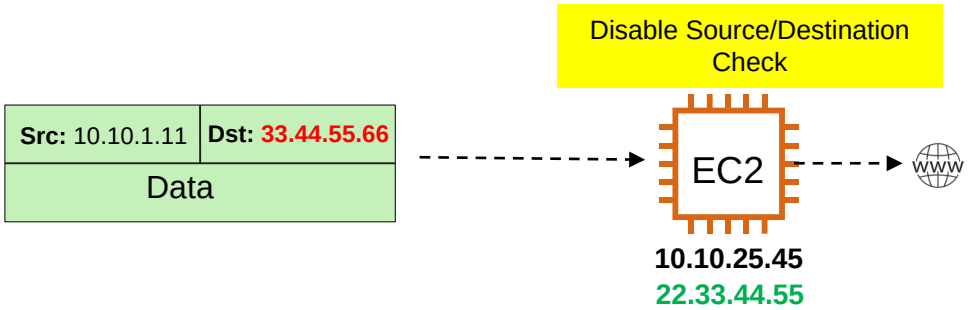




NAT Instance

- Must be in a Public Subnet
- Must have Public or Elastic IP
- Should be launched using AWS provided NAT AMIs
- **Disable Source/Destination Check**





NAT Instance

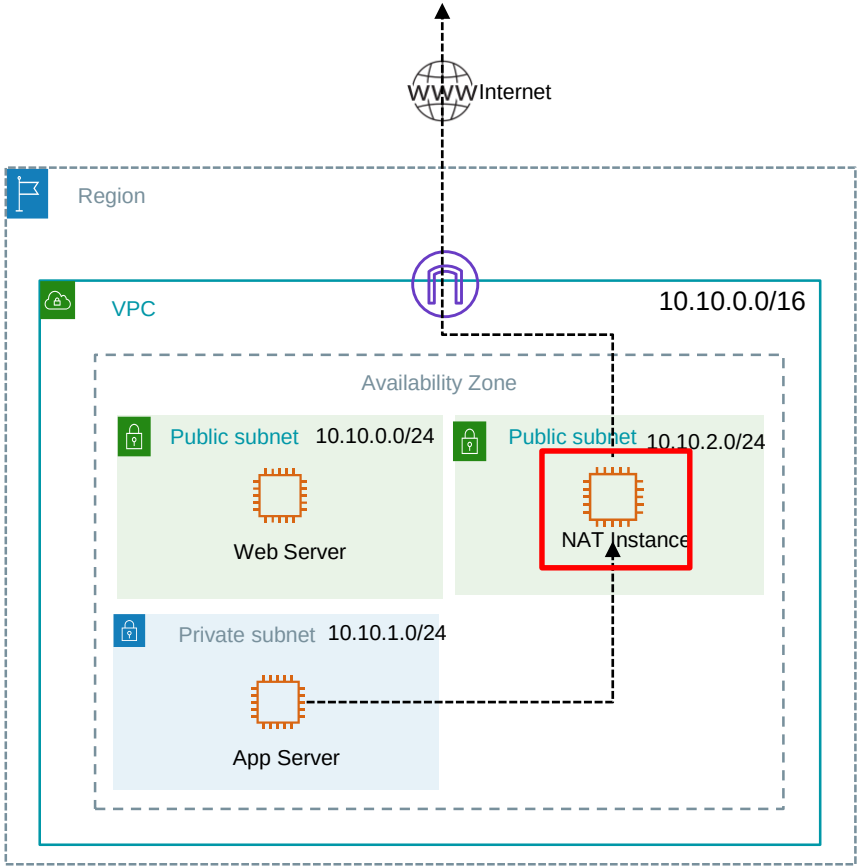
- Must be in a Public Subnet
- Must have Public or Elastic IP
- Should be launched using AWS provided NAT AMIs
- Disable Source/Destination Check
- **Security group inbound rules to allow http/s or ICMP (ping) traffic from Private subnets**

Inbound Rules

Protocol	Port	Source
HTTP	80	10.10.1.0/24
HTTPS	443	10.10.1.0/24
All ICMP - IPv4	All	10.10.1.0/24

Outbound Rules

Protocol	Port	Destination
All	All	0.0.0.0/0

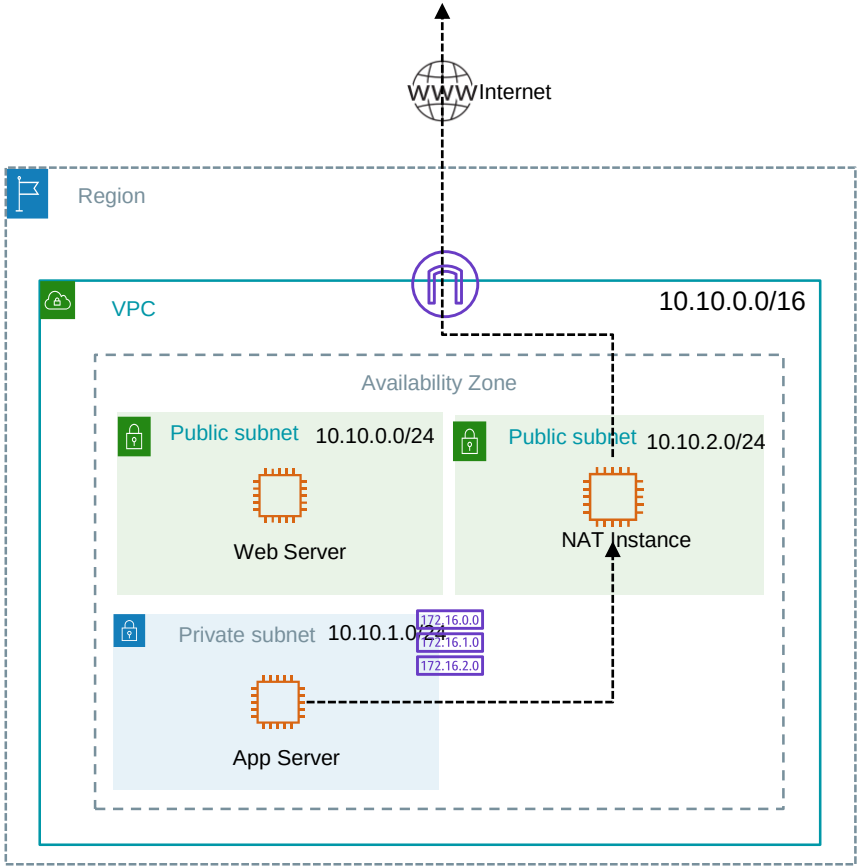


NAT Instance

- Must be in a Public Subnet
- Must have Public or Elastic IP
- Should be launched using AWS provided NAT AMIs
- Disable Source/Destination Check
- Security group inbound rules to allow http/s or ICMP traffic from Private subnet
- Update Private subnet route tables and route internet traffic through NAT EC2 instance

Private Subnet Route Table

Destination	Target
10.10.0.0/16	Local
0.0.0.0/0	**Nat Instance**

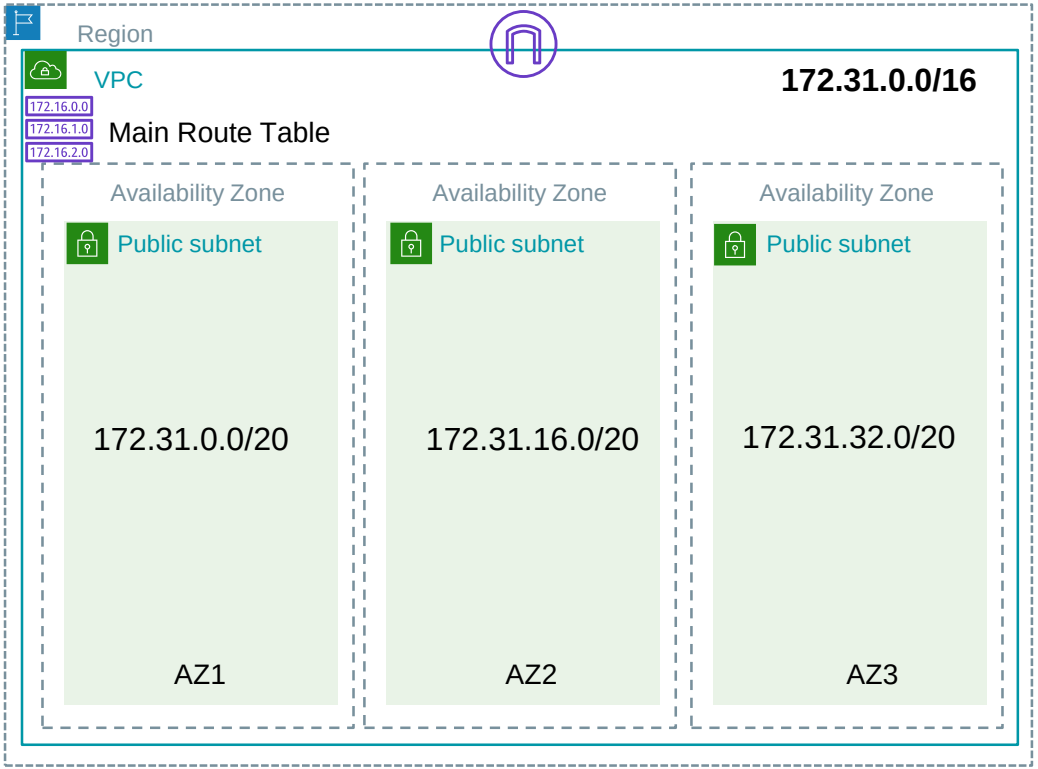


Default VPC

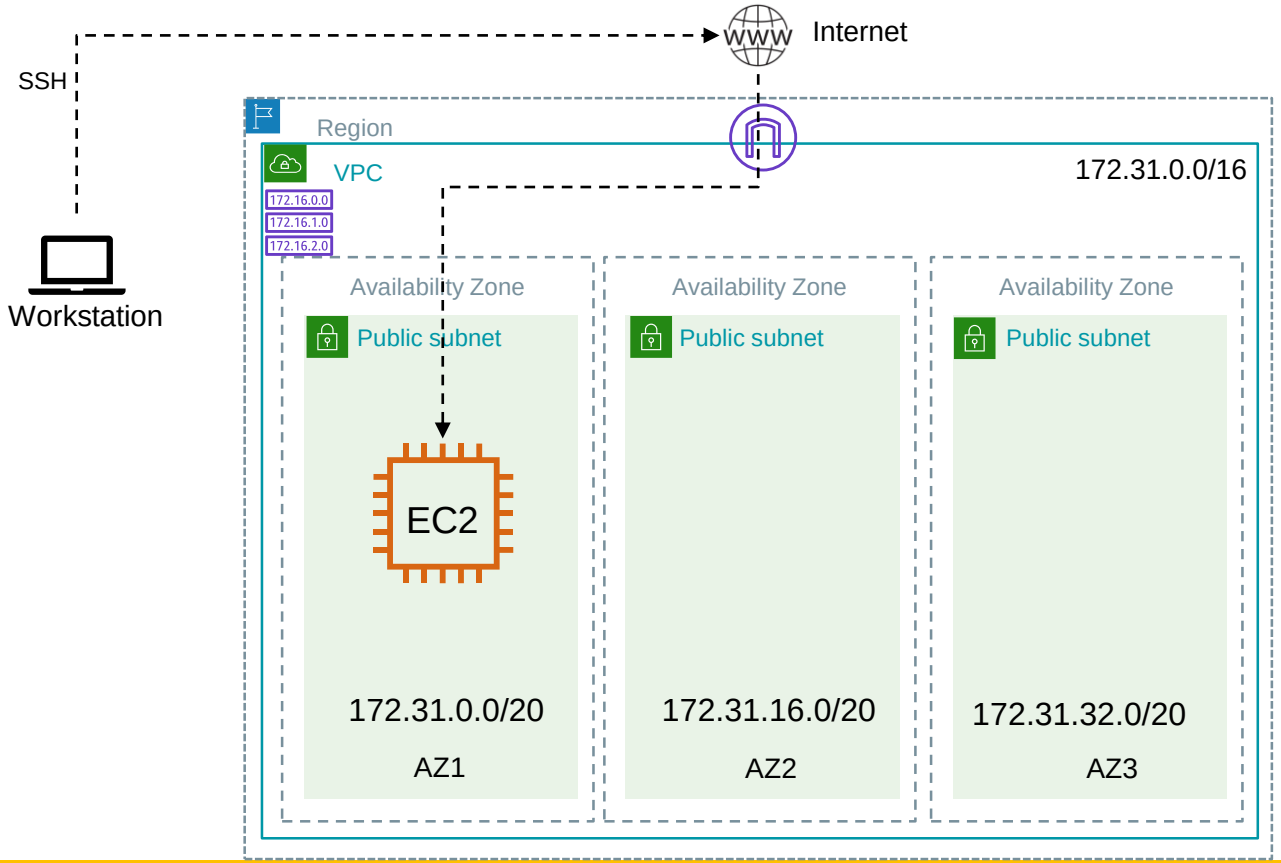
Default VPC

- 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
- If deleted, you can recreate default VPC

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



Demo - Launch a webserver in a default VPC



Main route table

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

Security Group – Inbound Rules

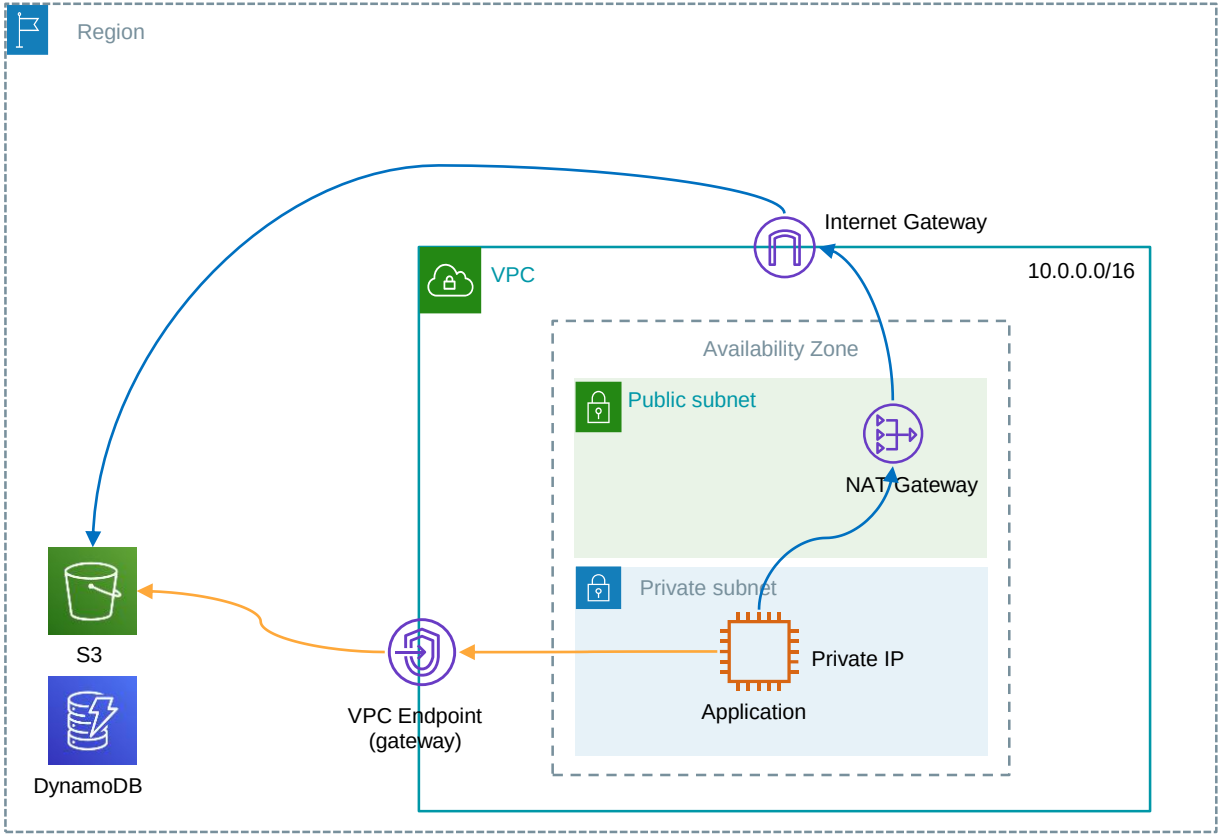
Protocol	Port	Source
HTTP	80	0.0.0.0/0
SSH	22	0.0.0.0/0

VPC Private Connectivity options

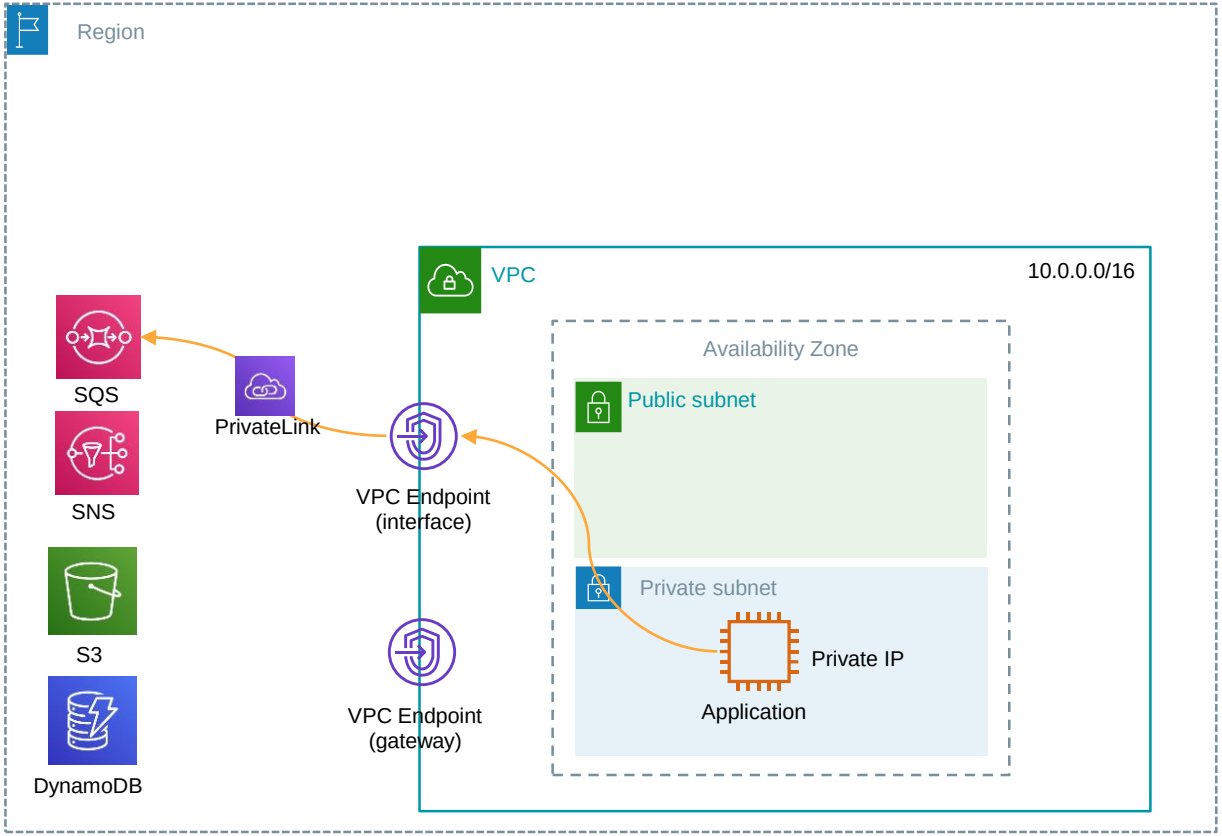
VPC connectivity options

- VPC Endpoints & PrivateLink
- VPC Peering connection
- Transit Gateway
- Site-2-Site VPN
- Client VPN
- Direct Connect

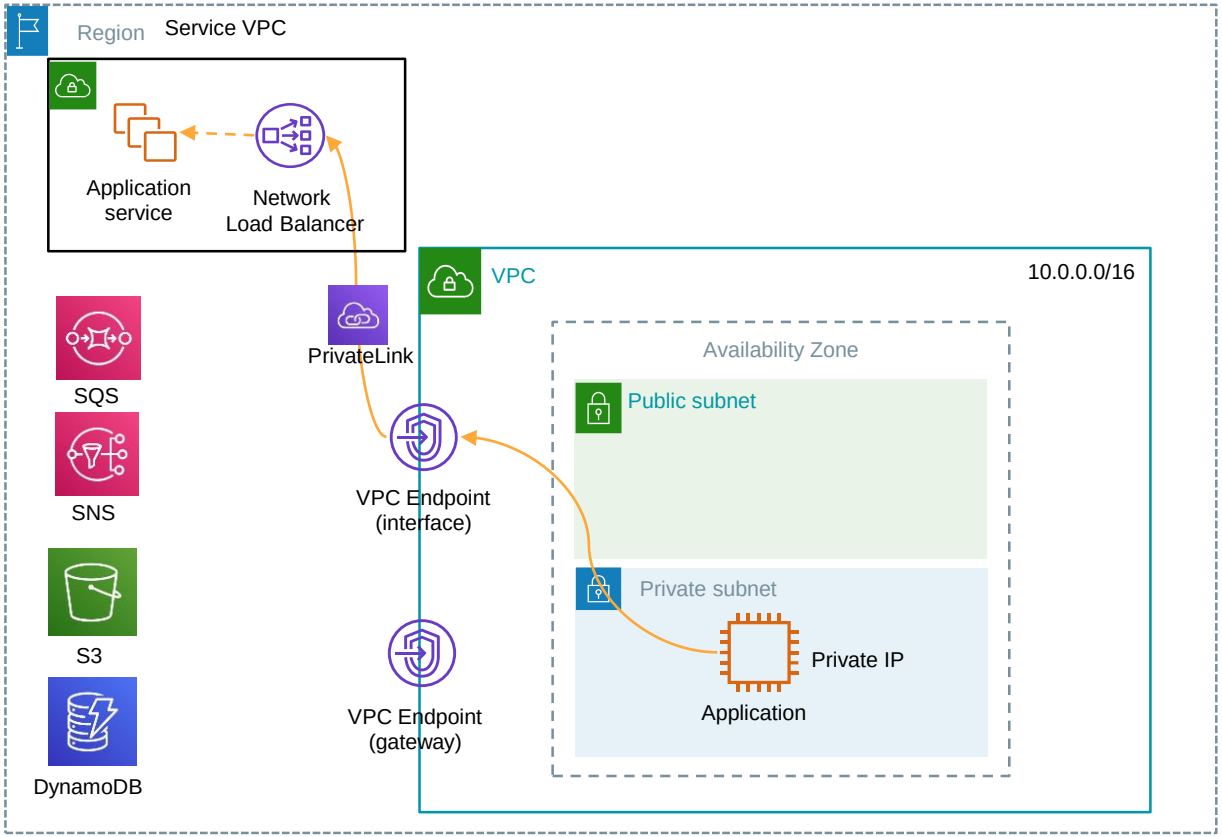
VPC Endpoints & Privatelink



VPC Endpoints & Privatelink



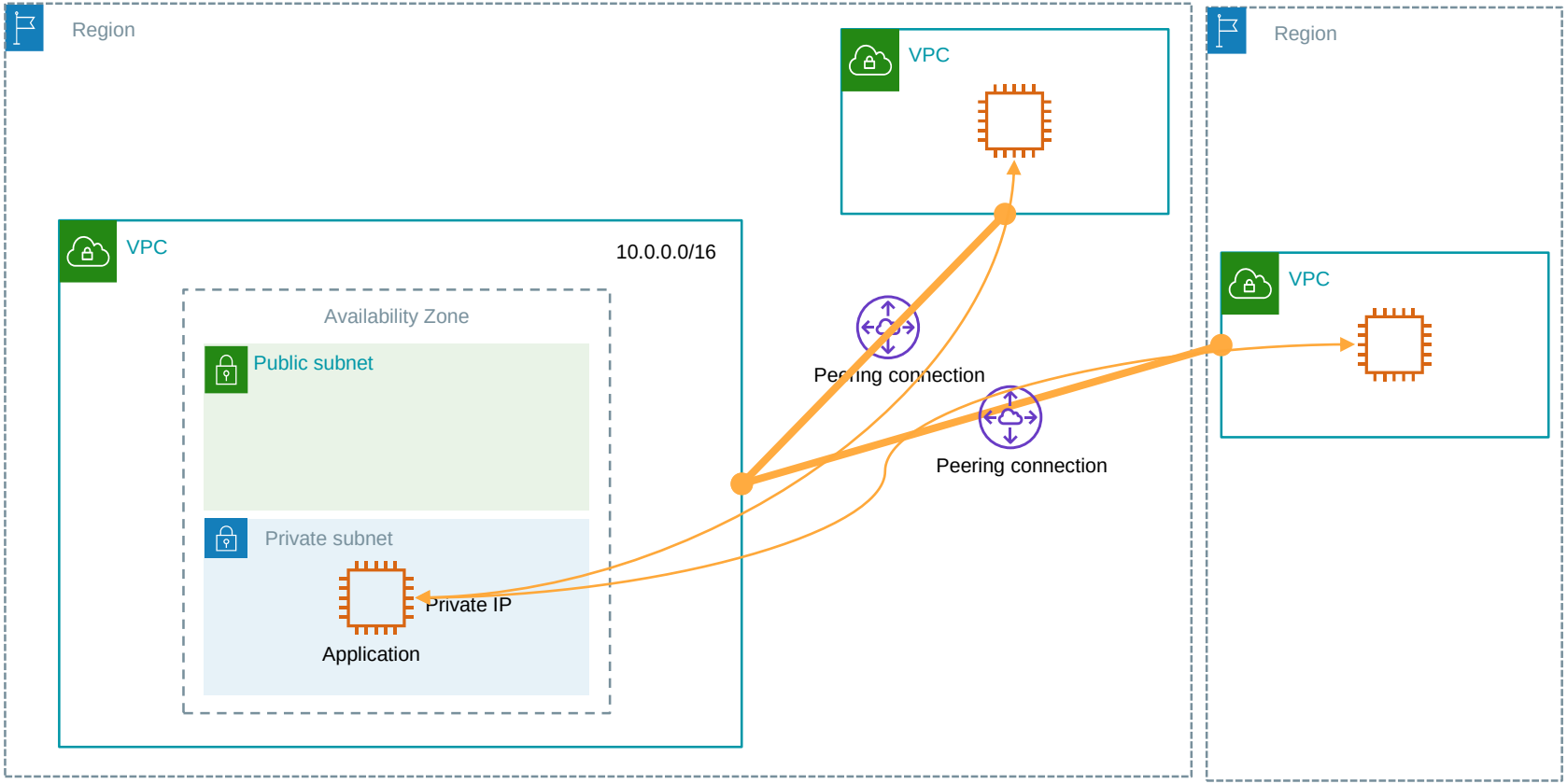
VPC Endpoints & Privatelink



VPC connectivity options

- VPC Endpoints & PrivateLink
- **VPC Peering connection**
- Transit Gateway
- Site-2-Site VPN
- Client VPN
- Direct Connect

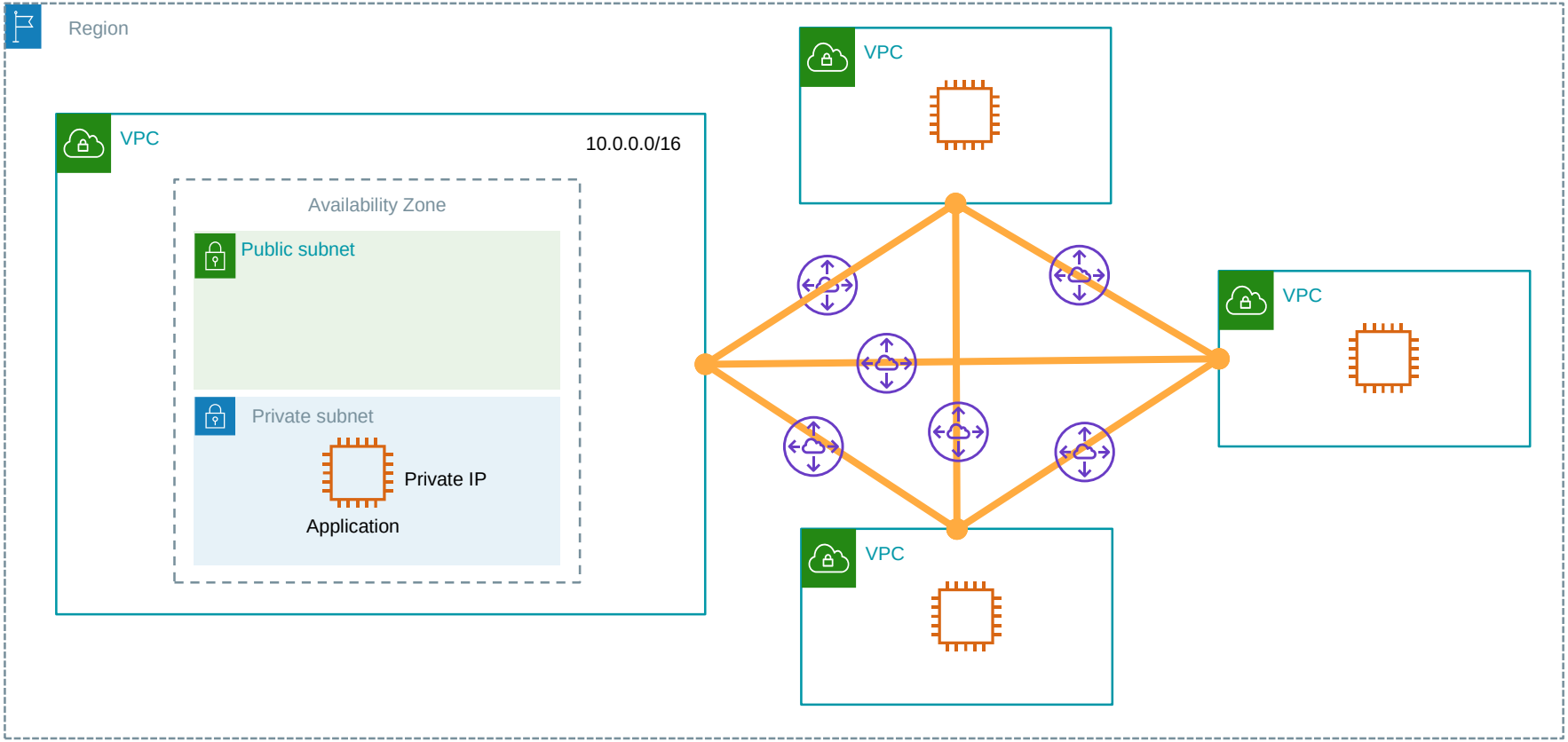
VPC Peering



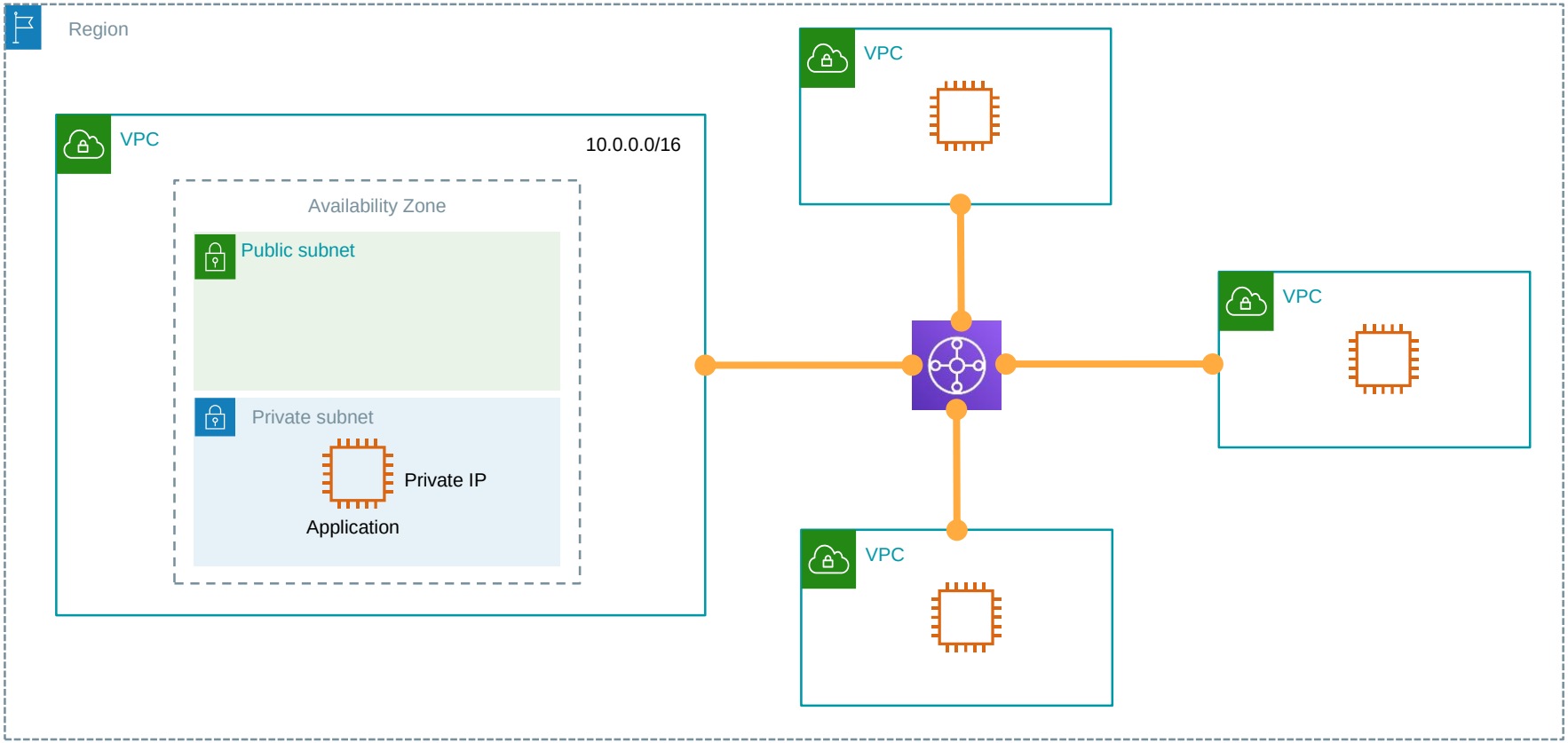
VPC connectivity options

- VPC Endpoints & PrivateLink
- VPC Peering connection
- **Transit Gateway**
- Site-2-Site VPN
- Client VPN
- Direct Connect

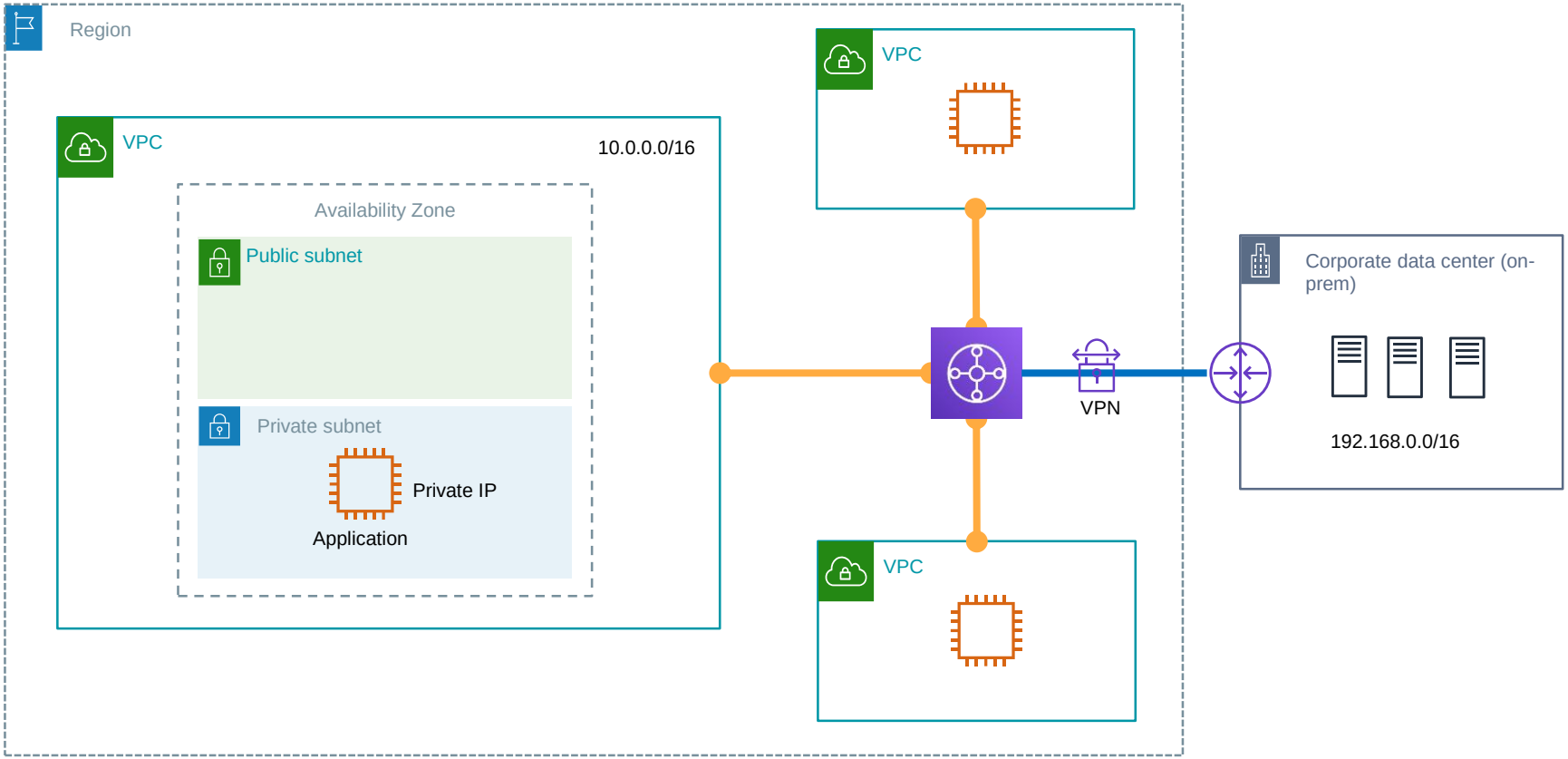
Transit Gateway



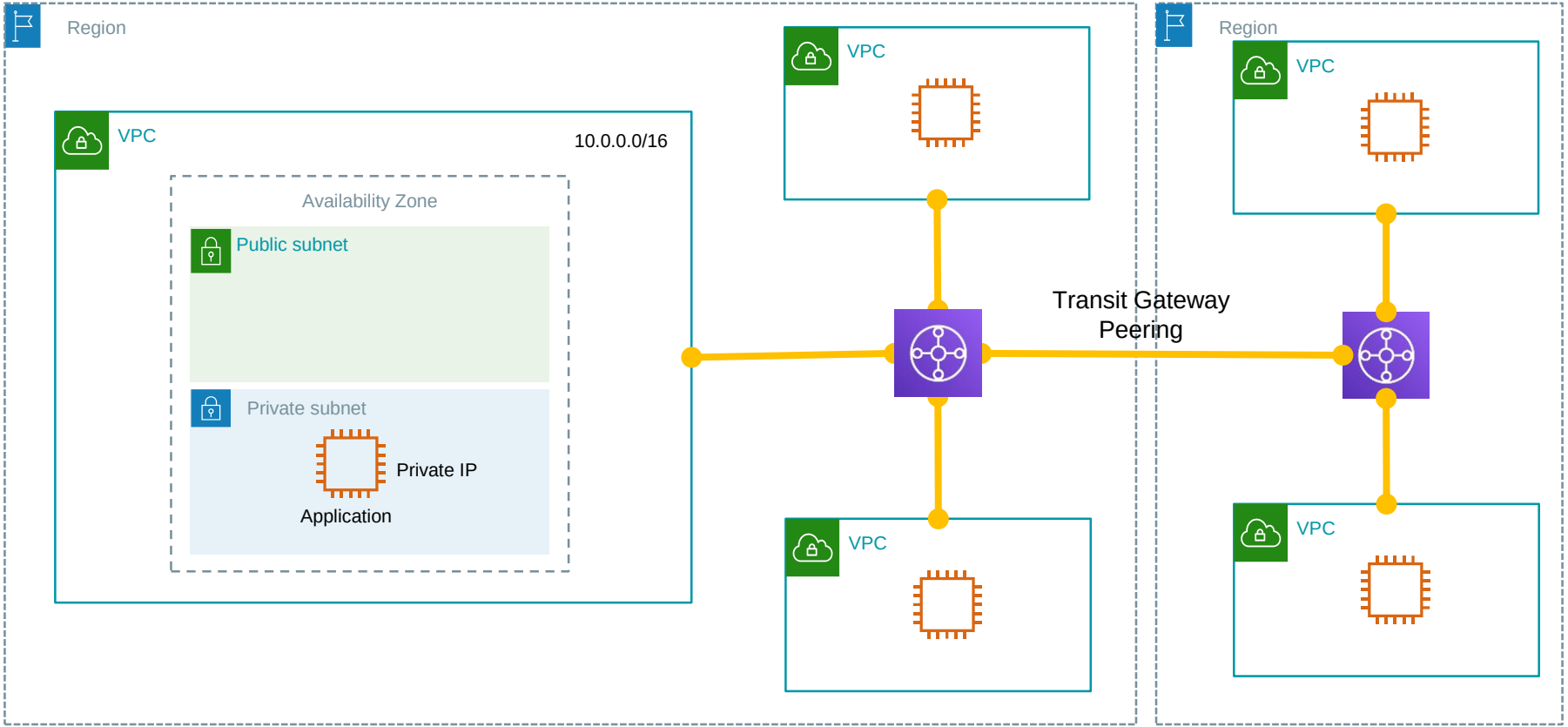
Transit Gateway



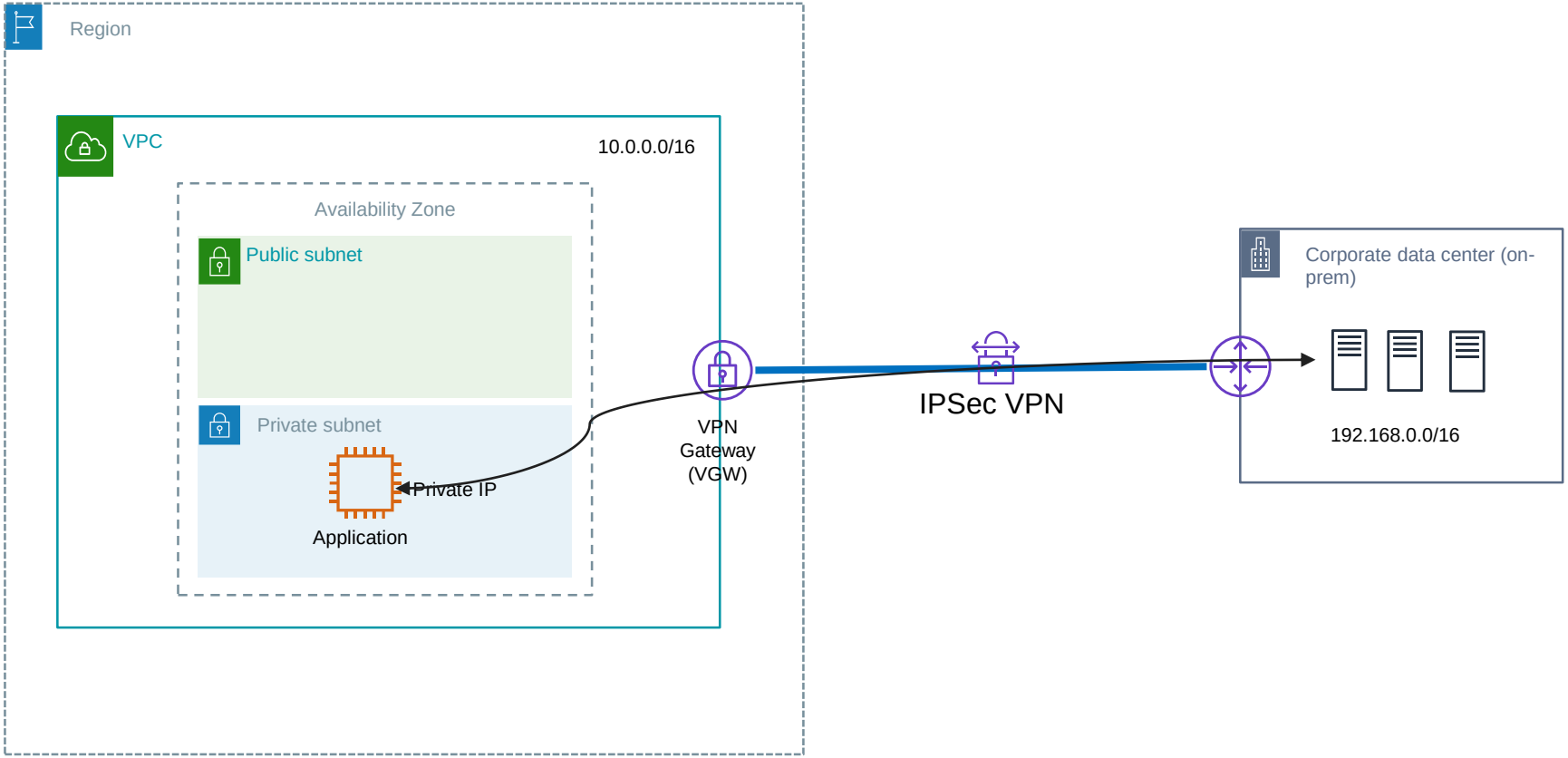
Transit Gateway with VPN attachment



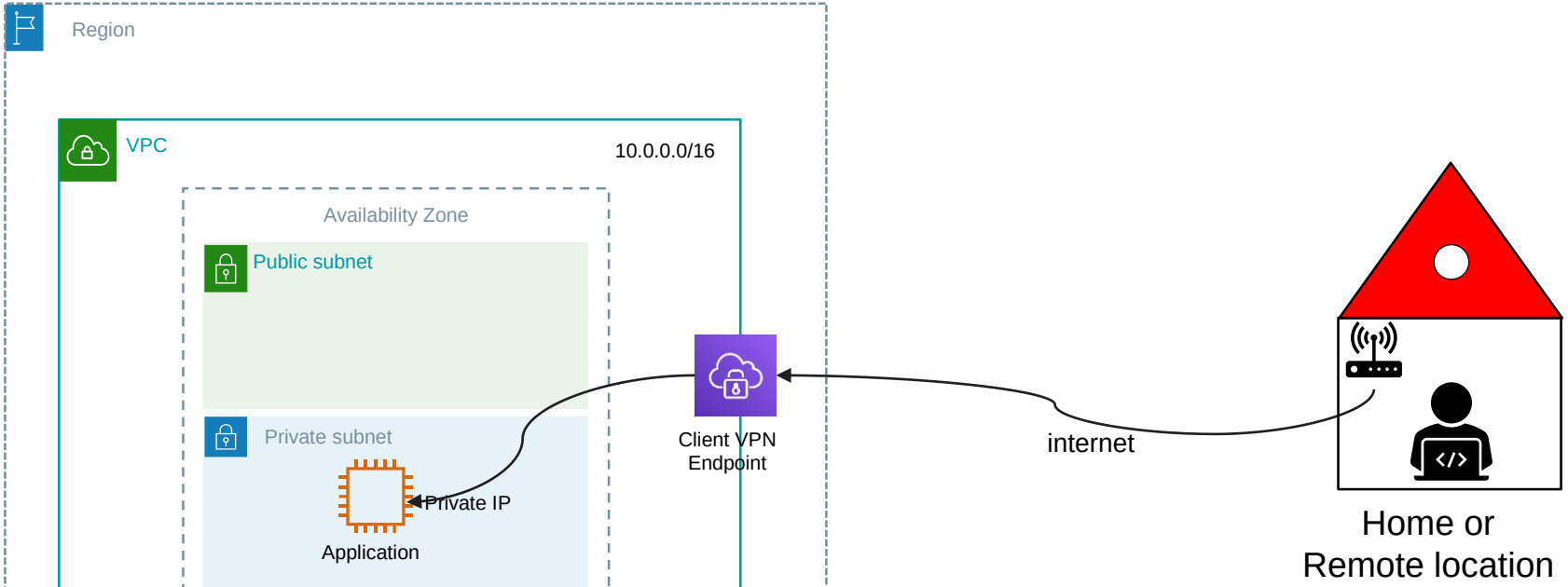
Transit Gateway peering



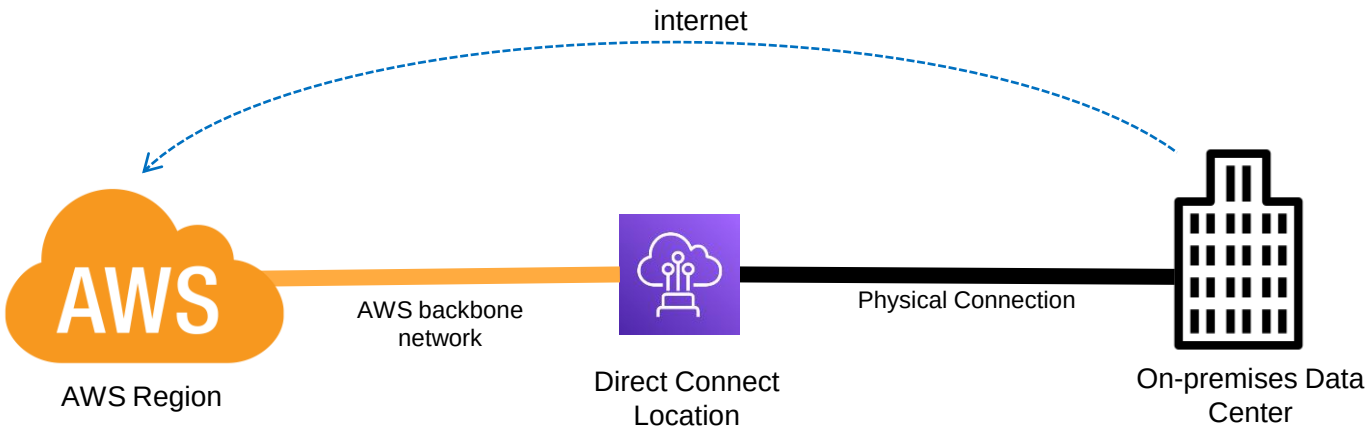
Site to Site VPN



Client to Site VPN

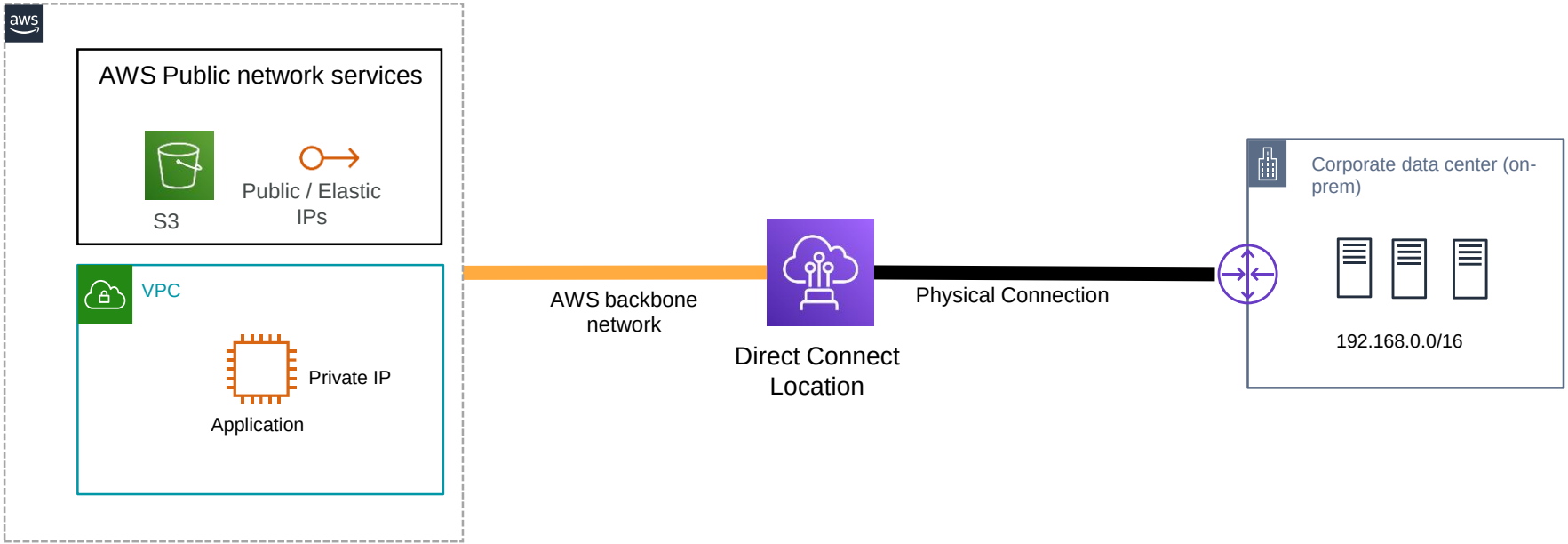


Direct Connect (DX)

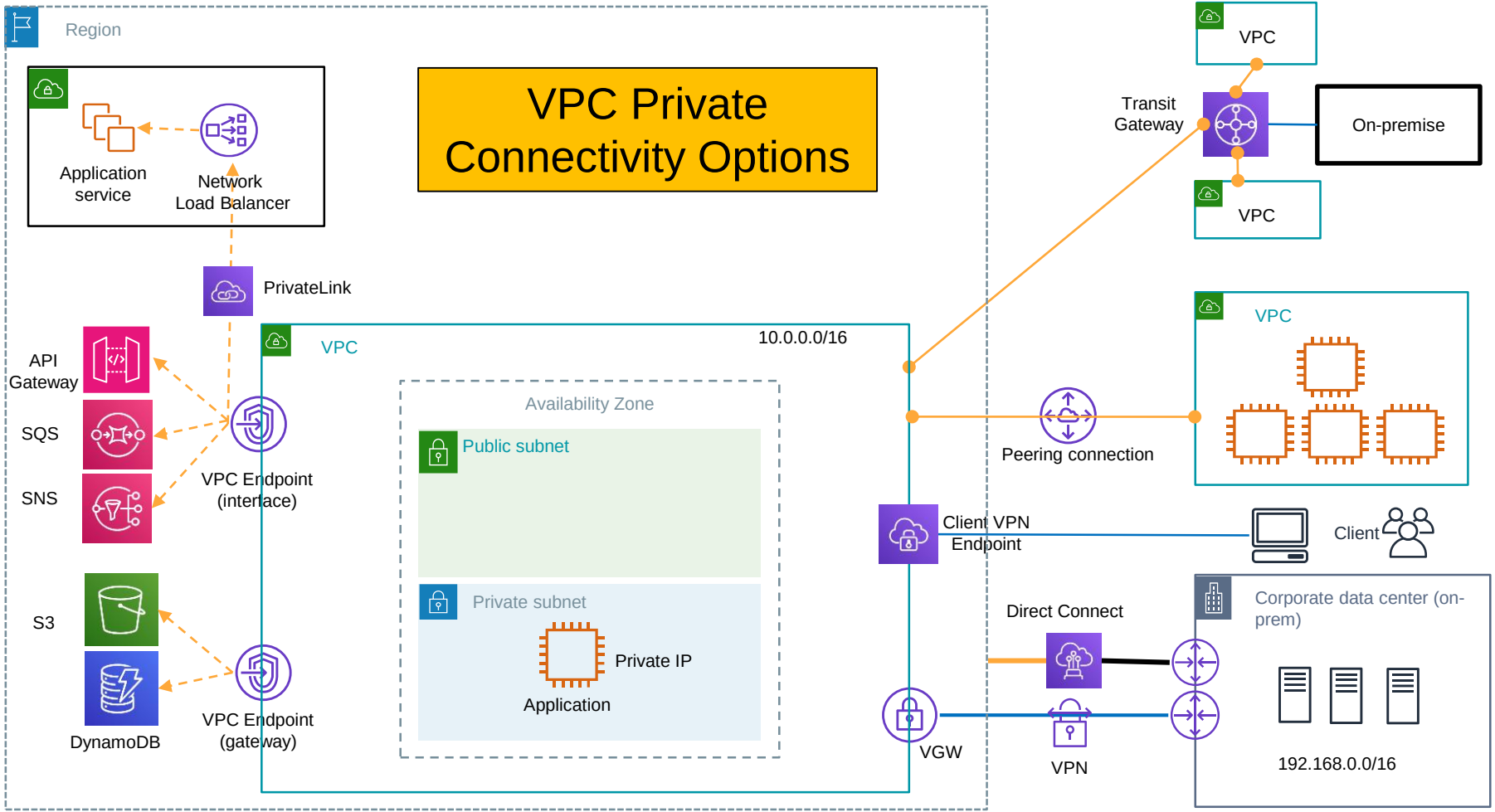


- Provides dedicated and consistent network
- Provides network bandwidth from 50 Mbps up to 100 Gbps over a single connection
- Low data transfer cost

Direct Connect

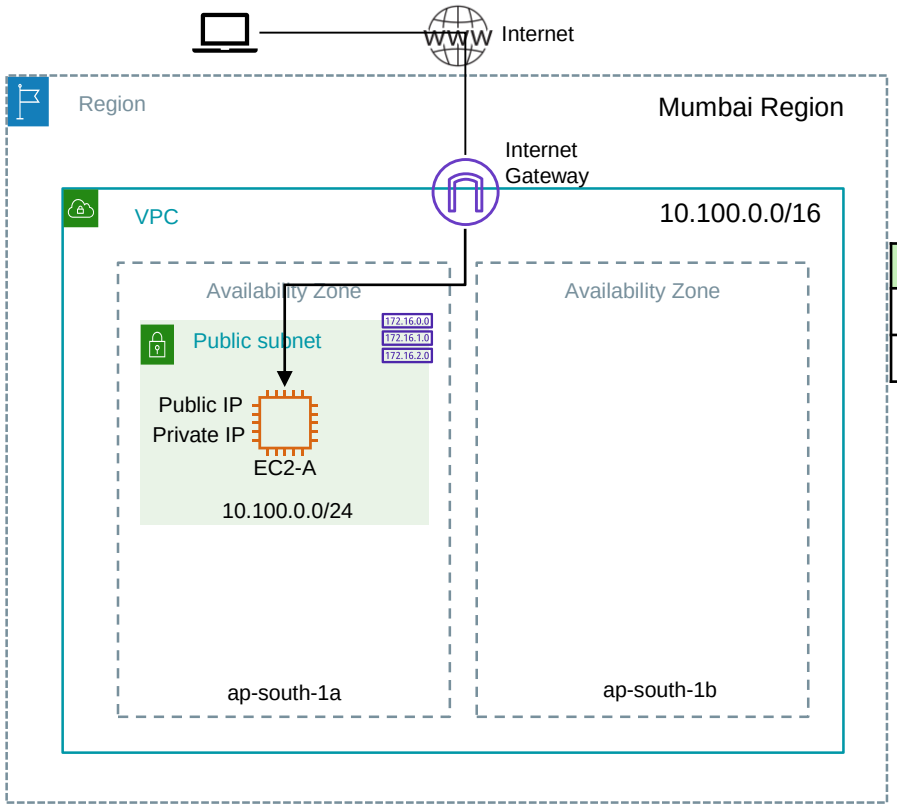


VPC Private Connectivity Options



Exercise – VPC with a single Public subnet

Exercise – Public Subnet



Public Subnet Route table

Destination	Target
10.100.0.0/16	Local
0.0.0.0/0	igw-xxxxxx

High level steps

1. Create a new VPC
2. Create an Internet Gateway & associate with your VPC
3. Create a Subnet in one of the availability zone. Enable Auto-assign Public IP for a subnet.
4. Create a Route table and add a route for destination (0.0.0.0/0) with target as an internet gateway
5. Associate route table with your subnet
6. Launch EC2 instance in your subnet.
7. Connect to EC2 instance over SSH using it's Public IP

Steps

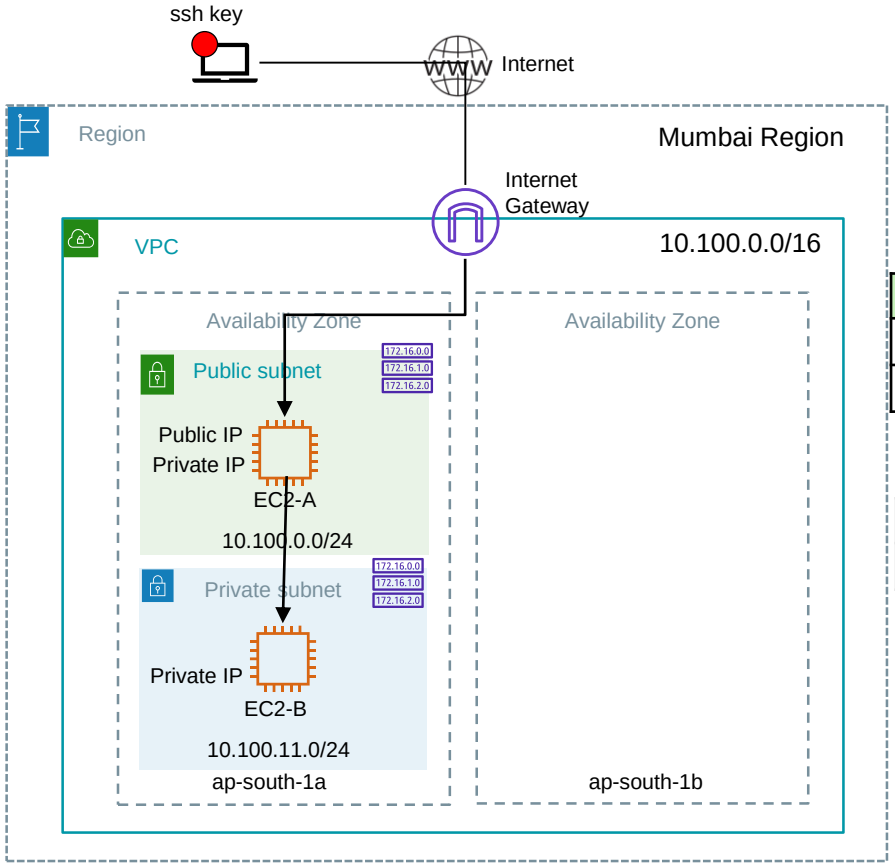
1. Create VPC
   a. AWS Console -> Go to VPC service -> Your VPCs -> Create VPC (Resources to create : VPC Only, Name tag: VPC-A, IPv4 CIDR block: Select IPv4 CIDR manual input (10.100.0.0/16) , Tenancy : Default - > Create VPC
2. Create Internet Gateway
   a. Internet Gateways -> Create internet gateway (Name tag: VPC-A-IGW) -> Create internet gateway
   b. Select Internet gateway -> Actions -> Attach to VPC -> Select your VPC (VPC-A) -> Attach Internet Gateway
3. Create Subnet
   a. Subnets -> Create subnet
   b. Select VPC ID: VPC-A
   c. Subnet 1 of 1 -> Subnet Name: VPC-A-Public, AZ: Select AZ 1, IPv4 CIDR block : 10.100.0.0/24) -> Create Subnet
   d. Select Subnet -> Actions -> Edit Subnet Settings - > Modify Auto-Assign IP Settings-> Enable -> Save
4. Create Route table
   a. Route Tables -> Create Route Table (Name: VPC-A-Public-RT, select VPC: VPC-A) -> Create route table
   b. Select Route table -> Routes -> Edit routes -> Add another route (Destination: 0.0.0.0/0, Target: Internet gateway -> igw-xxxxx) -> Save changes
5. Associate route table with the subnet
   a. Select Route table -> Subnet Associations -> Edit subnet associations -> Check the VPC-A-Public subnet -> Save associations

Steps

6. Launch EC2 instance in newly created Public Subnet
    a. Go to EC2 Service -> EC2 Dashboard -> Launch Instances
    b. Name: EC2-A
    c. Select Application and OS Images (Amazon Machine Image): Amazon Linux (default)
    d. Select instance type: t2.micro (default)
    e. Select key pair : *Your key-pair that you had created earlier in pre-requisites*
    f. Network settings -> Edit -> Select your VPC (VPC-A) and your public subnet
    g. Make sure Auto-Assign Public IP is enabled
    h. Firewall -> Create security group
        a. Name: EC2-A-SG
        b. Inbound Security group rule: Add rule (Type-> SSH, port Range-> 22, source type -> My IP
    i. Configure Storage -> 8GiB, gp3 (default)
    j. Launch Instance
7. Connect to EC2 instance with *Public IP* from your workstation using Putty or terminal with user *ec2-user*

Note: If you don't plan to continue with the next exercises at this moment, then terminate ec2 instance(s) and optionally delete the VPC. Otherwise you can continue with this setup until you finish first 4 exercises and then terminate/delete everything.

Exercise – Private Subnet



Public Subnet Route table

Destination	Target
10.100.0.0/16	Local
0.0.0.0/0	igw-xxxxxx

Private Subnet Route table

Destination	Target
10.100.0.0/16	Local

Continuing with earlier setup

1. Create a new subnet in Availability zone 1 (as shown)
2. Create a new route tables and associate with Private subnet. (route entries as shown)
3. Launch EC2 instance (EC2-B) in the Private subnet. Make sure Security Group for EC2-B allows SSH and ICMP (ping) from VPC CIDR.
4. Connect to EC2-A over SSH. Ping to EC2-B Private IP.
5. Bring/copy SSH key onto EC2-A and change file permissions to 400
6. From EC2-A terminal SSH into EC2-B using ssh key file
7. Once logged into EC2-B, try to ping google.com or wget google.com

Steps

1. Create Private subnet
    a. Subnets -> Create subnet, Select VPC ID: VPC-A
    b. Subnet 1 of 1 -> Subnet Name: VPC-A-Private, AZ: Select AZ 1, IPv4 CIDR: 10.100.11.0/24) -> Create Subnet
2. Create a Route table for Private subnet
    a. Route Tables -> Create Route Table (Name: VPC-A-Private-RT, select VPC: VPC-A) -> Create route table
    b. Select Route table -> Subnet Associations -> Edit subnet associations -> Check the VPC-A-Private subnet -> Save associations
3. Launch EC2-B instance in the Private Subnet
    a. Go to EC2 Service -> EC2 Dashboard -> Launch Instances
    b. Name: EC2-B
    c. Select AMI: Amazon Linux (default)
    d. Select instance type: t2.micro (default)
    e. Select key pair : *Your key-pair that you had created earlier*
    f. Network settings -> Edit -> Select your VPC (VPC-A) and Private subnet (VPC-B-Private)
    g. Firewall -> Create security group
        a. Security Group Name: EC2-B-SG
        b. Inbound Security group rule: Add rule for SSH (port 22) for source type (Custom) Source as 10.100.0.0/16
        c. Add security group rule
        d. Inbound Security group rule: Add rule for ICMP IPv4 for source as 10.100.0.0/16 (Type: All ICMP –IPV4, Source type- Custom as 10.100.0.0/16
    h. Configure Storage -> 8GiB, gp3 (default)
    i. Launch Instance and wait for the instance to be in running state

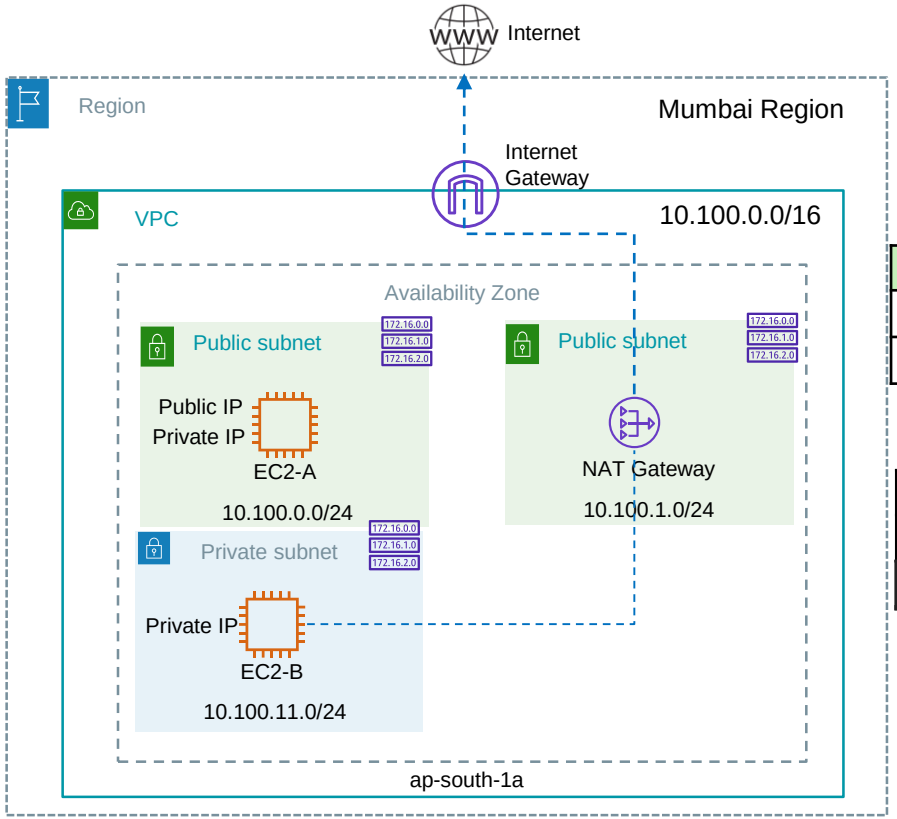
Steps

4. From EC2-A instance, ping to EC2-B private IP using command:
   $ping 10.100.11.x
5. Create a key.pem on EC2-A using any editor. Paste .pem file content and save the file. Change file permissions to 400 using command:
   $chmod 400 key.pem
6. SSH to EC2-B using command:
   $ssh –i key.pem ec2-user@10.100.11.x
7. Try to access the internet using following commands:
   $ping google.com
   $wget https://google.com

   [Above commands should not work, why? There is no outbound internet connectivity to the Private subnet]

Note: If you don't plan to continue with the next exercises at this moment, then terminate ec2 instance(s) and optionally delete the VPC. Otherwise you can continue with this setup until you finish first 6 exercises and then terminate/delete everything.

Exercise – NAT Gateway



Public Subnet Route table

Destination	Target
10.100.0.0/16	Local
0.0.0.0/0	igw-xxxxxx

Private Subnet Route table

Destination	Target
10.100.0.0/16	Local
0.0.0.0/0	Nat-gateway

Continuing from the earlier setup

1. Create a new Public subnet in Availability zone 1 (as shown)
2. Associate an existing Public route table with this subnet
3. Create a NAT Gateway in this Public subnet
4. Update the Private subnet route table and add route entry for destination 0.0.0.0/0 with target as nat gateway
5. While logged into EC2-B, try to access internet

Steps

1. Create a new Public subnet
    a. Subnets -> Create subnet
    b. Select VPC ID: VPC-A
    c. Subnet 1 of 1 -> Subnet Name: VPC-A-Public-NAT, AZ: Select AZ 1, IPv4 CIDR: 10.100.1.0/24)
    d. Save
2. Associate existing Public route table
    a. Select Public Route table (VPC-A-Public-RT) -> Subnet Associations -> Edit subnet associations -> Check the VPC-A-Public-NAT subnet -> Save associations
3. Create a nat gateway
    a. VPC console -> NAT Gateways -> Create NAT Gateway
    b. Name: VPC-A-NATGW
    c. Subnet: VPC-A-Public-NAT
    d. Connectivity Type: Public
    e. Elastic IP -> Allocate Elastic IP
    f. Create NAT gateway
    g. Wait until NAT gateway is available/ready.
4. Update Private subnet route table
    1. Route tables -> Select Private subnet (VPC-A-Private-RT) -> Routes
    2. Edit routes -> Add another route (Destination: 0.0.0.0/0, Target: nat gateway -> nat-xxxxx) -> Save changes

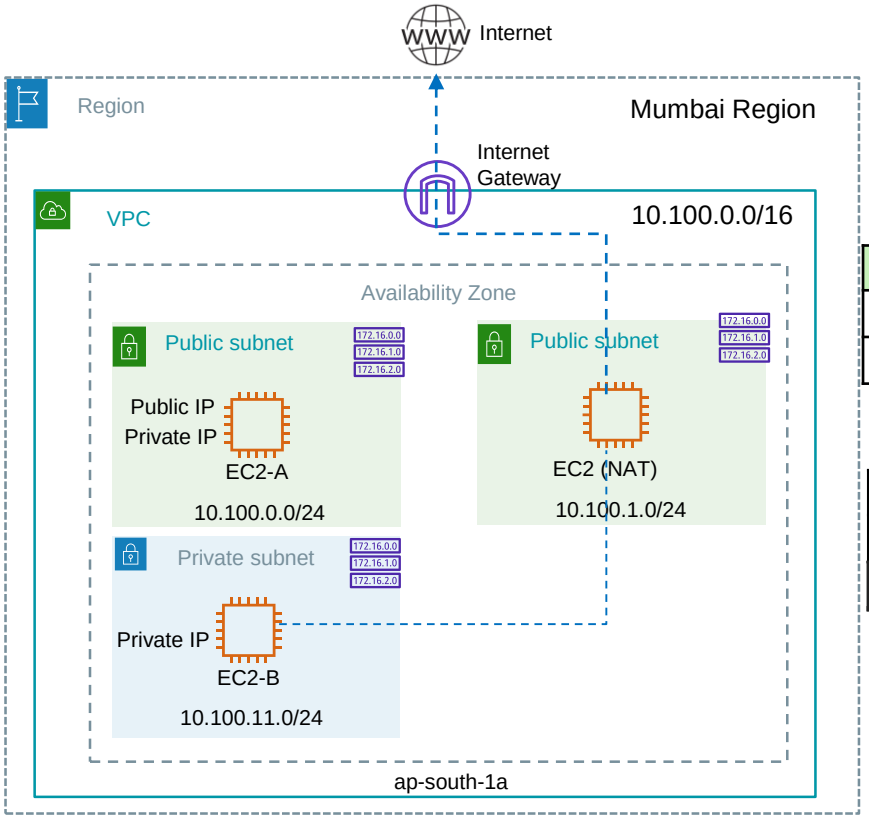
Steps

5. Log into EC2-B (SSH) and try to access internet using following commands:
   $ping google.com
   $wget https://google.com

[Above commands should work, why? There is now outbound internet connectivity via the NAT Gateway]

Note: If you don't plan to continue with the next exercises at this moment, then terminate ec2 instance(s), delete NAT gateway, release Elastic IP and optionally delete the VPC. Otherwise you can continue with this setup until you finish first 4 exercises and then terminate/delete everything.

Exercise – NAT Instance



Public Subnet Route table

Destination	Target
10.100.0.0/16	Local
0.0.0.0/0	igw-xxxxxx

Private Subnet Route table

Destination	Target
10.100.0.0/16	Local
0.0.0.0/0	**eni-xxxxxx**

Continuing from the earlier setup

1. Delete NAT gateway and release Elastic IP & remove route entry from Private route table
2. Launch EC2 instance in the NAT Public subnet using nat AMI (amzn-ami-vpc-nat-xxxx). Make sure EC2 instance gets Public IP and Security group to allow port 80 & 443 inbound traffic for VPC CIDR source.
3. Disable source/destination check for NAT EC2 instance
4. Update the Private subnet route table and modify route entry for destination 0.0.0.0/0 with target as NAT EC2 ENI
5. While logged into EC2-B, try to access internet

Steps

1. Delete NAT Gateway and release EIP
    a. NAT gateways -> Select your NAT gateway -> Actions -> Delete NAT Gateway
    b. Wait for NAT gateway to be deleted
    c. Elastic Ips -> Select your Elastic IP -> Actions -> Release Elastic IP addresses
2. Update private subnet route table
    a. Go to EC2 Service -> EC2 Dashboard -> Launch Instances
    b. Name: EC2-NAT
    c. Select AMI: amzn-ami-vpc-nat-xxxxx
    d. Select instance type: t2.micro (default)
    e. Select key pair : *Your key-pair that you had created earlier*
    f. Network settings -> Edit -> Select your VPC (VPC-A) and Public subnet (Public-Subnet-NAT)
    g. Make sure Auto-Assign Public IP is enabled
    h. Firewall -> Create security group
        a. Name: EC2-NAT-SG
        b. Inbound Security group rule: Add rule for HTTP (80) for source as 10.100.0.0/16
        c. Inbound Security group rule: Add rule for HTTPS (443) for source as 10.100.0.0/16
        d. Inbound Security group rule: Add rule for ICMP IPv4 for source as 10.100.0.0/16
    i. Configure Storage -> 8GiB, gp3 (default)
    j. Launch Instance and wait for the instance to be in running state

Steps

3. Disable source/destination check for EC2 NAT instance
    a. Select EC2-NAT instance -> Actions -> Networking -> Change source/destination check
    b. enable Stop -> Save
4. Update Private subnet route table
    a. Route tables -> Select Private subnet (VPC-A-Private-RT) -> Routes
    b. If you are continuing from the previous exercise, then you should see a Blackhole route because you have deleted NAT Gateway but did not remove this route. In that case, Edit routes and remove this rule -> Save.
    c. Edit routes -> Add new route (Destination: 0.0.0.0/0, Target: EC2 NAT instance -> Save. This should add EC2 instance ENI as target for this route (eni-xxxxxx)
5. Log into EC2-B (SSH) and try to access internet using following commands:

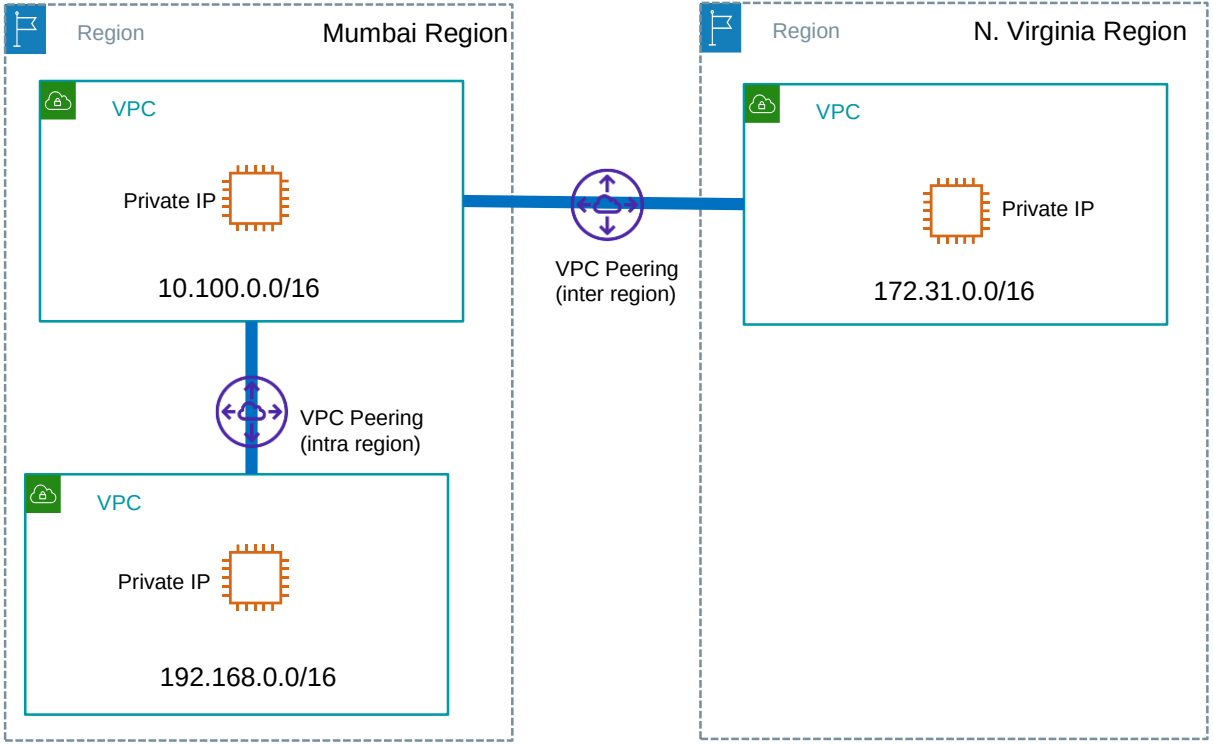
    $ping google.com
    $wget https://google.com

[Above commands should work, why? There is now outbound internet connectivity via the NAT EC2 instance]

Clean-up

a. If you are **not** continuing with next exercises
    a. Terminate all three EC2 instances
    b. Delete VPC-A (there is no adding cost even if you have VPC-A and subnets in your account)
b. If you want to continue with the next exercise
    a. Terminate EC2-NAT instance
    b. Update VPC-A-Private-RT and remove the route you added for 0.0.0.0/0 via the EC2 instance.
    c. Continue with next exercise

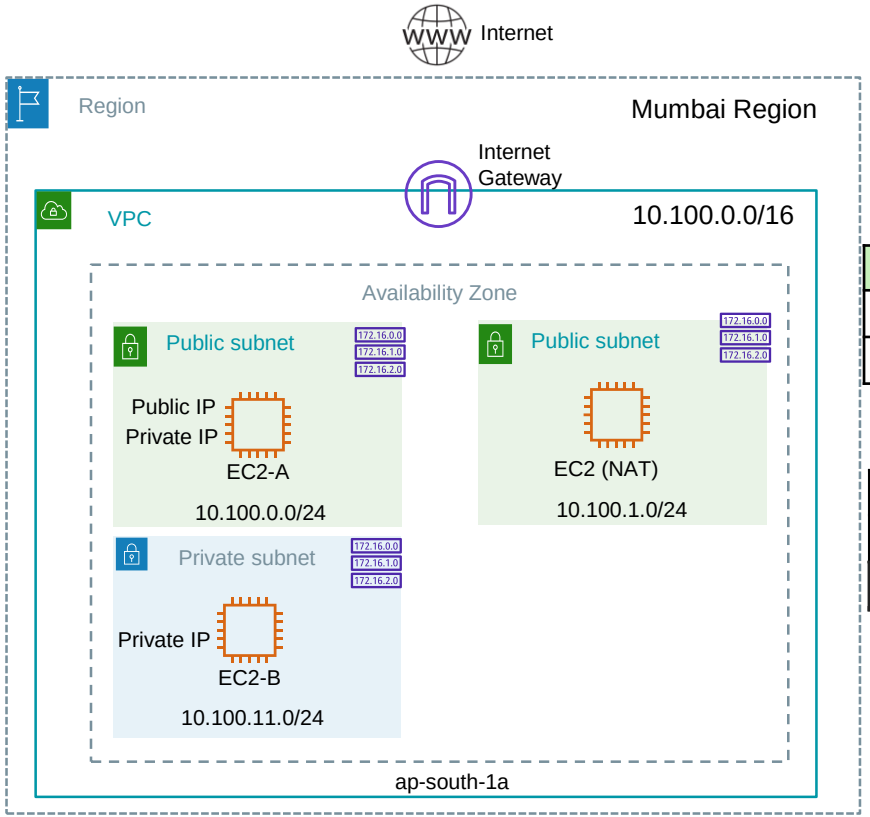
Exercise – VPC Peering



Exercise – VPC Peering

If you are continuing from the earlier setup

1. Delete EC2 NAT instance and release Elastic IP & remove route entry from Private route table



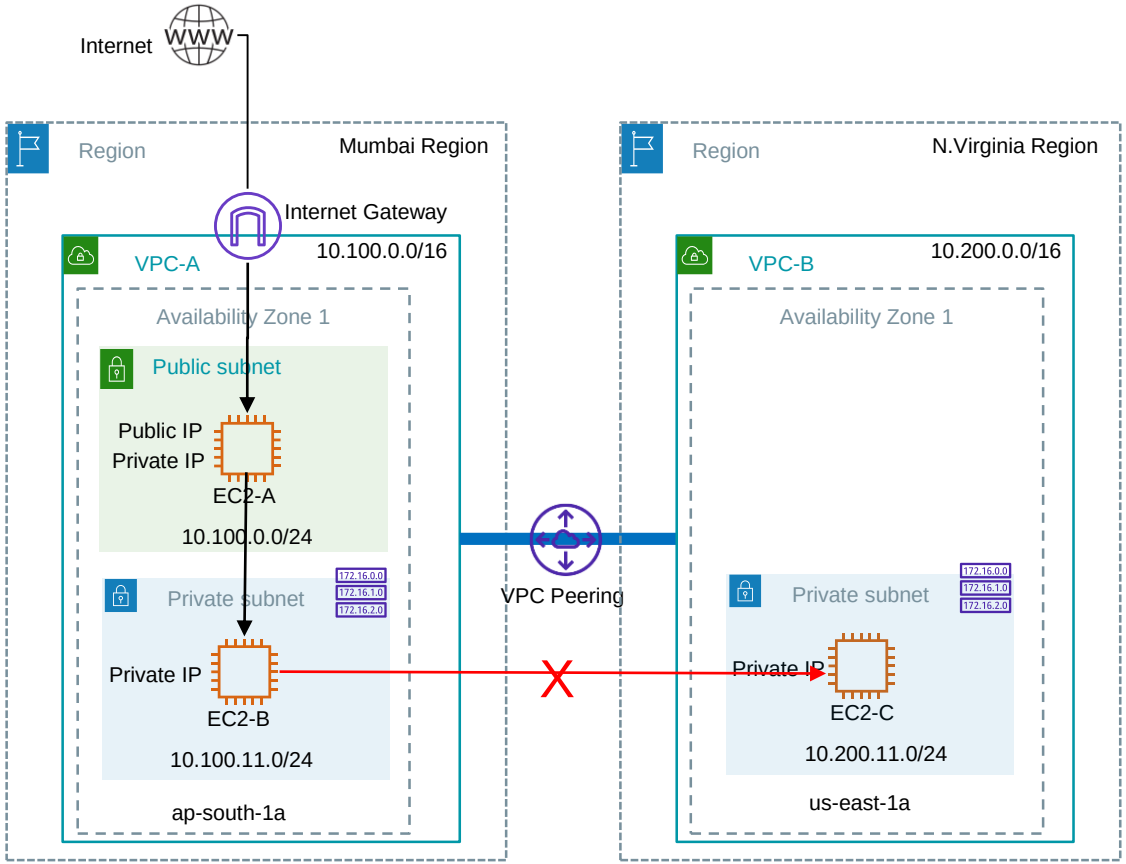
Public Subnet Route table

Destination	Target
10.100.0.0/16	Local
0.0.0.0/0	igw-xxxxxx

Private Subnet Route table

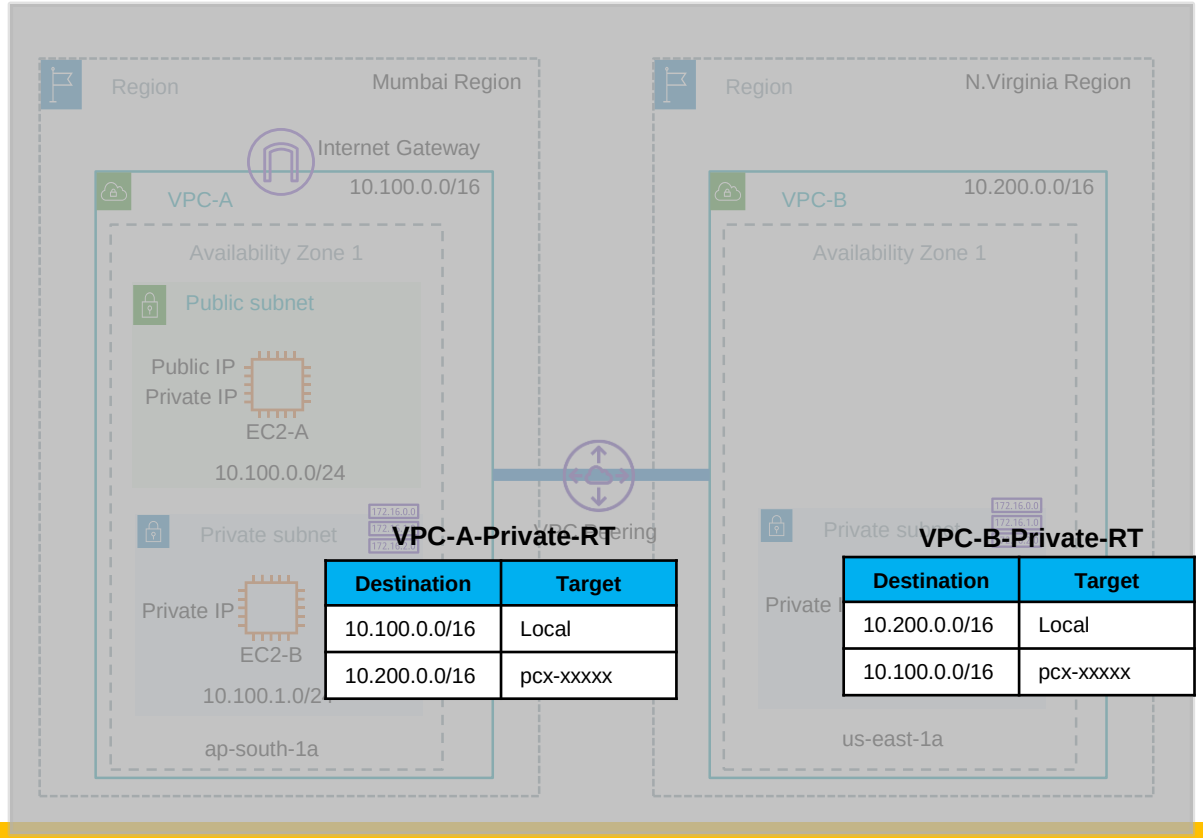
Destination	Target
10.100.0.0/16	Local
0.0.0.0/0	**eni-xxxxxx**

Exercise – VPC Peering



1. Create 2 VPCs in different AWS regions as shown. Create Internet gateway and associate with VPC-A. VPC-B do not need internet gateway.
2. Create Public and Private subnets as shown in respective VPCs
3. Launch EC2 instances in respective subnets. Only EC2-A should have Public IP. Have appropriate security groups to allow SSH from EC2-A to EC2-B and SSH and ping from EC2-B to EC2-C.
4. Login to EC2-A -> EC2-B and try to ping EC2-C. At this moment from EC2-B you should not be able to reach to EC2-C.
5. Create VPC peering between both the VPCs.

Exercise – VPC Peering



VPC-A-Private-RT

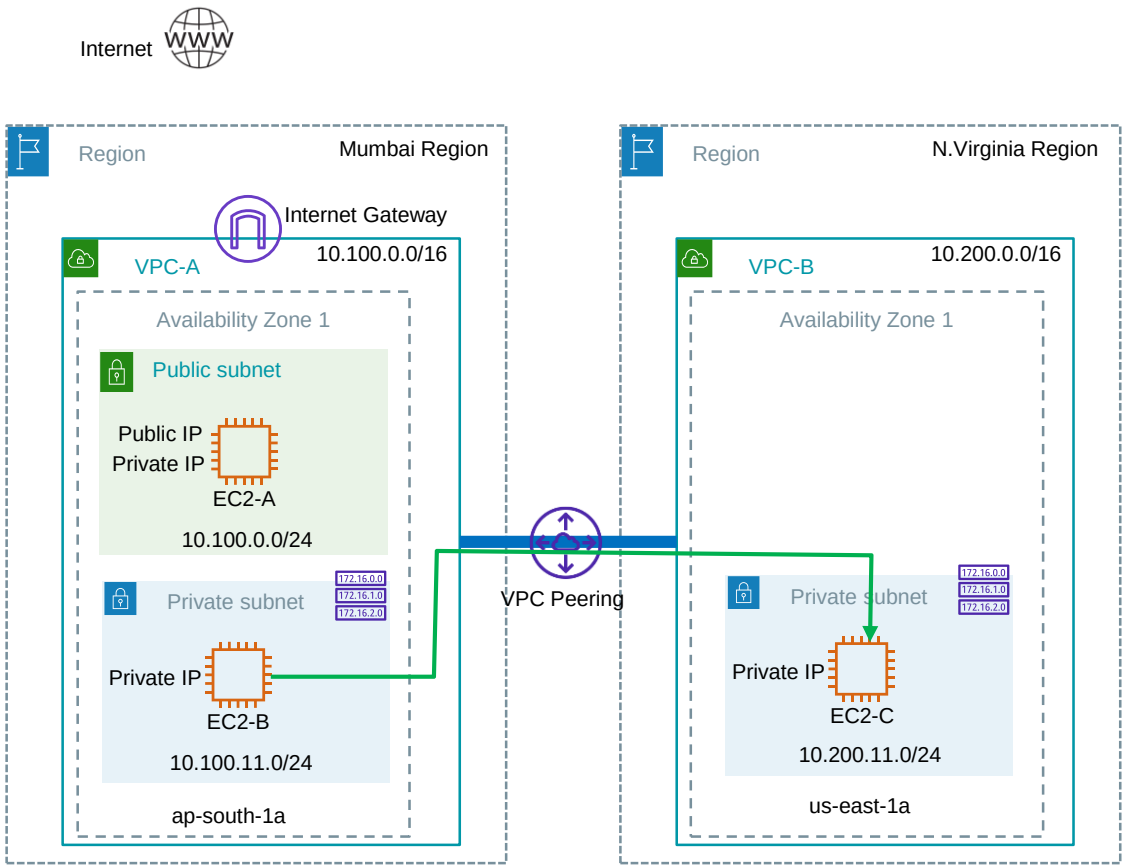
Destination	Target
10.100.0.0/16	Local
10.200.0.0/16	pcx-xxxxx

VPC-B-Private-RT

Destination	Target
10.200.0.0/16	Local
10.100.0.0/16	pcx-xxxxx

1. Create 2 VPCs in different AWS regions as shown. Create Internet gateway and associate with VPC-A. VPC-B do not need internet gateway.
2. Create Public and Private subnets as shown in respective VPCs
3. Launch EC2 instances in respective subnets. Only EC2-A should have Public IP. Have appropriate security groups to allow SSH from EC2-A to EC2-B and SSH and ping from EC2-B to EC2-C.
4. Login to EC2-A -> EC2-B and try to ping EC2-C. At this moment from EC2-B you should not be able to reach to EC2-C.
5. Create VPC peering between both the VPCs.
6. Modify both VPC Private subnet route tables and add routes for the destination VPC with VPC peering as target

Exercise – VPC Peering



1. Create 2 VPCs in different AWS regions as shown. Create Internet gateway and associate with VPC-A. VPC-B do not need internet gateway.
2. Create Public and Private subnets as shown in respective VPCs
3. Launch EC2 instances in respective subnets. Only EC2-A should have Public IP. Have appropriate security groups to allow SSH from EC2-A to EC2-B and SSH and ping from EC2-B to EC2-C.
4. Login to EC2-A -> EC2-B and try to ping EC2-C. At this moment from EC2-B you should not be able to reach to EC2-C.
5. Create VPC peering between both the VPCs.
6. Modify both VPC Private subnet route tables and add routes for the destination VPC with VPC peering as target
7. Communication should be successful in both direction between EC2-B and EC2-C

Steps

1. Create VPC-A and VPC-B in different AWS regions with non-overlapping CIDRs.
    a. Create VPC-A in Mumbai (ap-south-1) region with CIDR 10.100.0.0/16. Create and associate Internet gateway to VPC-A.
    b. Create VPC-B in N.Virginia (us-east-1) region with CIDR 10.200.0.0/16. No internet gateway.
2. Create subnets
    a. In VPC-A, create 1 Public subnet and 1 Private subnet. Create route tables and associate with corresponding subnets.
    b. In VPC-B, create 1 Private subnet. Create a route table and associate with the subnet.
3. Launch EC2 instances and login
    a. Launch Public EC2 instance in VPC-A Public Subnet (with Public IP) and associate security group to allow SSH (22) from MyIP or anywhere.
    b. Launch Private EC2 instance in VPC-A Private Subnet. Associate security group to allow SSH (22) from VPC-A CIDR (10.100.0.0/16)
    c. Launch Private EC2 instance in VPC-B Private Subnet. Assoicate security group to allow SSH (22) and ping (ICMP-IPv4) from VPC-A CIDR (10.100.0.0/16)
4. Connect to EC2-B and try to reach EC2-C
    a. SSH into EC2-A from your workstation -> From EC2-A, SSH into EC2-B (for this you need to bring your SSH key to EC2-A. Refer pre-requisites or troubleshooting section on how to do it)
    b. From EC2-B terminal, try to ping EC2-C private IP.

[This should not work, why? There is no connectivity between VPC-A and VPC-B.

Steps

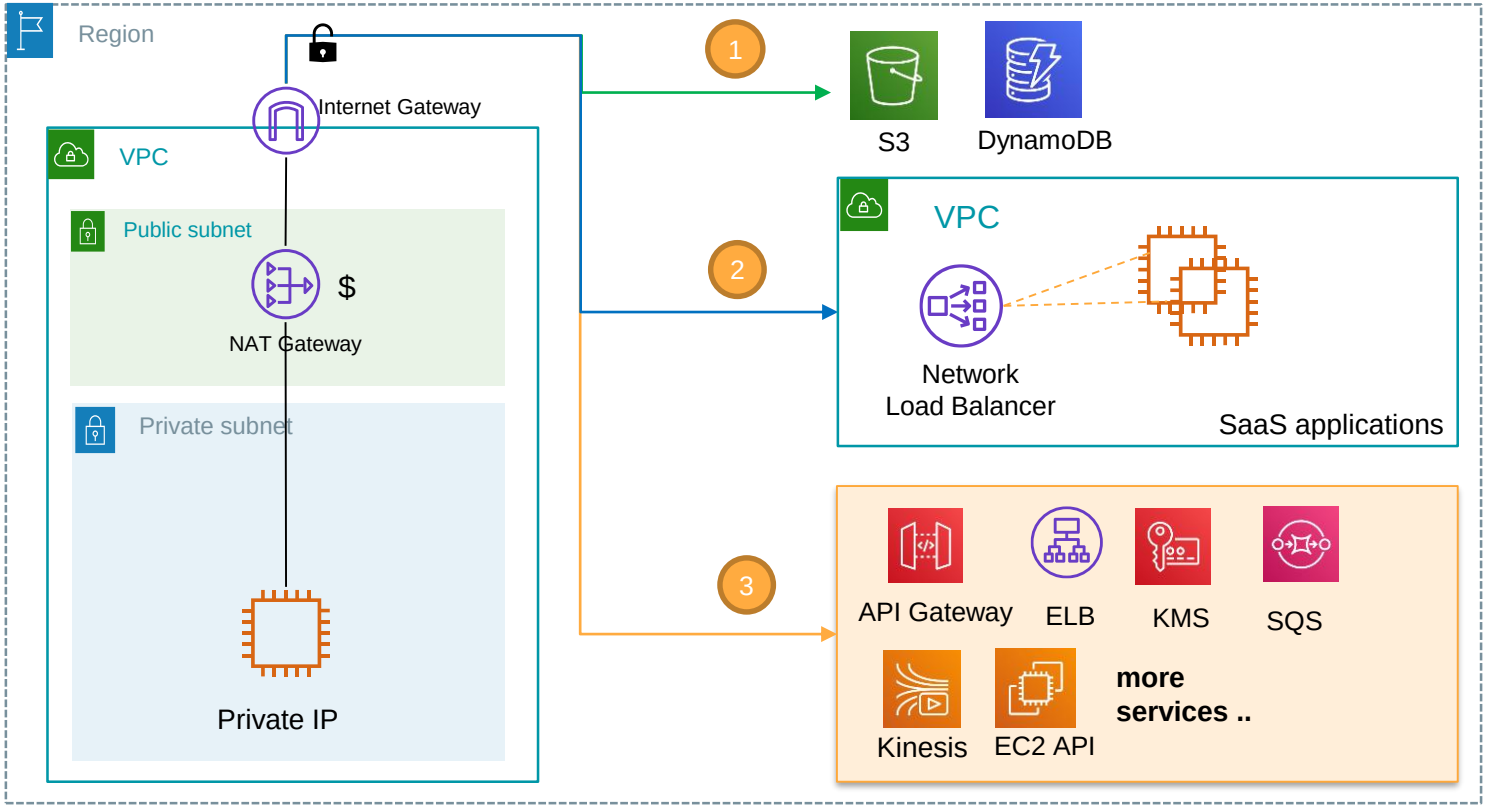
5. Create VPC Peering between both the VPCs
    a. Go to N.Virginia region VPC console -> Copy the VPC ID VPC-B (vpc-xxxxx)
    b. Go to Mumbai region VPC console -> Peering Connections -> Create peering connection
    c. Name: VPC-A-VPC-B-Peering
    d. Select local VPC: VPC-A
    e. Select another VPC to peer with: My Account
    f. Region: Another Region
    g. Select region: N. Virginia (us-east-1)
    h. VPC ID: *paste the VPC-B ID that you have copied.*
    i. Create peering connection
    j. Go to N.Virginia region VPC console -> Peering Connections -> Actions -> Accept request -> Accept
    k. From EC2-B terminal, try to ping EC2-C private IP again, does that work?
6. Update both Private subnet route tables and add route to reach to other VPC
    a. Mumbai region -> Select Private subnet (VPC-A-Private-RT) -> Routes -> Edit routes -> Add a route (Destination: 10.200.0.0/16, Target: Peering connection) -> Save changes
    b. N.Virginia region -> Select Private subnet (VPC-B-Private-RT) -> Routes -> Edit routes -> Add a route (Destination: 10.100.0.0/16, Target: Peering connection) -> Save changes
7. Log into EC2-B (SSH) and try to access EC2-C private IP using following commands:
    $ping 10.200.11.x

[Above commands should work, why? There is now connectivity and route between both VPCs Private subents.

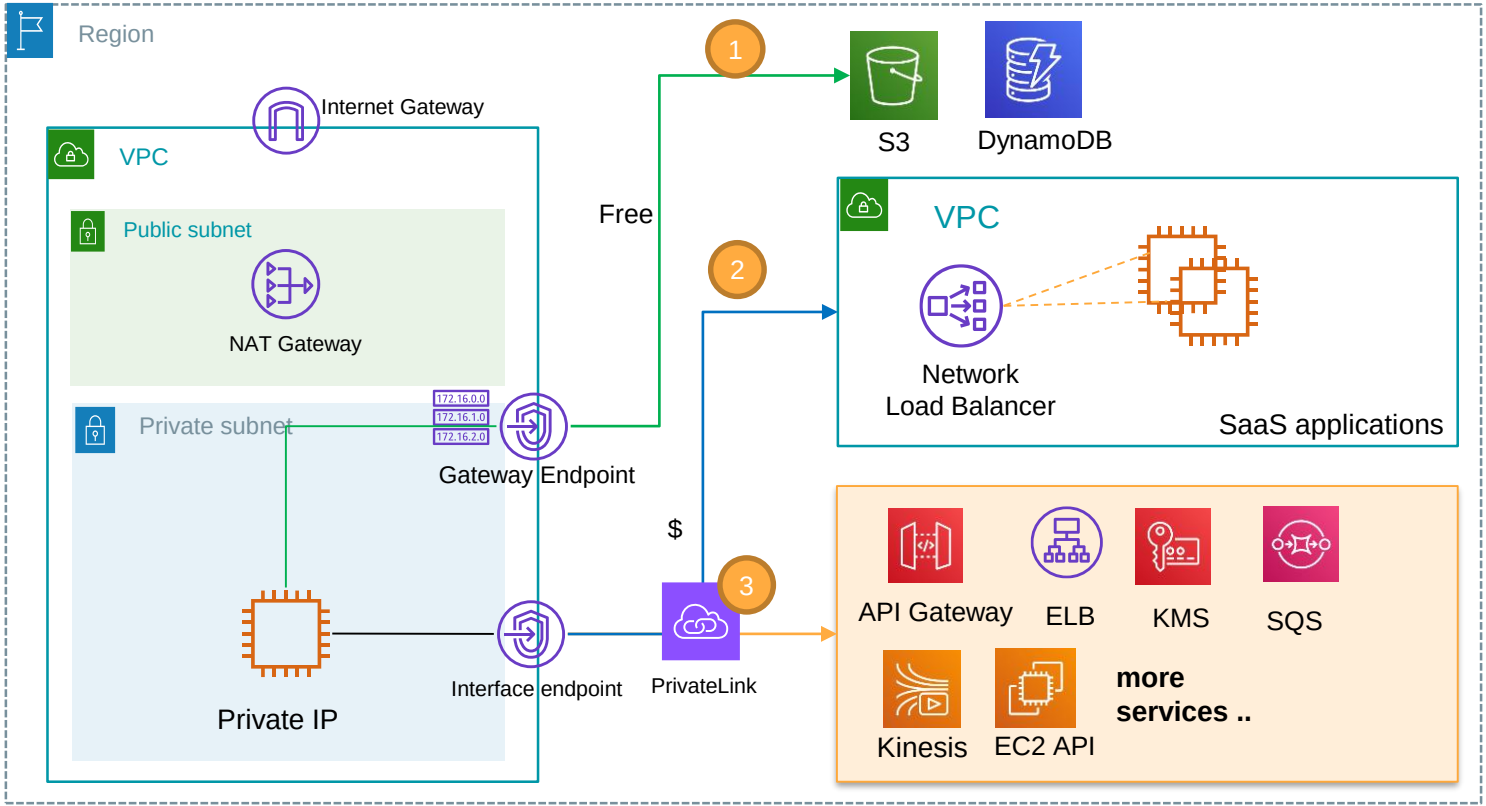
Clean-up

a. If you are **not** continuing with next exercises
    a. Terminate all three EC2 instances
    b. Delete VPC peering connection
    c. Delete VPC-B in N.Virginia region
    d. Delete VPC-A in Mumbai region (there is no adding cost even if you have VPC-A and subnets in your account)
b. If you want to continue with the next exercise
    a. Terminate EC2-C in VPC-B
    b. Delete VPC peering connection
    c. Delete VPC-B in N.Virginia region
    d. Continue with next exercise

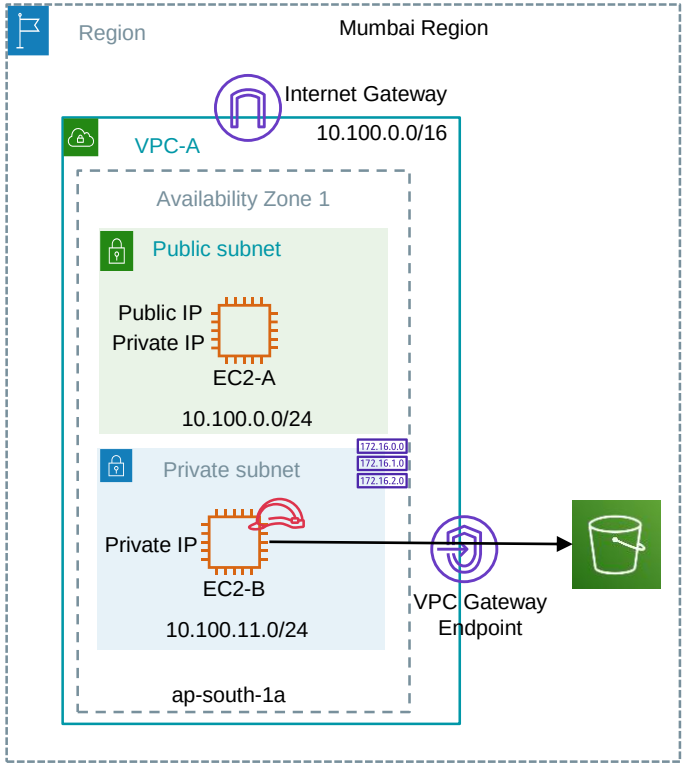
VPC Endpoints & PrivateLink



VPC Endpoints & PrivateLink



Exercise – VPC Gateway endpoint for S3



Private Subnet Route table

Destination	Target
10.100.0.0/16	Local
pl-xxxxxx	vpce-xxxxxx

1. Create a VPC, internet gateway, a public subnet and a private subnet as shown
2. Launch EC2 instances in respective subnets. Only EC2-A should have Public IP. Have appropriate security groups to allow SSH from EC2-A to EC2-B.
3. Create a S3 bucket in the same region, upload any sample file
4. Create an IAM role for EC2 instance and attach S3 read-only IAM policy. Associate role with EC2-B instance.
5. Login to EC2-B and try to download file from S3 using s3 CLI command. Does not work.
6. Create VPC endpoint for S3 and update Private subnet route table.
7. Try to download the same file from S3 again. Should work this time.

Steps

1. Create VPC and Subnets in any of the region
    a. Create VPC-A in Mumbai (ap-south-1) region with CIDR 10.100.0.0/16. Create and associate Internet gateway to VPC-A.
    b. Create 1 Public subnet and 1 Private subnet. Create route tables and associate with corresponding subnets.
2. Launch EC2 instances and login
    a. Launch Public EC2 instance in VPC-A Public Subnet (with Public IP) and associate security group to allow SSH (22) from MyIP or anywhere.
    b. Launch Private EC2 instance in VPC-A Private Subnet. Associate security group to allow SSH (22) from VPC-A CIDR (10.100.0.0/16)
3. Create S3 bucket and upload sample file
    a. S3 console -> Create Bucket
    b. Bucket Name: <unique bucket name>
    c. AWS Region: same region in which you are doing this exercise
    d. Create bucket
    e. Select same bucket -> Upload -> Add files -> Choose sample file from your local machine -> Upload
4. Create IAM role for EC2 to be able to download file from S3
    a. Go to IAM console -> Roles -> Create role
    b. Use case: EC2
    c. Click Next -> Permission policies -> Search for S3 and select "AmazonS3ReadOnlyAccess“ policy -> Next
    d. Role name: EC2_ROLE_FOR_S3_READONLY -> Create role

Steps

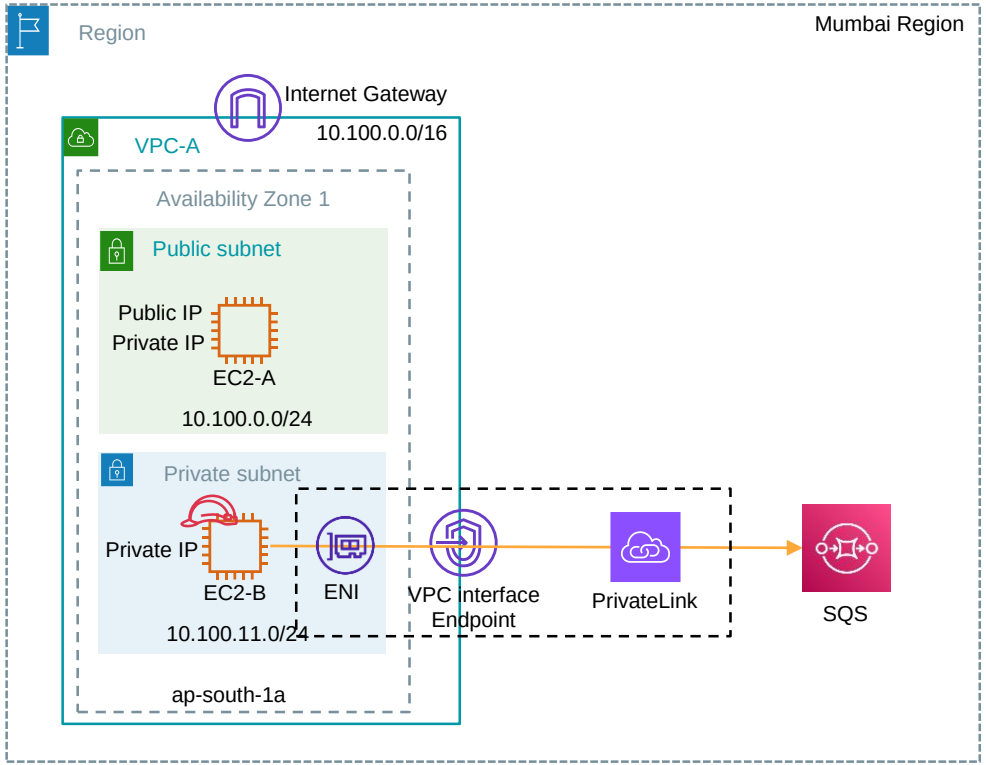
5. From EC2-B, try to download your file from S3
   a. SSH into EC2-A from your workstation
   b. From EC2-A, SSH into EC2-B (for this you need to bring your SSH key to EC2-A. Refer pre-requisites or troubleshooting section on how to do it)
   c. From EC2-B terminal, try to download file from S3. This command does not work as there is no connectivity to S3.
      $ aws s3 cp s3://bucket-name/filename /home/ec2-user/
6. Create VPC gateway endpoint for S3 and update route table
   a. VPC console -> Endpoints -> Create endpoint
   b. Name: my-s3-endpoint
   c. Services: search S3 and select “com.amazonaws.ap-south-1.s3” Type: Gateway
   d. VPC: VPC-A
   e. Route tables: Select Private subnet route table
   f. Create endpoint
   g. After endpoint is created successfully, route table should be updated with route for s3 prefix list with target as VPC endpoint.
7. Try to download your file from S3 again
   a. From EC2-B terminal, try to download file from S3.
      $ aws s3 cp s3://bucket-name/filename /home/ec2-user/

[Above commands should work, why? There is now connectivity and route between VPC and S3 through VPC endpoint.

Clean-up

a. If you are **not** continuing with next exercises
    a. Terminate both EC2 instances
    b. Delete VPC endpoint
    c. Delete VPC-A (there is no adding cost even if you have VPC-A and subnets in your account)
b. If you want to continue with the next exercise
    a. Delete VPC endpoint connection
    b. Update Private subnet route table and remove route for s3.

Exercise – VPC interface endpoint for SQS



Continuing from the earlier setup

0. Make sure to enable Private DNS for the VPC (DNS hostnames and DNS resolution)
1. Create a SQS queue in the same region
2. Create an IAM role for EC2 instance and attach SQS read-only IAM policy. Associate role with EC2-B instance.
3. Login to EC2-B and try to send message to the SQS queue using sqs CLI command. Does not work.
4. Create VPC interface endpoint for SQS in the Private Subnet. For Security group, allow inbound https/443.
5. Try to send the message to SQS queue again. Should work this time.

Steps

0. Create VPC, Subnets and EC2-A, EC2-B if not using the previous setup and Enable Private DNS
    a. VPC Console -> Select VPC-A -> Edit VPC Settings and select both
    b. Enable DNS resolution
    c. Enable DNS hostnames

1. Create SQS standard queue
    a. SQS console -> Create queue
    b. Type: Standard
    c. Name: my-queue
    d. Create queue
2. Create IAM role for EC2 to be able to send a message to SQS queue
    a. Go to IAM console -> Roles -> Create role
    b. Use case: EC2
    c. Click Next -> Permission policies -> Search for SQS and select "AmazonSQSFullAccess“ policy -> Next
    d. Role name: EC2_ROLE_FOR_SQS -> Create role

Steps

3. From EC2-B, try to send a message to you SQS queue
    a. SSH into EC2-A from your workstation
    b. From EC2-A, SSH into EC2-B (for this you need to bring your SSH key to EC2-A. Refer pre-requisites or troubleshooting section on how to do it)
    c. From SQS console, copy the queue URL (https://sqs.ap-south-1.amazonaws.com/xxxxxxxxxxxx/my-sqs)
    d. From EC2-B terminal, send a message. (This does not work)
    $ aws sqs send-message –queue-url <your sqs queue url> --message-body "Test message"
4. Create VPC interface endpoint for SQS in the private subnet.
    a. VPC console -> Security Groups -> Create security group
    b. Security group name: SG_FOR_VPC_INTERFACE_ENDPOINT, VPC: VPC-A
    c. Inbound Rule -> Add rule -> Type: HTTPS, Port: 443, Source: Custom 10.100.0.0/16 -> Create security group
    d. VPC console -> Endpoints -> Create endpoint
    e. Name: my-sqs-endpoint
    f. Service Category: AWS Services
    g. Services: search SQS and select "com.amazonaws.ap-south-1.sqs" Type: Interface
    h. VPC: VPC-A
    i. Subnets: ap-south-1a -> Subnet ID: VPC-A-Private
    j. Security groups -> Select SG_FOR_VPC_INERFACE_ENDPOINT
    k. Create endpoint

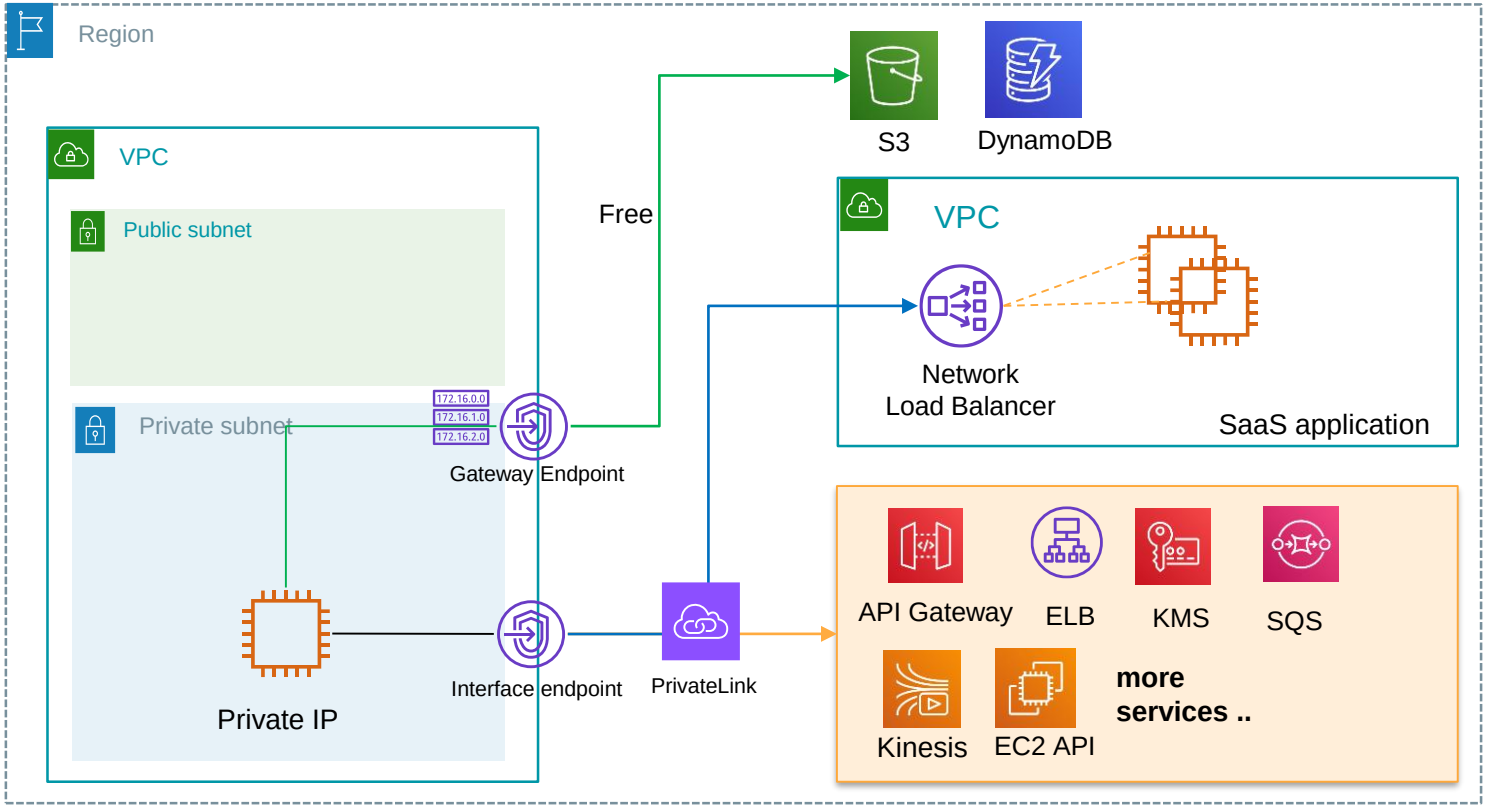
Steps

5. Send message from EC2-B to SQS queue again

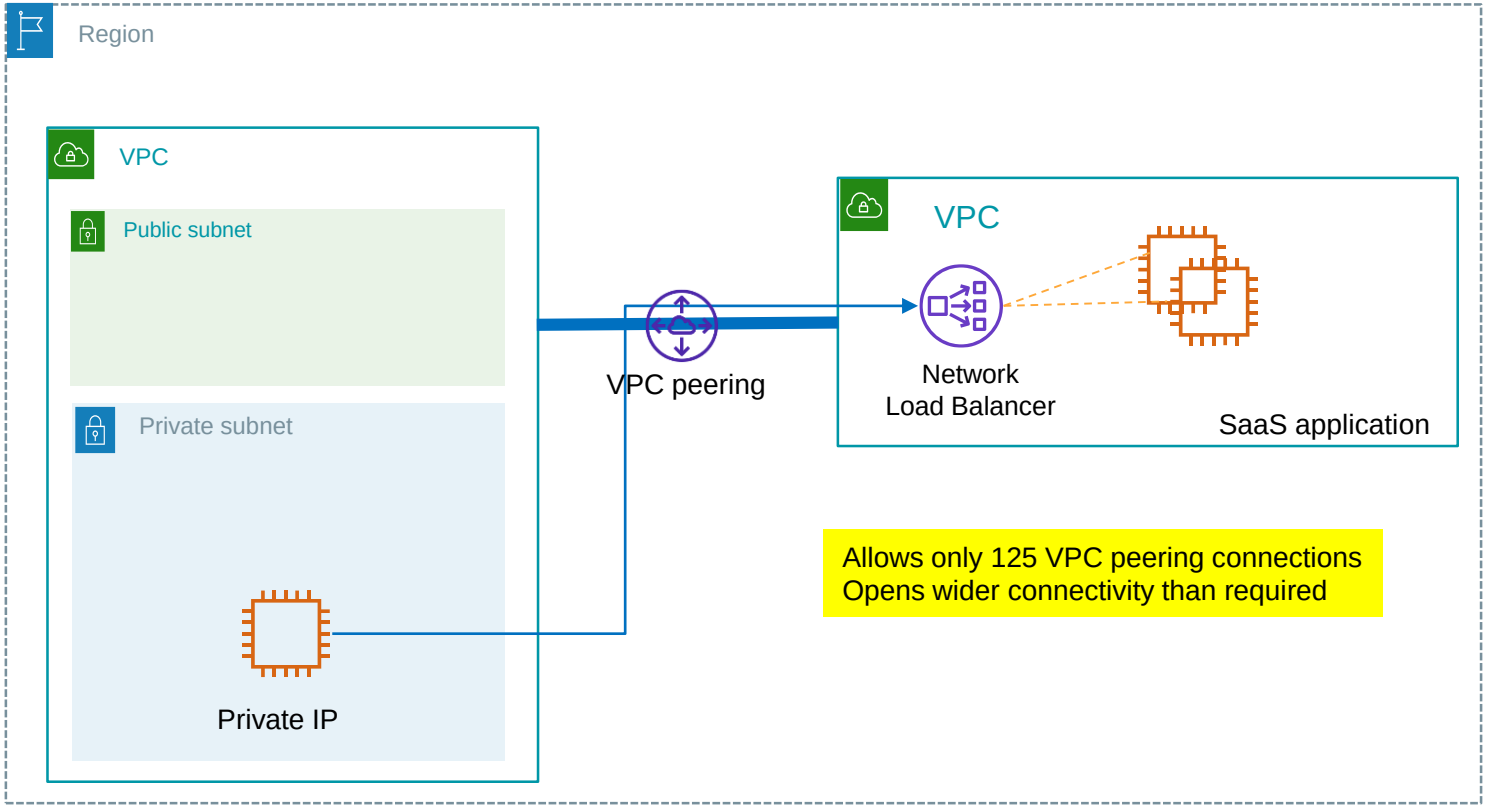
$ aws sqs send-message –queue-url <your sqs queue url> --message-body “Test message”

[Above commands should work, why? There is now connectivity and route between VPC and SQS through VPC interface endpoint.

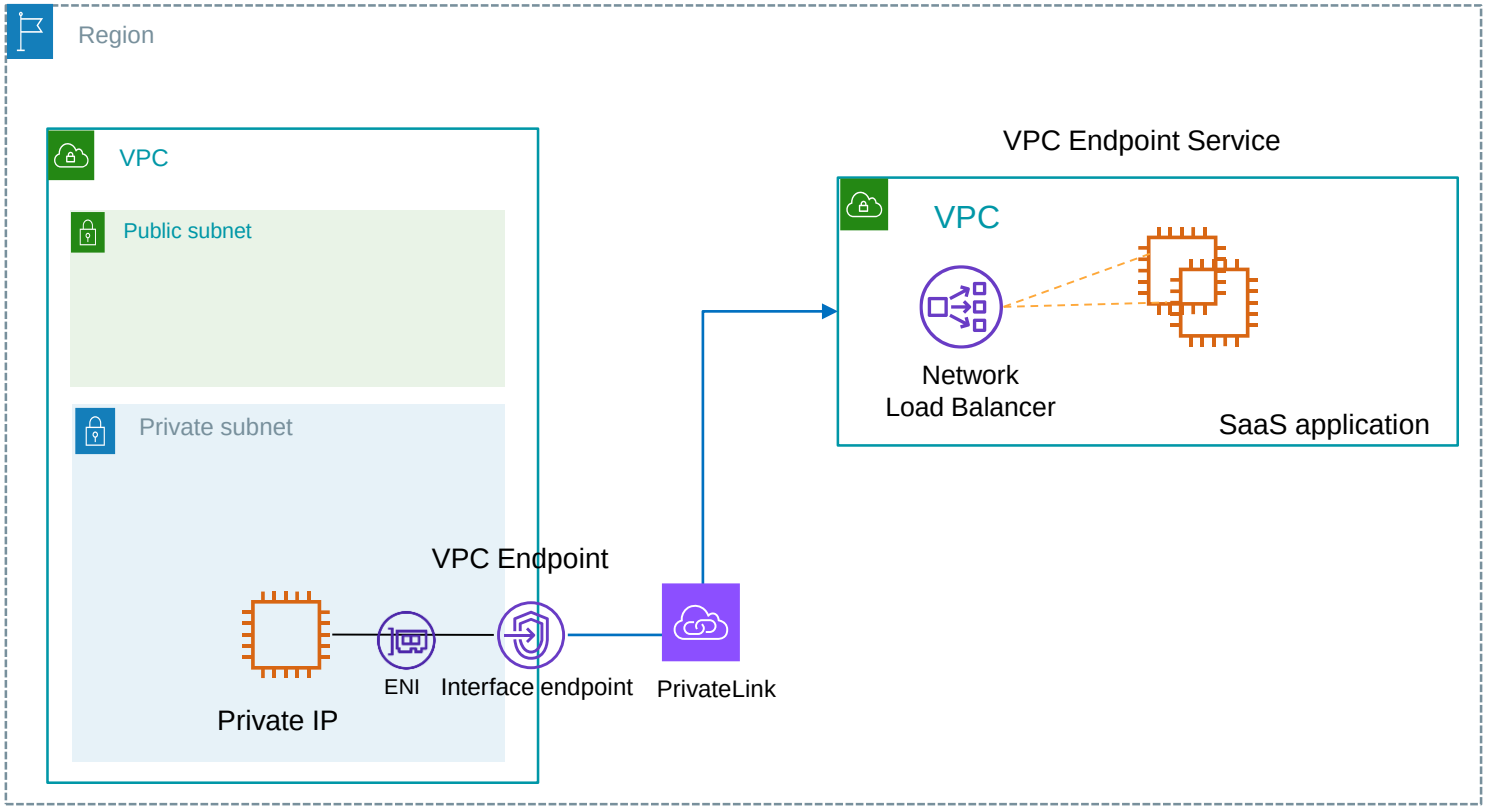
VPC PrivateLink for SaaS applications



VPC PrivateLink for SaaS applications



VPC PrivateLink for SaaS applications



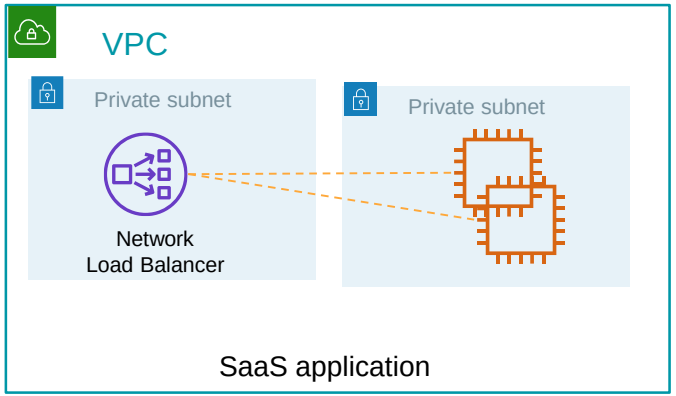
Exercise – VPC PrivateLink to access SaaS applications

Pre-requisite – Create an Amazon Machine Image (AMI) in the Service Provider VPC

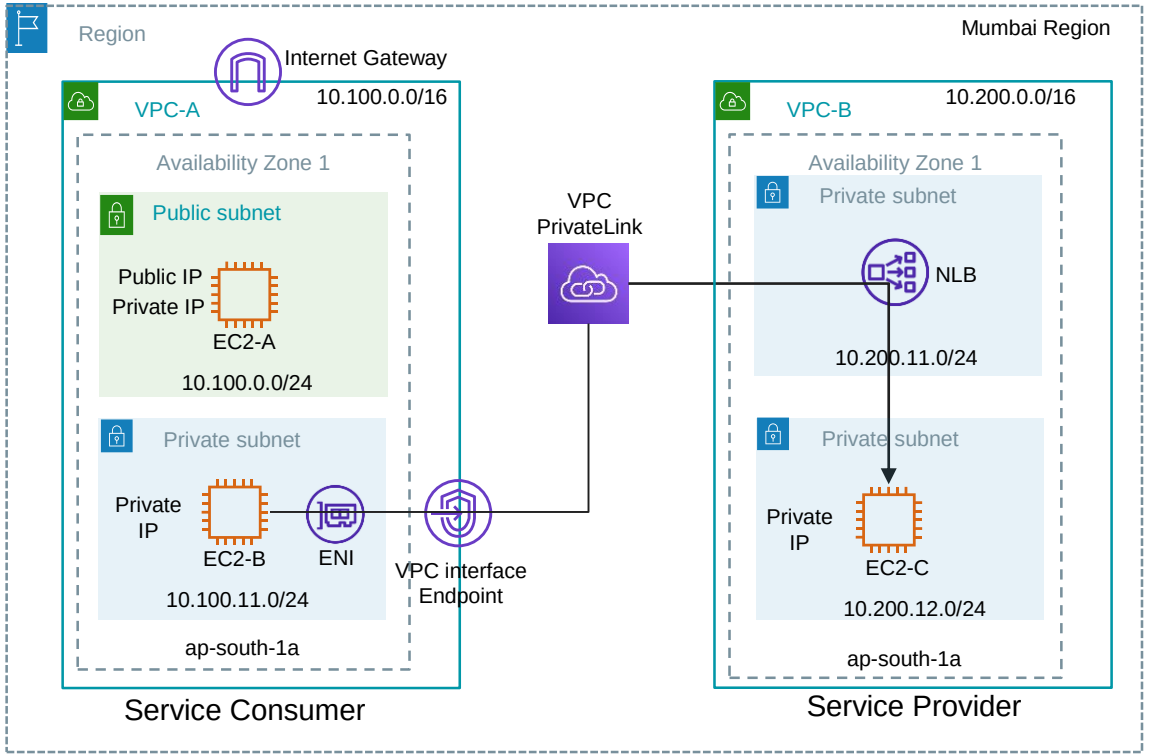
1. Launch EC2 instance (Amazon Linux) in a default VPC with following User Data script

```
#!/bin/bash
yum install httpd -y
service httpd start
chkconfig httpd on
echo "This is a SaaS application" > /var/www/html/index.html
echo successful
```

2. Verify that Web server is accessible by hitting EC2 Public IP
3. Stop the instance and create Amazon Machine Image (AMI)
4. Terminate EC2 instance



Exercise – VPC PrivateLink to SaaS applications (Part 1)



1. Create service consumer VPC, internet gateway, public subnet and a private subnet as shown
2. Launch EC2 instances in respective subnets. Only EC2-A should have Public IP. Have appropriate security groups to allow SSH from EC2-A to EC2-B.
3. Create service provider VPC in the same region and create two private subnets as shown
4. Launch a Web server EC2-C using pre-created web-server AMI as described in pre-requisites section
5. Create NLB and target group and add EC2-C as a target.
6. Create VPC endpoint service with NLB as a target.
7. Create VPC interface endpoint for above service in consumer VPC by selecting Private subnet.
8. From EC2-B, access VPC endpoint DNS, should be able to access a webserver

Steps

1. Create service consumer VPC and Subnets
    a. Create VPC-A in Mumbai (ap-south-1) region with CIDR 10.100.0.0/16. Create and associate Internet gateway to VPC-A.
    b. Create 1 Public subnet and 1 Private subnet. Create route tables and associate with corresponding subnets.
2. Launch EC2 instances and login
    a. Launch Public EC2 instance (EC2-A) in VPC-A Public Subnet (with Public IP) and associate security group to allow SSH (22) from MyIP or anywhere.
    b. Launch Private EC2 instance (EC2-B) in VPC-A Private Subnet. Associate security group to allow SSH (22) from VPC-A CIDR (10.100.0.0/16)
3. Create service provider VPC and Subnets
    a. Create VPC-B in Mumbai (ap-south-1) region with CIDR 10.200.0.0/16.
    b. Create 1 Private subnet for Network load balancer and another Private subnet for a webserver. Create a Private route table and associate with both the subnets.
4. Launch a webserver
    a. Launch EC2 instance (EC2-C) in VPC-B Private Subnet from the Amazon Machine Image (AMI) created as a pre-requisite. Security group to allow HTTP (80) from VPC-B CIDR or NLB Private subnet.

Steps

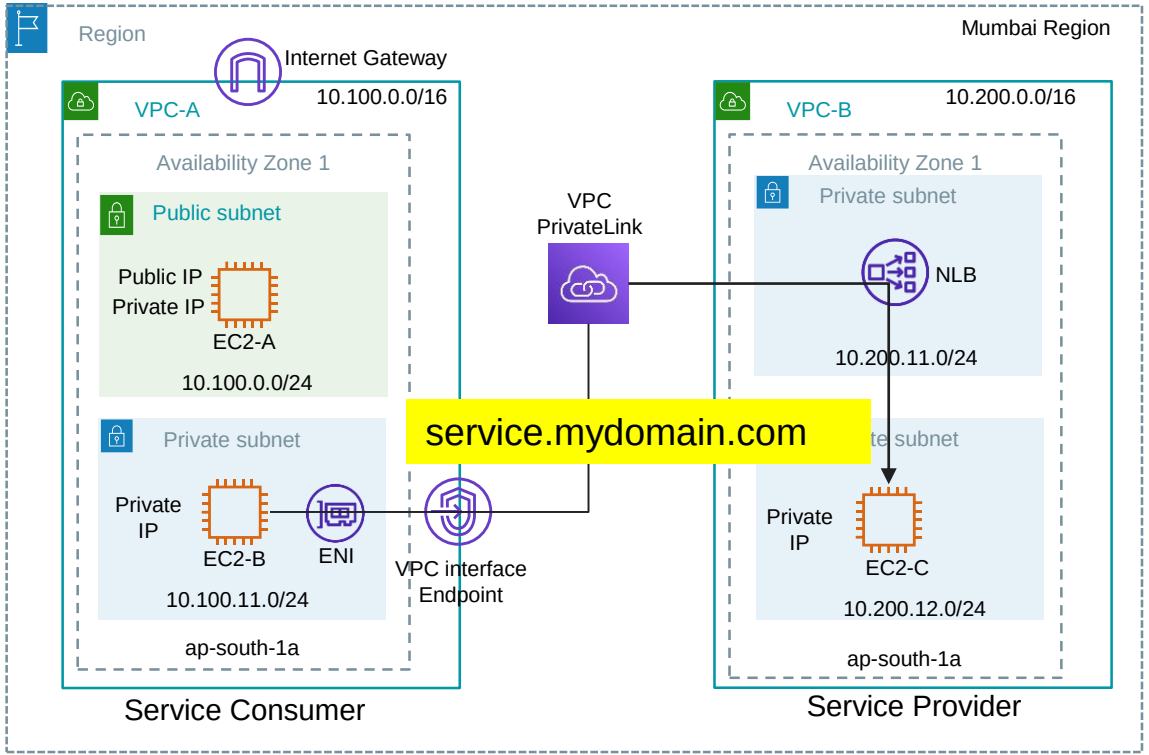
5. Create Network Load Balancer, Target group and register webserver EC2 instance
    a. EC2 console -> Load Balancers -> Target groups -> Create target group
    b. Choose target type: Instances, Target group name: Webserver, VPC: Select VPC-B, Protocol version: TCP/80 -> Next
    c. From Available instances, select a Webserver EC2 instance -> Include as pending below -> Create target group
    d. EC2 console -> Load Balancers -> Create load balancer -> Select Network Load Balancer
    e. Load balancer name: ServiceProviderNLB, Scheme: **internal,** VPC: Select VPC-B, Mappings: select AZ and Subnet that you created for NLB
    f. Listener and routing -> Protocol: TCP, Port: 80, Default action: Forward to: Select Webserver target group -> Create load balancer
6. Create VPC endpoint service using NLB as a target in service provider VPC
    a. VPC console -> Endpoint Services -> Create endpoint service
    b. Name: WebserverService, Load balancer type: Network
    c. Available load balancers -> Select ServiceProviderNLB
    d. Additional Settings -> Uncheck Acceptance required, Supported IP address types: IPv4 -> Create
    e. Note down the name of the service: com.amazonaws.vpce.us-east-1.vpce-svc-xxxxxxx
7. Create VPC endpoint in a consumer VPC
    a. VPC console -> Security groups -> Create security group -> Name: VPCEndpointSG -> Add inbound rule for HTTP/80 from VPC-A CIDR 10.100.0.0/16
    b. VPC console -> Endpoints -> Create endpoint

Steps

c. Name: WebserverServiceEndpoint, Service Category: Other endpoint services.
d. Service name: paste the service name that you copied -> Verify service
e. VPC: VPC-A, Select AZ and Subnet for VPC-A Private subnet, IP address type: IPv4, Security Groups: select VPCEndpointSG that you created above, Policy: Full access -> Create endpoint

8. Access a service from EC2-B using VPC endpoint
   a. Copy the regional DNS for the VPC endpoint
   b. SSH into EC2-B and run following command to access the Webserver Service using the VPC endpoint DNS
      $ curl <vpc endpoint regional endpoint dns>
   c. You should get a response from a webserver (EC2-B)

Exercise – VPC PrivateLink to access SaaS applications (Part 2) - Using Private DNS



Pre-requisites:
You should own the Domain name

1. Enabled Private DNS for VPC endpoint service
2. Verify the domain ownership
3. Enable Private DNS for VPC endpoint
4. From EC2-B, access the service using Private DNS.

Steps

1. Create service producer VPC and Subnets
    a. Create VPC-A in Mumbai (ap-south-1) region with CIDR 10.100.0.0/16. Create and associate Internet gateway to VPC-A.
    b. Create 1 Public subnet and 1 Private subnet. Create route tables and associate with corresponding subnets.
2. Launch EC2 instances and login
    a. Launch Public EC2 instance (EC2-A) in VPC-A Public Subnet (with Public IP) and associate security group to allow SSH (22) from MyIP or anywhere.
    b. Launch Private EC2 instance (EC2-B) in VPC-A Private Subnet. Associate security group to allow SSH (22) from VPC-A CIDR (10.100.0.0/16)
3. Create service consumer VPC and Subnets
    a. Create VPC-B in Mumbai (ap-south-1) region with CIDR 10.200.0.0/16.
    b. Create 1 Private subnet for Network load balancer and another Private subnet for a webserver. Create a Private route table and associate with both the subnets.
4. Launch a webserver
    a. Launch EC2 instance (EC2-C) in VPC-B Private Subnet from the Amazon Machine Image (AMI) created as a pre-requisite. Security group to allow HTTP (80) from VPC-B CIDR or NLB Private subnet.

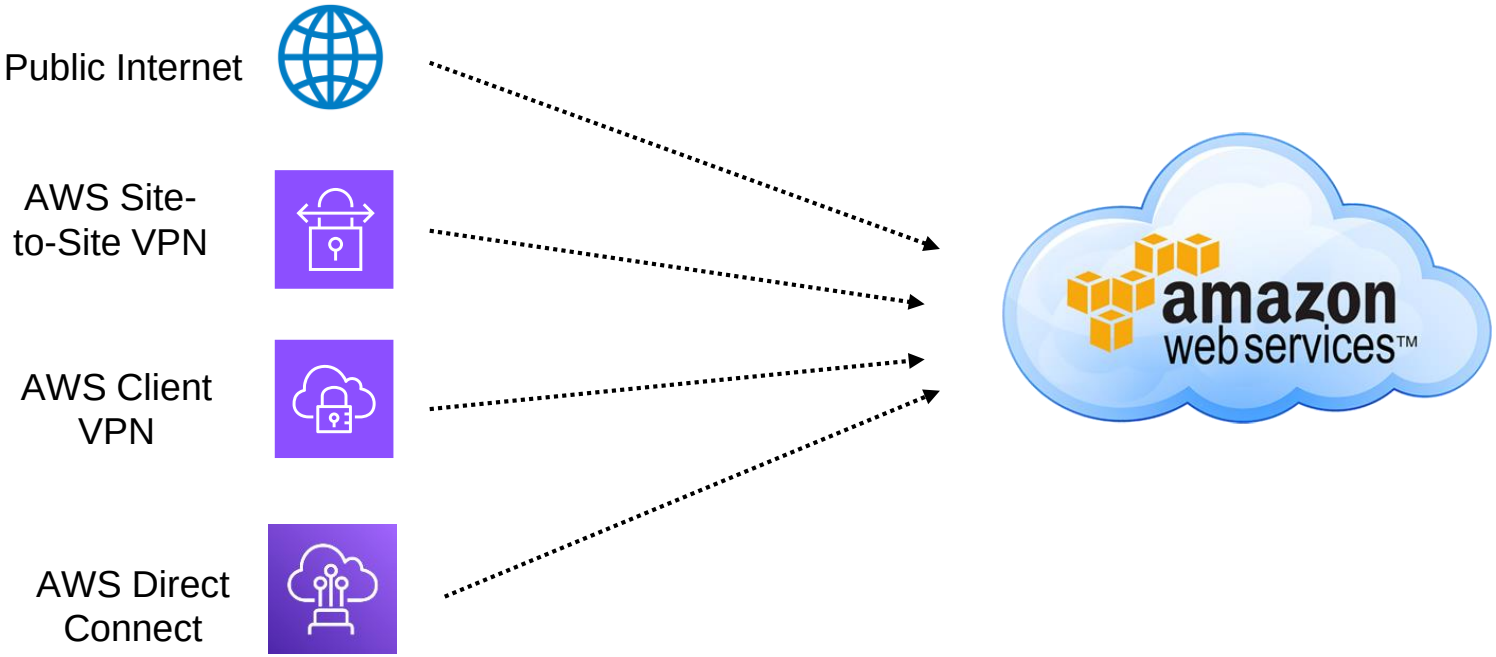
Steps

5. Create Network Load Balancer, Target group and register webserver EC2 instance
    a. EC2 console -> Load Balancers -> Target groups -> Create target group
    b. Choose target type: Instances, Target group name: Webserver, VPC: Select VPC-B, Protocol version: HTTP1 -> Next
    c. From Available instances, select a Webserver EC2 instance -> Include as pending below -> Create target group
    d. EC2 console -> Load Balancers -> Create load balancer -> Select Network Load Balancer
    e. Load balancer name: ServiceProviderNLB, Scheme: **internal,** VPC: Select VPC-B, Mappings: select AZ and Subnet that you created for NLB
    f. Listener and routing -> Protocol: TCP, Port: 80, Default action: Forward to: Select Webserver target group -> Create load balancer
6. Create VPC endpoint service using NLB as a target in service provider VPC
    a. VPC console -> Endpoint Services -> Create endpoint service
    b. Name: WebserverService, Load balancer type: Network
    c. Available load balancers -> Select ServiceProviderNLB
    d. Additional Settings -> Uncheck Acceptance required, Supported IP address types: IPv4 -> Create
    e. Note down the name of the service: com.amazonaws.vpce.us-east-1.vpce-svc-xxxxxxx
7. Create VPC endpoint in a consumer VPC
    a. VPC console -> Security groups -> Create security group -> Name: VPCEndpointSG (default rules)
    b. VPC console -> Endpoints -> Create endpoint
    c. Name: WebserverServiceEndpoint, Service Category: Other endpoint services.
    d. Service name: paste the service name that you copied -> Verify service
    e. VPC: VPC-A, Select AZ and Subnet for VPC-A Private subnet, IP address type: IPv4, Security Groups: select VPCEndpointSG that you created above, Policy: Full access -> Create endpoint

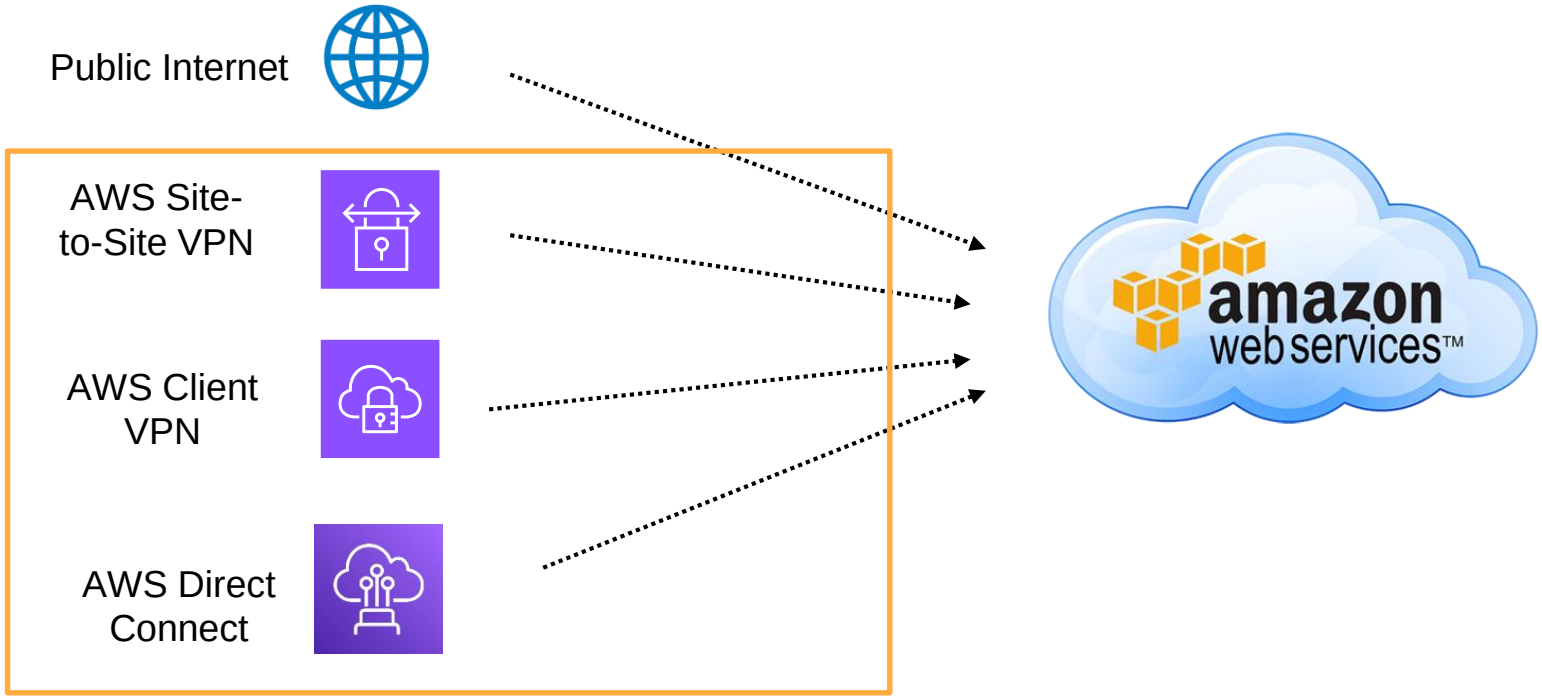
Cleanup

- Delete VPC endpoint in VPC-A (wait until deleted)
- Delete VPC endpoint service in VPC-B
- Terminate Webserver EC2 instance in VPC-B
- Delete Network Load Balancer in VPC-B
- Delete Target Group in VPC-B
- Delete VPC-B
- Optionally
    a. Terminate EC2 instances in VPC-A
    b. Delete VPC-A

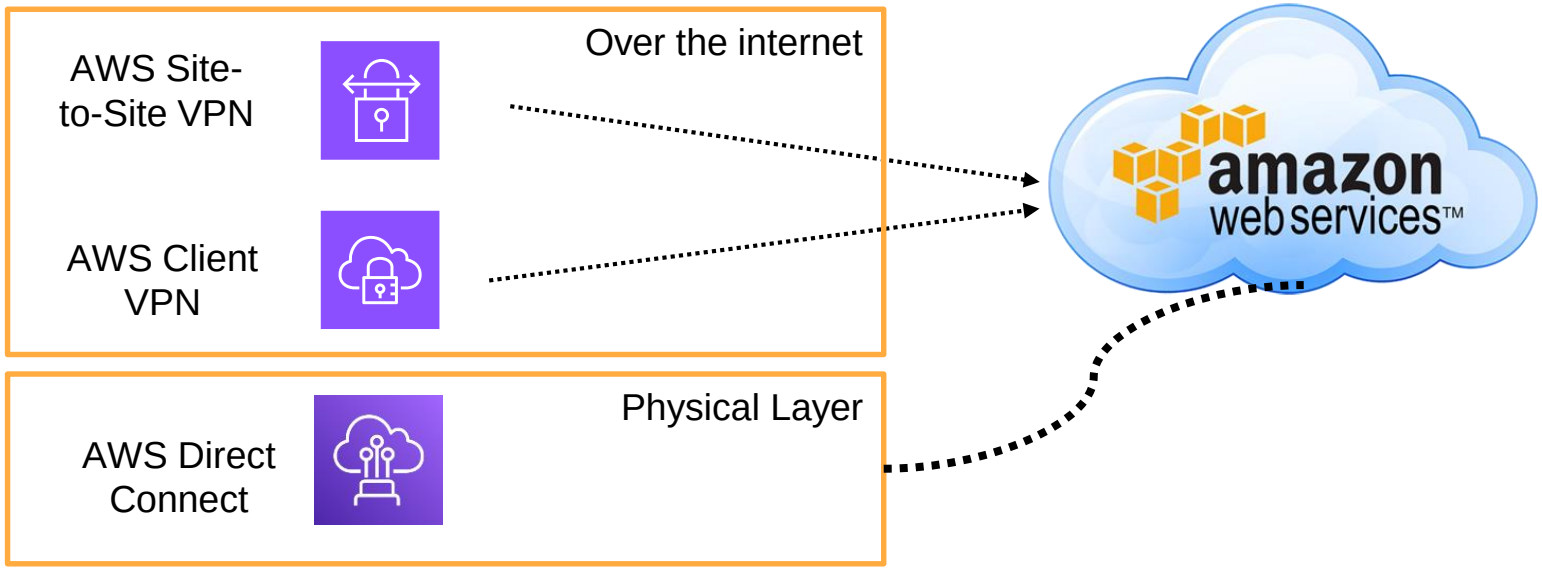
AWS Connectivity Options



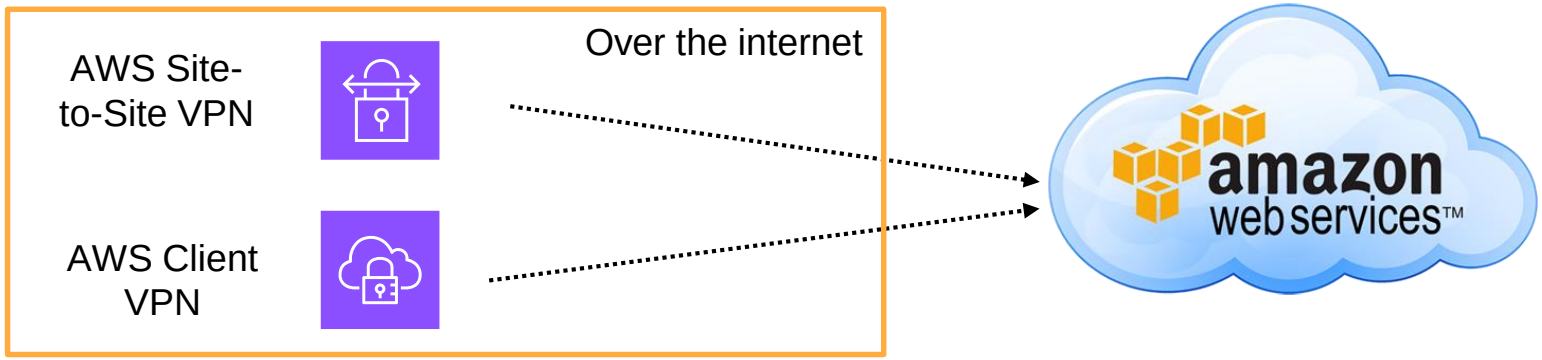
Hybrid Connectivity Options



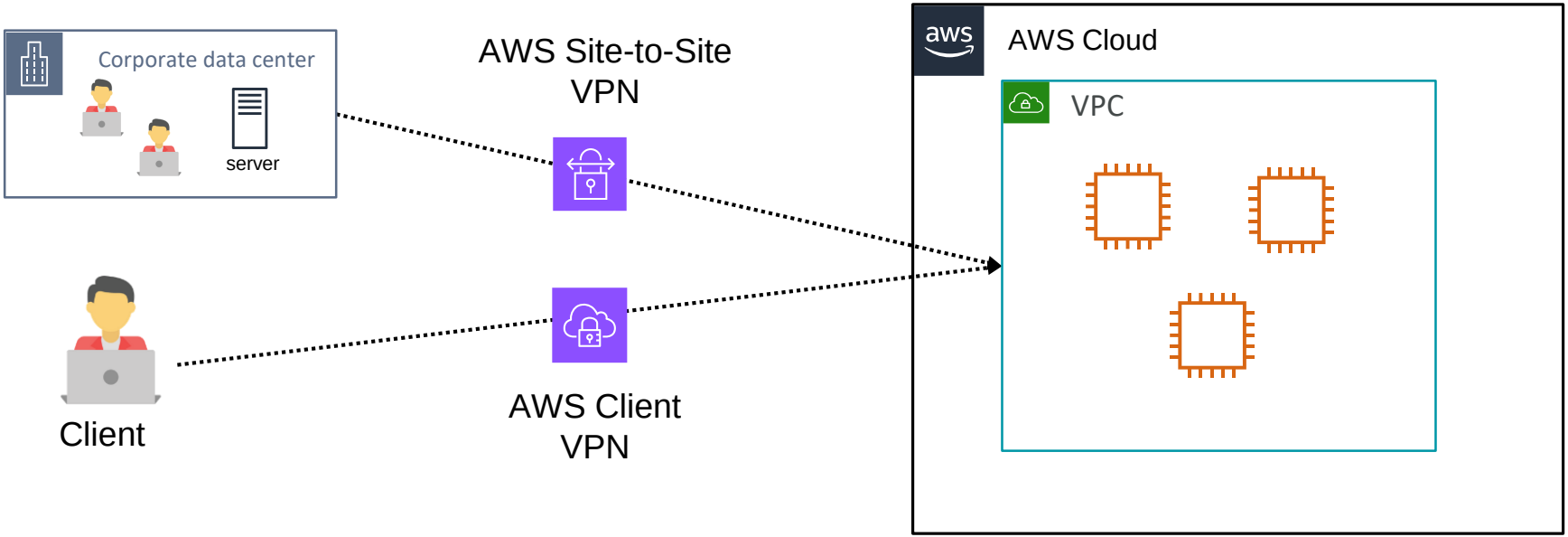
Hybrid Connectivity Options



Hybrid Connectivity Options

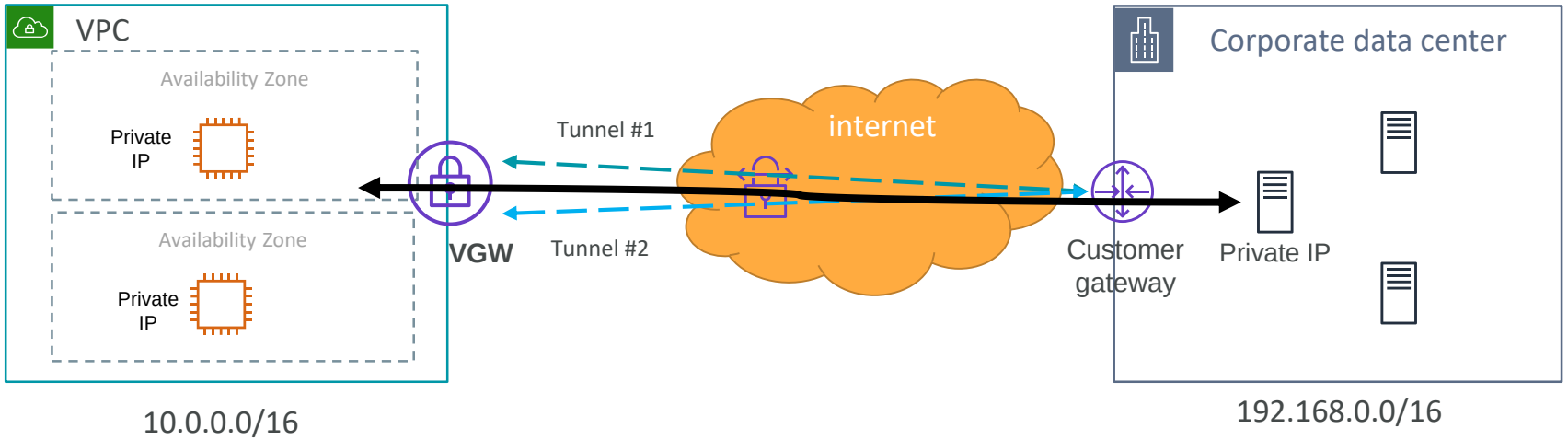


Hybrid Connectivity Options

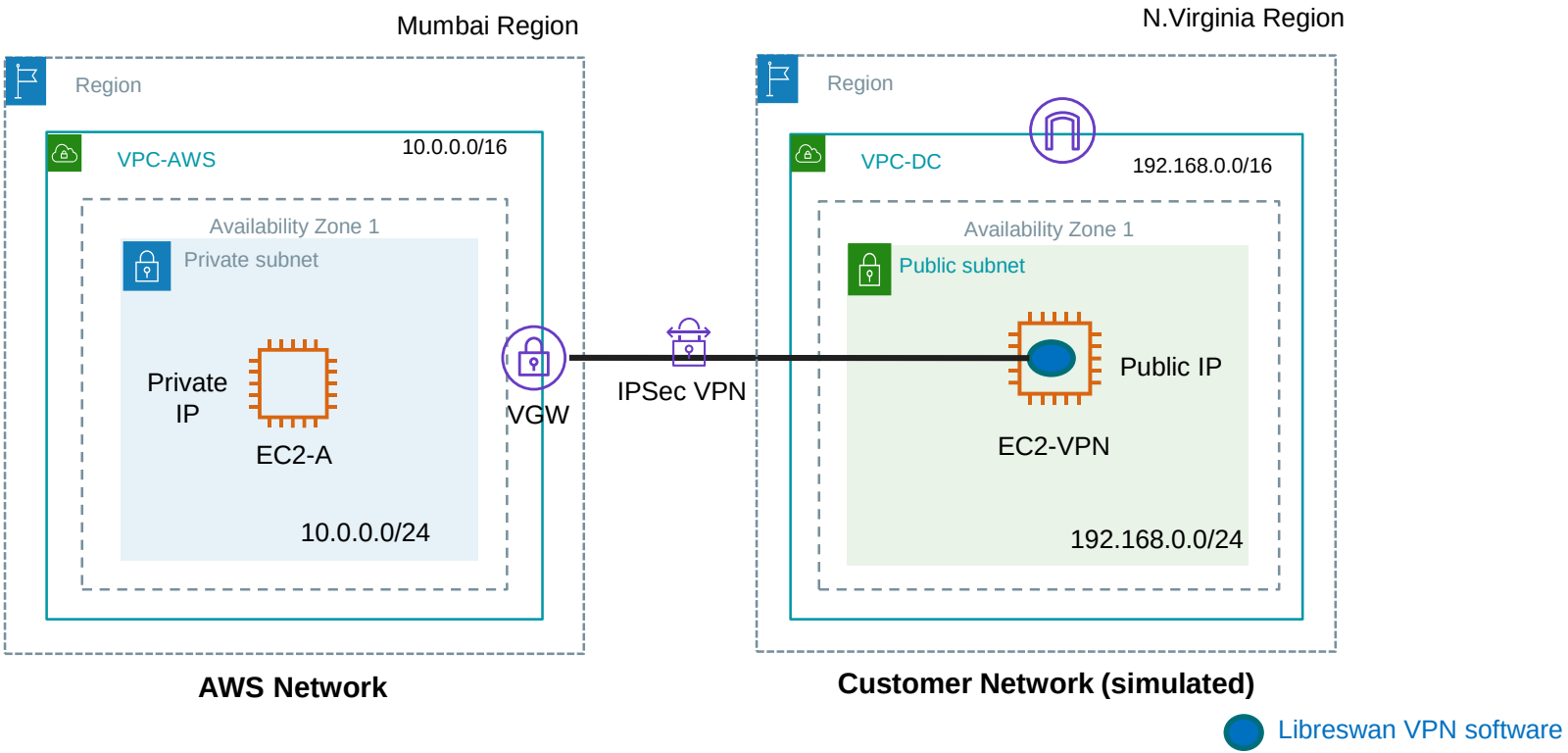


How Site-to-Site VPN works in AWS

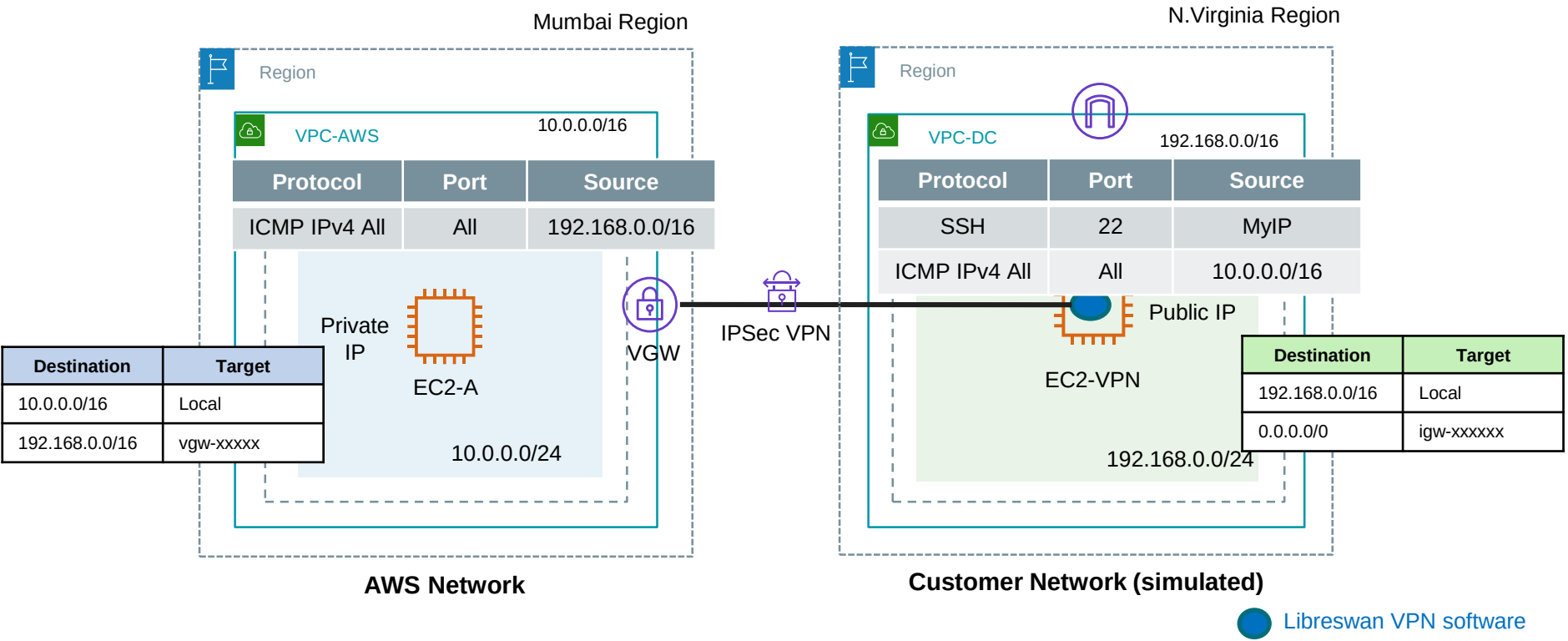
- VPN connection is established between AWS network and Customer corporate network
- At AWS end, VPN is terminated on Virtual Private Gateway (VGW) and Customer gateway (CGW) on Customer end of the network
- AWS Site-to-Sote VPN creates 2 IP Security (IPSec) Tunnels for High Availability



Exercise – AWS Site-to-Site VPN



Exercise – AWS Site-to-Site VPN



High level steps

1. Create VPCs in two different regions as shown in the diagram. One acts as an AWS network and other as a on-premises DC network.

2. Launch EC2 instances in both the VPCs. For VPC-DC, launch it using **Amazon Linux 2023 AMI** so that you can install Libreswan. In Security group for both EC2 instances allow ICMP traffic from the other network. EC2-VPN will have Public IP. You should also open SSH from MyIP or 0.0.0.0/0.

3. Create a Virtual Private Gateway (VGW) in Mumbai region and associate it with the VPC-AWS. Add route in VPC-AWS Private subnet route table to route all the traffic (0.0.0.0/0) via the VGW. Create Customer gateway using the EC2-VPN Public IP.

4. Create a VPN connection in Mumbai region. Use EC2-VPN Public IP with static routing with VPC-DC CIDR.

5. Download VPN configuration file for Openswan from VPN connection console.

6. Install Libreswan on EC2-VPN. You need to add Libreswan fedora repository. After you install libreswan follow the instructions in VPN configuration file. You need to replace values for various fields like IP and CIDR range as per your environment, start the IPSec VPN service.

7. Access the AWS side EC2-A instance Private IP from EC2-VPN. You should be able to.

Steps

Refer the PDF document provided with this lecture for the detailed steps.

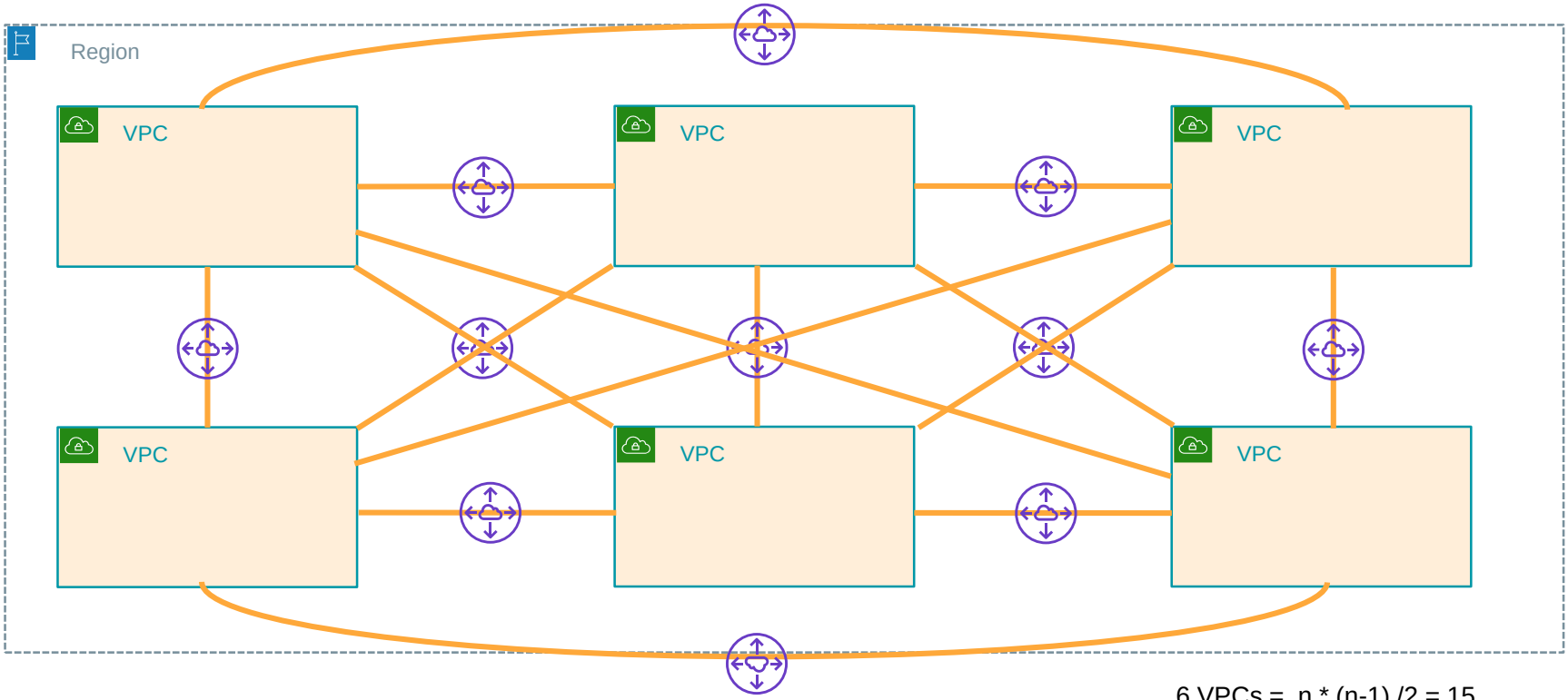
Cleanup

1. Terminate both EC2 instances
2. Delete VPN Connection
3. Dissociate VPN Gateway with VPC
4. Delete VPN Gateway
5. Delete Customer Gateway
6. Delete VPC-DC
7. Delete VPC-AWS

What is Transit Gateway?

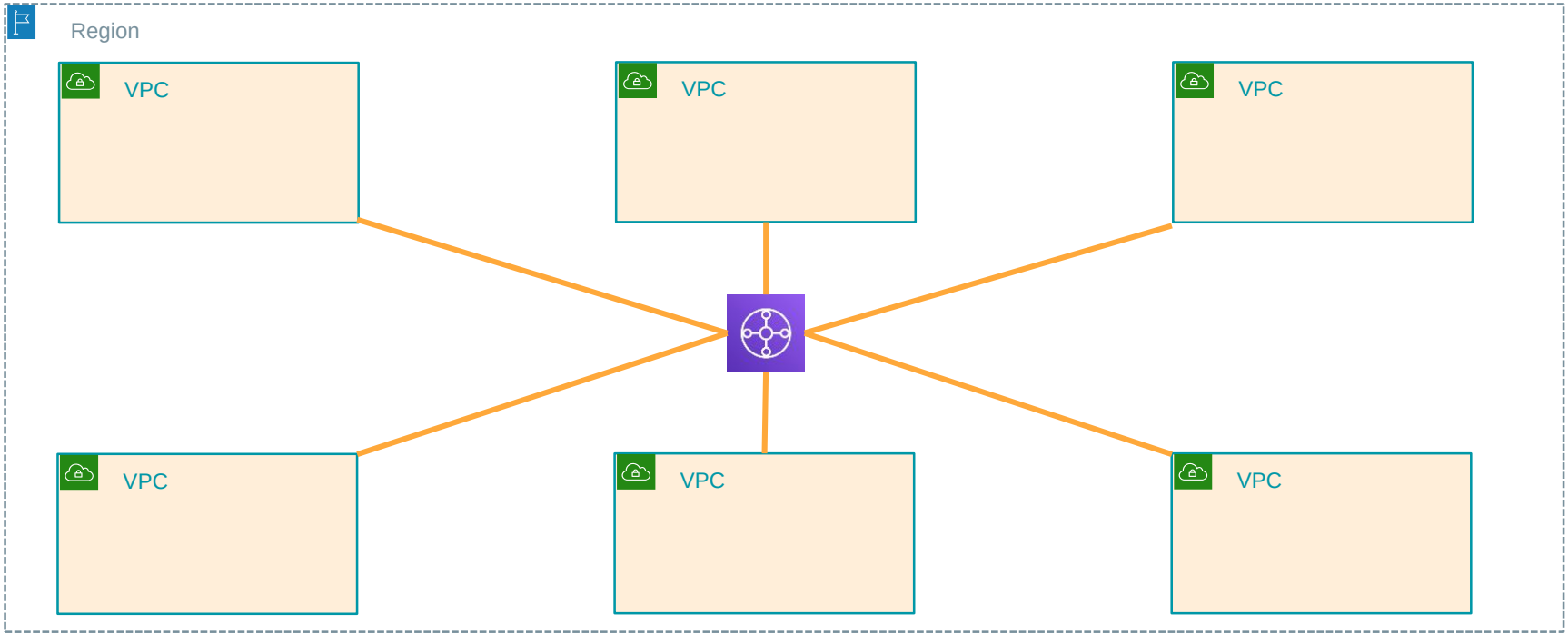
- Allows customers to interconnect thousands of Virtual Private Clouds (VPCs) and on-premises networks.
- Transit Gateway supports attachments to
  - One or more VPCs
  - VPN
  - Direct Connect Gateway
  - Peering connection with another Transit Gateway
  - A Connect SD-WAN/third-party network appliance

Multiple VPCs without Transit Gateway

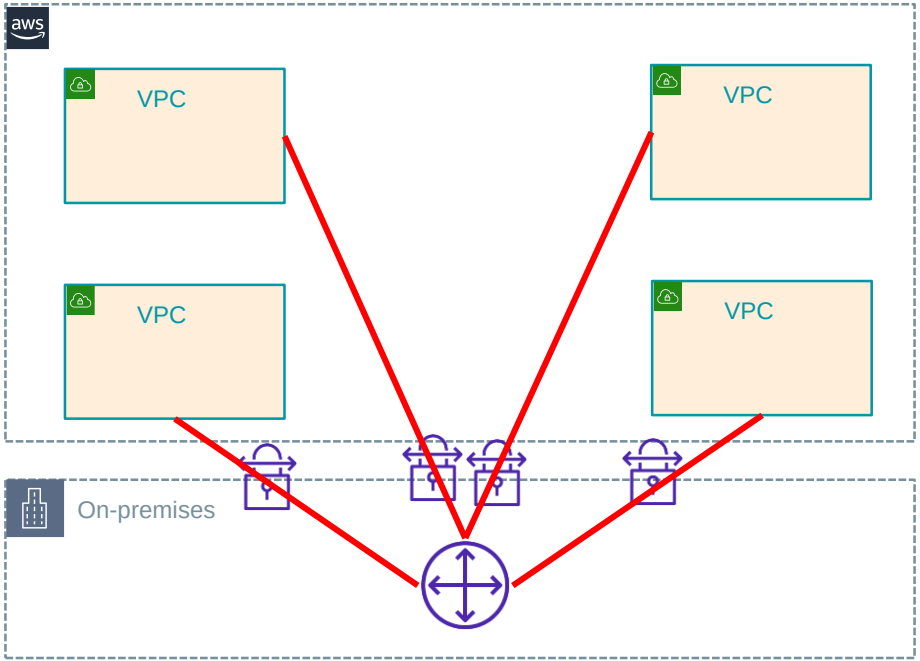


6 VPCs = n * (n-1) /2 = 15

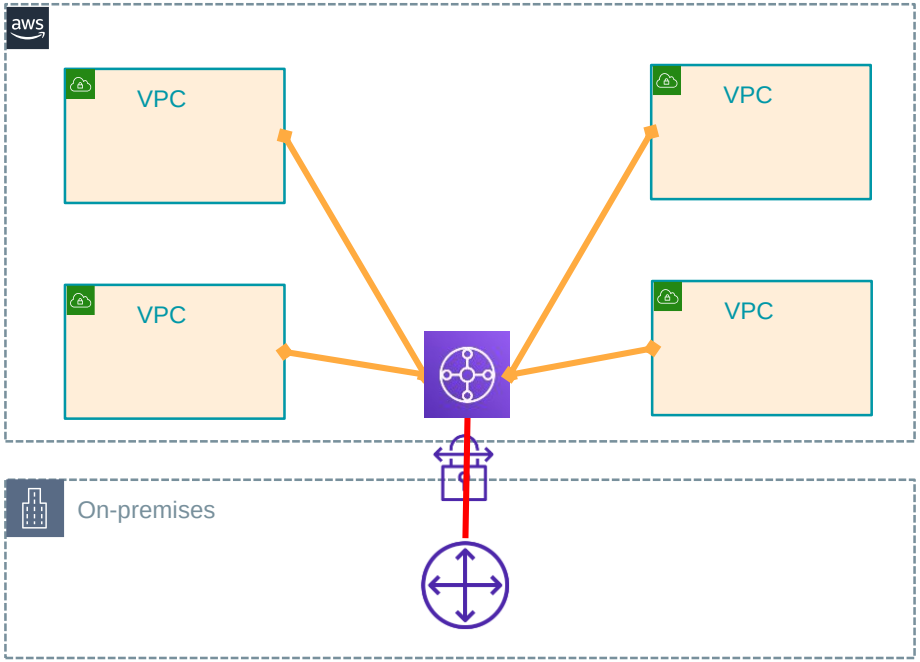
Multiple VPCs with Transit Gateway



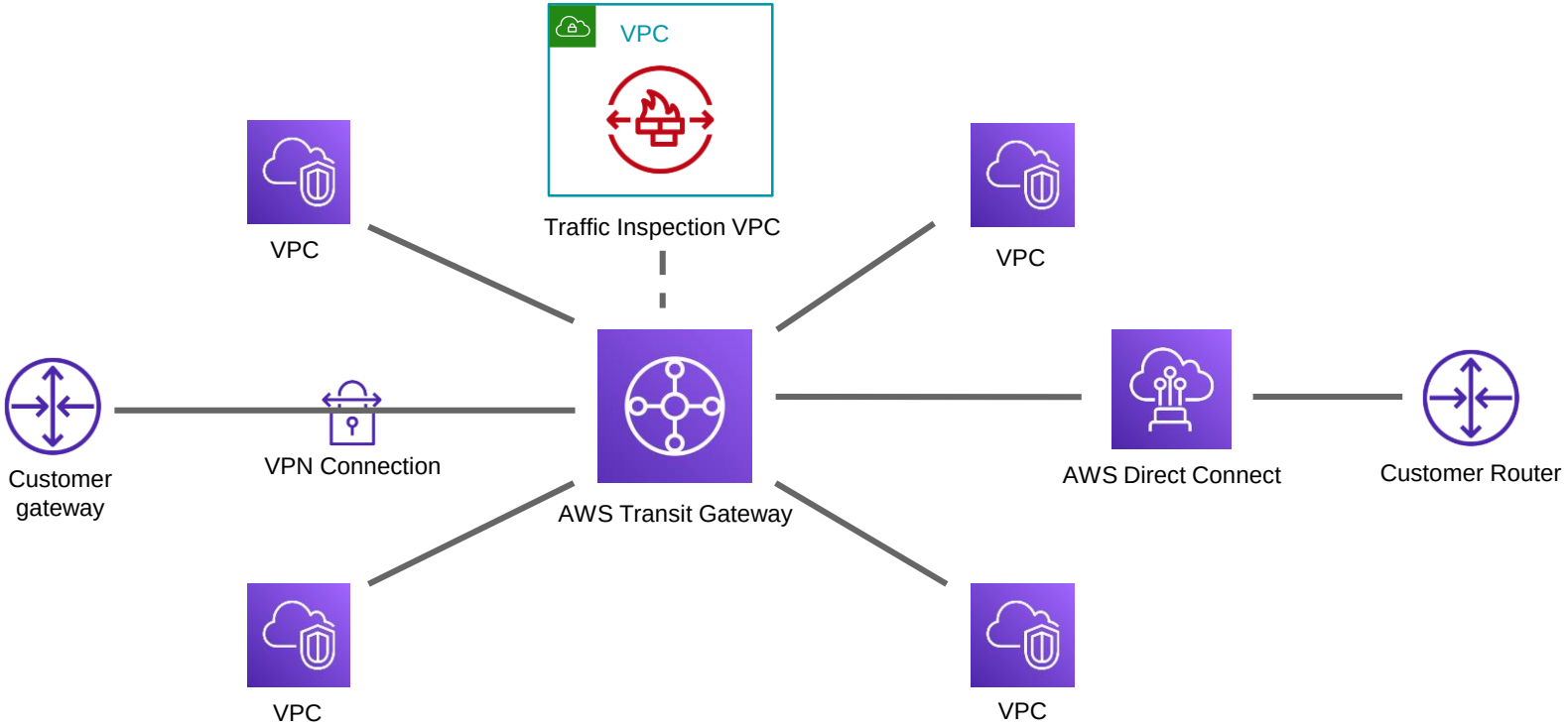
Multiple VPCs and VPN



Multiple VPCs and VPN with Transit Gateway

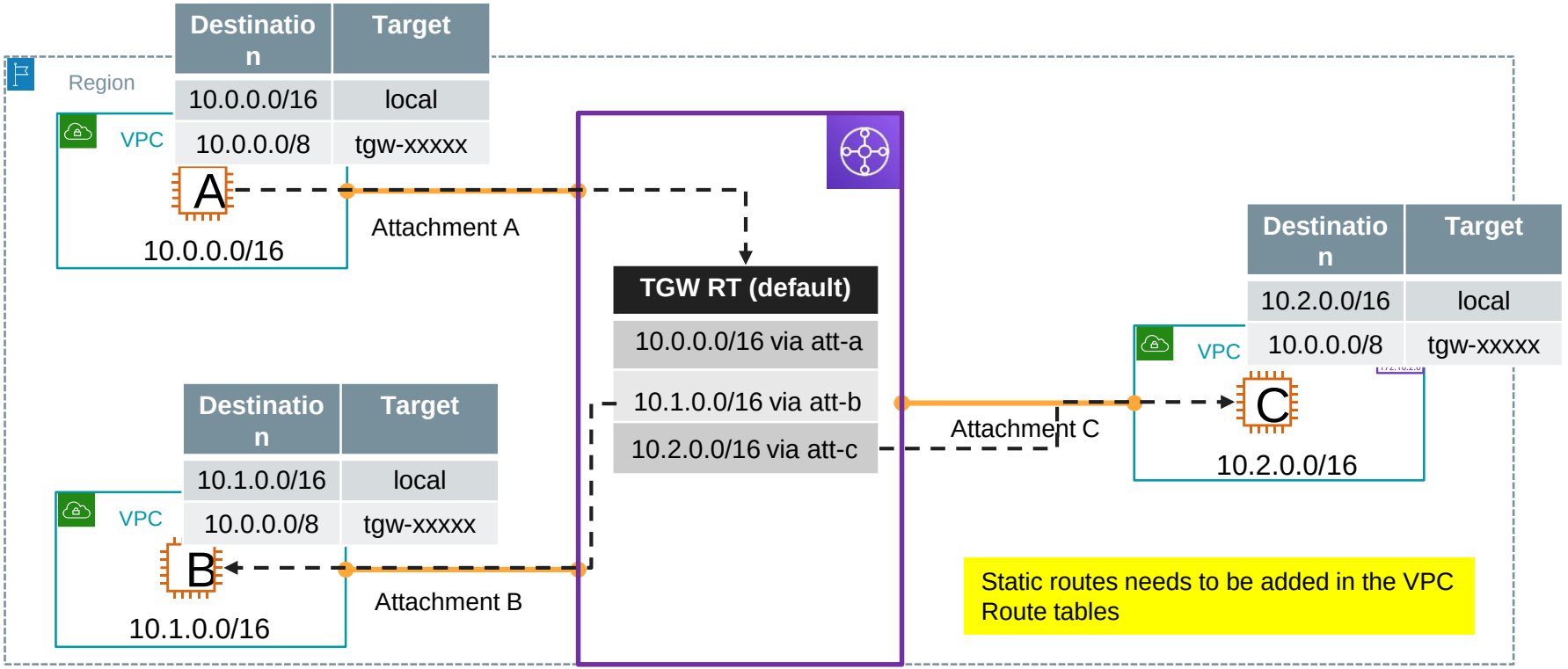


Transit Gateway attachments – VPC, VPN, DX



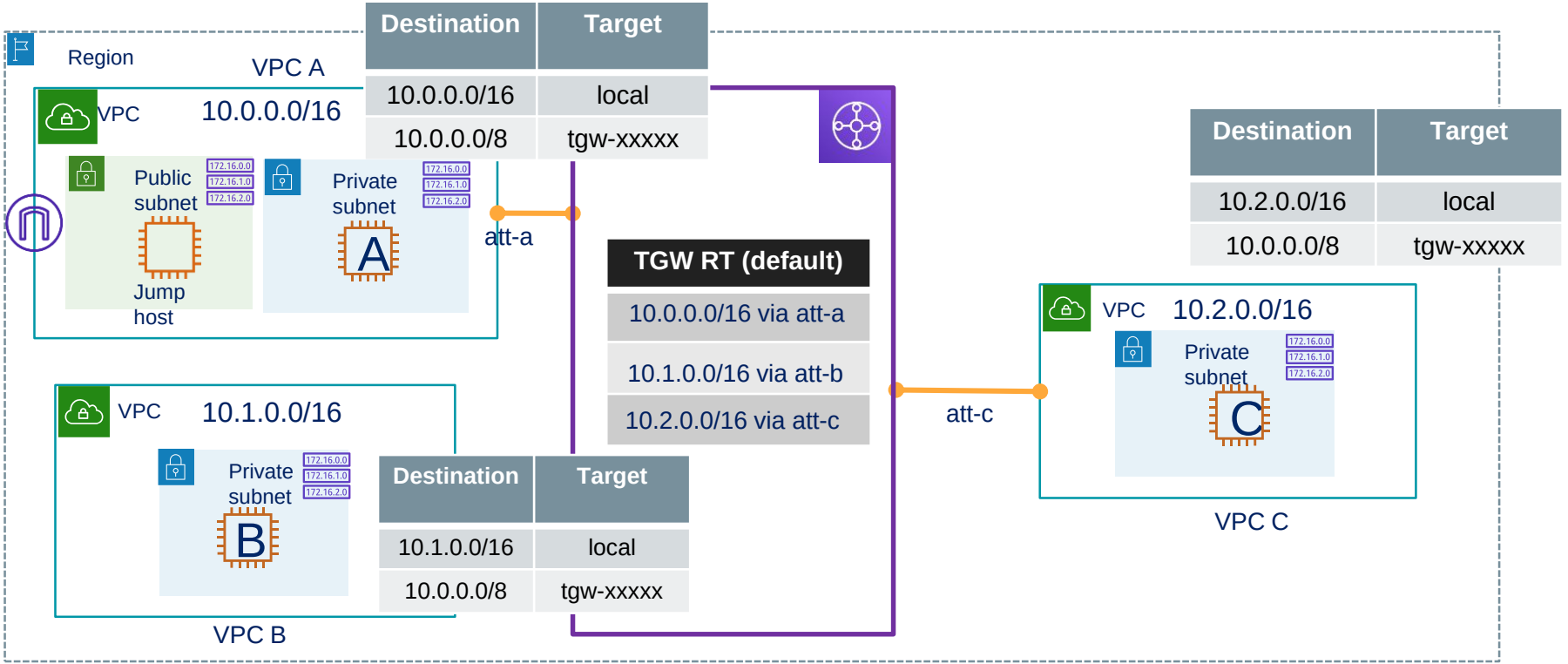
Transit Gateway VPC attachments

VPC Route tables



Transit Gateway Lab – Three VPCs with full connectivity

Lab setup

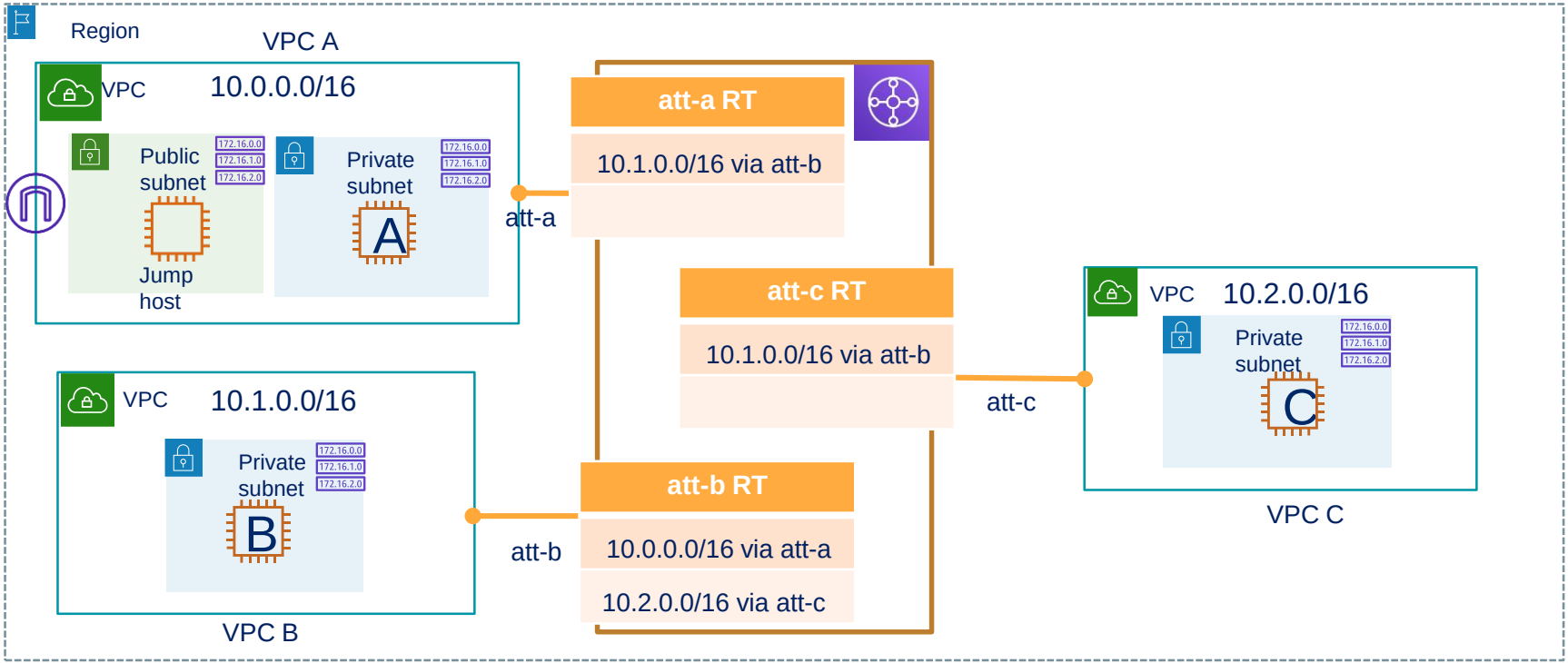


Lab Steps

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

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

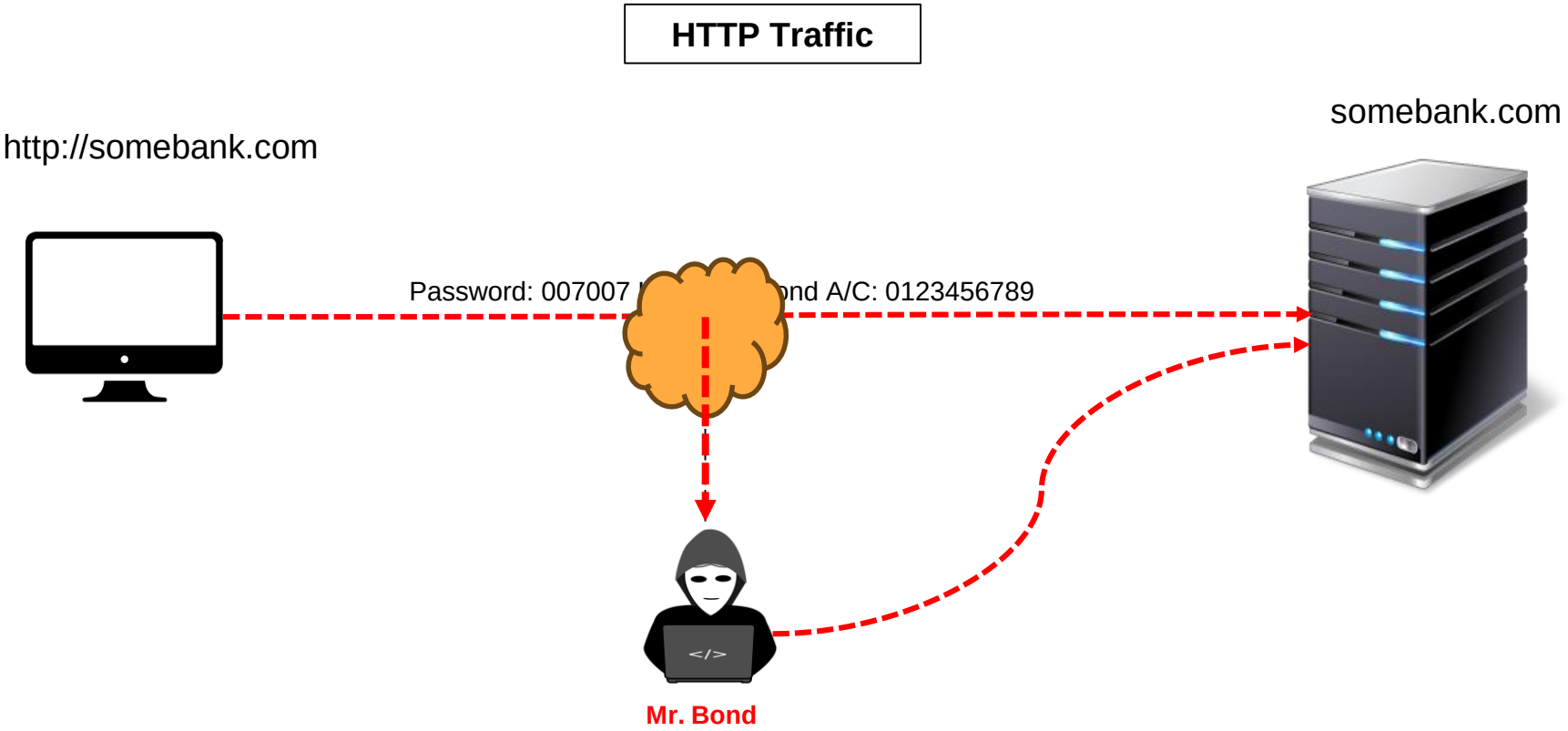
Cleanup – Very important

1. Terminate all four EC2 instances
2. Delete Transit Gateway attachments
3. Delete Transit Gateway
4. Delete all four VPCs

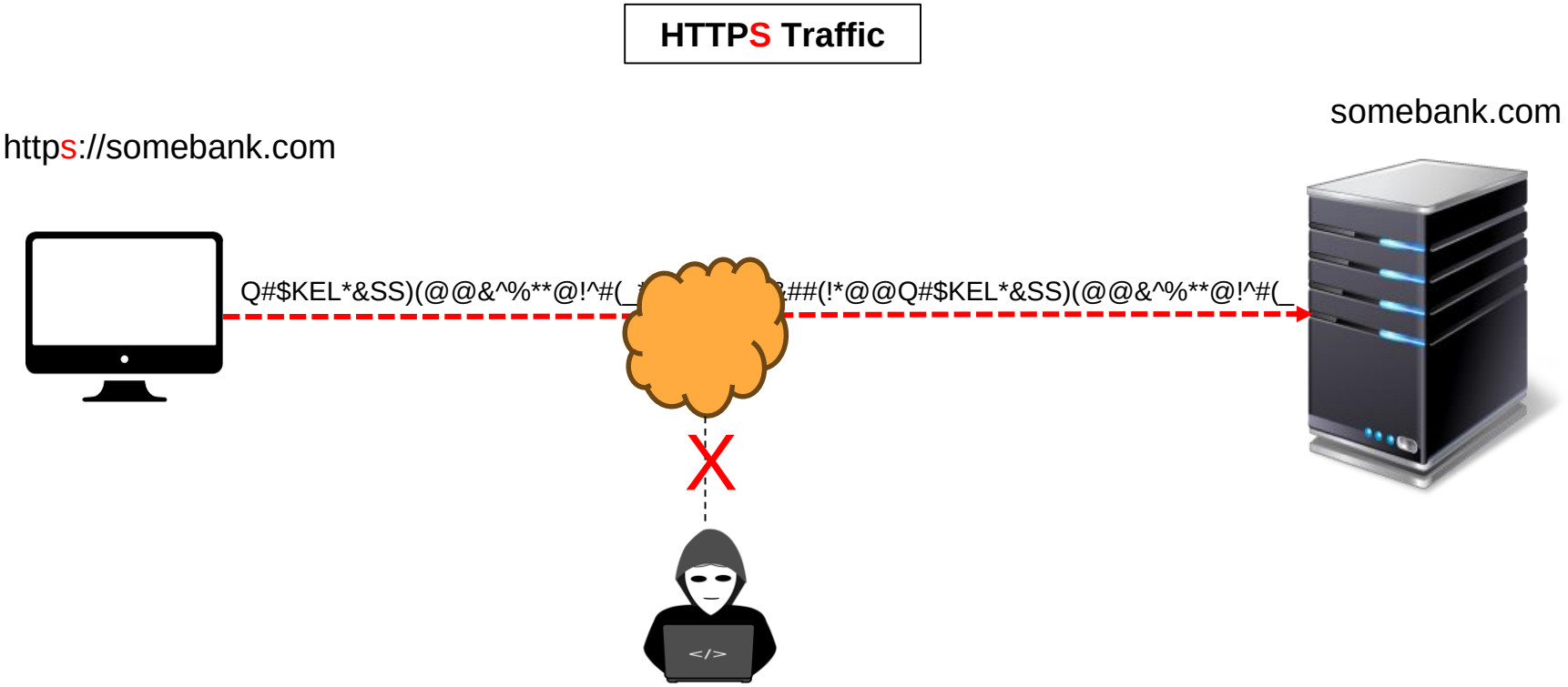
Secure Network with HTTPs and SSL/TLS

How it all works?

Why HTTPS?



Why HTTPS?


HTTPS Traffic
somebank.com
https://somebank.com
Q#$KEL*&SS)(@@&^%**@!^#(_
##(!*@@Q#$KEL*&SS)(@@&^%**@!^#(_
X
</>

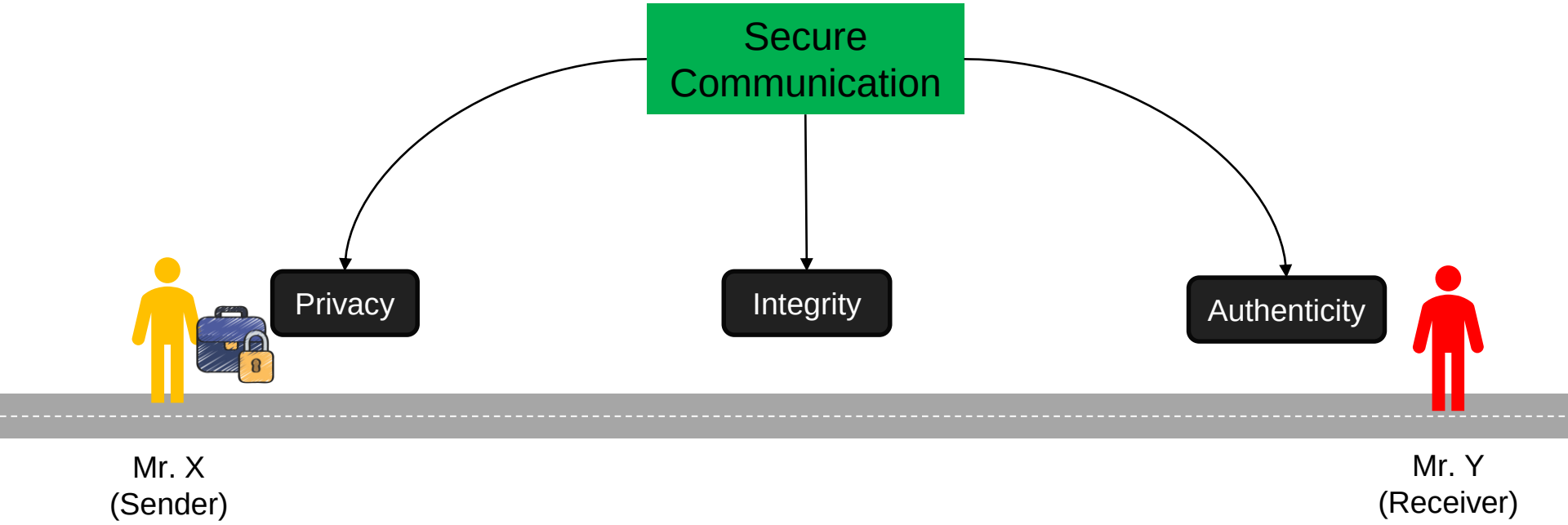
How HTTPS work?

Client

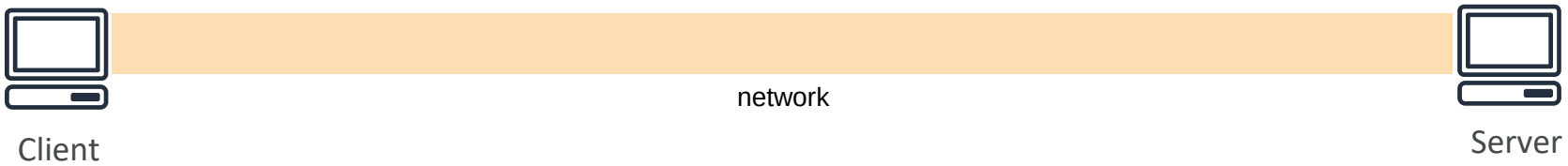
Server

7	Application	HTTPS	7 Application
6	Presentation	SSL/TLS	6 Presentation
5	Session		5 Session
4	Transport		4 Transport
3	Network		3 Network
2	Datalink		2 Datalink
1	Physical	Electronic / Light / Radio signals via Ethernet cable or optical fiber or Air	1 Physical

How SSL/TLS work?



How SSL/TLS work?



How SSL/TLS work?

- Authenticity
- Privacy
- Integrity

Certificate Authority (CA)

CA Signature

Trust store

Server Public Cert

Server Public key

Session key

SSL/TLS

network

Server Public Cert

Server Private Key

Session key

Client

Server

Data+Hash

Encrypt/Decrypt

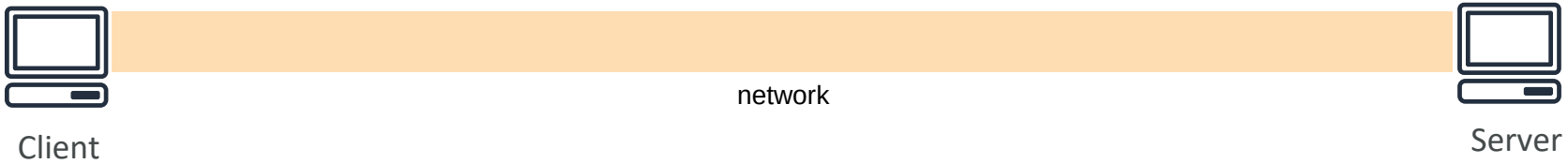
Data Transfer

Decrypt/Encrypt

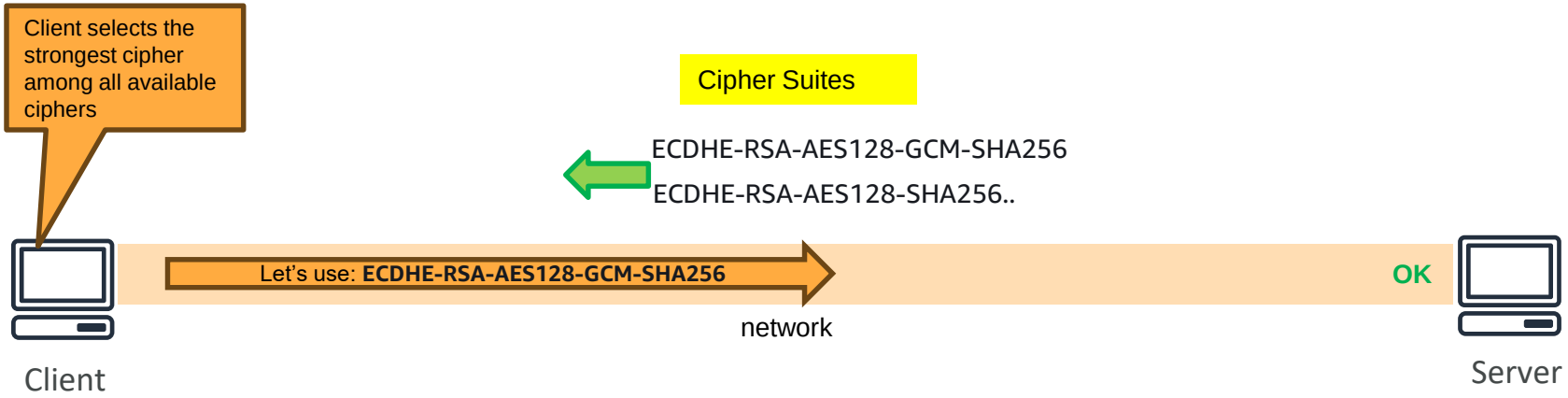
Data+Hash

How SSL/TLS work?

How does Client & Server knows which Encryption algorithms and Hashing algorithms to use?



How SSL/TLS work?



Session key generation algorithm: ECDHE

Asymmetric encryption Algorithm & Key: RSA

Symmetric encryption Algorithm & Key: AES-128, AES-192, AES-128-GCM

Integrity check Algorithm & Key: SHA128, SHA256, SHA384

Putting it together..

Client

1. Client contacts the server – ClientHello
2. Server sends it's TLS Certificate and Public key along with available Cipher suites - ServerHello
3. Client verifies server certificate issuer signature with a local Trusted Root Certificate authority
4. Client and Server negotiate the strongest type of encryption, keys that each can support (Cipher suite)
5. Client encrypts a session key with the server's Public key and sends it to the server
6. Server decrypts a session key with its private key and session is established
7. Session key now used to encrypt and decrypt the data transmitted between client and server

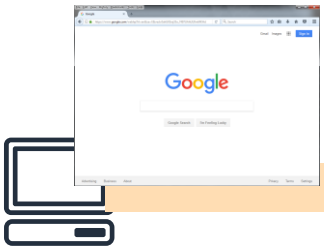
Server

Putting it together..

Certificate Authority (CA)

Trust store

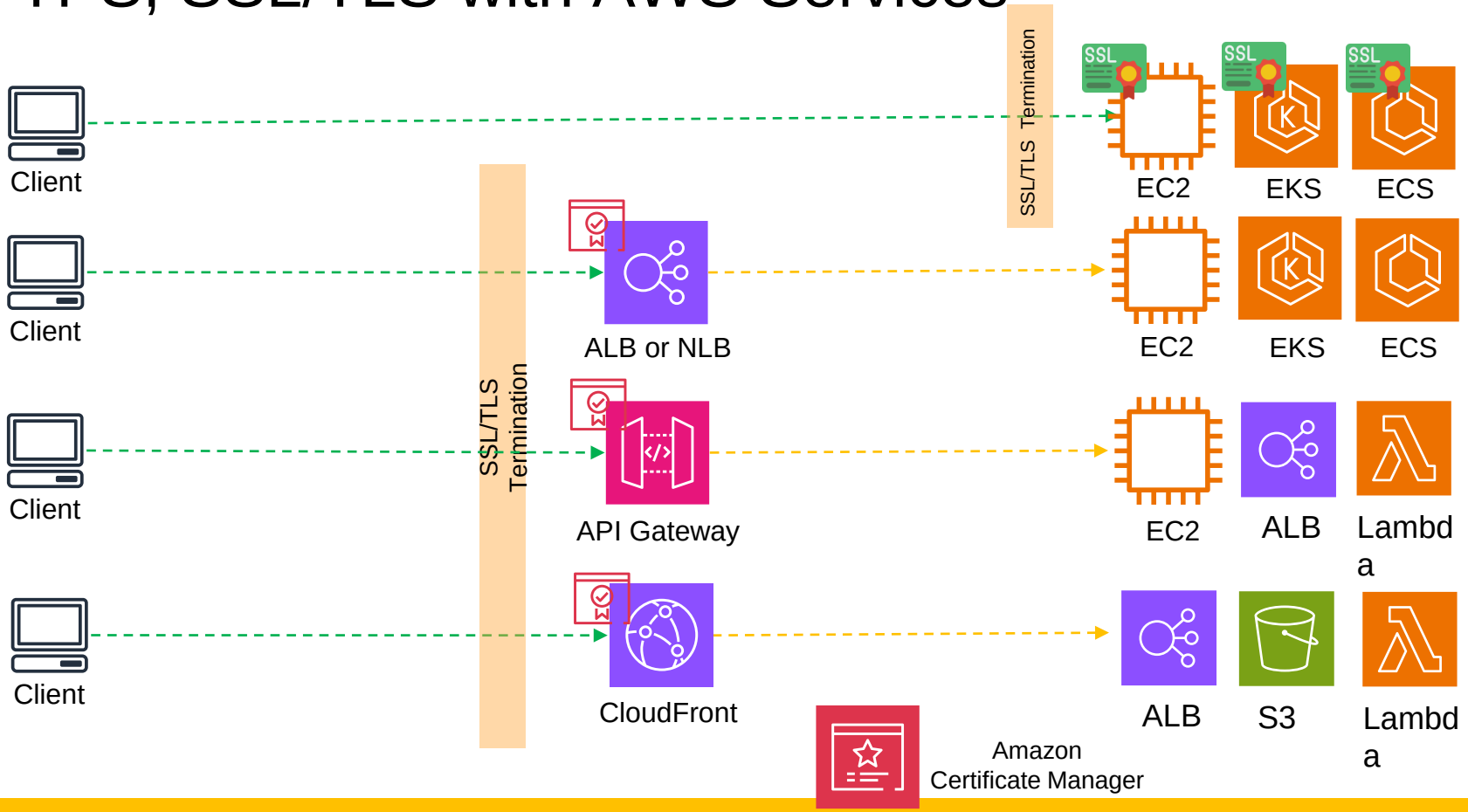
But where is this Trust store in my machine?



Client

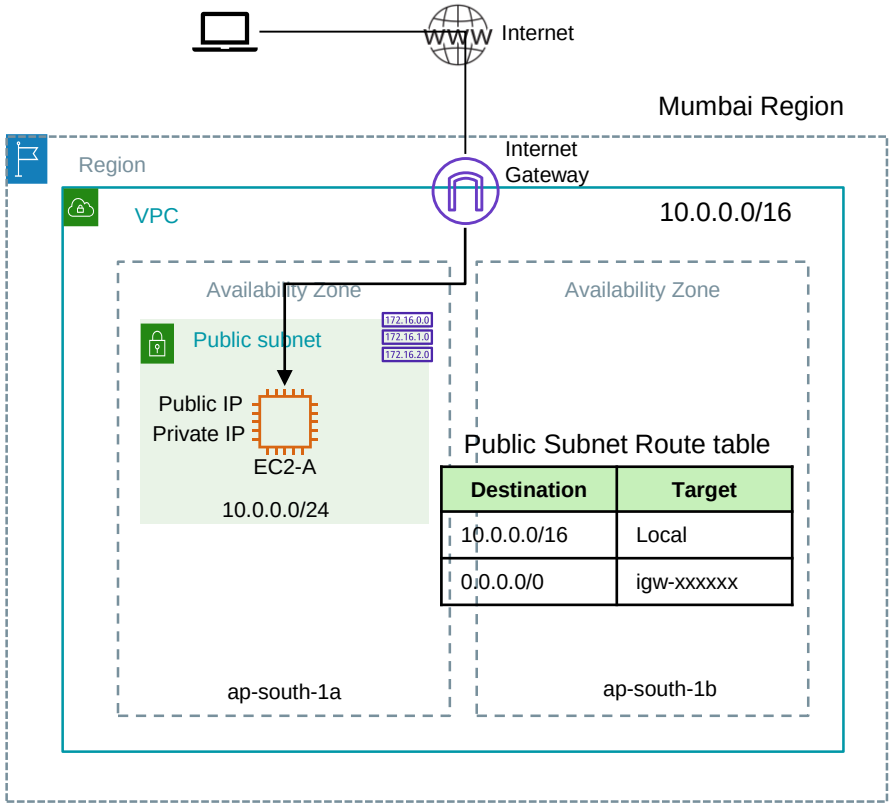
Server

HTTPS, SSL/TLS with AWS Services



Launch a simple Webserver and access using custom domain name

Exercise – A simple web server



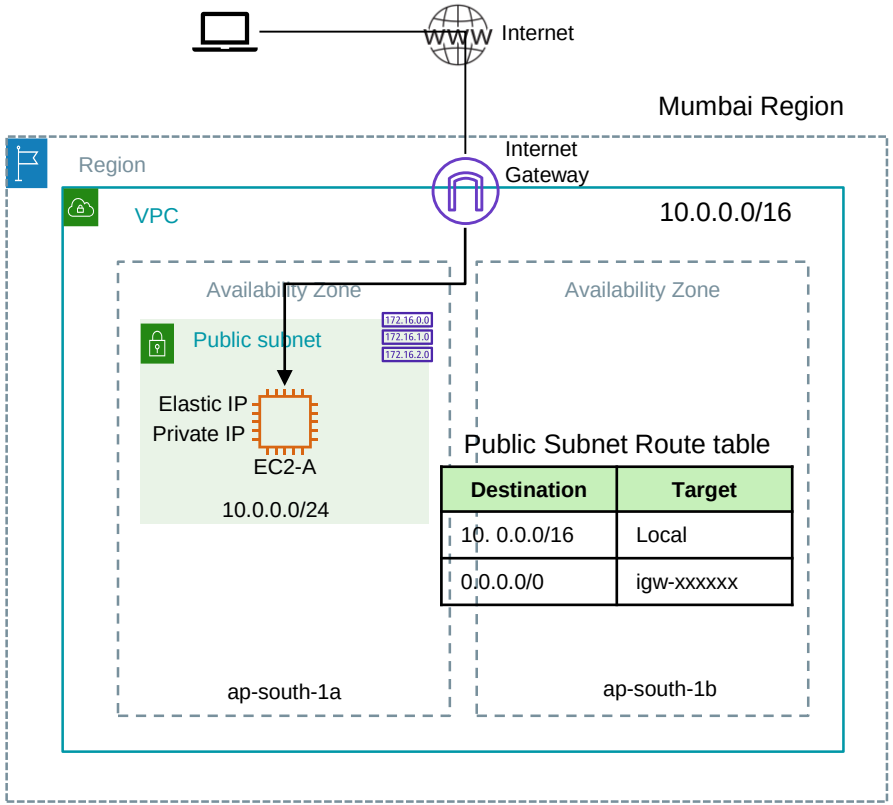
High level steps

1. Setup network as shown
2. Launch EC2 instance and assign Public IP. Allow SSH and HTTP in security group.
3. Connect to instance over SSH and install a httpd web server. Refer commands from the next slide.
4. Allocate Elastic IP and assign it to EC2 instance.
5. Verify that you can access web page over the internet using EC2 Elastic IP

Web server installation steps

- Connect to EC2 instance over SSH and execute following commands:
  sudo yum install httpd -y
  sudo systemctl start httpd.service
  sudo systemctl enable httpd.service
- Create /var/www/html/index.html and add some text e.g. <h1>This is a webserver</h1>
- Save the file

Exercise – Access webserver using custom domain name



Continuing from earlier setup

Pre-requisite for following steps:

- You should have your public domain name and DNS pointing to Route53 public hosted zone
- Refer Labs prerequisites section if you haven't done this earlier.

1. In Route53 Public hosted zone, create A record pointing to EC2 Public IP
2. Access webserver using your custom domain name.

Clean-up

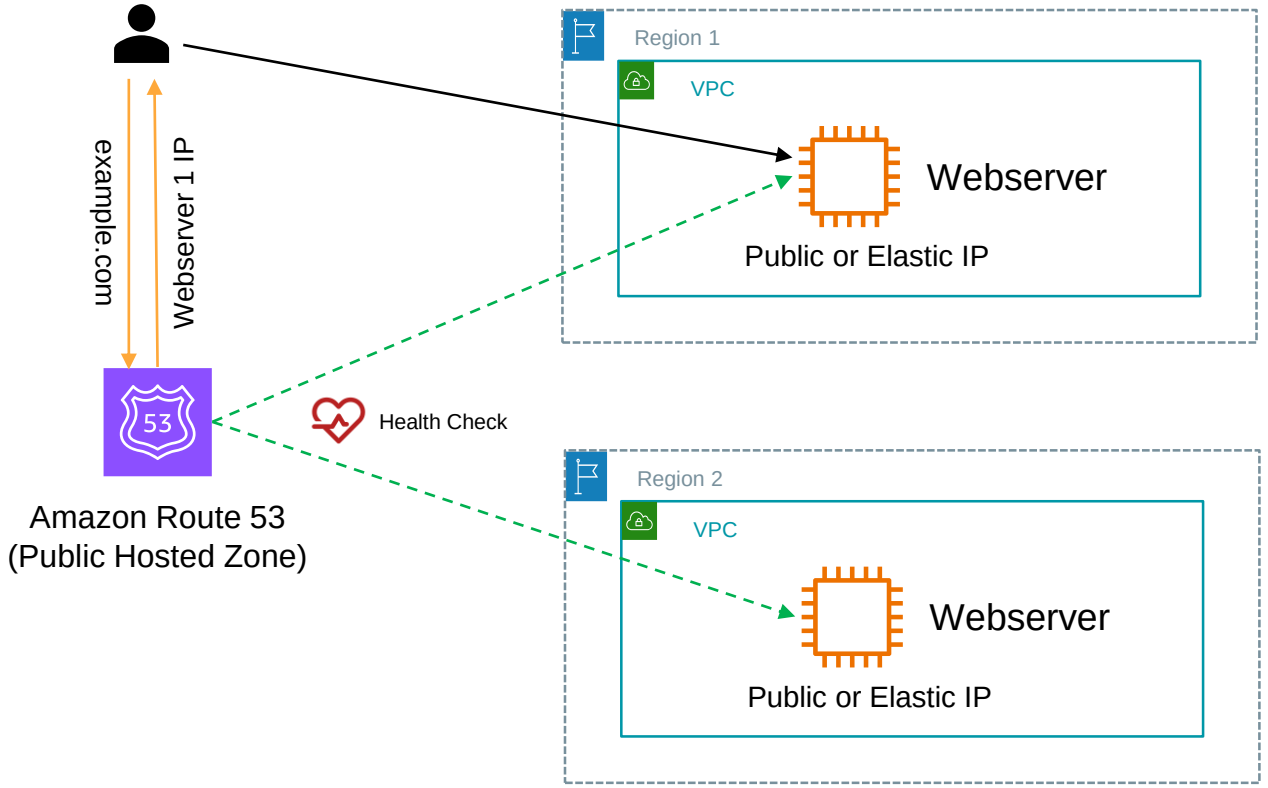
1. Terminate EC2 instance
1. Release Elastic IP

Route53 for Region level failover

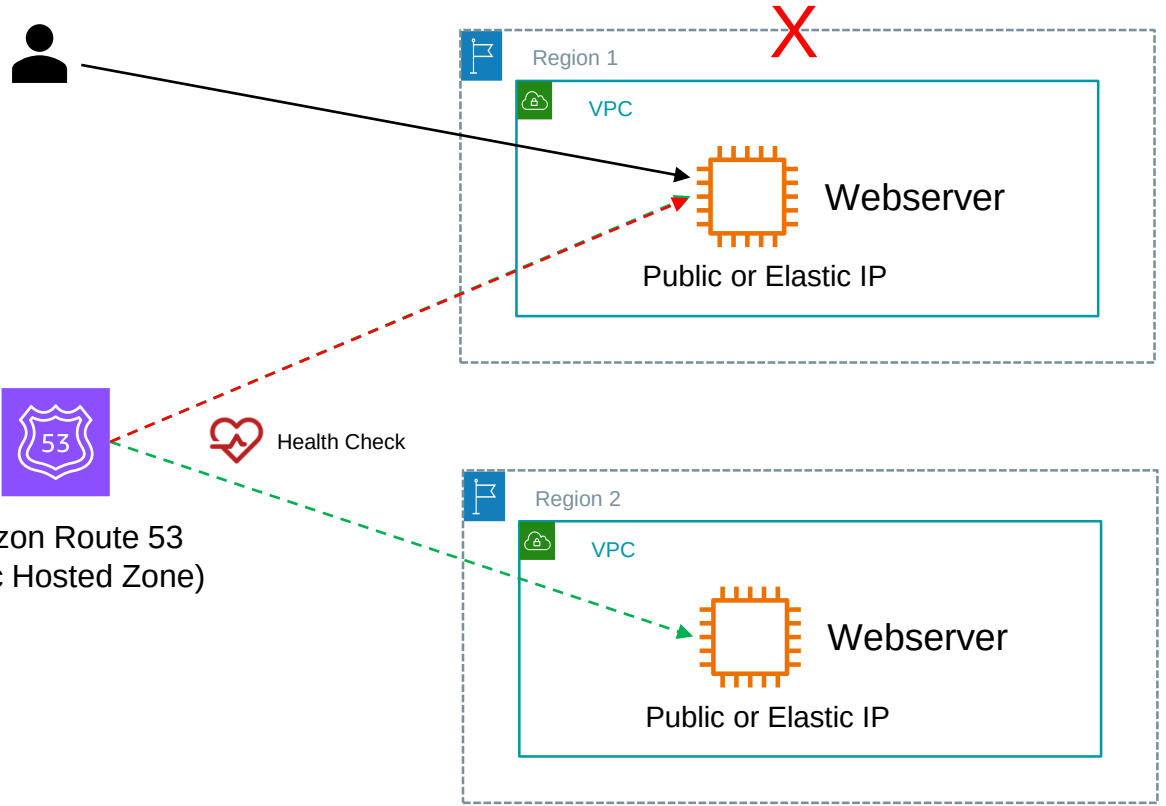
Failover Routing policy

Primary = Web Server 1 IP

Secondary = Web Server 2 IP



Route53 for Region level failover



Failover Routing policy

Primary = Web Server 1 IP

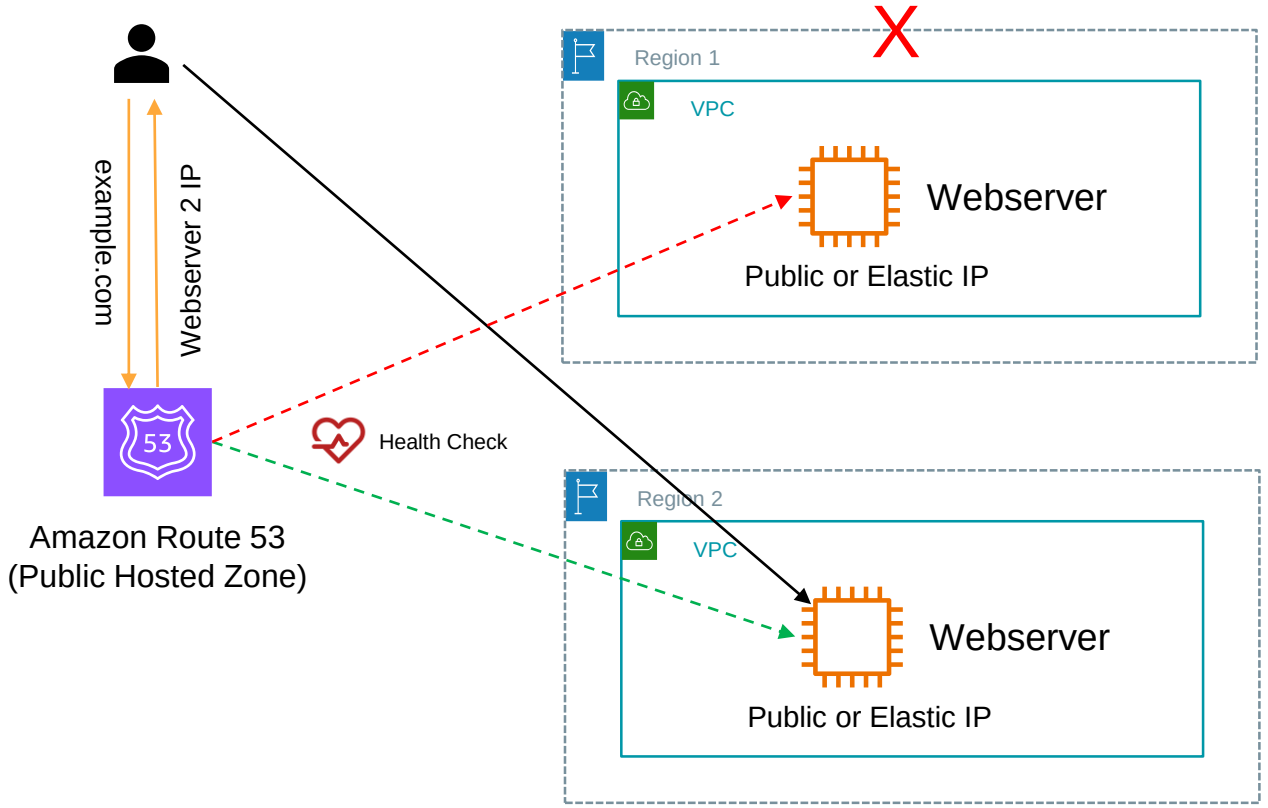
Secondary = Web Server 2 IP

Route53 for Region level failover

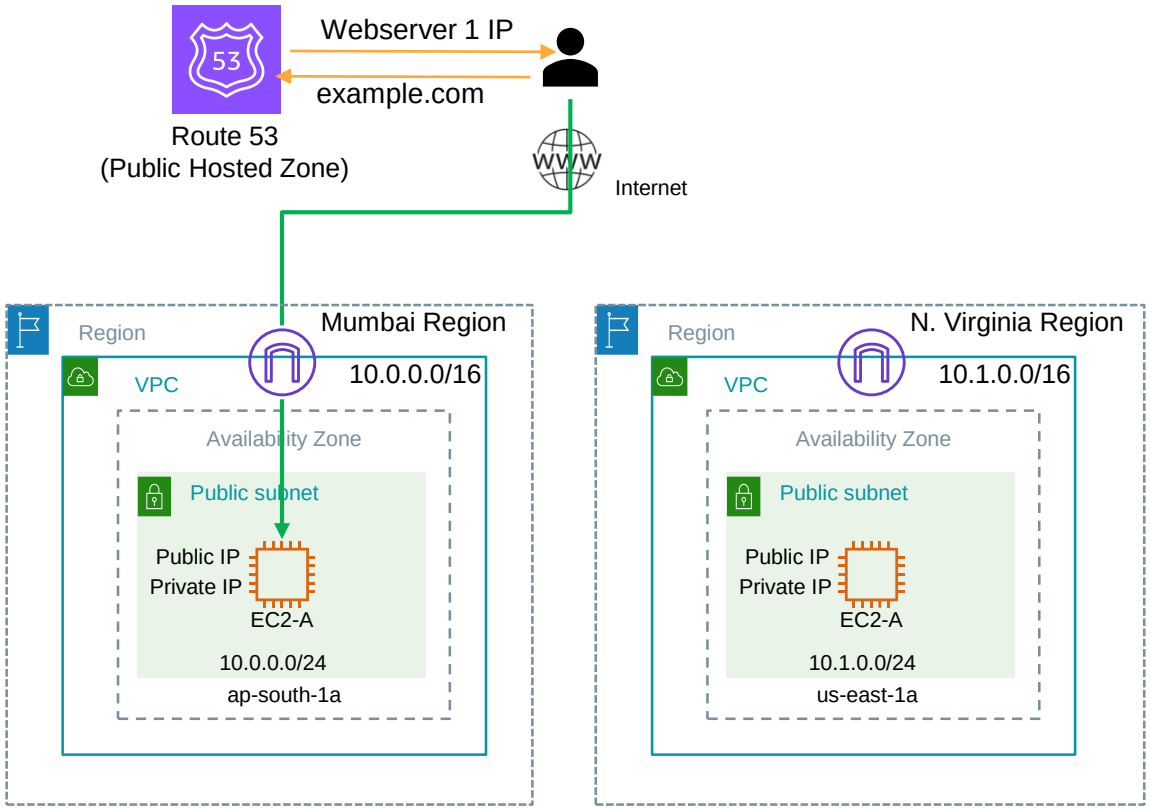
Failover Routing policy

Primary = Web Server 1 IP

Secondary = Web Server 2 IP



Exercise – Region level failover using Route53



1. Create VPC & Subnet network as shown
2. Launch Web server 1 in Mumbai region and Web server 2 in N.Virginia region using EC2 userdata from next slide. Verify that webserver are accessible over the web.
3. In Route53 create health checks for both the servers. User server Public IP, port 80 and /index.html path.
4. In Route53 create two A records – One as a primary pointing to Mumbai & other as a secondary pointing to N. Viriginia Web server Public IP. Use health-checks and short TTL (60 sec)
5. Stop the Mumbai region Webserver. Wait for TTL to expire. The DNS should failover to N. Virginia webserver.

Clean-up: Terminate both EC2 instances

Exercise – Region level failover using Route53

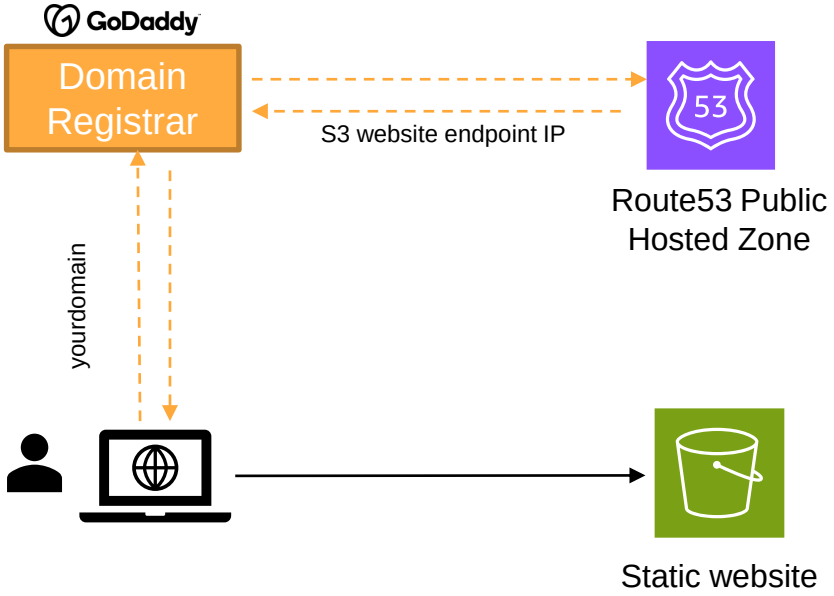
Userdata for Webserver 1 in Mumbai region:

```
#!/bin/bash
yum install httpd -y
systemctl start httpd.service
systemctl enable httpd.service
echo "<h1>This website is hosted in AWS Mumbai Region</h1>" > /var/www/html/index.html
echo "Configured successfully"
```

Userdata for Webserver 2 in N. Virginia region:

```
#!/bin/bash
yum install httpd -y
systemctl start httpd.service
systemctl enable httpd.service
echo "<h1> This website is hosted in AWS North Virginia Region </h1>" > /var/www/html/index.html
echo "Configured successfully"
```

Exercise: Hosting a static website on S3 over http



Pre-requisites

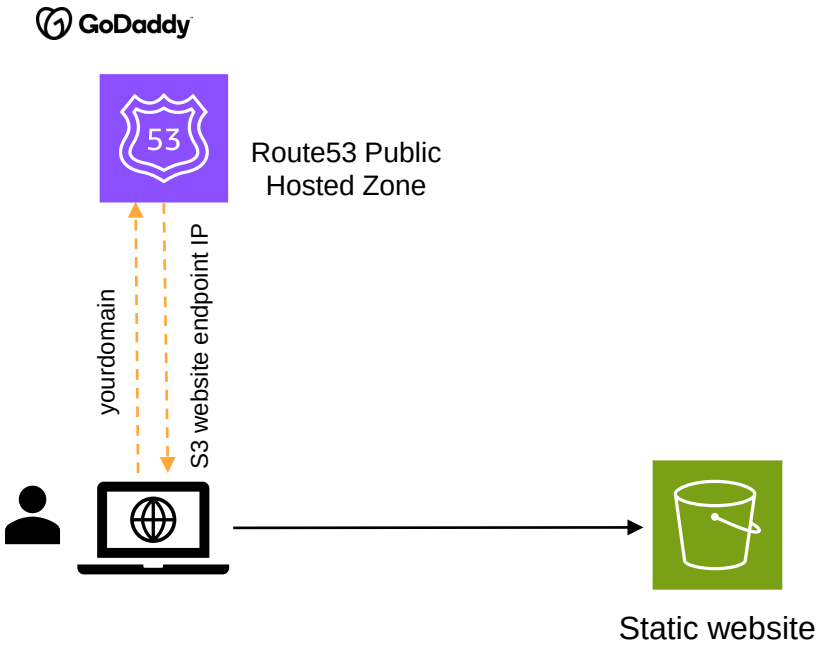
1. Your own Public domain.

If you purchased this domain name from other domain name providers e.g. Godaddy, namecheap etc.

2. Create Amazon Route53 Public Hosted Zone for same domain and update Rout53 Nameservers into your domain registrar DNS settings.

Note:
Refer the course prerequisites section video for how to perform this Step 2.

Exercise - Hosting a static website on S3 over http



1. Create a S3 buckets for your website. We will host website with e.g. www.spacemission.in
2. Upload static website content to your www bucket
3. Enable bucket for website hosting with index document as index.html
4. Modify bucket permissions to allow Public access. Unblock “Block all public access” settings and add bucket policy.
5. Access bucket using http://bucket-name.s3-website-region.amazonaws.com
6. In Route 53 Public Hosted Zone, create an alias record for your www domain name pointing to S3 bucket endpoint. Access website using http://www.domain-name
7. (Optional) Configure the root domain S3 bucket to redirect requests to www subdomain. Create an alias record in Route53 and point root domain to this bucket website endpoint

Steps

1. Go to S3 console and create S3 bucket with www subdomain
    a. Go to S3 console
    b. Create Bucket -> Bucket Name: www.mydomain.com, Region: ap-south-1 -> Create bucket
2. Upload any static html content to www S3 bucket
    a. Download the sample website contents from https://github.com/chetanagrawal/aws-networking-exercises/raw/main/S3/sample-website.zip
    b. Unzip it locally and upload all the files and folders directly inside the bucket. Do not upload under any subfolder. index.html should be directly inside the bucket and not under any folder.
3. Enable website hosting for S3 bucket
    1. Go to www bucket -> Properties -> Static Website Hosting -> Edit -> Enable
    2. Specify index.html as index document -> Save changes
4. Modify Bucket permissions to allow Public Access
    1. Go to bucket -> Permissions -> Block public access (bucket settings) -> Edit -> Uncheck “Block all public access” -> Save changes
    2. Bucket Policy -> Edit -> Paste the bucket policy from next page. Make sure to replace your bucket name.

Bucket policy

```
{
  "Version":"2012-10-17",
  "Statement":[{
        "Sid":"PublicReadGetObject",
        "Effect":"Allow",
        "Principal": "*",
      "Action":["s3:GetObject"],
      "Resource": ["arn:aws:s3:::your-bucket-name/*" ]
   }
  ]
}
```

Replace your-bucket-name with your actual bucket name

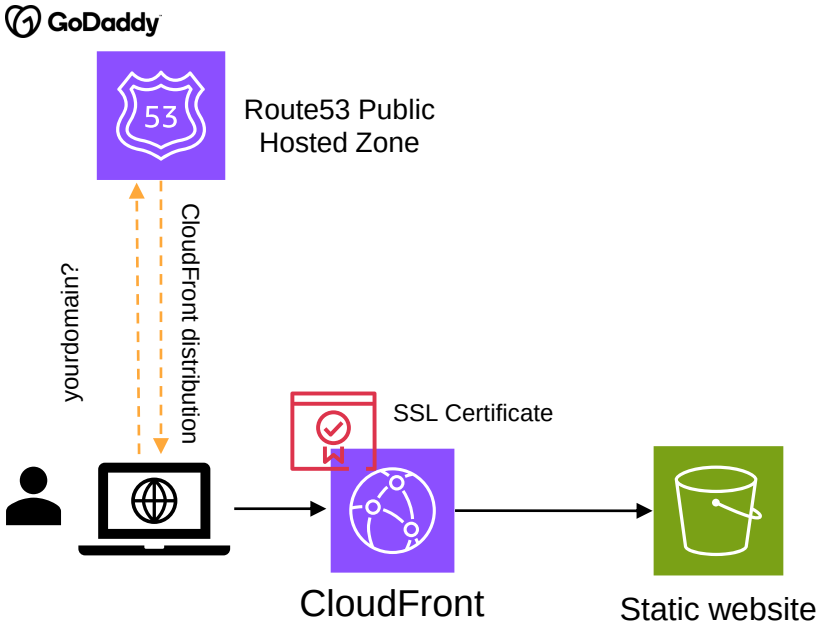
Steps

5. Access website using AWS provided s3 bucket endpoint.
    a. Open your browser and try accessing http://bucket-name.s3-website-region.amazonaws.com
    b. Should be able to see your *space* website if everything done properly
6. Create DNS record in Route53
    a. It's a prerequisite for this step to have your own domain name and linked to Route 53 Public Hosted Zone. If you haven't done this earlier, refer pre-requisite section videos for details and then proceed with following steps.
    b. Go to Route 53 console -> Hosted Zones -> select your domain name.
    c. Create record -> Toggle Alias (Should be ON) -> Route traffic to -> Select S3 website endpoint -> Select the region in which you have created your bucket -> Create records
7. Access website using your domain name.
    a. Open your browser and try accessing http://www.yourdomain
    b. Should be able to see your space website if everything done properly

Steps

8. Redirect your root domain traffic (yourdomain) to www subdomain (www.yourdomain)
    1. Create another bucket with name yourdomain
    2. Go to Bucket -> Properties -> Static Website Hosting -> Edit -> Enable
    3. Redirect requests for an object -> enter www.yourdomain -> Save changes
    4. Repeat the steps for Route53 and create another record for yourdomain pointing to this bucket.
    5. Try accessing website using http://yourdomain. Browser should be redirected to http://www.yourdomain and website should be accessible using main domain.

Exercise - Hosting a static website on S3 over https



Continuing from the earlier setup

1. Create TLS certificate for your domain name and www subdomain using Amazon Certificate Manager (ACM) in **N.Virginia region**
2. Create CloudFront distribution with www website endpoint as a origin. Set viewer protocol policy to redirect HTTP to HTTPS. Set the alternate domain names (CNAME) for your www subdomain.
3. Update Route53 record for www to point to CloudFront distribution
4. Access website using your domain name. It should redirect to https://www.domainname and website should be displayed

Steps

1. Create SSL/TLS certificate
    1. Go to Amazon Certificate Manager Console -> **Change region to N.Virginia**
    2. Request Public Certificate -> enter your domain name e.g. www.yourdomain -> Request
    3. For domain verification, choose Create records in Route53 -> Create records
    4. Wait for up to 5 mins for Domain verification to be successful.
2. Go to CloudFront console and create distribution with origin as S3 website endpoint (www)
    a. CloudFront console -> Create Distribution -> Origin domain: select www s3 bucket endpoint -> Use website endpoint
    b. Viewer Protocol Policy -> Redirect HTTP to HTTPS
    c. Setting -> Alternate domain name -> Add item -> add www.yourdomain
    d. Select Custom SSL Certificate -> Choose the certificate you created in Step 1.
    e. Create distribution
3. Create DNS record in Route53
    a. Go to Route 53 console -> Hosted Zones -> select your domain name.
    b. Edit record for www subdomain -> Route traffic to -> Alias to CloudFront distribution -> Select your CloudFront distribution -> Save

Cleanup

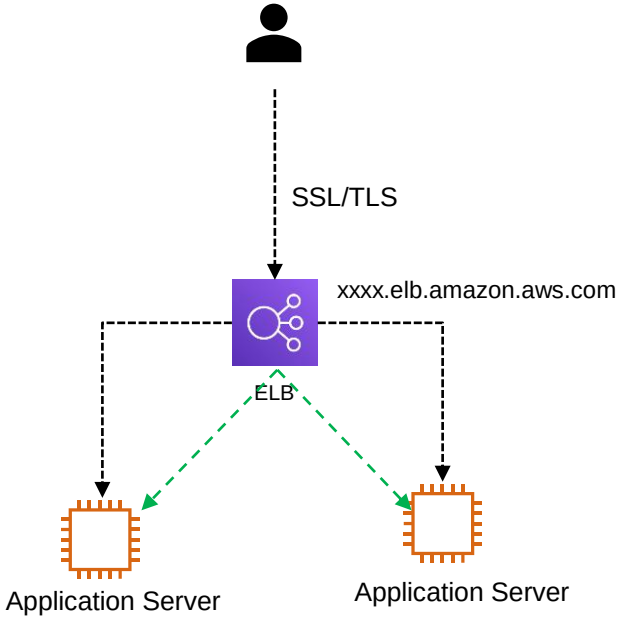
If you don’t want this website

1. Delete CloudFront Distribution
2. Delete both S3 buckets
3. Delete Route53 records
4. Delete Route53 Public Hosted Zone (Otherwise there will be $0.5 per month cost for this hosted zone)

AWS Elastic Load Balancer

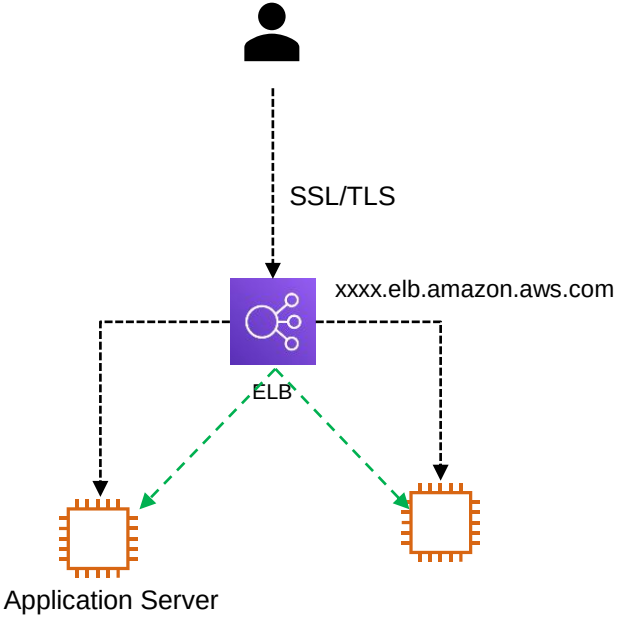
AWS Elastic Load Balancer

- Load Balancer distributes incoming traffic to multiple downstream servers (e.g. EC2 instances, Containers, IP addresses and Lambda functions)
- Supports protocols like HTTP, HTTPS, HTTP/2, TCP, UDP, gRPC
- Expose a single point of access (DNS) to your application
- Seamlessly handle failures of downstream instances by performing health checks to the instances
- Provide SSL/TLS termination for your websites and web applications
- High availability across Availability zones
- Separate public traffic from private traffic
- Supports various routing mechanisms



Why to use ELB?

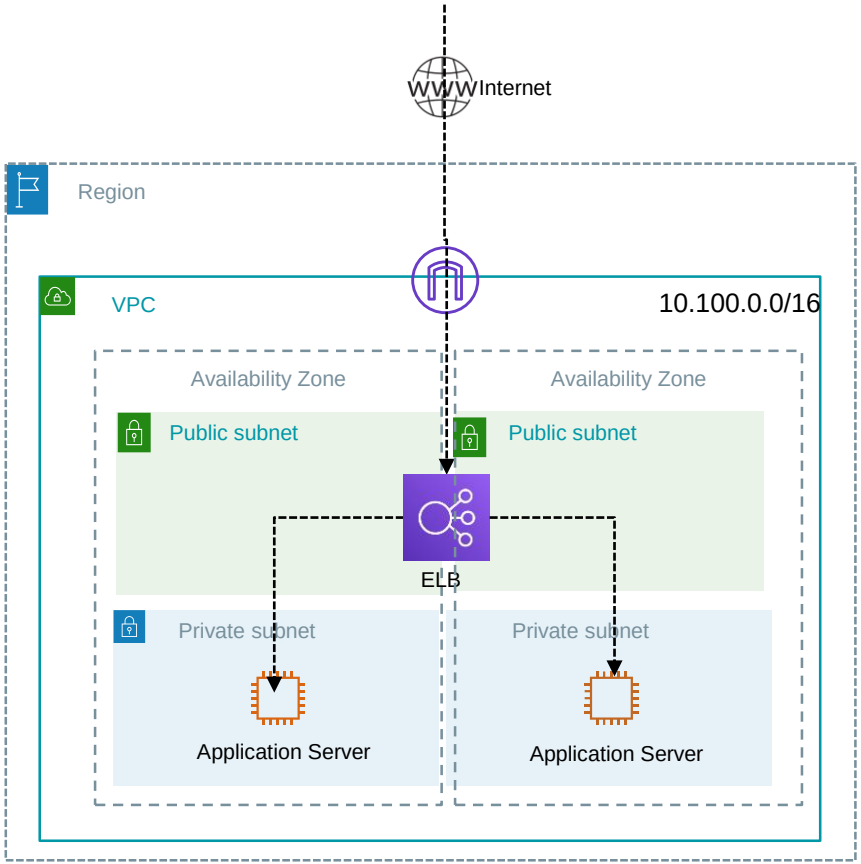
- It’s a fully managed AWS service which means AWS takes care of upgrades, maintenance, high availability , reliability and scaling
- AWS provides configurations and mechanism for traffic engineering
- We can also setup our load balancer on EC2 instance(s) but we have to then take care of high availability, scalability and maintenance.
- ELB is integrated with AWS services
    - EC2
    - Auto Scaling Groups
    - Amazon ECS
    - AWS Certificate Manager (ACM)
    - Amazon CloudWatch
    - Amazon Route 53
    - AWS WAF
    - AWS Global Accelerator



Types of Load Balancers

AWS offers following types of Load Balancers

1. **Classic Load Balancer** (CLB)
   - HTTP, HTTPS, TCP, SSL/TLS (secure TCP)
2. **Application Load Balancer** (ALB)
   - HTTP, HTTPS, WebSocket
3. **Network Load Balancer** (NLB)
   - TCP, TLS (secure TCP), UDP
4. **Gateway Load Balancer** (GWLB)
   - Operates at layer 3 – IP Protocol



OSI Network Layers

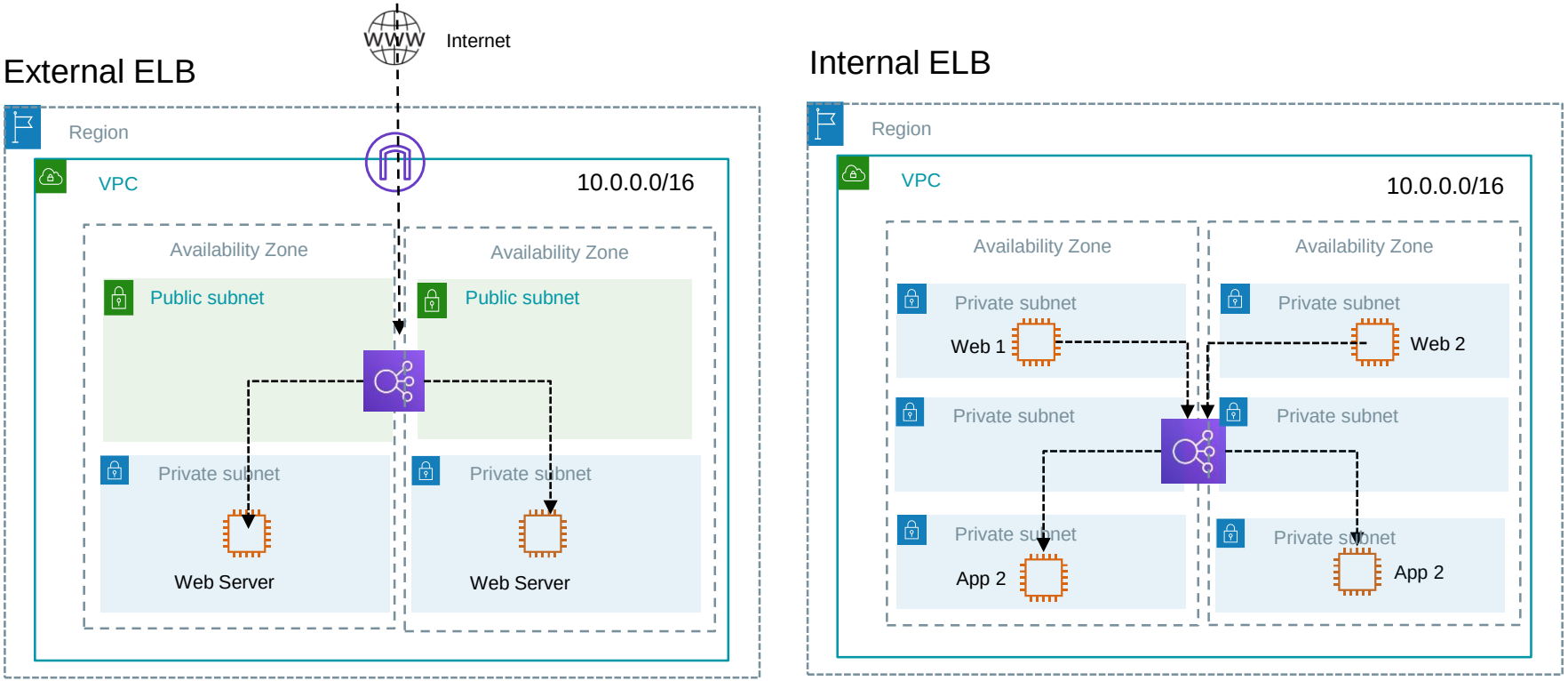
Client

Server

7 Application	 Application Load Balancer	**HTTP, HTTPS, HTTP/2, gRPC**		7 Application
6 Presentation				6 Presentation
5 Session		**SSL/TLS**		5 Session
4 Transport	 Network Load Balancer	**TCP, UDP**		4 Transport
3 Network	 Gateway Load Balancer	**IP**		3 Network
2 Datalink				2 Datalink
1 Physical	Electronic / Light / Radio signals via Ethernet cable or optical fiber or Air			1 Physical

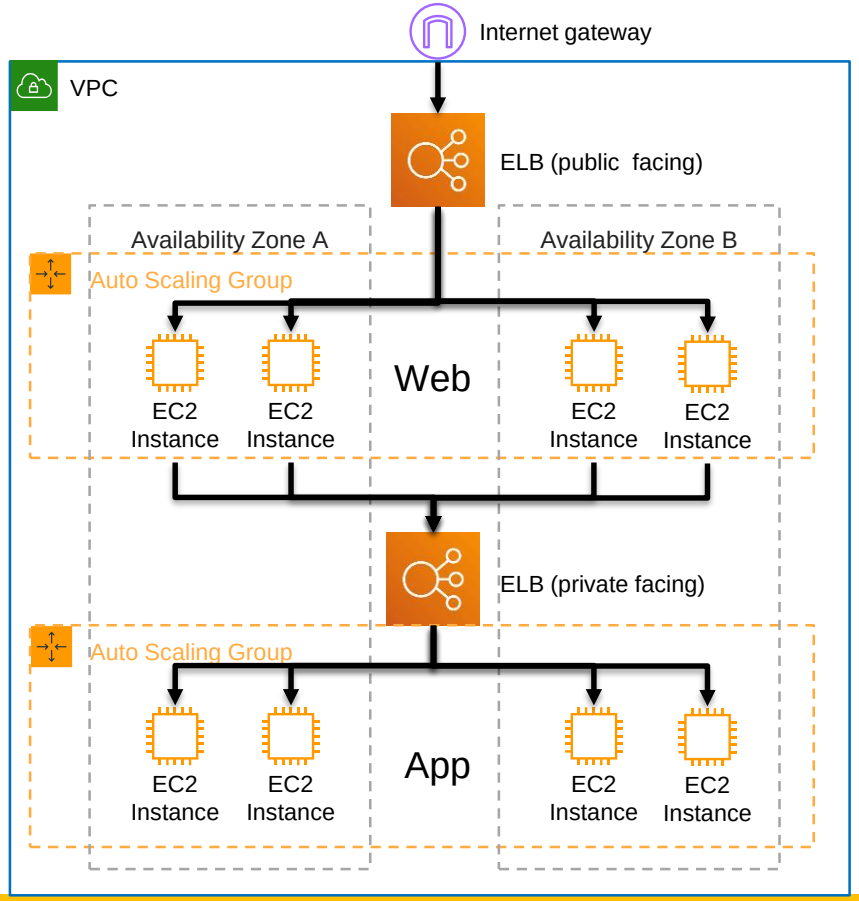
External and internal Load Balancers

Load balancers can be **external** (Public facing) or **internal** (Private facing)



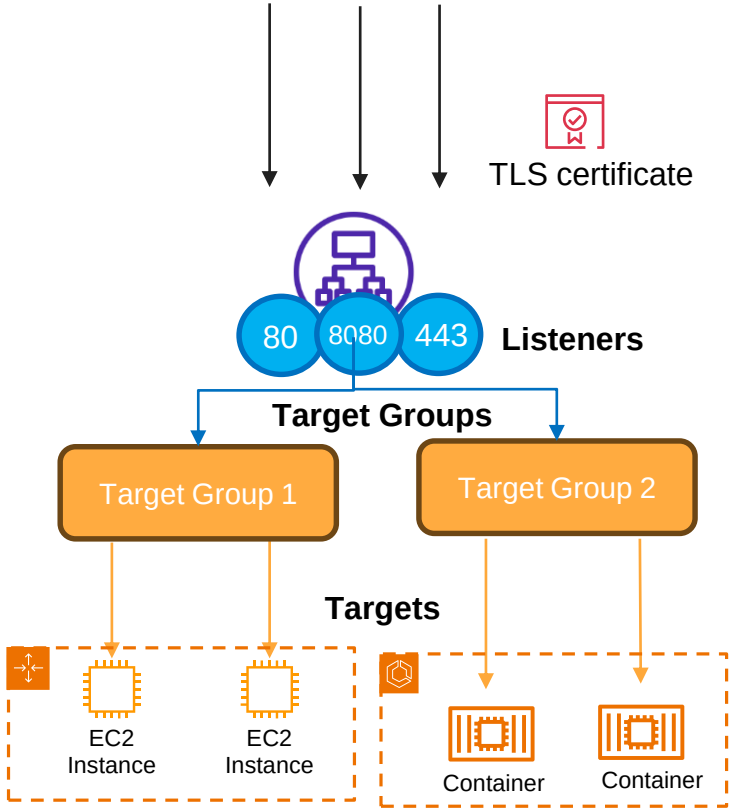
Example architecture:

- The external (public facing) load balancer serves the traffic from the internet
- The internal (private) load balancer serves traffic from within the VPC e.g. Web server to App servers
- Load balancers works with Auto Scaling group (ASG) enabling automatic scaling of the backend ec2 instances

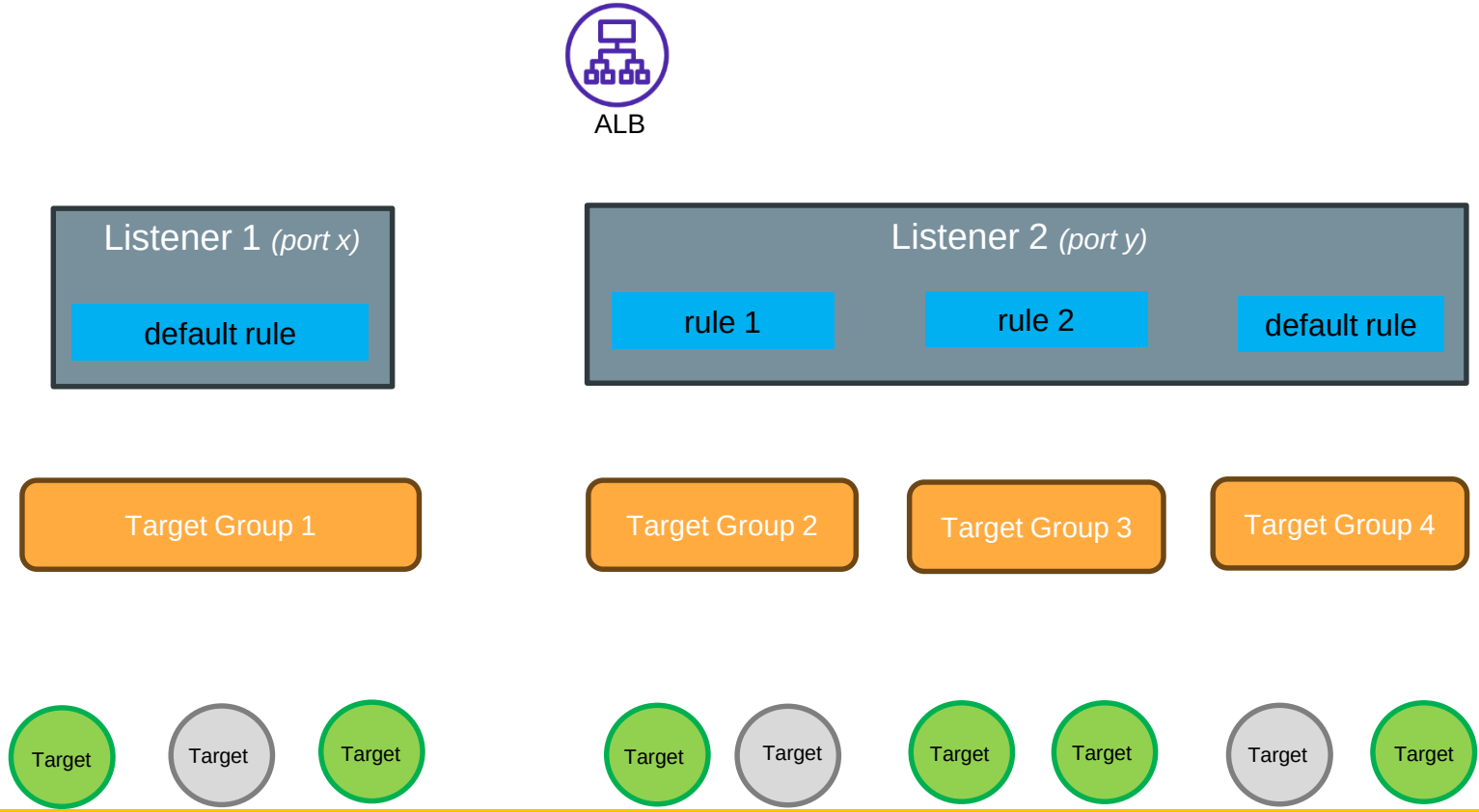


Application Load Balancer

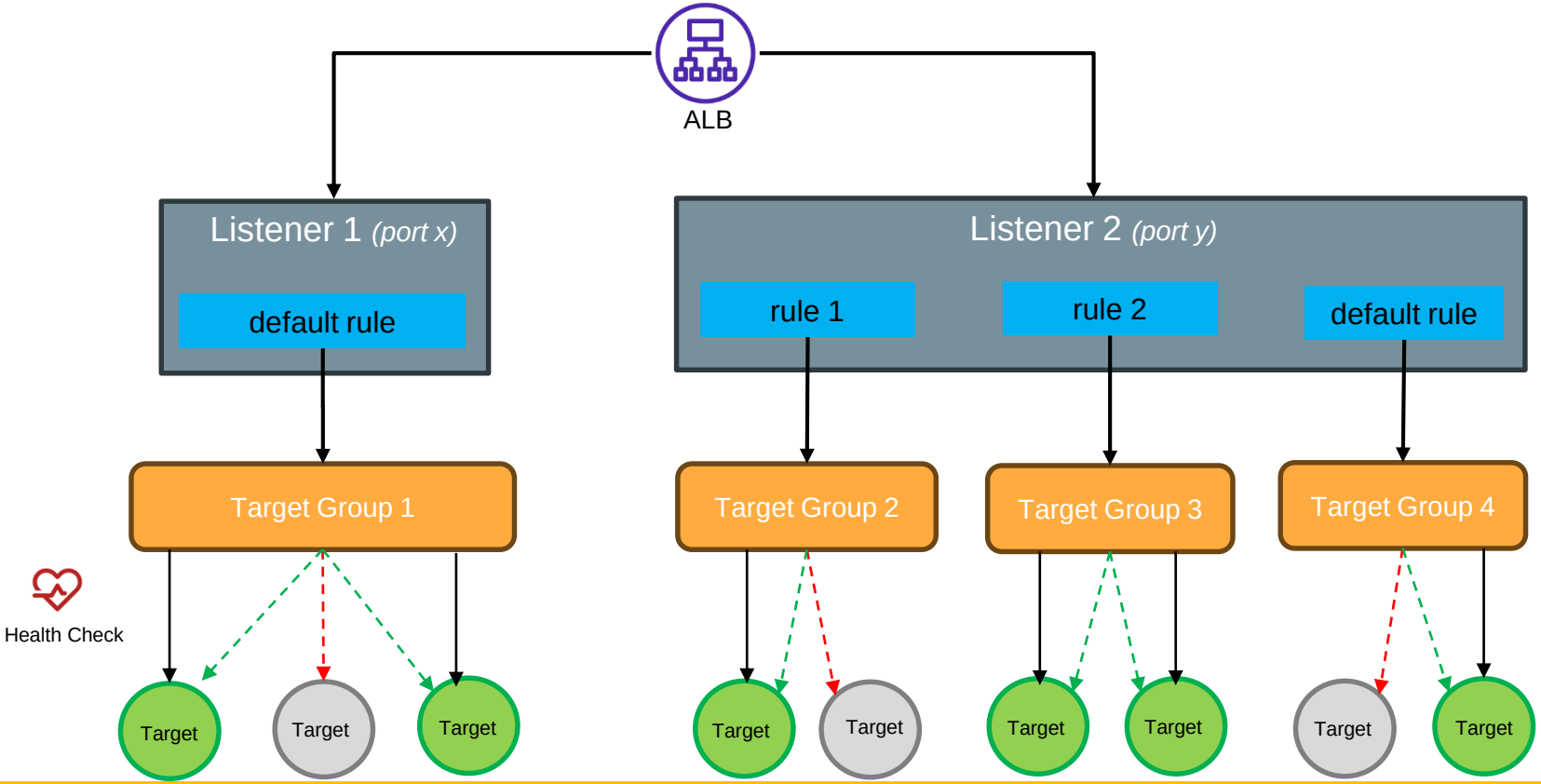
- Operates at Layer 7
- Supported protocols HTTP, HTTPS, WebSocket, HTTP/2 and gRPC
- Supports multiple listeners e.g. http, https, custom
- Load balancing to multiple Target groups which contains targets across EC2 instances, container tasks, IP addresses
- Support for returning custom HTTP responses or fixed responses
- Supports redirects (e.g., from HTTP to HTTPS)
- Supports SSL/TLS termination
- Supports authentication using Amazon Cognito and federation (AD, OIDC, SAML etc.)
- Supports routing host based, URL/path based, source-ip, http-header, query-string



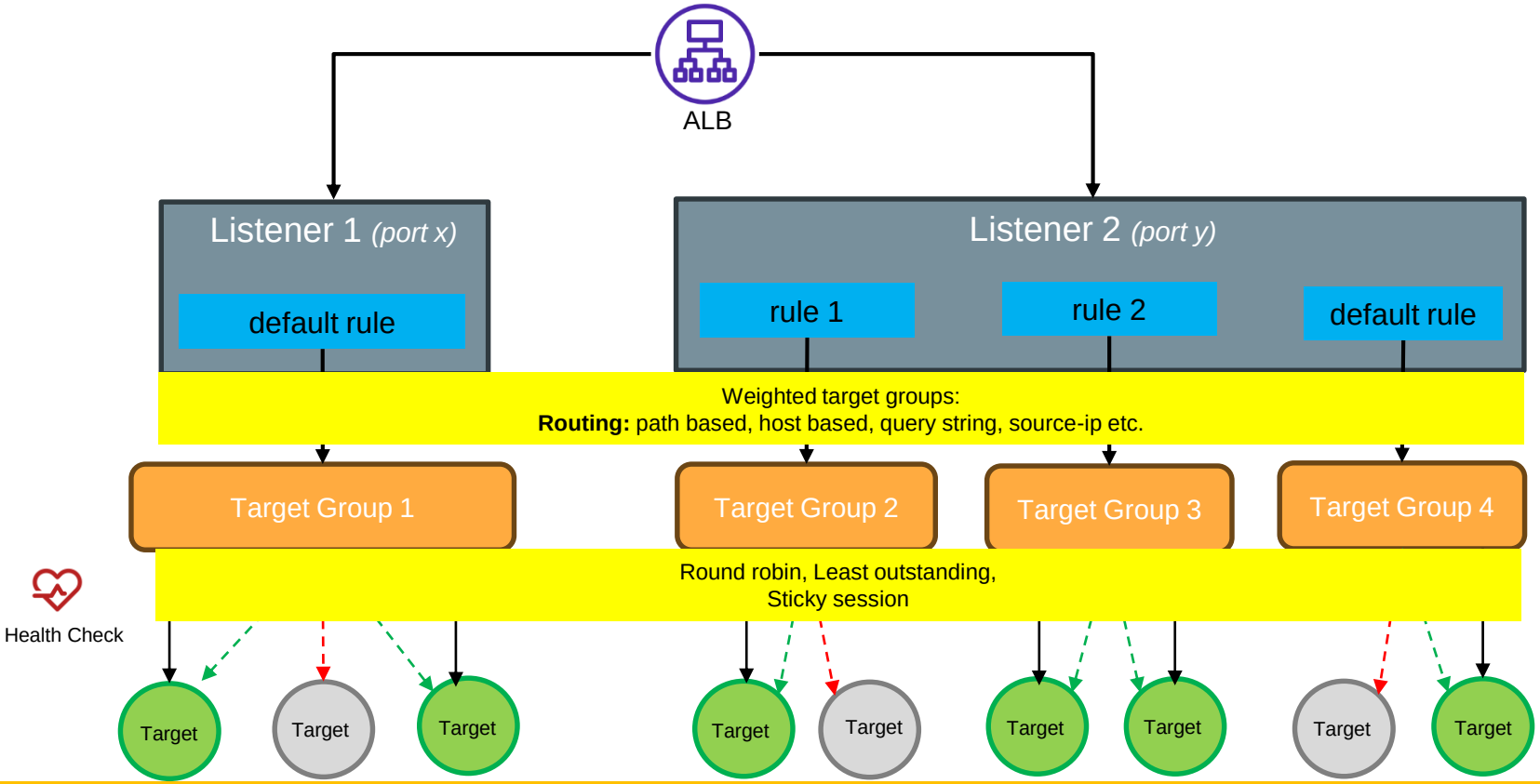
ALB components



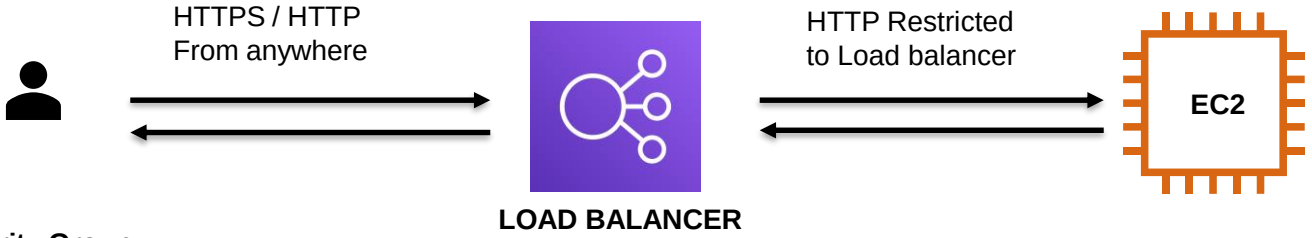
ALB -> Listener -> Rules -> Target group -> Targets



ALB traffic routing



Load Balancer Security Groups



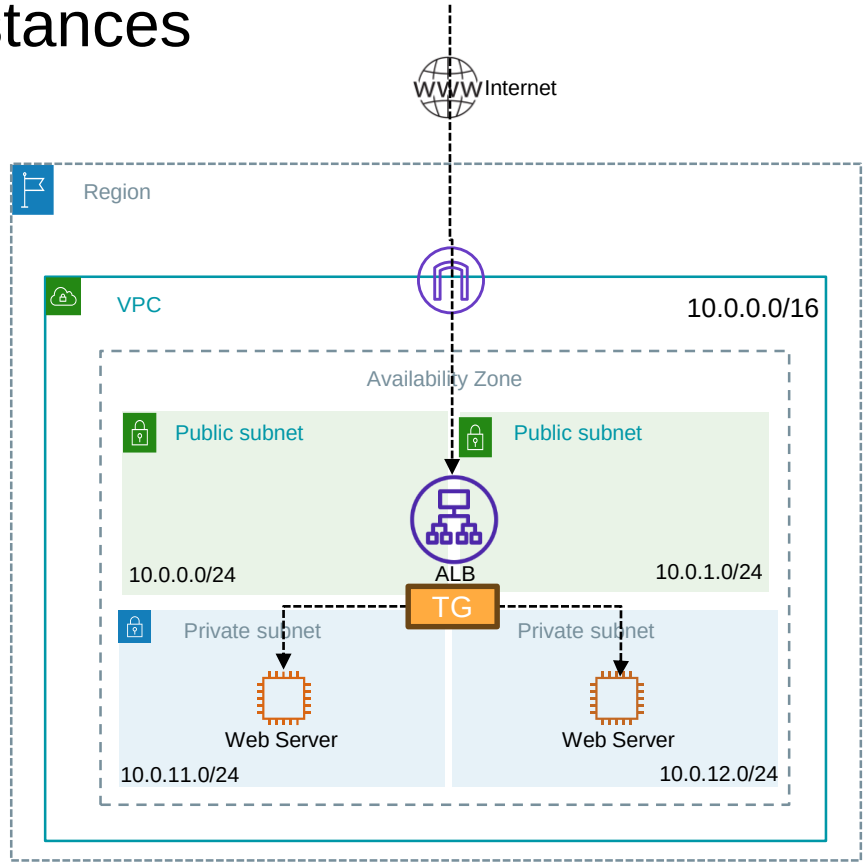
Load Balancer Security Group:

Type	Protocol	Port range	Source	Description
HTTP	TCP	80	0.0.0.0/0	Allow HTTP
HTTPS	TCP	443	0.0.0.0/0	Allow HTTPS

Application Security Group: Allow traffic only from Load Balancer

Type	Protocol	Port range	Source	Description
HTTP	TCP	80	sg-0bc494c71e6cf1896 / ALB-SG	Allow traffic from ALB

Exercise – ALB with 1 Target group and 2 backend EC2 instances



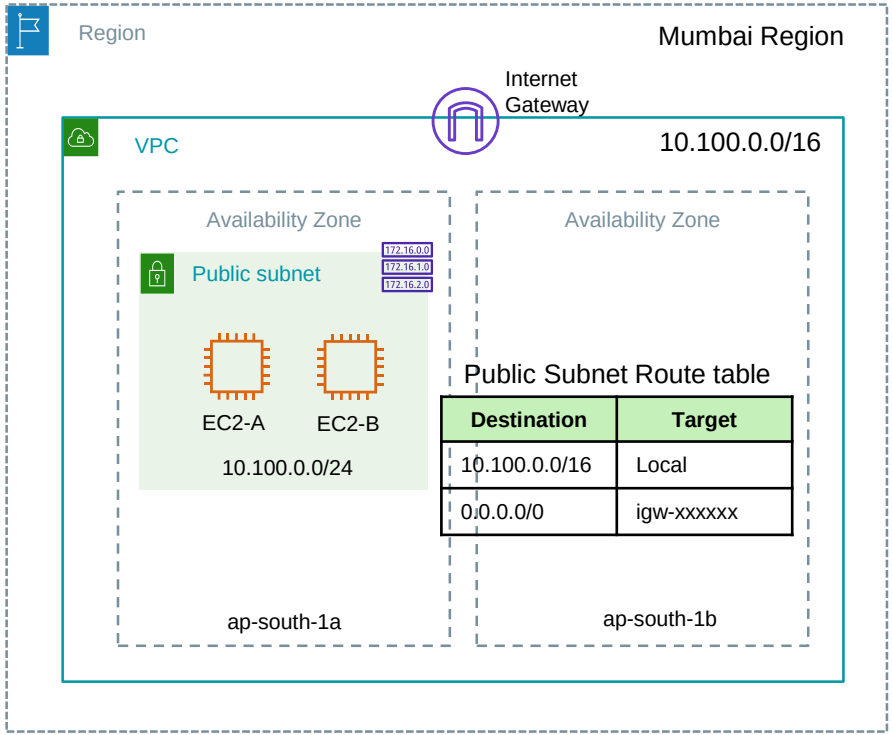
Do you see a problem in doing this exercise?

1. How will you connect to EC2 instances launched in the Private subnets?
2. How will you install httpd package?

Solution:

1. Create a machine image of the EC2 instance which already has webserver configured and running
2. Use this AMI to launch EC2 instances in the Private subnets

Pre-requisites for ALB exercises – Create webserver AMIs



Continuing with earlier VPC setup or create a new VPC with a Public subnet as shown

1. Launch Webserver 1 and Web server 2 with EC2 Userdata provided in the next slide. Open Security group to allow HTTP for 0.0.0.0/0.
2. Wait for instances to be running and try to access both the webservers over HTTP using EC2 Public IPs. Should be able to access.
3. Stop both the instances
4. Create AMI (Amazon Machine Image) for both the instances
5. Terminate both the instances

Pre-requisite – Create webserver AMIs

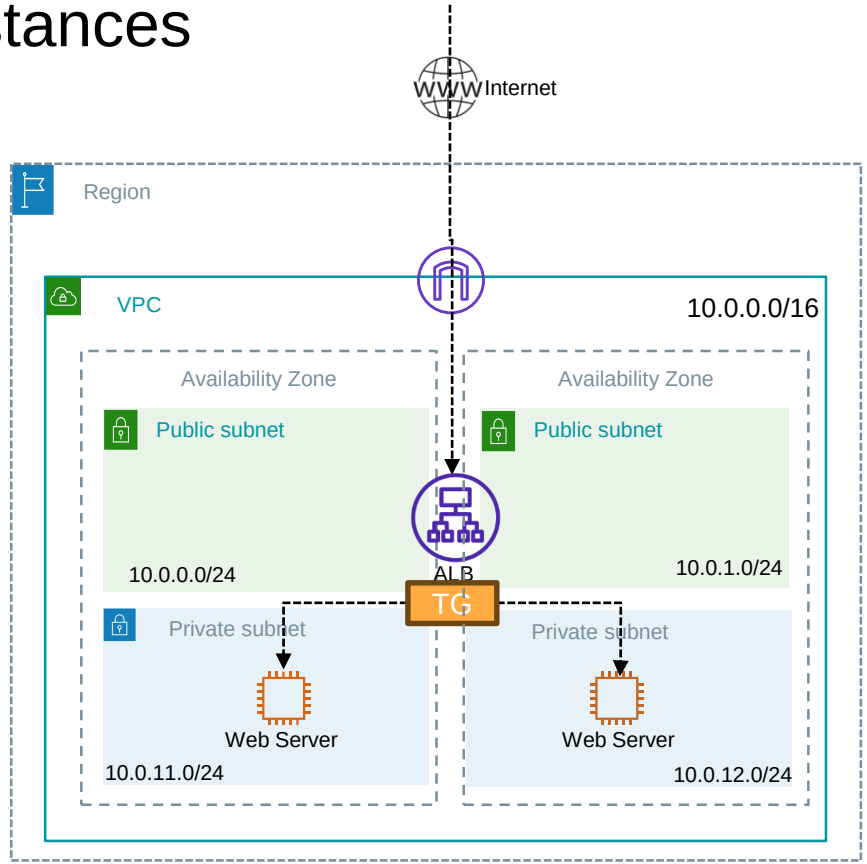
Userdata for Webserver 1:

```
#!/bin/bash
yum install httpd -y
systemctl start httpd.service
systemctl enable httpd.service
echo "<h1>This is Webserver 1 (INDIA)</h1>" > /var/www/html/index.html
echo "Configured successfully"
```

Userdata for Webserver 2:

```
#!/bin/bash
yum install httpd -y
systemctl start httpd.service
systemctl enable httpd.service
echo "<h1>This is Webserver 2 (UK) </h1>" > /var/www/html/index.html
echo "Configured successfully"
```

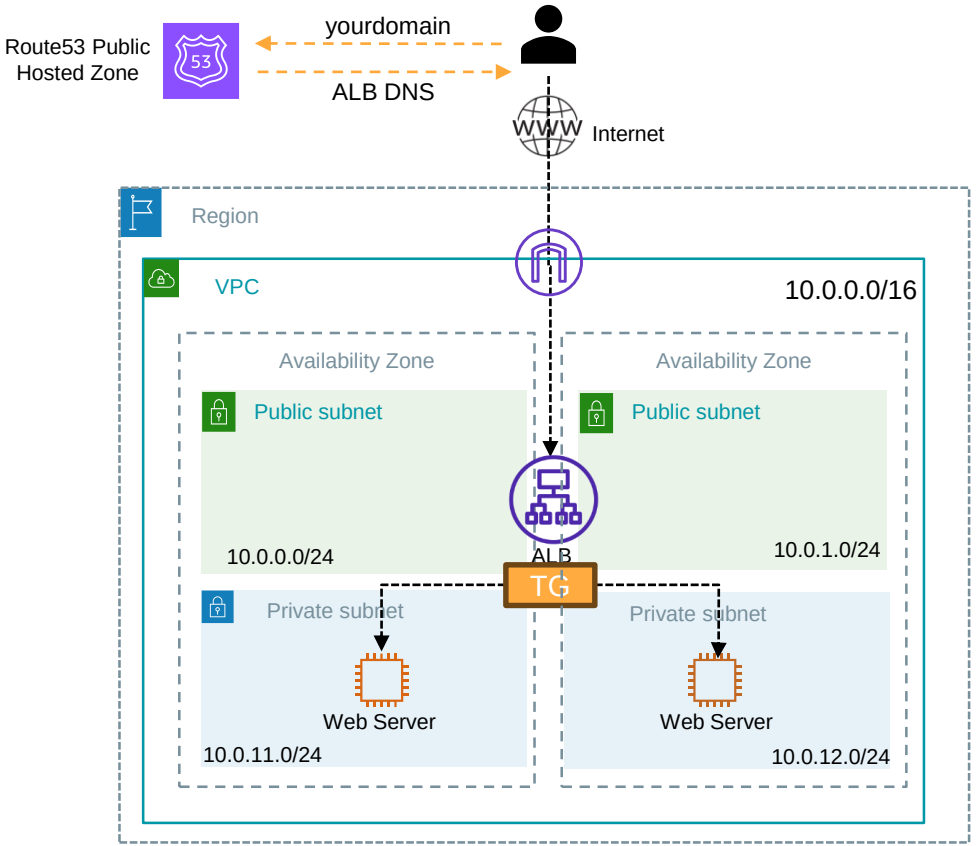
Exercise – ALB with 1 Target group and 2 backend EC2 instances



High level steps

1. Setup VPC & subnets network as shown
2. Launch Web server 1 and Web server 2 in respective private subnets using AMIs created during earlier exercise. Security group to allow HTTP for VPC CIDR
3. Create ALB Target group and add both EC2 instances. Configure health check on port 80 with /index.html
4. Create ALB with HTTP listener and forward the traffic to Target group. Open ALB Security group for HTTP traffic.
5. Access ALB over the browser using AWS provided ALB DNS

Exercise – ALB with custom domain name



Continuing with earlier setup

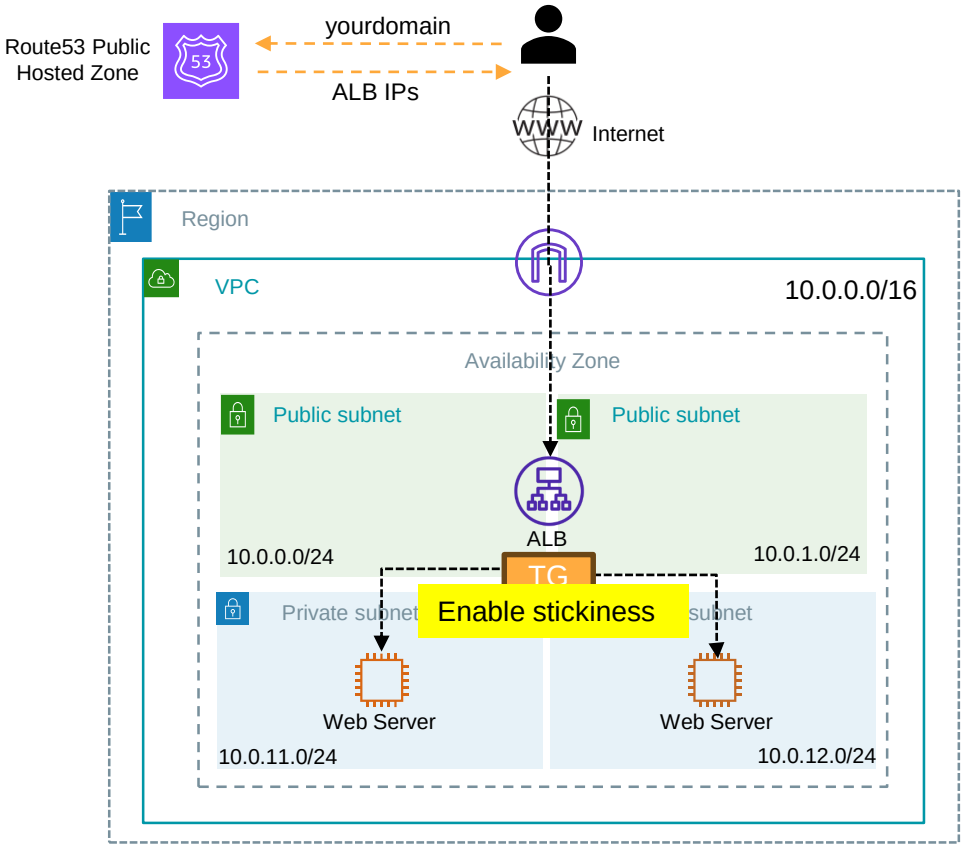
Pre-requisite for this exercise:

- You should have your public domain name and DNS should be pointing to Route53 public hosted zone
- Refer Labs prerequisites section if you haven't done this earlier.

1. In Route53 Public hosted zone create an A (Alias) record and point it to ALB DNS

2. Wait for some time and access application using your custom domain name

Exercise – ALB Sticky session

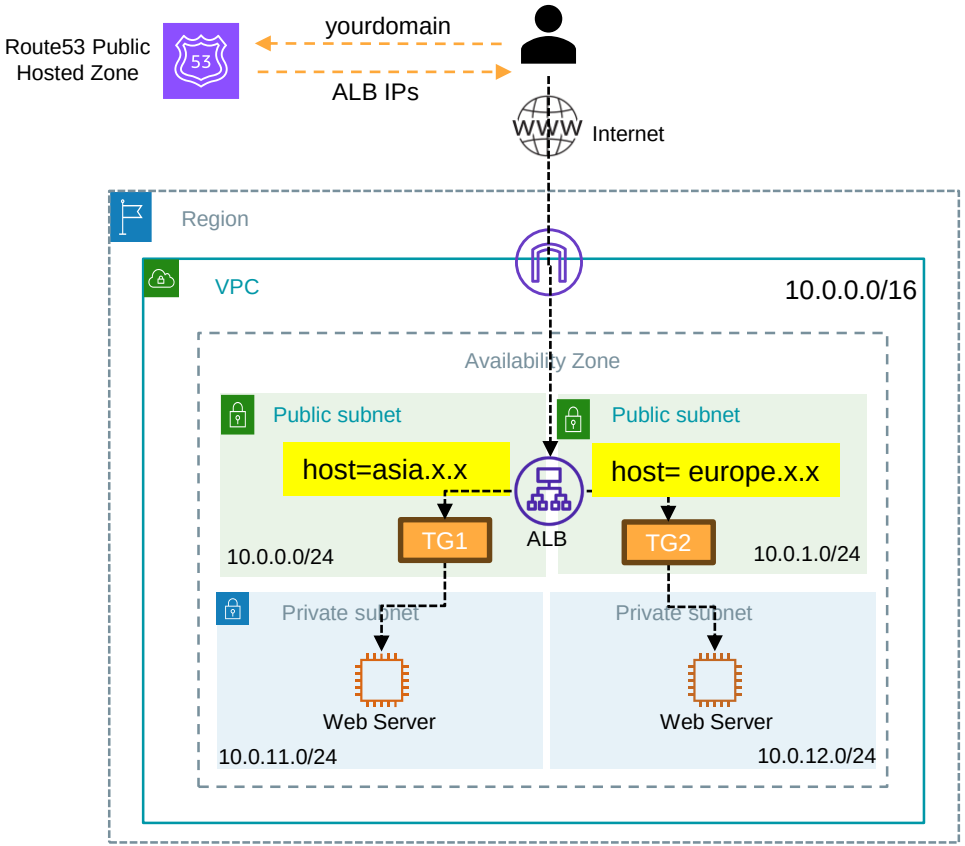


Continuing with earlier setup

1. Update target group attribute to enable Session stickiness for 30 seconds duration

2. Try accessing website with your custom domain name, refresh couple of times, you should see that you land onto the same EC2 instance. Wait for 30 seconds and try again.

Exercise – ALB host based routing



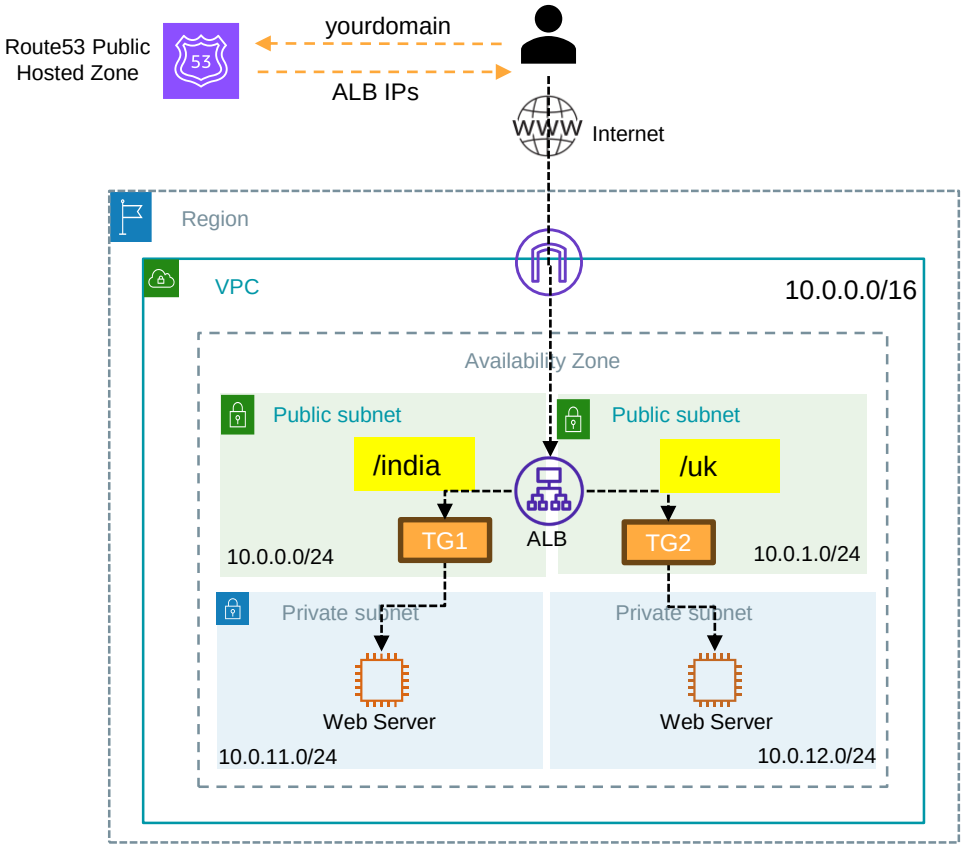
Continuing with earlier setup

1. Create 2 target groups and register one EC2 instance in each target group as per the intended traffic routing (Webserver 1 for Asia and Webserver 2 for Europe).

2. In ALB listener rule, add a rule for host based routing where traffic with host=asia.x.x is sent to TG1 and host=europe.x.x is sent to TG2

3. This time create 2 separate Route53 record sets e.g asia.x.x and europe.x.x and point both these records to ALB DNS (using Alias record)

Exercise – ALB path based routing



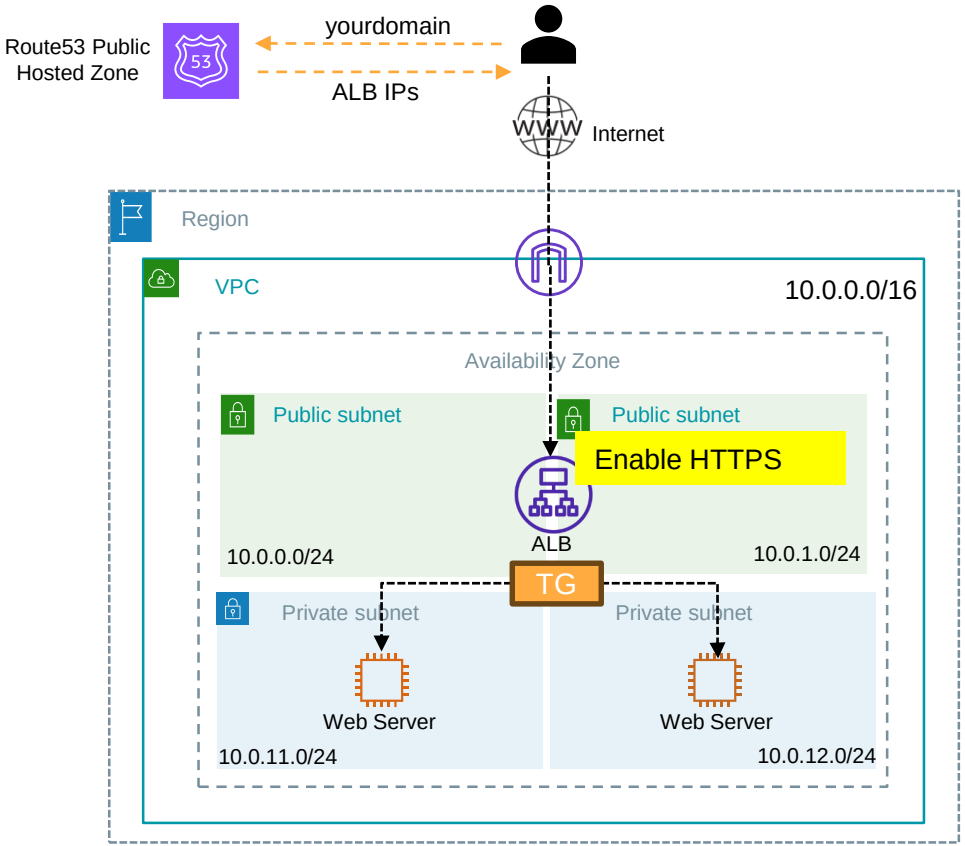
Continuing with earlier setup

1. Create 2 target groups and register one EC2 instance in each target group as per the intended traffic routing (Webserver 1 for /india and Webserver 2 for /uk).

2. Create Load Balancer listener rule to route traffic to respective Target groups with matching paths i.e. /india to TG1 and /uk to TG2

3. Access ALB DNS and append path /india. Traffic should be routed to web server for India. Do the same for /uk path and traffic should be routed to other webserver.

Exercise – Enable HTTPS



Continuing with earlier setup

1. Get TLS certificate from Amazon ACM for your domain name. Validate the domain ownership through ACM portal.
2. Modify ALB listener to HTTPS (443) and associate TLS Certificate. Forward traffic to same target group as earlier.
3. Update ALB security group to allow HTTPS (443) traffic
4. Access website using https://yourdomain

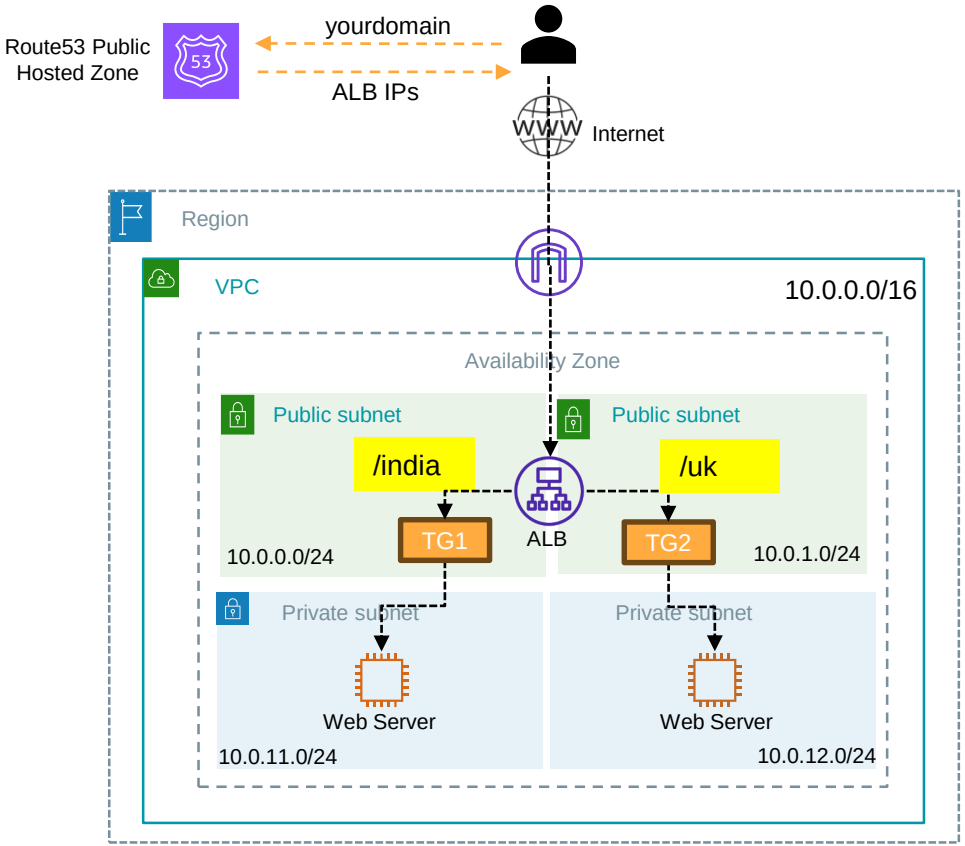
Cleanup – Very important

1. Terminate both the EC2 instances
2. Deregister AMIs
3. Delete Load Balancer Target groups
4. Delete Load Balancer
5. Delete Route53 records
6. You can keep the VPC (there is no cost) and you can use it for your next assignment.

Assignment – ALB path based routing

- Complete the ALB path based routing assignment as per the instructions given in the PDF with this lecture (in the resources section)
- Make sure you terminate all EC2 instances and ALB after this assignment

Assignment – ALB path based routing



0 **Use the existing VPC with 2 Public Subnets and 2 Private Subnets** or create a new VPC and subnets if you don't have it.

1 Create EC2 AMIs for both the backend webservers. Use userdata given in the next slide.

2 Launch Web server 1 and Web server 2 in respective private subnets using AMIs created above. Security group to allow HTTP traffic.

3 Create 2 target groups and register one EC2 instance in each target group as per the intended traffic routing (Webserver 1 for /india and Webserver 2 for /uk).

4 Create Load Balancer listener rules to route traffic to respective Target groups with matching paths i.e. /india to TG1 and /uk to TG2

5 Access web application with domain name and append path /india. Traffic should be routed to web server for India. Do the same for /uk path and traffic should be routed to the other webserver.

Assignment Pre-requisite – Create webserver AMIs

Userdata for Webserver 1:

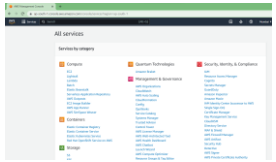
```
#!/bin/bash
yum install httpd -y
systemctl start httpd.service
systemctl enable httpd.service
echo "<h1>This is Webserver 1</h1>" > /var/www/html/index.html
mkdir /var/www/html/india
echo "<h1>Hi from India</h1>" > /var/www/html/india/index.html
echo "Configured successfully"
```

Userdata for Webserver 2:

```
#!/bin/bash
yum install httpd -y
systemctl start httpd.service
systemctl enable httpd.service
echo "<h1>This is Webserver 2</h1>" > /var/www/html/index.html
mkdir /var/www/html/uk
echo "<h1>Hi from UK</h1>" > /var/www/html/uk/index.html
echo "Configured successfully"
```

Automating AWS infrastructure deployment

Automate AWS infrastructure deployment



AWS Console

AWS CLI

AWS SDK

c++, go, java, Java script, Kotlin, .Net, Node.js, PHP, python, ruby, rust, swift

ANSIBLE

CHEF

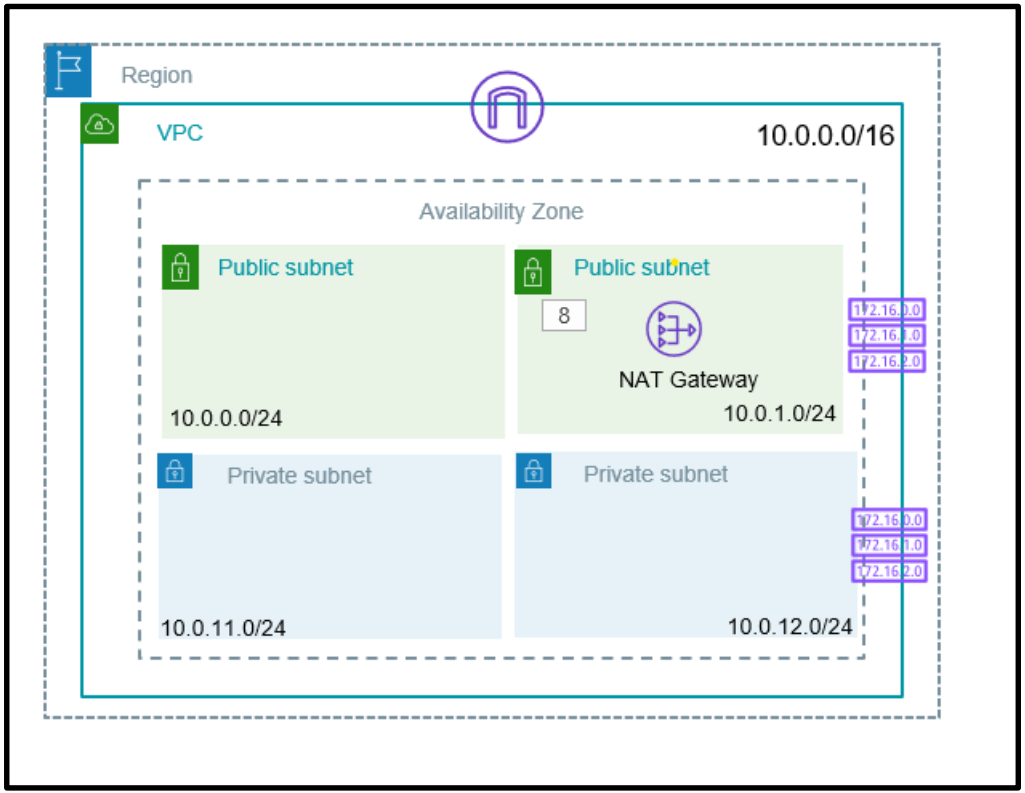
Terraform

AWS CloudFormation

yaml or json

CDK

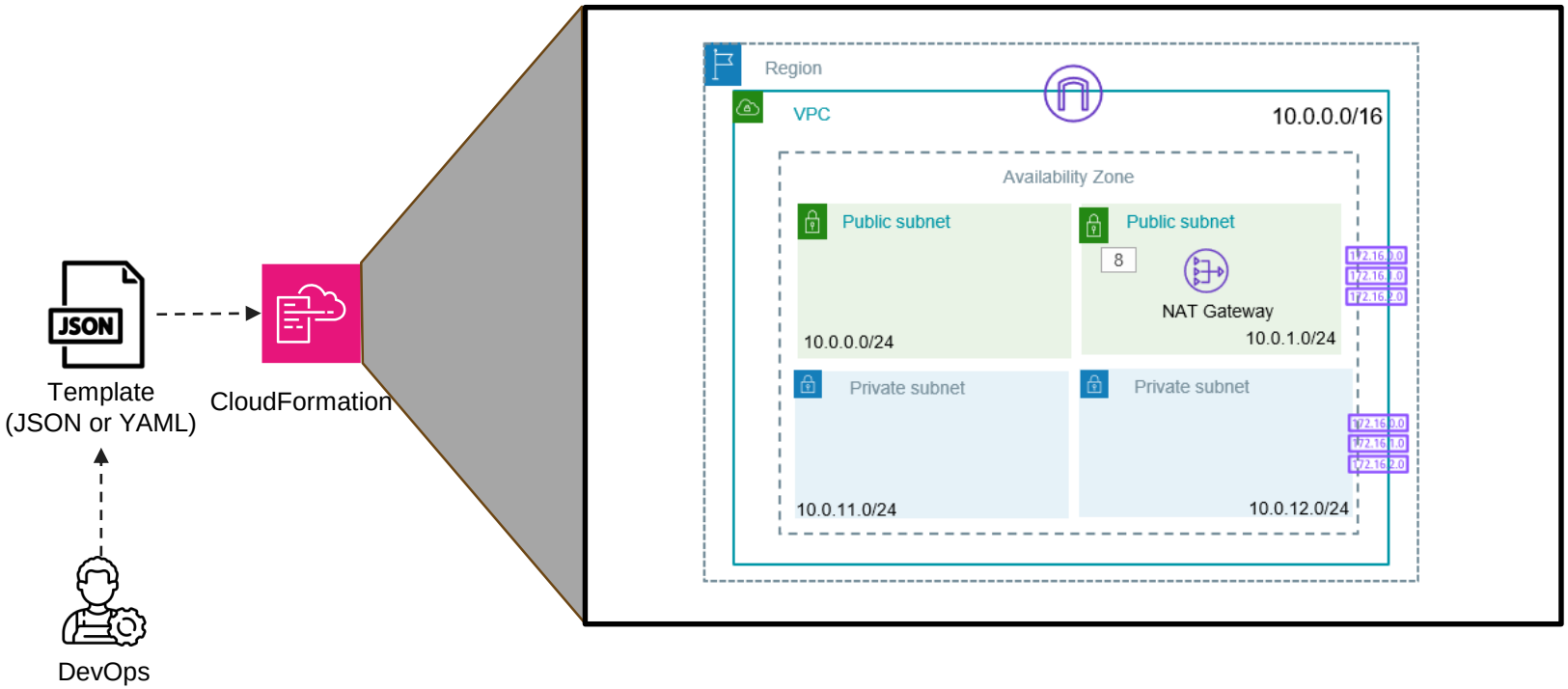
TypeScript, JavaScript, Python, Java, C#/. Net, Go



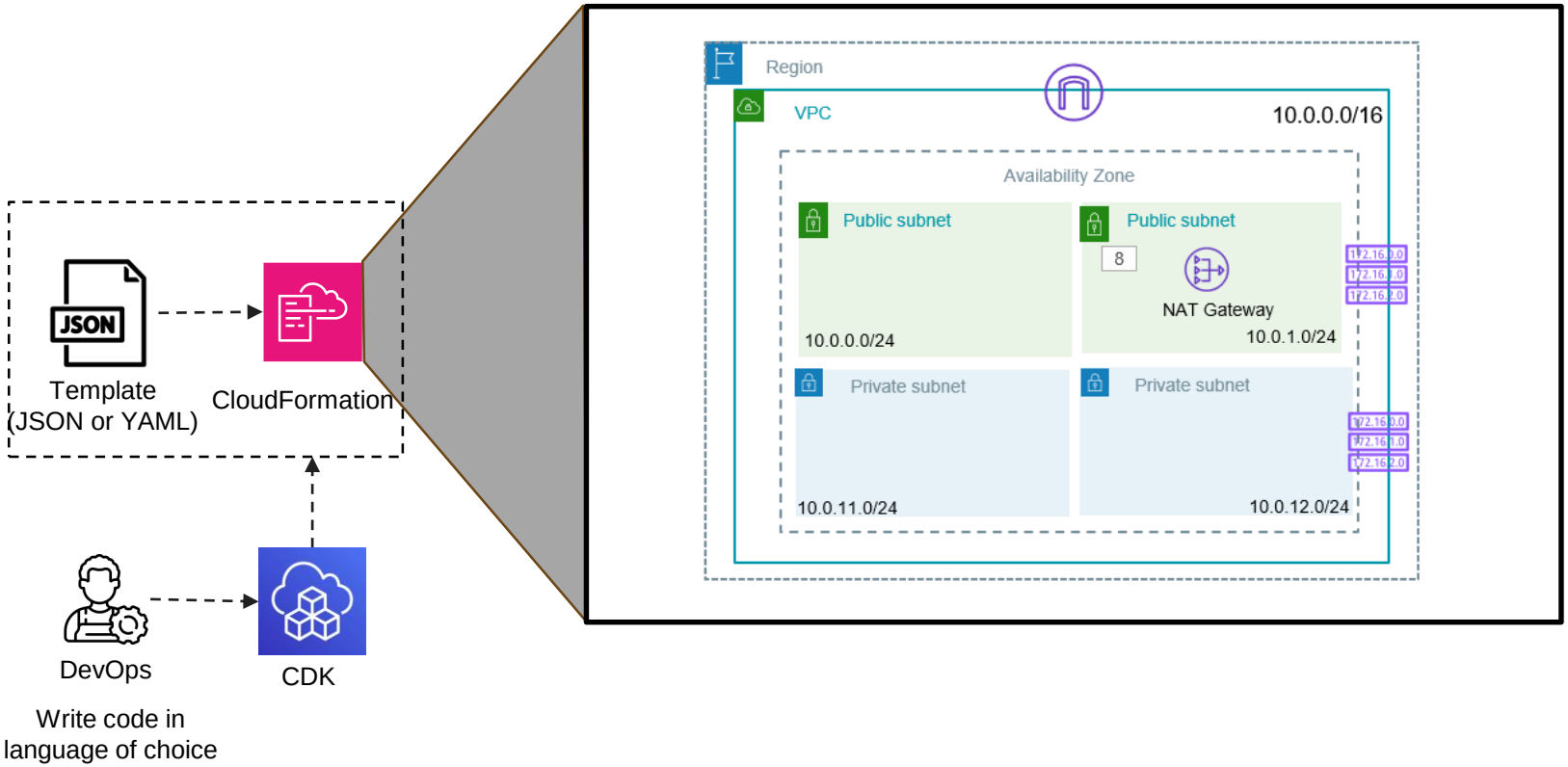
Using AWS CLI

```
aws ec2 create-vpc --cidr-block 10.0.0.0/16 --tag-specifications ResourceType=vpc,Tags=[{Key=Name,Value=MyVPC}]
aws ec2 create-internet-gateway
aws ec2 attach-internet-gateway --vpc-id vpc-xxxxxxxxxx --internet-gateway-id igw-xxxxxxxxxx
aws ec2 create-subnet --vpc-id vpc-xxxxxxxxxx --cidr-block 10.0.0.0/24 --availability-zone ap-south-1a
aws ec2 create-subnet --vpc-id vpc-xxxxxxxxxx --cidr-block 10.0.1.0/24 --availability-zone ap-south-1b
aws ec2 create-subnet --vpc-id vpc-xxxxxxxxxx --cidr-block 10.0.11.0/24 --availability-zone ap-south-1a
aws ec2 create-subnet --vpc-id vpc-xxxxxxxxxx --cidr-block 10.0.12.0/24 --availability-zone ap-south-1b
aws ec2 create-route-table --vpc-id vpc-xxxxxxxxxx
aws ec2 create-route --route-table-id rtb-xxxxxxxxxx --destination-cidr-block 0.0.0.0/0 --gateway-id igw-xxxxxxxxxx
aws ec2 associate-route-table --route-table-id rtb-xxxxxxxxxx --subnet-id subnet-xxxxxxxxxx
aws ec2 create-route-table --vpc-id vpc-xxxxxxxxxx
aws ec2 associate-route-table --route-table-id rtb-xxxxxxxxxx --subnet-id subnet-xxxxxxxxxx
aws ec2 allocate-address --domain vpc
aws ec2 create-nat-gateway --subnet-id subnet-xxxxxxxxxx --allocation-id eipalloc-xxxxxxxxxx
aws ec2 create-route --route-table-id rtb-xxxxxxxxxx --destination-cidr-block 0.0.0.0/0 --gateway-id nat-xxxxxxxxxx
```

Using AWS CloudFormation



Using AWS CDK



Cleanup

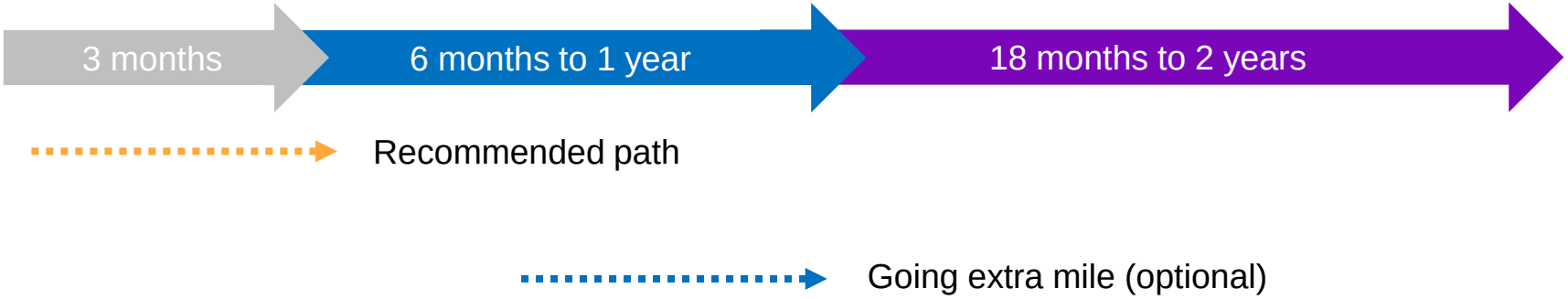
1. Delete CloudFormation stack. Make sure that stack is deleted successfully.

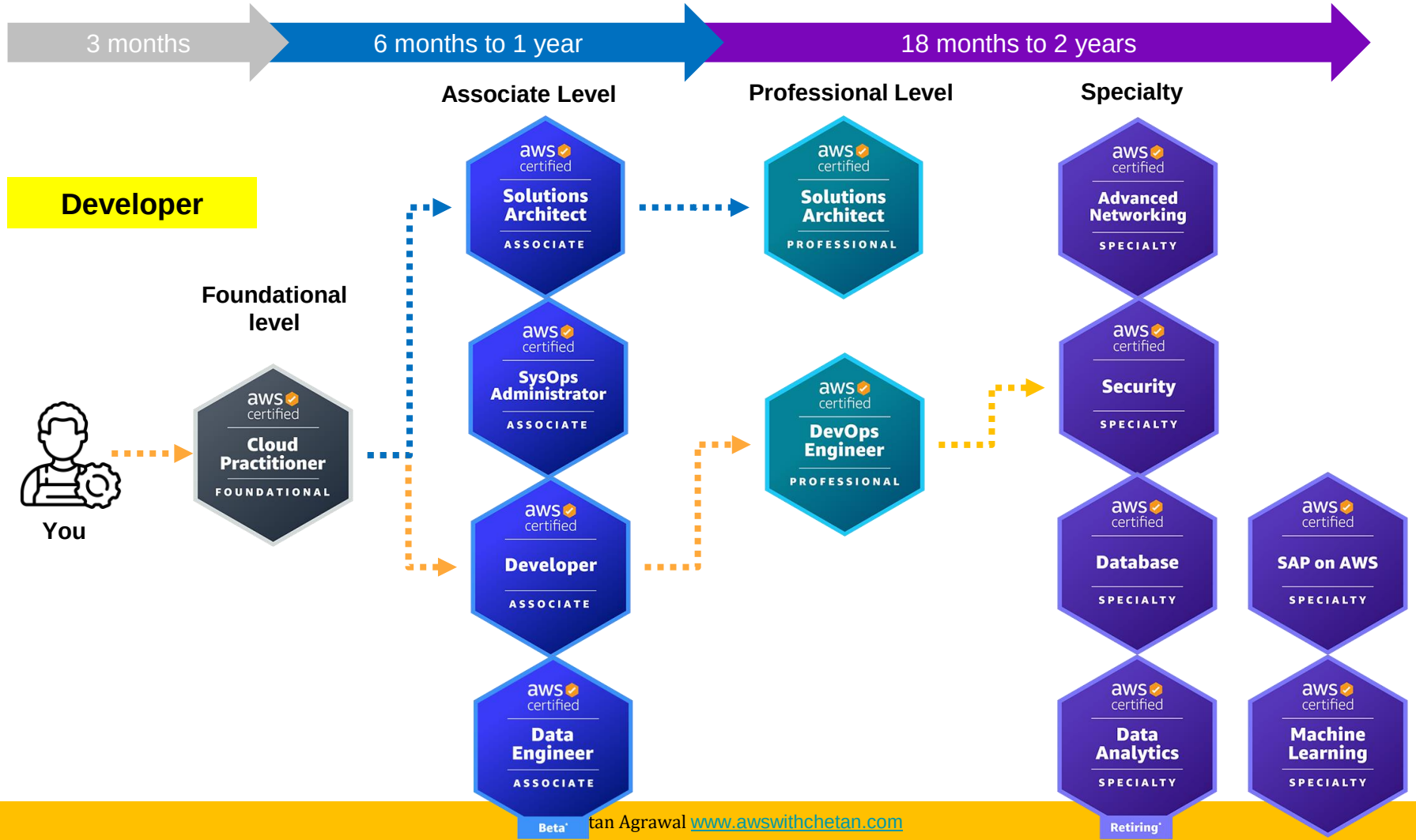
Congratulations !

What's next?

- Try to get as much hands-on experience with real projects
- Try small labs from https://aws.amazon.com/getting-started/hands-on/
- Read AWS blogs: https://aws.amazon.com/blogs/networking-and-content-delivery/
- Read AWS whitepapers: https://aws.amazon.com/whitepapers
- Target AWS Certifications

..but.. Which AWS Certification?




3 months
6 months to 1 year
18 months to 2 years
Associate Level
Professional Level
Specialty
Developer
Foundational level
You
aws certified Cloud Practitioner FOUNDATIONAL
aws certified Solutions Architect ASSOCIATE
aws certified SysOps Administrator ASSOCIATE
aws certified Developer ASSOCIATE
aws certified Data Engineer ASSOCIATE
Beta
aws certified Solutions Architect PROFESSIONAL
aws certified DevOps Engineer PROFESSIONAL
aws certified Advanced Networking SPECIALTY
aws certified Security SPECIALTY
aws certified Database SPECIALTY
aws certified Data Analytics SPECIALTY
Retiring
aws certified SAP on AWS SPECIALTY
aws certified Machine Learning SPECIALTY

3 months

6 months to 1 year

18 months to 2 years

DevOps

Foundational level

Associate Level

Professional Level

Specialty

You

aws certified Cloud Practitioner FOUNDATIONAL

aws certified Solutions Architect ASSOCIATE

aws certified SysOps Administrator ASSOCIATE

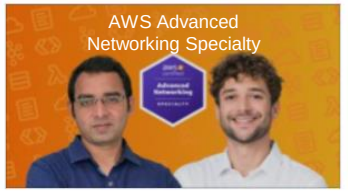
aws certified Developer ASSOCIATE

aws certified Data Engineer ASSOCIATE Beta

aws certified Solutions Architect PROFESSIONAL

aws certified DevOps Engineer PROFESSIONAL

aws certified Advanced Networking SPECIALTY



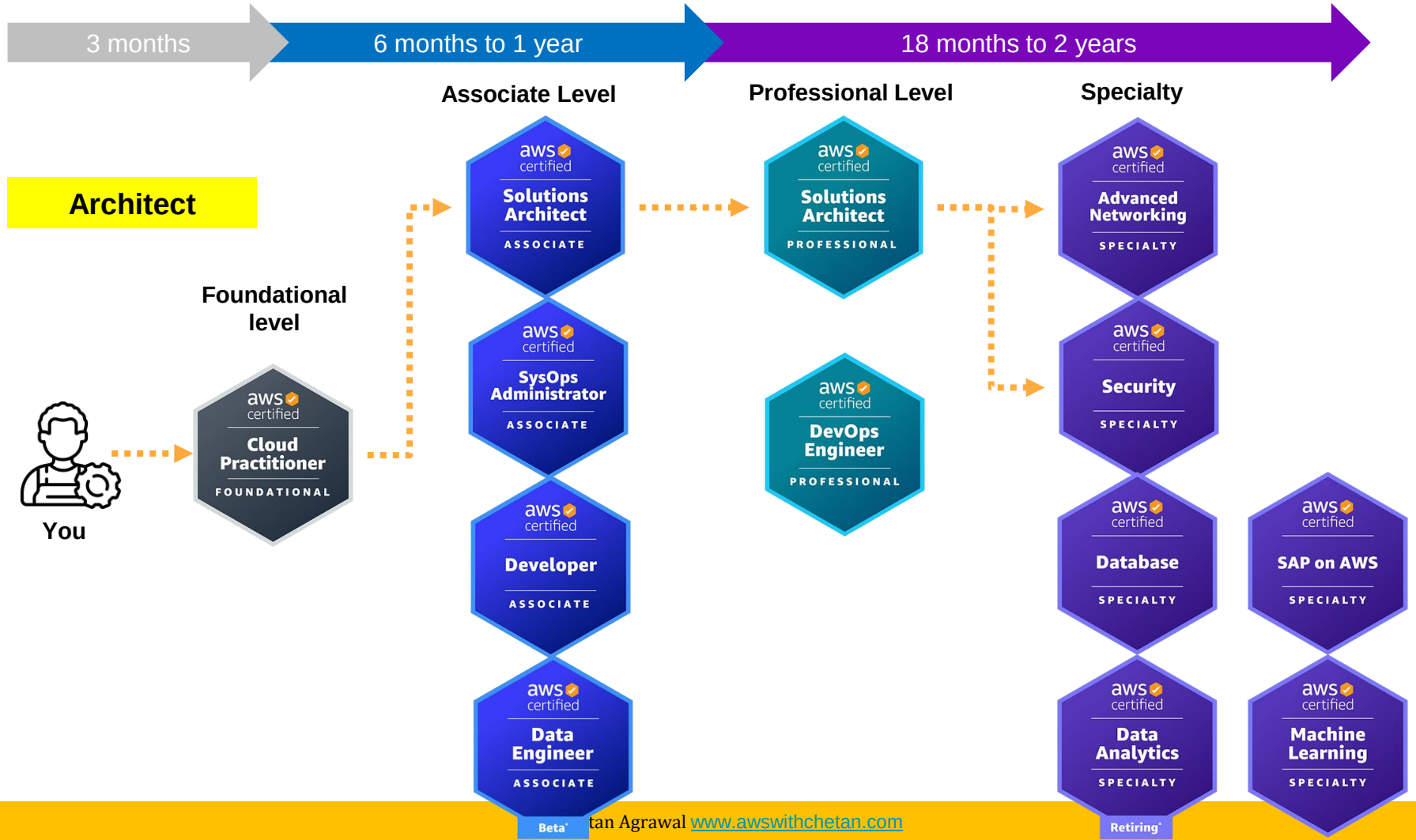
aws certified Security SPECIALTY

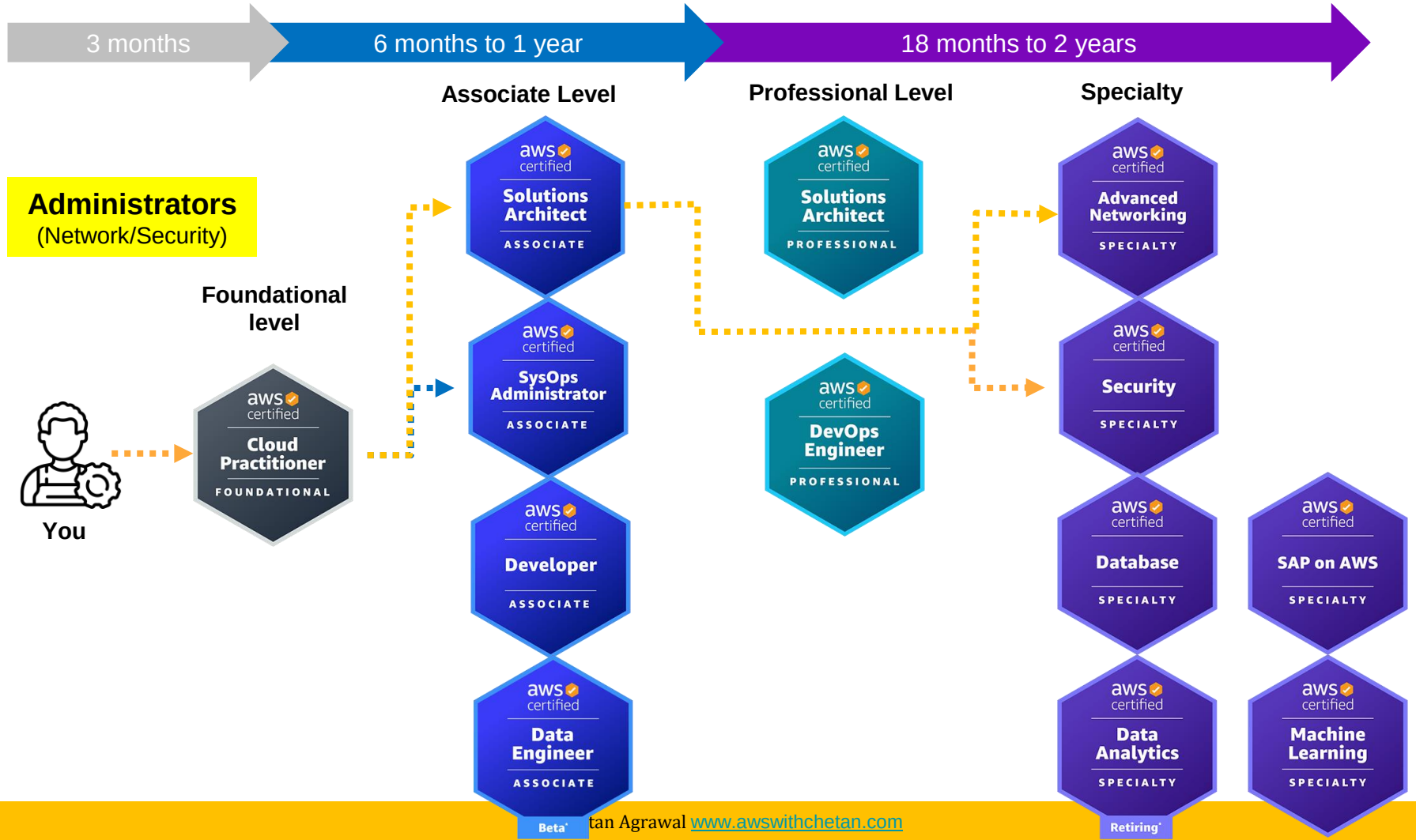
aws certified Database SPECIALTY

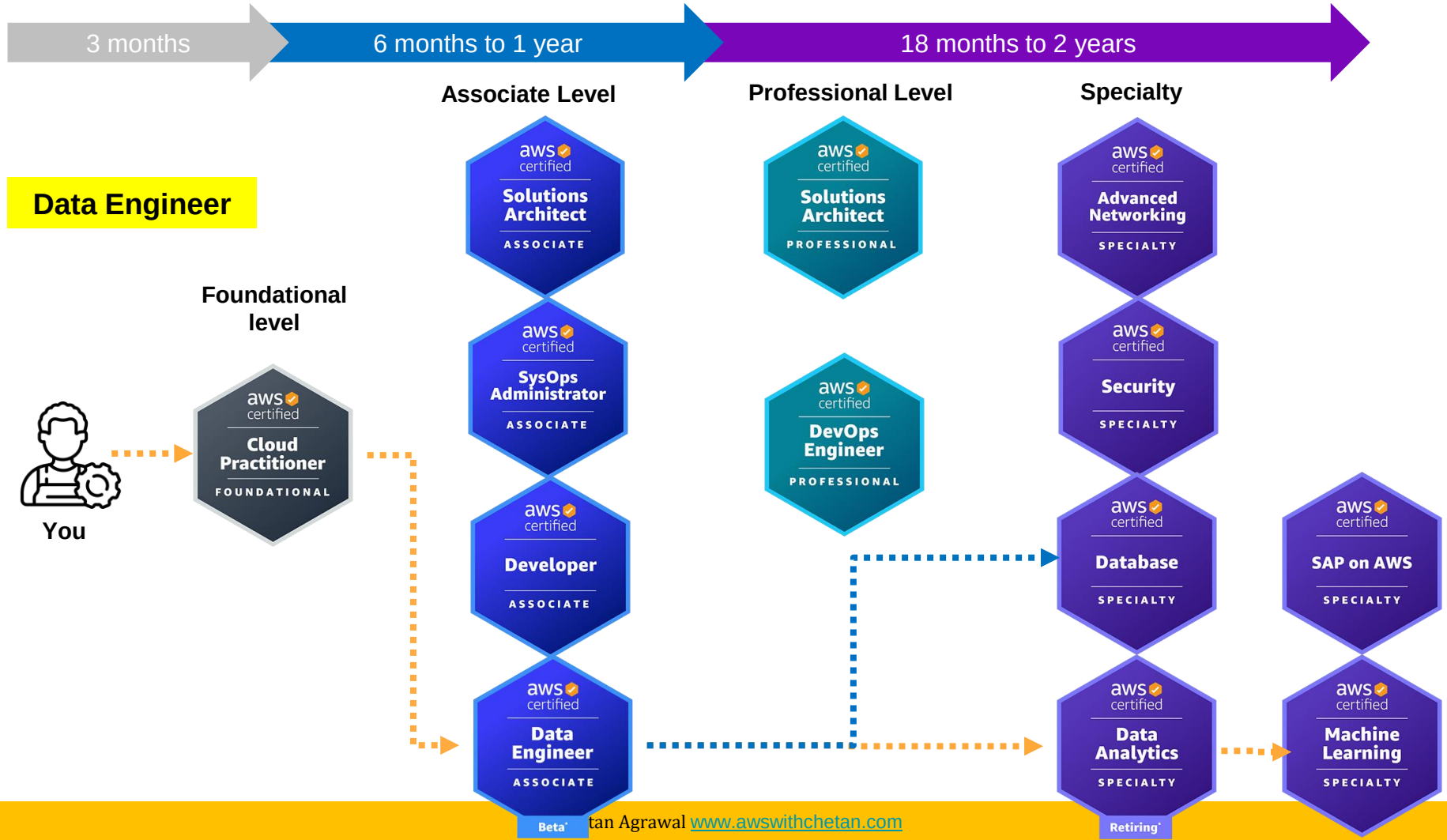
aws certified Data Analytics SPECIALTY Retiring

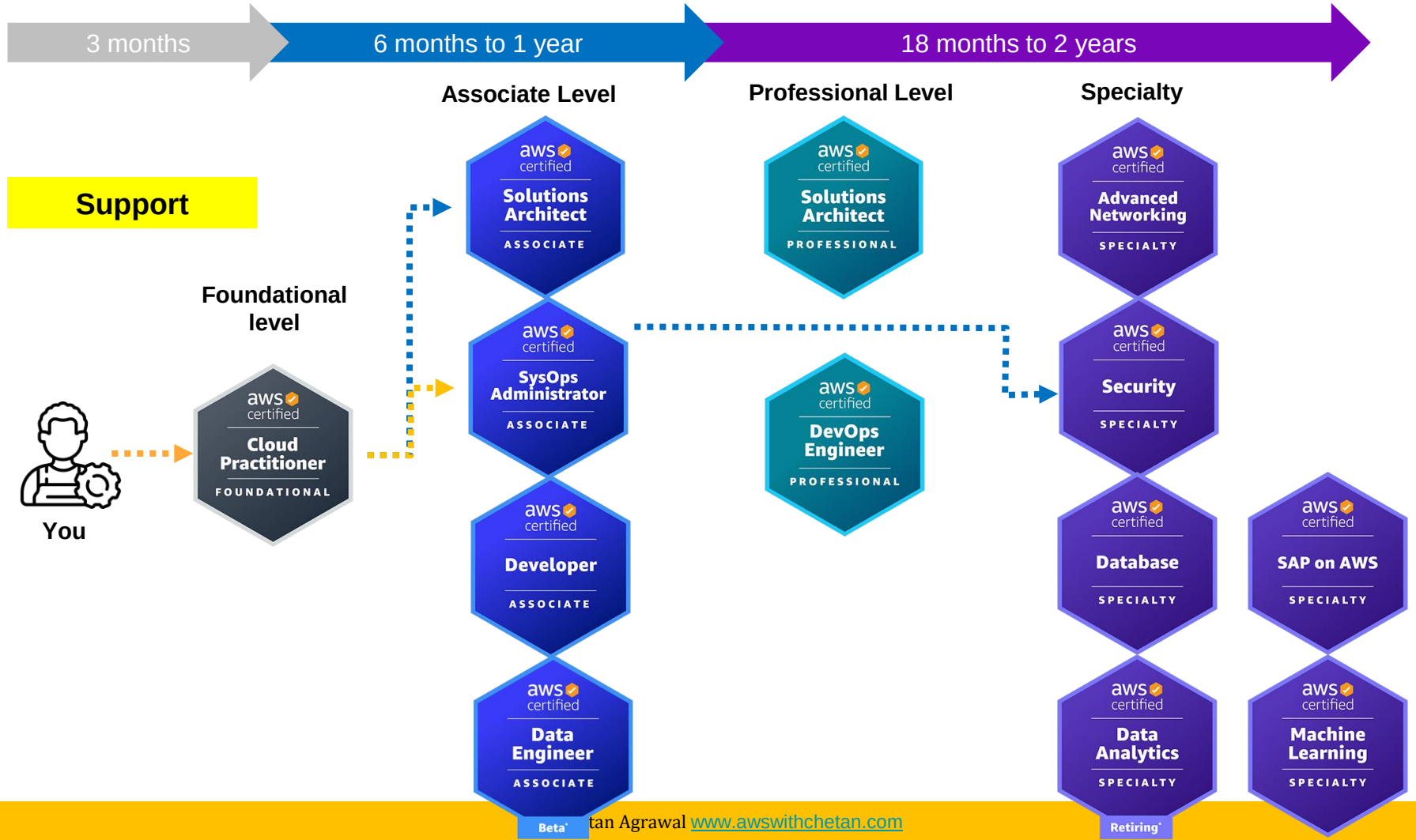
aws certified SAP on AWS SPECIALTY

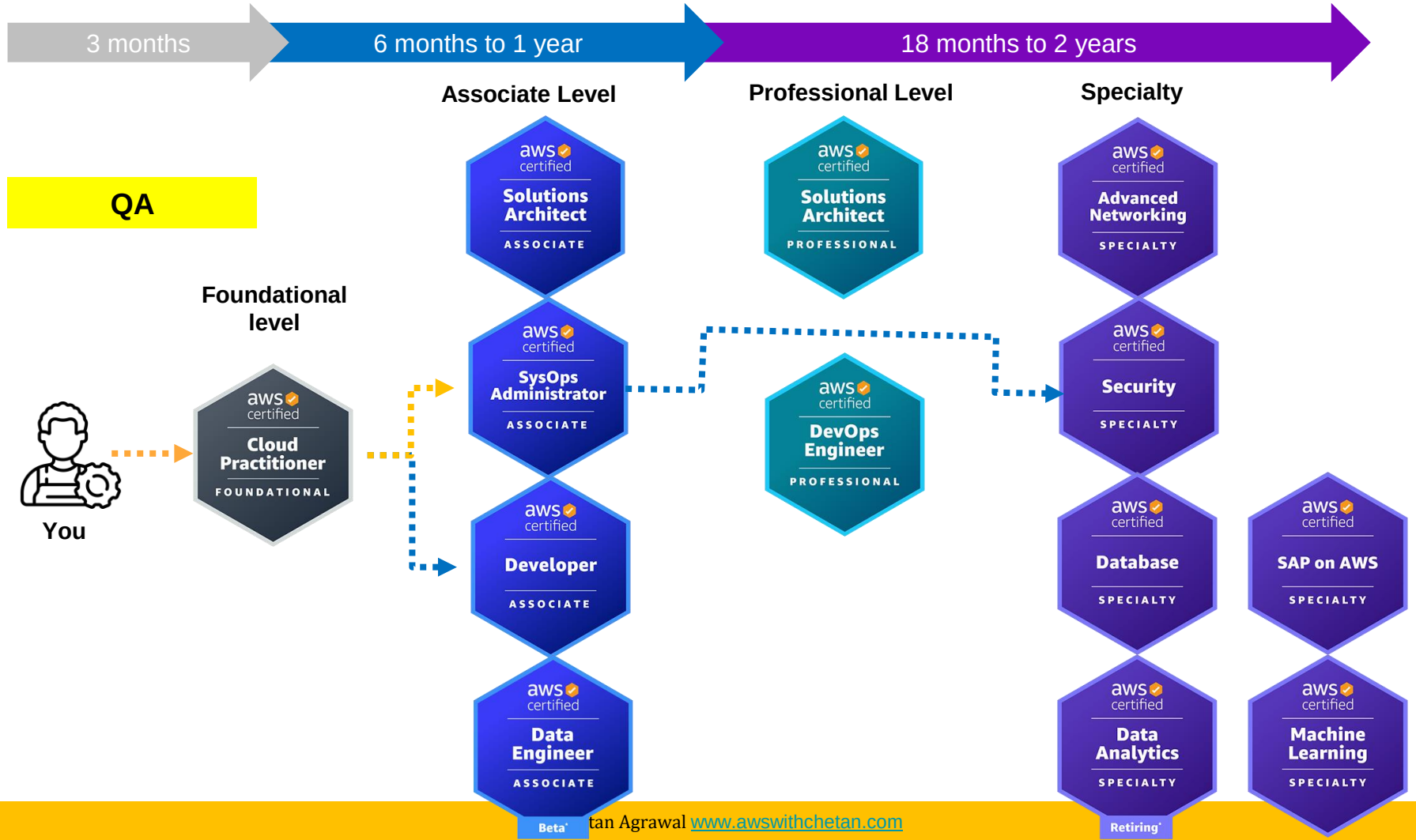
aws certified Machine Learning SPECIALTY


3 months
6 months to 1 year
18 months to 2 years
Associate Level
Professional Level
Specialty
Architect
Foundational level
You
aws certified Cloud Practitioner FOUNDATIONAL
aws certified Solutions Architect ASSOCIATE
aws certified SysOps Administrator ASSOCIATE
aws certified Developer ASSOCIATE
aws certified Data Engineer ASSOCIATE
Beta
aws certified Solutions Architect PROFESSIONAL
aws certified DevOps Engineer PROFESSIONAL
aws certified Advanced Networking SPECIALTY
aws certified Security SPECIALTY
aws certified Database SPECIALTY
aws certified Data Analytics SPECIALTY
Retiring
aws certified SAP on AWS SPECIALTY
aws certified Machine Learning SPECIALTY


3 months
6 months to 1 year
18 months to 2 years
Associate Level
Professional Level
Specialty
Administrators
(Network/Security)
Foundational level
You
aws certified Cloud Practitioner FOUNDATIONAL
aws certified Solutions Architect ASSOCIATE
aws certified SysOps Administrator ASSOCIATE
aws certified Developer ASSOCIATE
aws certified Data Engineer ASSOCIATE
Beta
aws certified Solutions Architect PROFESSIONAL
aws certified DevOps Engineer PROFESSIONAL
aws certified Advanced Networking SPECIALTY
aws certified Security SPECIALTY
aws certified Database SPECIALTY
aws certified Data Analytics SPECIALTY
Retiring
aws certified SAP on AWS SPECIALTY
aws certified Machine Learning SPECIALTY


3 months
6 months to 1 year
18 months to 2 years
Associate Level
Professional Level
Specialty
Data Engineer
Foundational level
You
aws certified Cloud Practitioner FOUNDATIONAL
aws certified Solutions Architect ASSOCIATE
aws certified SysOps Administrator ASSOCIATE
aws certified Developer ASSOCIATE
aws certified Data Engineer ASSOCIATE
Beta
aws certified Solutions Architect PROFESSIONAL
aws certified DevOps Engineer PROFESSIONAL
aws certified Advanced Networking SPECIALTY
aws certified Security SPECIALTY
aws certified Database SPECIALTY
aws certified Data Analytics SPECIALTY
Retiring
aws certified SAP on AWS SPECIALTY
aws certified Machine Learning SPECIALTY


3 months
6 months to 1 year
18 months to 2 years
Associate Level
Professional Level
Specialty
Support
Foundational level
You
aws certified Cloud Practitioner FOUNDATIONAL
aws certified Solutions Architect ASSOCIATE
aws certified SysOps Administrator ASSOCIATE
aws certified Developer ASSOCIATE
aws certified Data Engineer ASSOCIATE
Beta
aws certified Solutions Architect PROFESSIONAL
aws certified DevOps Engineer PROFESSIONAL
aws certified Advanced Networking SPECIALTY
aws certified Security SPECIALTY
aws certified Database SPECIALTY
aws certified Data Analytics SPECIALTY
Retiring
aws certified SAP on AWS SPECIALTY
aws certified Machine Learning SPECIALTY


3 months
6 months to 1 year
18 months to 2 years
Associate Level
Professional Level
Specialty
QA
Foundational level
You
aws certified Cloud Practitioner FOUNDATIONAL
aws certified Solutions Architect ASSOCIATE
aws certified SysOps Administrator ASSOCIATE
aws certified Developer ASSOCIATE
aws certified Data Engineer ASSOCIATE
Beta
aws certified Solutions Architect PROFESSIONAL
aws certified DevOps Engineer PROFESSIONAL
aws certified Advanced Networking SPECIALTY
aws certified Security SPECIALTY
aws certified Database SPECIALTY
aws certified Data Analytics SPECIALTY
Retiring
aws certified SAP on AWS SPECIALTY
aws certified Machine Learning SPECIALTY

3 months | 6 months to 1 year | 18 months to 2 years

Managers

Foundational level

You

aws certified **Cloud Practitioner** FOUNDATIONAL

AWS Cost & Usage Report

AWS Cost Explorer

AWS Financial Management Guide

Associate Level

aws certified **Solutions Architect** ASSOCIATE

aws certified **SysOps Administrator** ASSOCIATE

aws certified **Developer** ASSOCIATE

aws certified **Data Engineer** ASSOCIATE Beta

Professional Level

aws certified **Solutions Architect** PROFESSIONAL

aws certified **DevOps Engineer** PROFESSIONAL

Specialty

aws certified **Advanced Networking** SPECIALTY

aws certified **Security** SPECIALTY

aws certified **Database** SPECIALTY

aws certified **Data Analytics** SPECIALTY Retiring

aws certified **SAP on AWS** SPECIALTY

aws certified **Machine Learning** SPECIALTY

3 months → 6 months to 1 year

FinOps

Foundational level

You

AWS Cost & Usage Report

AWS Cost Explorer

AWS Financial Management Guide

Associate Level

Professional Level

Specialty

Best luck !

Continue your journey and connect on:

https://www.youtube.com/@AWSwithChetan

https://www.awswithchetan.com