

Multitier architecture of Cloud Apps To explain the concepts of multitier architecture (which is also called n-tier architecture), let us start with understanding Client-Server Relationship.

In computing, a client is a piece of computer hardware or software that accesses a service made available by a server as part of the client-server model of computer networks. The server is often (but not always) on another computer system, in which case the client accesses the service by way of a network. A client can be any device – a computer, a tablet or a mobile.

Therefore, Client-server denotes a relationship between cooperating programs in an application, composed of clients initiating requests for services and servers providing that function or service. There are 3 main categories of client-server computing:

One-Tier Architecture consists of a simple program running on a single computer without requiring access to the network. User requests don't manage any network protocols, therefore the code is simple and the network is relieved of the extra traffic.

Two-Tier Architecture consists of the client, the server, and the protocol that links the two tiers. The Graphical User Interface code resides on the client host and the domain logic resides on the server host. Domain logic or business logic is the part of the program that encodes the real-world business rules that determine how data can be created, stored, and changed. Business logic should be distinguished from business rules. Business logic is the portion of an enterprise system which determines how data is transformed or calculated, and how it is routed to people or software (workflow). Business rules are formal expressions of business policy. Anything that is a process or procedure is business logic, and anything that is neither a process nor a procedure is a business rule. For example, welcoming a new visitor is a process (workflow) consisting of steps to be taken, whereas saying every new visitor must be welcomed is a business rule. Further, business logic is procedural whereas business rules are declarative.

Multi-Tier Architecture (often referred to as *n*-tier architecture) or multilayered architecture is a client-server architecture in which presentation, application processing and data management functions are physically separated. The most widespread use of multitier architecture is the three-tier architecture.

Three-tier architecture is a client-server software architecture pattern in which the user interface (presentation tier), functional process logic (logic or application tier), computer data storage and data access (data tier) are developed and maintained as independent modules, most often on separate platforms.

The presentation tier is the front end layer in the 3-tier system and consists of the user interface. This user interface is often a graphical one accessible through a web browser or web-based application and which displays content and information useful to an end user. This tier is often built on web technologies such as HTML5, JavaScript, CSS, or through other popular web development frameworks, and communicates with others layers through API calls. An application program interface (API) is code that allows two software programs to communicate with each other.

The logic or application tier contains the functional business logic which drives an application's core capabilities. The logical tier is pulled out from the presentation tier and, as its own layer, it

controls an application's functionality by performing detailed processing. It's often written in Java, .NET, C#, Python, C++, etc.

The data tier comprises of the database/data storage system and data access layer. Examples of such database

systems are MySQL, Oracle, PostgreSQL, Microsoft SQL Server, MongoDB, etc. Data is accessed by the application layer via API calls.

The typical structure for a 3-tier architecture deployment would have the presentation tier deployed to a desktop, laptop, tablet or mobile device either via a web browser or a web-based application utilizing a web server. The underlying application tier is usually hosted on one or more application servers, but can also be hosted in the cloud, or on a dedicated workstation depending on the complexity and processing power needed by the application. And the data layer would normally comprise of one or more relational databases, big data sources, or other types of database systems hosted either on-premises or in the cloud.

A simple example of a 3-tier architecture in action would be logging into a media account such as Netflix and watching a video. You start by logging in either via the web or via a mobile application. Once you've logged in you might access a specific video through the Netflix interface which is the presentation tier used by you as an end user. Once you've selected a video that information is passed on to the application tier which will query the data tier to call the information or in this case a video back up to the presentation tier. This happens every time you access a video from most media sites.

So, now the question arises, why do we need this complex multitier architecture?

As mentioned, modularizing different tiers of an application gives development teams the ability to develop and enhance a product with greater speed than developing a singular code base because a specific layer can be upgraded with minimal impact on the other layers. It can also help improve development efficiency by allowing teams to focus on their core competencies. Many development teams have separate developers who specialize in front-end, server back-end, and data back-end development, by modularizing these parts of an application you no longer have to rely on full stack developers and can better utilize the specialties of each team.

Scalability is another great advantage of a 3-layer architecture. By separating out the different layers you can scale each independently depending on the need at any given time. For example, if you are receiving many web requests but not many requests which affect your application layer, you can scale your web servers without touching your application servers. Similarly, if you are receiving many large application requests from only a handful of web users, you can scale out your application and data layers to meet those requests without touch your web servers. This allows you to load balance each layer independently, improving overall performance with minimal resources.

Additionally, the independence created from modularizing the different tiers gives you many deployment options. For example, you may choose to have your web servers hosted in a public or private cloud while you're application and data layers may be hosted onsite. Or you may have your application and data layers hosted in the cloud while your web servers may be locally hosted, or any combination thereof.

By having disparate layers you can also increase reliability and availability by hosting different parts of your application on different servers and utilizing cached results. With a full stack system you have to worry about a server going down and greatly affecting performance throughout your entire system, but with a 3-layer application, the increased independence created when physically separating different parts of an application minimizes performance issues when a server goes down. Hope, next time when you will go to hire an architect, this will help you.

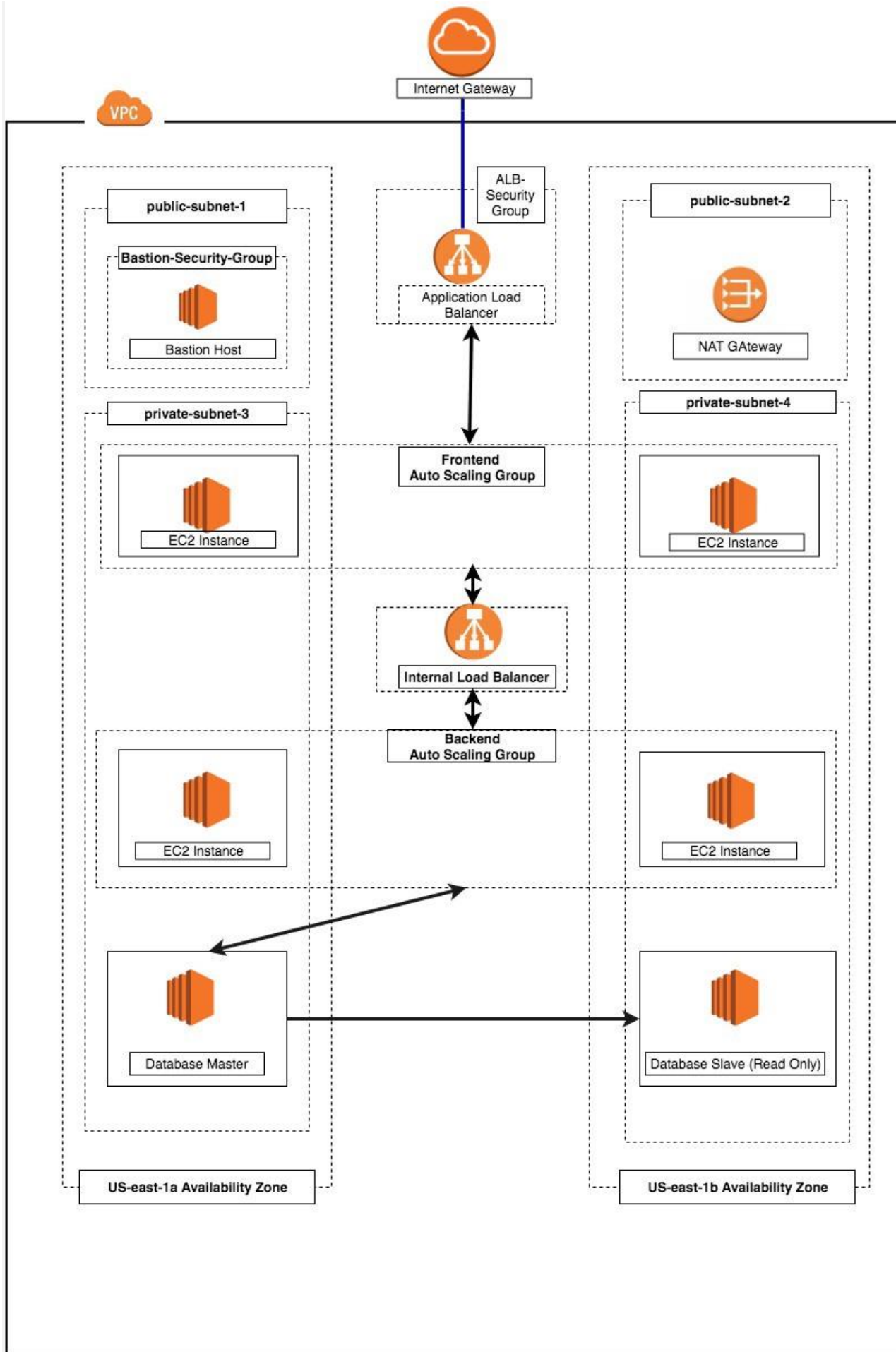
<https://www.linkedin.com/pulse/decoding-few-technical-jargons-recruiters-frontend-backend-pandey/>

Designing a Three-Tier Architecture in AWS

<https://medium.com/the-andela-way/designing-a-three-tier-architecture-in-aws-e5c24671f124>
[Kenechukwu Nnamani](#)

A three-tier architecture is a software architecture pattern where the application is broken down into three logical tiers: the presentation layer, the business logic layer and the data storage layer. This architecture is used in a client-server application such as a web application that has the frontend, the backend and the database. Each of these layers or tiers does a specific task and can be managed independently of each other. This is a shift from the monolithic way of building an application where the frontend, the backend and the database are both sitting in one place.

Amazon Web Service (AWS) is a cloud platform that provides different cloud computing services to their customers. To view a list of all AWS services and products, click on this [link](#). In this article, we shall be making use of the following AWS services to design and build a three-tier cloud infrastructure: [Elastic Compute Cloud \(EC2\)](#), [Auto Scaling Group](#), [Virtual Private Cloud \(VPC\)](#), [Elastic Load Balancer \(ELB\)](#), [Security Groups](#) and the [Internet Gateway](#). Our infrastructure will be designed to be highly available and fault tolerant.



What are we solving for?

1. **Modularity:** The essence of having a three-tier architecture is to modularize our application such that each part can be managed independently of each other. With modularity, teams can focus on different tiers of the application and changes made as quickly as possible. Also, modularization helps us recover quickly from an unexpected disaster by focusing solely on the faulty part.
2. **Scalability:** Each tier of the architecture can scale horizontally to support the traffic and request demand coming to it. This can easily be done by adding more EC2 instances to each tier and load balancing across them. For instance, assuming we have two EC2 instances serving our backend application and each of the EC2 instances is working at 80% CPU utilization, we can easily scale the backend tier by adding more EC2 instances to it so that the load can be distributed. We can also automatically reduce the number of the EC2 instances when the load is less.
3. **High Availability:** With the traditional data centre, our application is sitting in one geographical location. If there is an earthquake, flooding or even power outage in that location where our application is hosted, our application will not be available. With AWS, we can design our infrastructure to be highly available by hosting our application in different locations known as the [availability zones](#).
4. **Fault Tolerant:** We want our infrastructure to comfortably adapt to any unexpected change both to traffic and fault. This is usually done by adding a redundant system that will account for such a hike

in traffic when it does occur. So instead of having two EC2 instances working at 50% each, such that when one instance goes bad, the other instance will be working at 100% capacity until a new instance is brought up by our Auto Scaling Group, we have extra instance making it three instances working at approximately 35% each. This is usually a tradeoff made against the cost of setting up a redundant system.

5. **Security:** We want to design an infrastructure that is highly secured and protected from the prying eyes of hackers. As much as possible, we want to avoid exposing our interactions within the application over the internet. This simply means that the application will communicate within themselves with a private IP. The presentation (frontend) tier of the infrastructure will be in a private subnet (the subnet with no public IP assigned to its instances) within the VPC. Users can only reach the frontend through the application load balancer. The backend and the database tier will also be in the private subnet because we do not want to expose them over the internet. We will set up the Bastion host for remote SSH and a NAT gateway for our private subnets to access the internet. The AWS security group helps us limit access to our infrastructure setup.

Before we get started

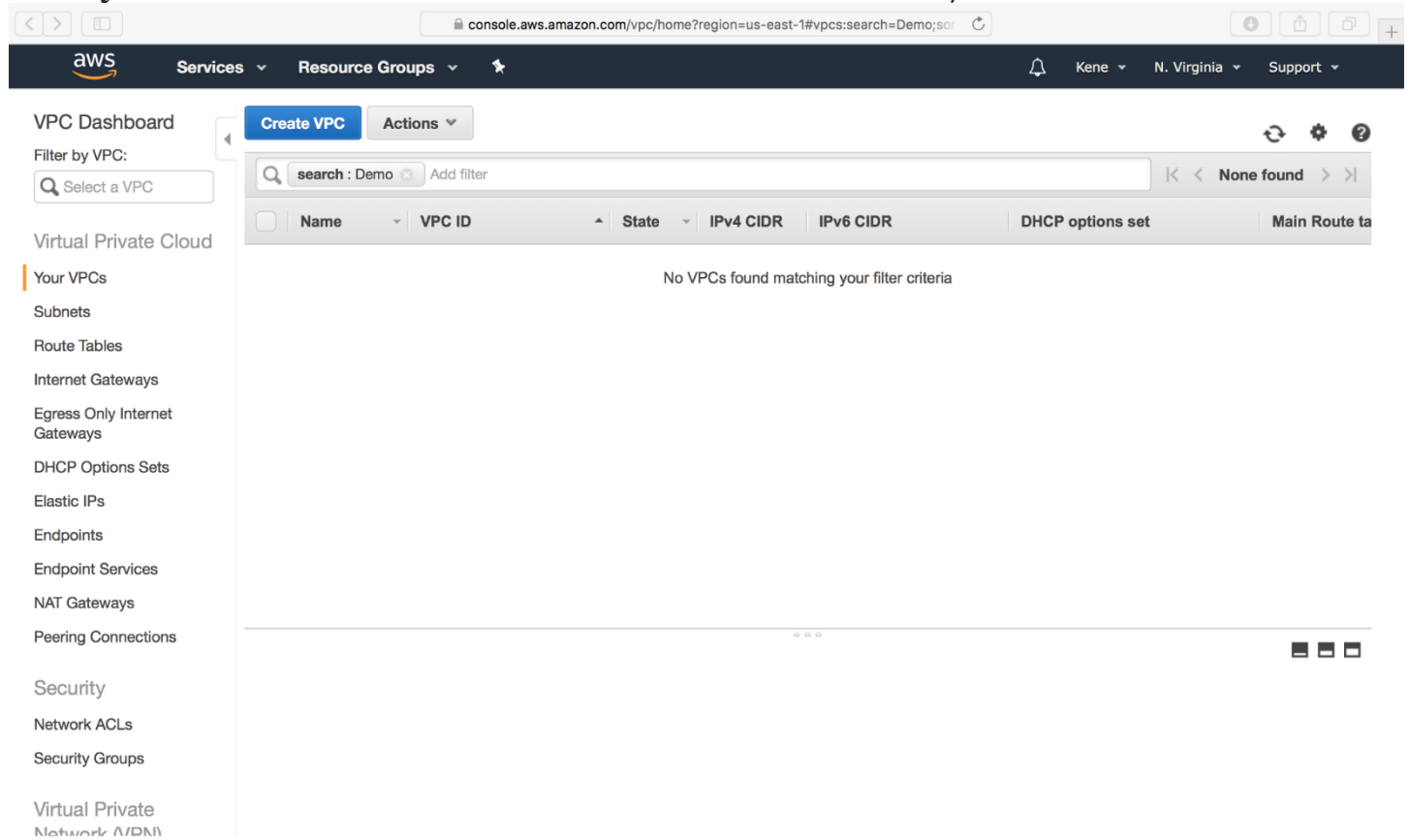
To follow along, you need to have an [AWS](#) account. We shall be making use of the AWS free-tier resources so we do not incur charges while learning.

Note: At the end of this tutorial, you need to stop and delete all the resources such as the EC2 instances, Auto Scaling Group, Elastic Load Balancer etc you set up. Otherwise, you get charged for it when you keep them running for a long.

Let's Begin

1. **Setup the Virtual Private Cloud (VPC):** VPC stands for Virtual Private Cloud (VPC). It is a virtual network where you create and manage your AWS resource in a more secure and scalable manner. Go to the VPC section of the AWS services, and click on the **Create VPC** button.

Give your VPC a name and a [CIDR](#) block of 10.0.0.0/16



Create VPC

console.aws.amazon.com/vpc/home?region=us-east-1#CreateVpc:

aws Services Resource Groups

VPCs > Create VPC

Create VPC

A VPC is an isolated portion of the AWS cloud populated by AWS objects, such as Amazon EC2 instances. You must specify an IPv4 address range for your VPC. Specify the IPv4 address range as a Classless Inter-Domain Routing (CIDR) block; for example, 10.0.0.0/16. You cannot specify an IPv4 CIDR block larger than /16. You can optionally associate an Amazon-provided IPv6 CIDR block with the VPC.

Name tag ⓘ

IPv4 CIDR block* ⓘ

IPv6 CIDR block ☒ No IPv6 CIDR Block ⓘ
☐ Amazon provided IPv6 CIDR block

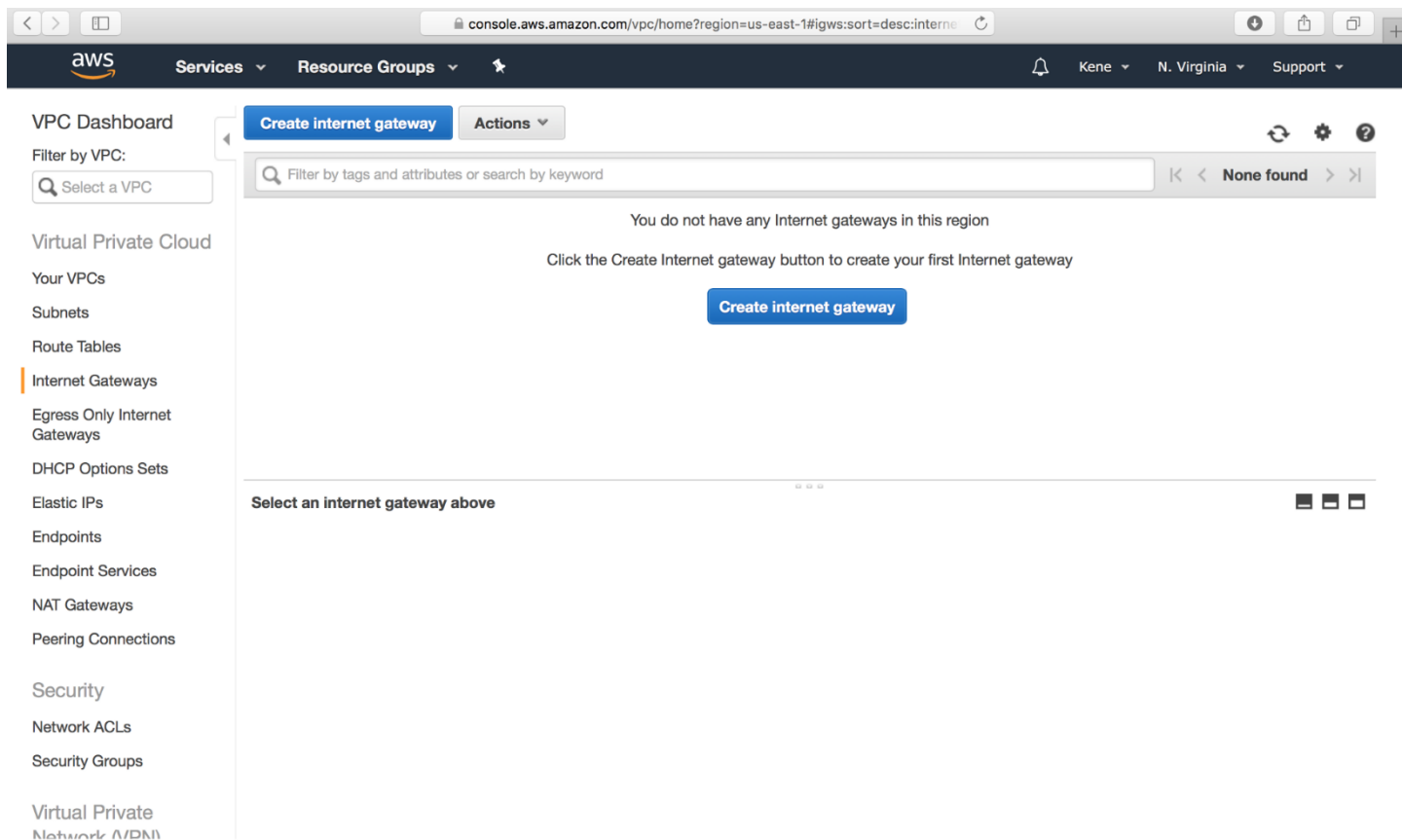
Tenancy ⓘ

* Required

Cancel Create

Create VPC

2. Setup the Internet Gateway: The Internet Gateway allows communication between the EC2 instances in the VPC and the internet. To create the Internet Gateway, navigate to the **Internet Gateways** page and then click on **Create internet gateway** button.



Create internet gateway

This screenshot shows the 'Create internet gateway' form in the AWS console. The browser's address bar shows the URL 'console.aws.amazon.com/vpc/home?region=us-east-1#Create%20Internet%20Gateway'. The breadcrumb navigation at the top reads 'Internet gateways > Create internet gateway'. The main heading is 'Create internet gateway'. Below the heading, a descriptive text states: 'An internet gateway is a virtual router that connects a VPC to the internet. To create a new internet gateway specify the name for the gateway below.' There is a text input field labeled 'Name tag' with the value 'Demo-IG' and an information icon to its right. At the bottom left, there is a note '* Required'. At the bottom right, there are two buttons: 'Cancel' and 'Create'.

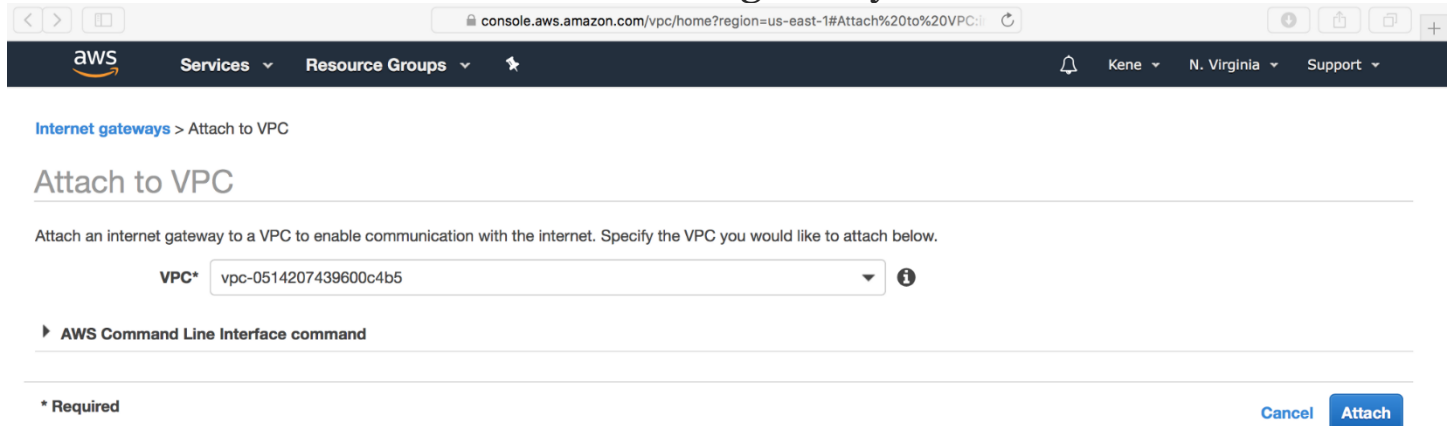
Create Internet Gateway

We need to attach our VPC to the internet gateway. To do that:

a. we select the internet gateway

b. Click on the **Actions** button and then select Attach to VPC.

c. Select the VPC to attach the internet gateway and click **Attach**

The screenshot shows the AWS Management Console interface for the 'Attach to VPC' action. At the top, the breadcrumb navigation reads 'Internet gateways > Attach to VPC'. The main heading is 'Attach to VPC'. Below this, a descriptive text states: 'Attach an internet gateway to a VPC to enable communication with the internet. Specify the VPC you would like to attach below.' There is a dropdown menu labeled 'VPC*' with the value 'vpc-0514207439600c4b5' selected. To the right of the dropdown is an information icon. Below the dropdown, there is a section titled 'AWS Command Line Interface command' with a right-pointing arrow. At the bottom left, there is a note '* Required'. At the bottom right, there are two buttons: 'Cancel' and 'Attach'.

Attach the VPC to the internet gateway

3. Create 4 Subnets: The subnet is a way for us to group our resources within the VPC with their IP range. A subnet can be public or private. EC2 instances within a public subnet have public IPs and can directly access the internet while those in the private subnet does not have public IPs and can only access the internet through a [NAT](#) gateway.

For our setup, we shall be creating the following subnets with the corresponding IP ranges.

- demo-public-subnet-1 | CIDR (10.0.1.0/24) | Availability Zone (us-east-1a)
- demo-public-subnet-2 | CIDR (10.0.2.0/24) | Availability Zone (us-east-1b)

- demo-private-subnet-3 | CIDR (10.0.3.0/24) | Availability Zone (us-east-1a)
- demo-private-subnet-4 | CIDR(10.0.4.0/24) | Availability Zone (us-east-1b)

Subnets > Create subnet

Create subnet

Specify your subnet's IP address block in CIDR format; for example, 10.0.0.0/24. IPv4 block sizes must be between a /16 netmask and /28 netmask, and can be the same size as your VPC. An IPv6 CIDR block must be a /64 CIDR block.

Name tag

VPC*

VPC CIDRs	CIDR	Status	Status Reason
	10.0.0.0/16	associated	

Availability Zone

IPv4 CIDR block*

* Required

[Cancel](#) [Create](#)

create subnets

VPC Dashboard

Filter by VPC:

Virtual Private Cloud

Your VPCs

Subnets

Route Tables

Internet Gateways

Egress Only Internet Gateways

DHCP Options Sets

Classic IPs

[Create subnet](#) [Actions](#)

search : Demo [Add filter](#)

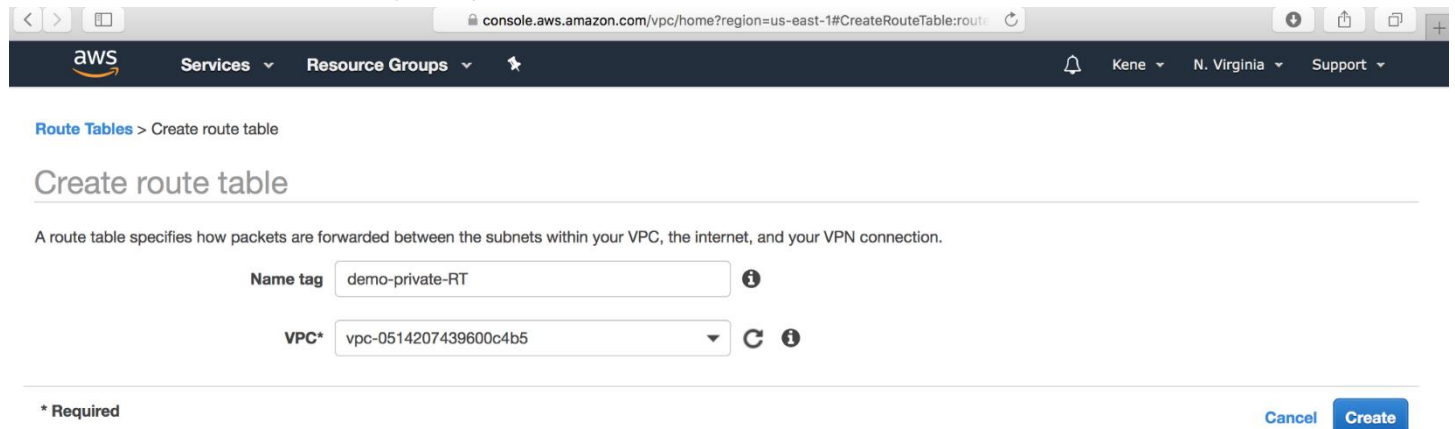
	Name	Subnet ID	State	VPC	IPv4 CIDR	Available IPv4
<input type="checkbox"/>	demo-private-subnet-3	subnet-042f12f4806a1fa66	available	vpc-0514207439600c4b5 ...	10.0.3.0/24	251
<input type="checkbox"/>	demo-public-subnet-2	subnet-0955b5c3444d27dd8	available	vpc-0514207439600c4b5 ...	10.0.2.0/24	251
<input type="checkbox"/>	demo-private-subnet-4	subnet-0a74970565af18075	available	vpc-0514207439600c4b5 ...	10.0.4.0/24	251
<input type="checkbox"/>	demo-public-subnet-1	subnet-0a7c26278766da22f	available	vpc-0514207439600c4b5 ...	10.0.1.0/24	251

four subnets in our VPC

4. Create Two Route Tables: Route tables is a set of rule that determines how data moves within our network. We need two route tables; private route table and public route table. The public route table will define which subnets

that will have direct access to the internet (ie public subnets) while the private route table will define which subnet goes through the NAT gateway (ie private subnet).

To create route tables, navigate over to the **Route Tables** page and click on **Create route table** button.



Route Tables > Create route table

Create route table

A route table specifies how packets are forwarded between the subnets within your VPC, the internet, and your VPN connection.

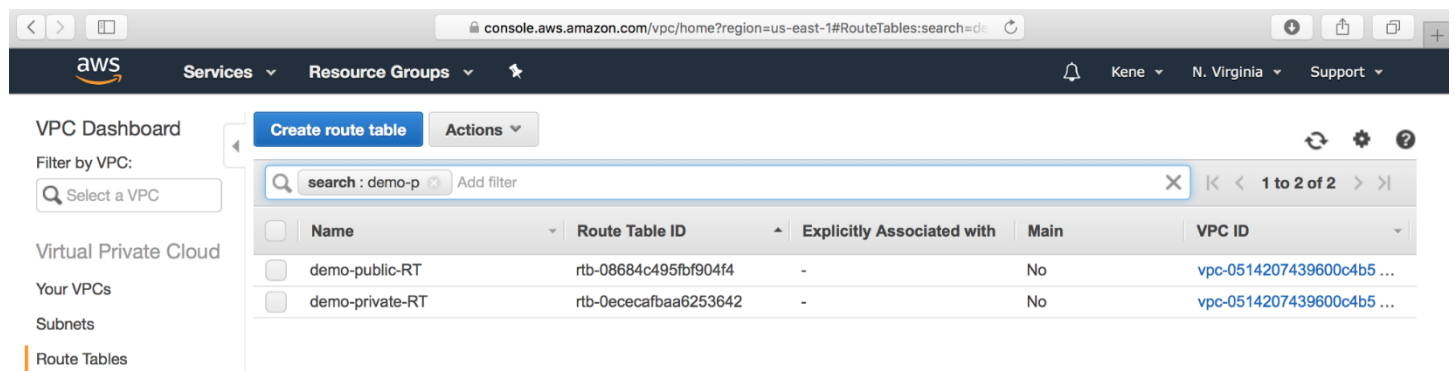
Name tag

VPC*

* Required

[Cancel](#) [Create](#)

Create Route Table



VPC Dashboard

Filter by VPC:

Virtual Private Cloud

Your VPCs

Subnets

Route Tables

[Create route table](#) [Actions](#)

search : demo-p [Add filter](#)

<input type="checkbox"/>	Name	Route Table ID	Explicitly Associated with	Main	VPC ID
<input type="checkbox"/>	demo-public-RT	rtb-08684c495fbf904f4	-	No	vpc-0514207439600c4b5 ...
<input type="checkbox"/>	demo-private-RT	rtb-0ececfa6aa6253642	-	No	vpc-0514207439600c4b5 ...

Private and Public Route Tables

The public and the private subnet needs to be associated with the public and the private route table respectively.

To do that, we select the route table and then choose the **Subnet Association** tab.

console.aws.amazon.com/vpc/home?region=us-east-1#RouteTables:search=de

Services Resource Groups

VPC Dashboard

Filter by VPC: Select a VPC

Virtual Private Cloud

Your VPCs

Subnets

Route Tables

Internet Gateways

Egress Only Internet Gateways

DHCP Options Sets

Elastic IPs

Endpoints

Endpoint Services

NAT Gateways

Peering Connections

Security

Network ACLs

Create route table Actions

search : demo-p Add filter

1 to 2 of 2

Name	Route Table ID	Explicitly Associated with	Main	VPC ID
demo-public-RT	rtb-08684c495fbf904f4	-	No	vpc-0514207439600c4b5 ...
demo-private-RT	rtb-0ececfa6253642	-	No	vpc-0514207439600c4b5 ...

Route Table: rtb-08684c495fbf904f4

Summary Routes Subnet Associations Route Propagation Tags

Edit subnet associations

None found

Subnet ID	IPv4 CIDR	IPv6 CIDR
-----------	-----------	-----------

You do not have any subnet associations.

Subnet Associations

console.aws.amazon.com/vpc/home?region=us-east-1#EditRouteTableSubnet/

Services Resource Groups

Route Tables > Edit subnet associations

Edit subnet associations

Route table rtb-08684c495fbf904f4 (demo-public-RT)

Associated subnets subnet-0955b5c3444d27dd8 subnet-0a7c26278766da22f

Filter by attributes or search by keyword

1 to 4 of 4

Subnet ID	IPv4 CIDR	IPv6 CIDR	Current Route 1
subnet-0955b5c3444d27dd8 demo-public-subnet-2	10.0.2.0/24	-	Main
subnet-0a7c26278766da22f demo-public-subnet-1	10.0.1.0/24	-	Main
subnet-042f12f4806a1fa66 demo-private-subnet-3	10.0.3.0/24	-	Main
subnet-0a74970565af18075 demo-private-subnet-4	10.0.4.0/24	-	Main

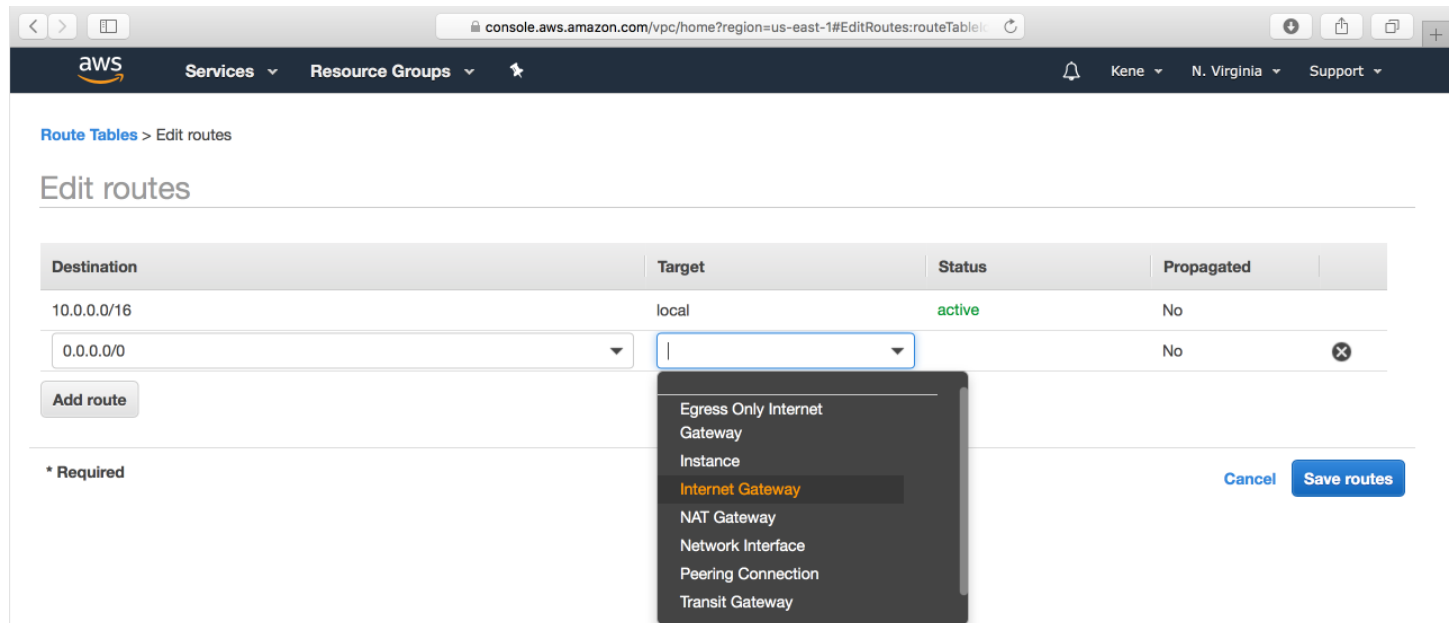
* Required

Cancel Save

Select the public subnet for the public route table

We also need to route the traffic to the internet through the **internet gateway** for our public route table.

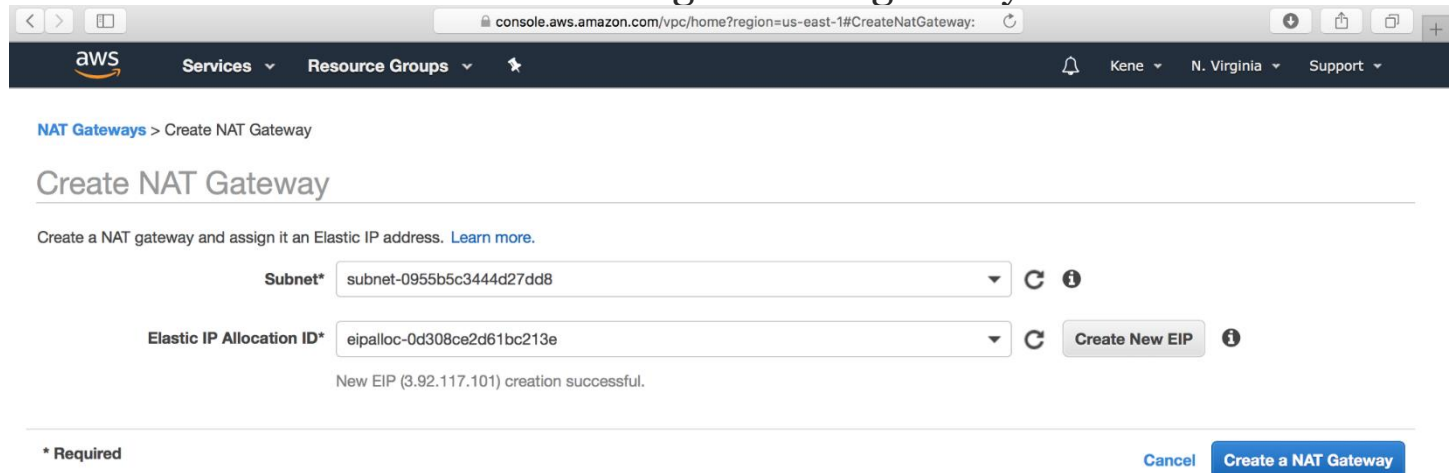
To do that we select the public route table and then choose the **Routes** tab. The rule should be similar to the one shown below:



Edit Route for the public route table

5. Create the NAT Gateway: The NAT gateway enables the EC2 instances in the private subnet to access the internet. The NAT Gateway is an AWS managed service for the NAT instance. To create the NAT gateway, navigate to the NAT Gateways page, and then click on the **Create NAT Gateway**.

Please ensure that you know the Subnet ID for the **demo-public-subnet-2**. This will be needed when creating the NAT gateway.



Create NAT Gateway

Now that we have the NAT gateway, we are going to edit the private route table to make use of the NAT gateway to access the internet.

The screenshot shows the 'Edit routes' page in the AWS console. The URL is `console.aws.amazon.com/vpc/home?region=us-east-1#EditRoutes:routeTableId=rtb-0e9cafbba6253642`. The page has a header with the AWS logo, navigation tabs (Services, Resource Groups), and user information (Kene, N. Virginia, Support). The breadcrumb is 'Route Tables > Edit routes'. The main heading is 'Edit routes'. Below it is a table with columns: Destination, Target, Status, and Propagated. The first row shows a route for destination `10.0.0.0/16` with target `local`, status `active`, and `No` propagated. The second row shows a route for destination `0.0.0.0/0` with target `nat-` (selected from a dropdown), status `active`, and `No` propagated. A dropdown menu is open for the target, showing `nat-04192c1270260ce54`. There is an 'Add route' button on the left and 'Cancel' and 'Save routes' buttons on the right. A note '* Required' is at the bottom left.

Destination	Target	Status	Propagated
10.0.0.0/16	local	active	No
0.0.0.0/0	nat-	active	No

Edit the Private Route Table

The screenshot shows the 'Route Tables' page in the AWS console. The URL is `console.aws.amazon.com/vpc/home?region=us-east-1#RouteTables:search=demo-p`. The page has a header with the AWS logo, navigation tabs (Services, Resource Groups), and user information (Kene, N. Virginia, Support). The breadcrumb is 'Route Tables'. The main heading is 'Route Tables'. Below it is a table with columns: Name, Route Table ID, Explicitly Associated with, Main, and VPC ID. The first row is 'demo-public-RT' with ID `rtb-08684c495fb904f4`, associated with 2 subnets, not main, and VPC ID `vpc-0514207439600c4b5`. The second row is 'demo-private-RT' with ID `rtb-0e9cafbba6253642`, associated with 2 subnets, not main, and VPC ID `vpc-0514207439600c4b5`. Below the table is a section for 'Route Table: rtb-0e9cafbba6253642' with tabs: Summary, Routes, Subnet Associations, Route Propagation, and Tags. The 'Routes' tab is selected. There is an 'Edit routes' button. Below the tabs is a 'View' dropdown set to 'All routes'. Below that is a table with columns: Destination, Target, Status, and Propagated. The first row shows a route for destination `10.0.0.0/16` with target `local`, status `active`, and `No` propagated. The second row shows a route for destination `0.0.0.0/0` with target `nat-04192c1270260ce54`, status `active`, and `No` propagated.

Name	Route Table ID	Explicitly Associated with	Main	VPC ID
demo-public-RT	rtb-08684c495fb904f4	2 subnets	No	vpc-0514207439600c4b5 ...
demo-private-RT	rtb-0e9cafbba6253642	2 subnets	No	vpc-0514207439600c4b5 ...

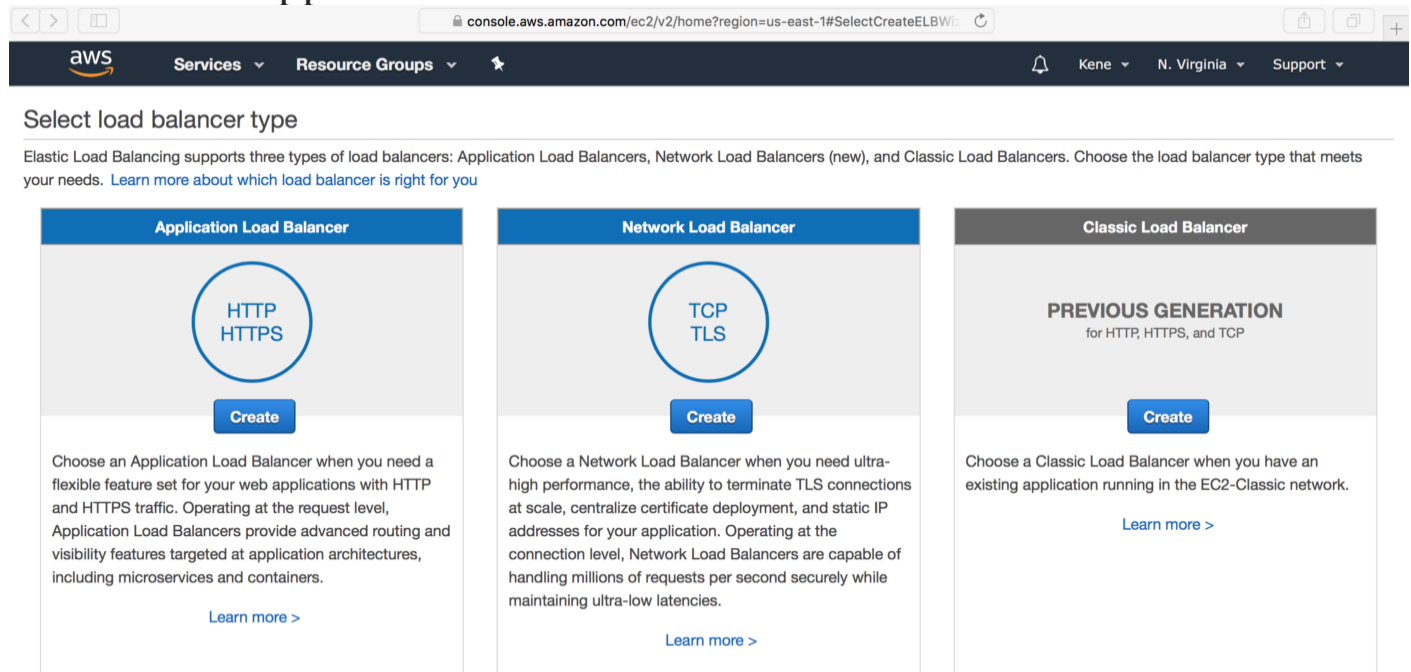
Destination	Target	Status	Propagated
10.0.0.0/16	local	active	No
0.0.0.0/0	nat-04192c1270260ce54	active	No

Edit Private Route Table to use NAT Gateway for private EC2 instances

6. Create Elastic Load Balancer: From our architecture, our frontend tier can only accept traffic from the elastic load balancer which connects directly with the internet gateway while our backend tier will receive traffic through the internal load balancer. The essence of the load balancer is to

distribute load across the EC2 instances serving that application. If however, the application is using sessions, then the application needs to be rewritten such that sessions can be stored in either the Elastic Cache or the DynamoDB. To create the two load balancers needed in our architecture, we navigate to the **Load Balancer** page and click on **Create Load Balancer**.

a. Select the Application Load Balancer.



Select Application Load Balancer

b. Click on the **Create button**

c. Configure the Load Balancer with a name. Select **internet facing for the load balancer that we will use to communicate with the frontend and **internal** for the one we will use for our backend.**

Step 1: Configure Load Balancer

Basic Configuration

To configure your load balancer, provide a name, select a scheme, specify one or more listeners, and select a network. The default configuration is an Internet-facing load balancer in the selected network with a listener that receives HTTP traffic on port 80.

Name (i) Demo-ELB

Scheme (i) ☒ internet-facing ☐ internal

IP address type (i) ipv4

Listeners

A listener is a process that checks for connection requests, using the protocol and port that you configured.

Load Balancer Protocol	Load Balancer Port
HTTP	80

Add listener

Internet Facing Load Balancer for the Frontend tier

Step 1: Configure Load Balancer

Basic Configuration

To configure your load balancer, provide a name, select a scheme, specify one or more listeners, and select a network. The default configuration is an Internet-facing load balancer in the selected network with a listener that receives HTTP traffic on port 80.

Name (i) Demo-internal-ELB

Scheme (i) ☐ internet-facing ☒ internal

IP address type (i) ipv4

Listeners

A listener is a process that checks for connection requests, using the protocol and port that you configured.

Load Balancer Protocol	Load Balancer Port
HTTP	80

Add listener

Internal Load Balancer for the Backend Tier

d. Under the Availability Zone, for the **internet facing Load Balancer**, we will select the two **public subnets** while for our **internal Load Balancer**, we will select the two **private subnet**.

Availability Zones

Specify the Availability Zones to enable for your load balancer. The load balancer routes traffic to the targets in these Availability Zones only. You can specify only one subnet per Availability Zone. You must specify subnets from at least two Availability Zones to increase the availability of your load balancer.

VPC ⓘ vpc-0514207439600c4b5 (10.0.0.0/16) | Demo-VPC

<input type="checkbox"/> Availability Zone	Subnet ID	Subnet IPv4 CIDR	Name	
<input checked="" type="checkbox"/> us-east-1a	subnet-0a7c26278766da22f	10.0.1.0/24	demo-public-subnet-1	Change subnet...
<input checked="" type="checkbox"/> us-east-1b	subnet-0955b5c3444d27dd8	10.0.2.0/24	demo-public-subnet-2	Change subnet...

Availability Zone for the Internet Facing Load Balancer

Availability Zones

Specify the Availability Zones to enable for your load balancer. The load balancer routes traffic to the targets in these Availability Zones only. You can specify only one subnet per Availability Zone. You must specify subnets from at least two Availability Zones to increase the availability of your load balancer.

VPC ⓘ vpc-0514207439600c4b5 (10.0.0.0/16) | Demo-VPC

<input type="checkbox"/> Availability Zone	Subnet ID	Subnet IPv4 CIDR	Name	
<input checked="" type="checkbox"/> us-east-1a	subnet-042f12f4806a1fa66	10.0.3.0/24	demo-private-subnet-3	Change subnet...
<input checked="" type="checkbox"/> us-east-1b	subnet-0a74970565af18075	10.0.4.0/24	demo-private-subnet-4	Change subnet...

▸ Tags

Availability Zone for the internal Load Balancer

e. Under the Security Group, we **only need** to allow ports that the application needs. For instance, we need to allow **HTTP port 80 and/or HTTPS port 443** on our **internet facing load balancer**. For the **internal load balancer**, we only open the port that the backend runs on (eg: port 3000) and the make such port **only open to the security group of the frontend**. This will allow only the frontend to have access to that port within our architecture.

f. Under the **Configure Routing**, we need to configure our Target Group to have the **Target type** of **instance**. We will give the **Target Group** a name that will enable us to identify it. This is will be needed when we will create our **Auto Scaling Group**. For example, we can name the Target Group of our frontend to be **Demo-Frontend-TG**

Skip the Register Targets and then go ahead and review the configuration and then click on the **Create** button.

7. Auto Scaling Group: We can simply create like two EC2 instances and directly attach these EC2 instances to our load balancer. The problem with that is that our application will no longer scale to accommodate traffic or shrink when there is no traffic to save cost. With Auto Scaling Group, we can achieve this feat. Auto Scaling Group is can automatically adjust the size of the EC2 instances serving the application based on need. This is what makes it a good approach rather than directly attaching the EC2 instances to the load balancer.

To create an Auto Scaling Group, navigate to the **Auto Scaling Group** page, Click on the **Create Auto Scaling Group** button.

a. Auto Scaling Group needs to have a common configuration that instances within it **MUST** have. This common configuration is made possible with the help of the **Launch Configuration**. In our Launch configuration, under the Choose AMI, the best practice is to choose the AMI which contains the application and its dependencies bundled together. You can also create your custom AMI in AWS.

1. Choose AMI2. Choose Instance Type3. Configure details4. Add Storage5. Configure Security Group6. Review

Create Launch Configuration

Cancel and Exit

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Quick Start

My AMIs

AWS Marketplace

Community AMIs

Ownership


☒ Owned by me☐ Shared with me

Architecture

☐ 32-bit☐ 64-bit

Root device type


☐ EBS☐ Instance store

Custom Frontend AMI - ami-061a6d34b817848f2

Root device type: ebsVirtualization type: hvmOwner: 839162139045

Select


64-bit

Custom Postgres Database AMI - ami-093d27bf3a986ace7

Root device type: ebsVirtualization type: hvmOwner: 839162139045

Select

64-bit

Custom Backend AMI - ami-0aae9f86142dc1126

Root device type: ebsVirtualization type: hvmOwner: 839162139045

Select

64-bit

Custom AMI for each tier of our application

b. Choose the appropriate instance type. For a demo, I recommend you choose t2.micro (free tier eligible) so that you do not incur charges.

c. Under the Configure details, give the Launch Configuration a name, eg **Demo-Frontend-LC**. Also, under the **Advance Details** dropdown, the **User data** is provided for you to type in a command that is needed to install dependencies and start the application.

console.aws.amazon.com/ec2/autoscaling/home?region=us-east-1#CreateLaunch

aws Services Resource Groups

1. Choose AMI 2. Choose Instance Type 3. Configure details 4. Add Storage 5. Configure Security Group 6. Review

Create Launch Configuration

Name ⓘ Demo-Frontend-LC

Purchasing option ⓘ ☐ Request Spot Instances

IAM role ⓘ None

Monitoring ⓘ ☐ Enable CloudWatch detailed monitoring [Learn more](#)

▼ **Advanced Details**

Kernel ID ⓘ Use default

RAM Disk ID ⓘ Use default

User data ⓘ ☒ As text ☐ As file ☐ Input is already base64 encoded

```
npm start
```

IP Address Type ⓘ ☒ Only assign a public IP address to instances launched in the default VPC and subnet. (default)
☐ Assign a public IP address to every instance.
☐ Do not assign a public IP address to any instances.
Note: this option only affects instances launched into an Amazon VPC.

[Cancel](#) [Previous](#) [Skip Storage](#) [Next: Add Storage](#)

d. Again under the security group, we want to only allow the ports that are necessary for our application.

e. Review the Configuration and Click on **Create Launch Configuration** button. Go ahead and create a new key pair. Ensure you download it before proceeding.

f. Now we have our Launch Configuration, we can finish up with the creating our Auto Scaling Group. Use the below image as a template for setting up yours.

Services
Resource Groups

Kene
N. Virginia
Support

1. Configure Auto Scaling group details
2. Configure scaling policies
3. Configure Notifications
4. Configure Tags
5. Review

Create Auto Scaling Group

Group name

Frontend-ASG

Launch Configuration

Demo-Frontend-LC

Group size

Start with 2 instances

Network

vpc-0514207439600c4b5 (10.0.0.0/16) | Demo-VPC

Create new VPC

Subnet

subnet-042f12f4806a1fa66(10.0.3.0/24) | demo-private-subnet-3 | us-east-1a

subnet-0a74970565af18075(10.0.4.0/24) | demo-private-subnet-4 | us-east-1b

Create new subnet

No public IP addresses will be assigned

None of the instances in this Auto Scaling group will be assigned a public IP address because you have not chosen to launch in your default VPC and subnet.

You can ensure a public IP address is assigned to instances launched with this configuration by selecting only default subnets of your default VPC.

[Learn more](#) about IP addressing in an Amazon VPC.

Auto Scaling Group 1

Advanced Details

Load Balancing

☒ Receive traffic from one or more load balancers

Learn about Elastic Load Balancing

Classic Load Balancers

Target Groups

Demo-Frontend-TG

Health Check Type

☒ ELB
☐ EC2

Health Check Grace Period

300 seconds

Monitoring

Amazon EC2 Detailed Monitoring metrics, which are provided at 1 minute frequency, are not enabled for the launch configuration Demo-Frontend-LC. Instances launched from it will use Basic Monitoring metrics, provided at 5 minute frequency.

Learn more

Instance Protection

Service-Linked Role

AWSServiceRoleForAutoScaling

View Role in IAM

Cancel

Next: Configure scaling policies

Auto Scaling Group 2

g. Under the Configure scaling policies, we want to add one instance when the CPU is greater than or equal to 80% and to scale down when the CPU is less than or equal to 50%. Use the image as a template.

Create Auto Scaling Group

You can optionally add scaling policies if you want to adjust the size (number of instances) of your group automatically. A scaling policy is a set of instructions for making such adjustments in response to an Amazon CloudWatch alarm that you assign to it. In each policy, you can choose to add or remove a specific number of instances or a percentage of the existing group size, or you can set the group to an exact size. When the alarm triggers, it will execute the policy and adjust the size of your group accordingly. [Learn more](#) about scaling policies.

☐ Keep this group at its initial size

☒ Use scaling policies to adjust the capacity of this group

Scale between and instances. These will be the minimum and maximum size of your group.

Increase Group Size

Name:

Execute policy when: [awsec2-Frontend-ASG-CPU-Utilization](#) [Edit](#) [Remove](#)
breaches the alarm threshold: CPUUtilization >= 80 for 900 seconds
for the metric dimensions AutoScalingGroupName = Frontend-ASG

Take the action: instances <= CPUUtilization < +infinity

[Add step](#) ⓘ

Instances need: seconds to warm up after each step

[Create a simple scaling policy](#) ⓘ

Scale-up

Decrease Group Size

Name:

Execute policy when: [awsec2-Frontend-ASG-High-CPU-Utilization](#) [Edit](#) [Remove](#)
breaches the alarm threshold: CPUUtilization <= 50 for 900 seconds
for the metric dimensions AutoScalingGroupName = Frontend-ASG

Take the action: instances >= CPUUtilization > -infinity

[Add step](#) ⓘ

[Create a simple scaling policy](#) ⓘ

[Cancel](#)

[Previous](#)

[Review](#)

[Next: Configure Notifications](#)

Scale Down

h. We can now go straight to Review and then Click on the **Create Auto Scaling group** button. This process is to be done for both the frontend tier and the backend tier but not the data storage tier.

We have almost setup or architecture. However, we cannot SSH into the EC2 instances in the private subnet. This is because have not created our bastion host. So the last part of this article will show how to create the bastion host.

8. Bastion Host: The bastion host is just an EC2 instance that sits in the public subnet. The best practice is to only allow SSH to this instance from your trusted IP. To create a bastion host, navigate to the EC2 instance page and create an EC2 instance in the **demo-public-subnet-1** subnet within our VPC. Also, ensure that it has public IP.

The screenshot shows the 'Configure Instance Details' step in the AWS Management Console. The navigation bar at the top includes the AWS logo, 'Services', 'Resource Groups', and a star icon. On the right, there are links for 'Kene', 'N. Virginia', and 'Support'. Below the navigation bar, a progress bar shows seven steps: 1. Choose AMI, 2. Choose Instance Type, 3. Configure Instance (active), 4. Add Storage, 5. Add Tags, 6. Configure Security Group, and 7. Review.

The main content area is titled 'Step 3: Configure Instance Details'. It contains several configuration options:

- Number of instances:** Set to 1. A link 'Launch into Auto Scaling Group' is available.
- Purchasing option:** A checkbox for 'Request Spot instances' is present.
- Network:** A dropdown menu shows 'vpc-0514207439600c4b5 | Demo-VPC'. A link 'Create new VPC' is available. Below the dropdown, it says 'No default VPC found. Create a new default VPC.'
- Subnet:** A dropdown menu shows 'subnet-0a7c26278766da22f | demo-public-subnet-1 | us-east-1'. A link 'Create new subnet' is available. Below the dropdown, it says '250 IP Addresses available'.
- Auto-assign Public IP:** Set to 'Enable'.
- Placement group:** A checkbox for 'Add instance to placement group' is present.
- Capacity Reservation:** Set to 'Open'. A link 'Create new Capacity Reservation' is available.
- IAM role:** Set to 'None'. A link 'Create new IAM role' is available.
- Shutdown behavior:** Set to 'Stop'.
- Enable termination protection:** A checkbox for 'Protect against accidental termination' is present.
- Monitoring:** A checkbox for 'Enable CloudWatch detailed monitoring' is present. Below it, a link 'Additional charges apply.' is available.
- Tenancy:** Set to 'Shared - Run a shared hardware instance'.

At the bottom right, there are four buttons: 'Cancel', 'Previous', 'Review and Launch' (highlighted in blue), and 'Next: Add Storage'.

Bastion Host EC2 instance in public subnet

The screenshot shows the 'Configure Security Group' step in the AWS Management Console. The navigation bar at the top includes the AWS logo, 'Services', 'Resource Groups', and a star icon. On the right, there are links for 'Kene', 'N. Virginia', and 'Support'. Below the navigation bar, a progress bar shows seven steps: 1. Choose AMI, 2. Choose Instance Type, 3. Configure Instance, 4. Add Storage, 5. Add Tags, 6. Configure Security Group (active), and 7. Review.

The main content area is titled 'Step 6: Configure Security Group'. It contains the following information:

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. [Learn more](#) about Amazon EC2 security groups.

Assign a security group: ☒ Create a new security group ☐ Select an existing security group

Security group name: launch-wizard-1

Description: launch-wizard-1 created 2019-01-30T11:59:55.231+01:00

Type	Protocol	Port Range	Source	Description
SSH	TCP	22	Custom 1.2.3.4/32	My IP

At the bottom left, there is an 'Add Rule' button.

Security Group of the Bastion Host

We also need to allow SSH from our private instances from the Bastion Host.
<https://medium.com/the-andela-way/designing-a-three-tier-architecture-in-aws-e5c24671f124>