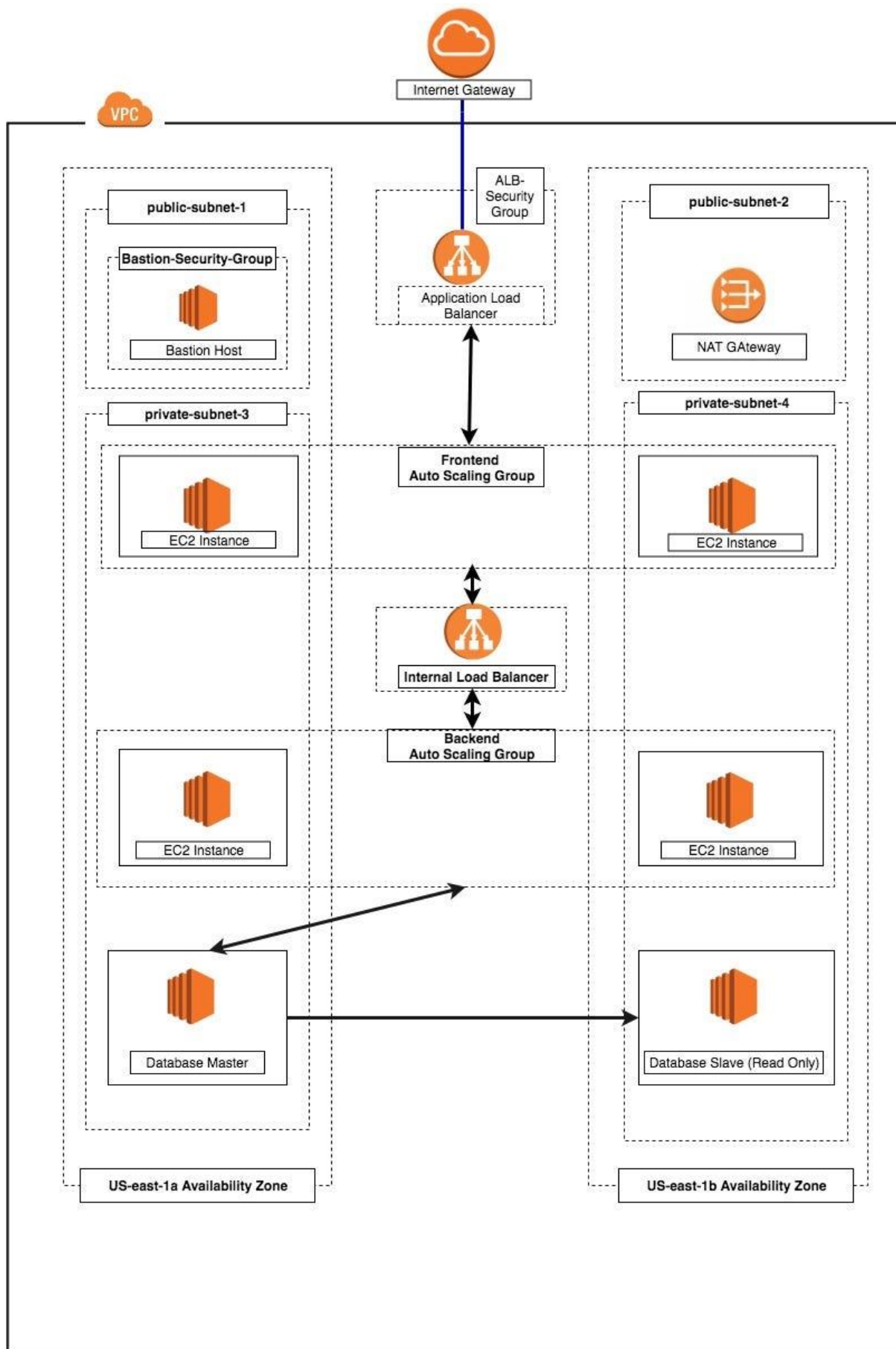


Designing a Three-Tier Architecture in AWS

Introduction

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.



three-tier architecture

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

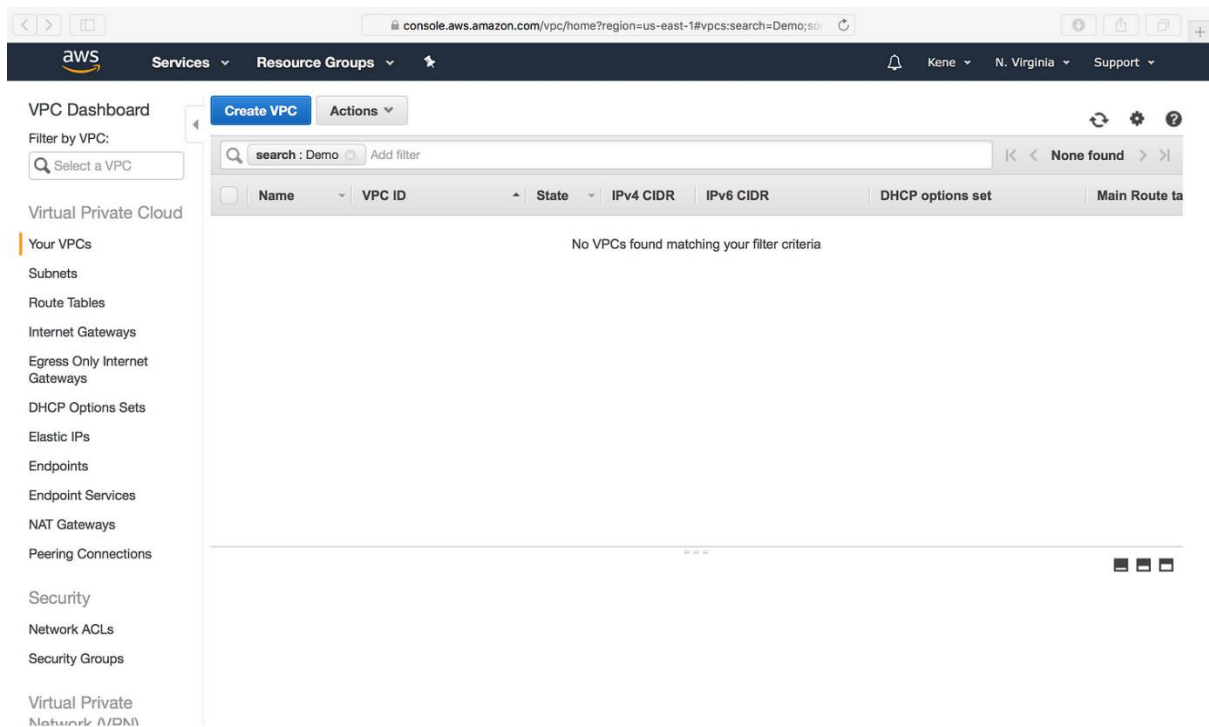
To follow along, you need to have an [AWS](#) account. We shall be making use of the AWS free-tier resources so we do not incur charges while learning.

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



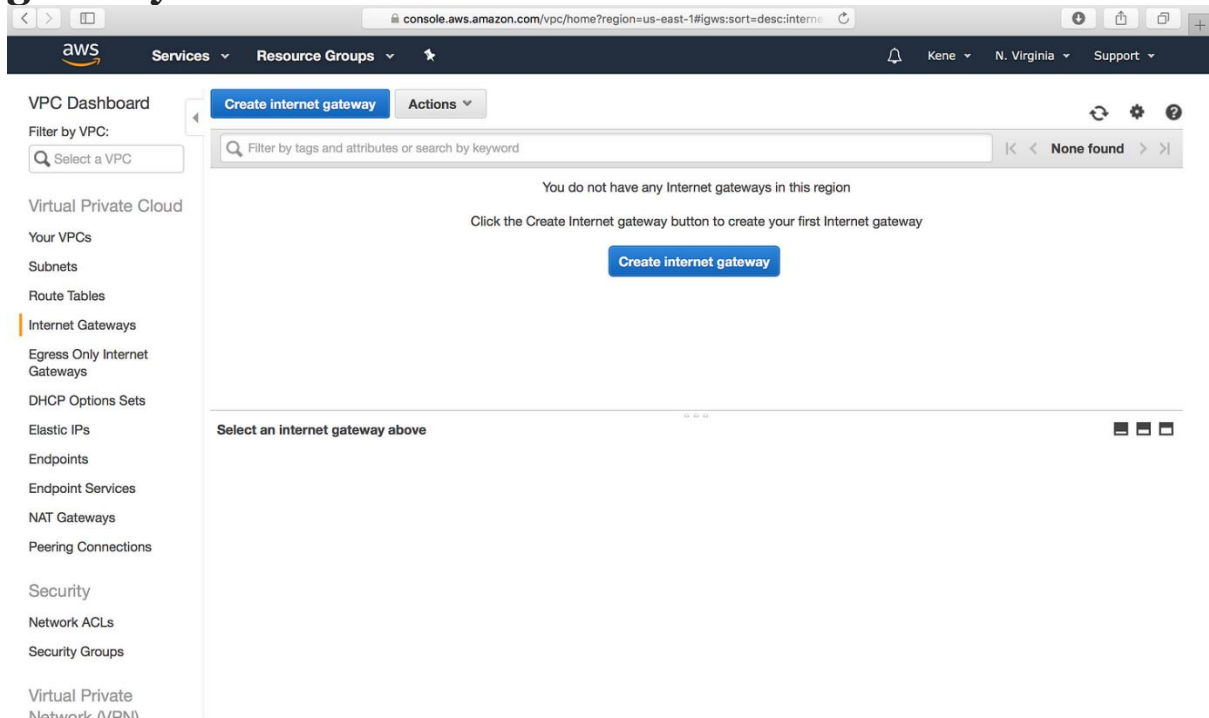
Create VPC

The screenshot shows the 'Create VPC' form in the AWS console. The breadcrumb is 'VPCs > Create VPC'. The title is 'Create VPC'. Below the title is a descriptive paragraph: '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.' The form has four main fields: 'Name tag' with the value 'Demo-VPC', 'IPv4 CIDR block*' with the value '10.0.0.0/16', 'IPv6 CIDR block' with radio buttons for 'No IPv6 CIDR Block' (selected) and 'Amazon provided IPv6 CIDR block', and 'Tenancy' with a dropdown menu set to 'Default'. At the bottom left is a '* Required' label, and at the bottom right are 'Cancel' and 'Create' buttons.

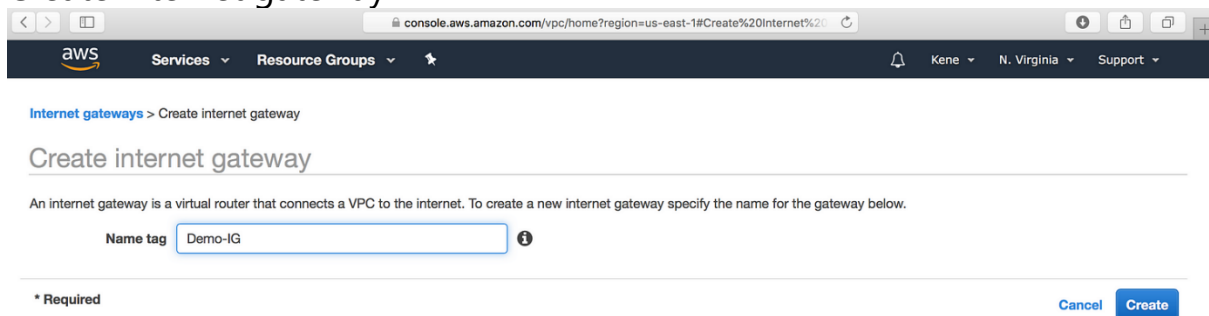
Create VPC

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



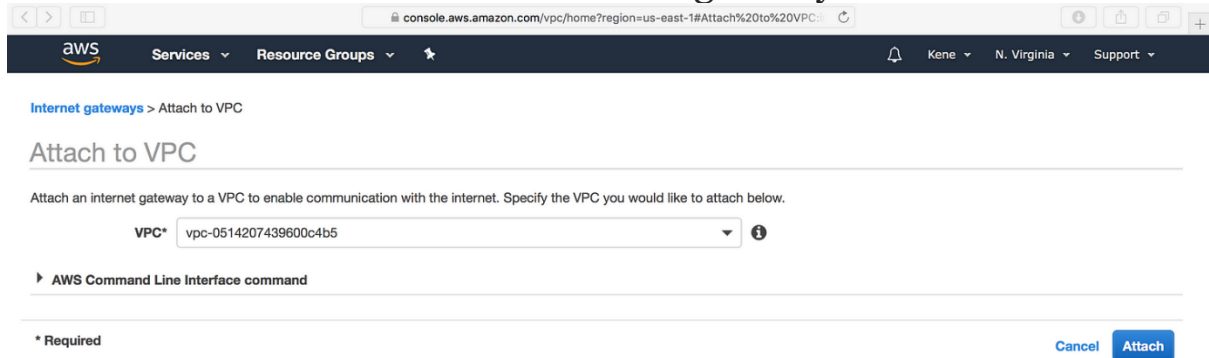
Create Internet Gateway

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. At the top, there's a navigation bar with the AWS logo, 'Services', 'Resource Groups', and user information. Below this, the breadcrumb 'Internet gateways > Attach to VPC' is visible. The main heading is 'Attach to VPC'. A sub-header reads: '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. Below the dropdown is a link for 'AWS Command Line Interface command'. At the bottom right, there are 'Cancel' and 'Attach' buttons. A small asterisk note '* Required' is at the bottom left.

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: demo-public-subnet-1 ⓘ

VPC*: vpc-0514207439600c4b5 ⓘ

VPC CIDRs	CIDR	Status	Status Reason
	10.0.0.0/16	associated	

Availability Zone: us-east-1a ⓘ

IPv4 CIDR block*: 10.0.1.0/24 ⓘ

* 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

elastic IP

Create subnet Actions

search: Demo Add filter

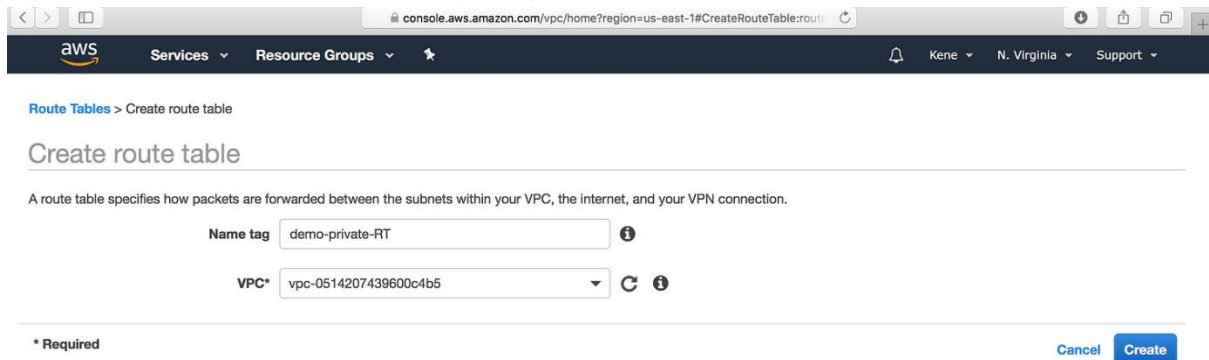
Name	Subnet ID	State	VPC	IPv4 CIDR	Available IPv4
demo-private-subnet-3	subnet-042f12f4806a1fa66	available	vpc-0514207439600c4b5 ...	10.0.3.0/24	251
demo-public-subnet-2	subnet-0955b5c3444d27dd8	available	vpc-0514207439600c4b5 ...	10.0.2.0/24	251
demo-private-subnet-4	subnet-0a74970565af18075	available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

1 to 4 of 4

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.

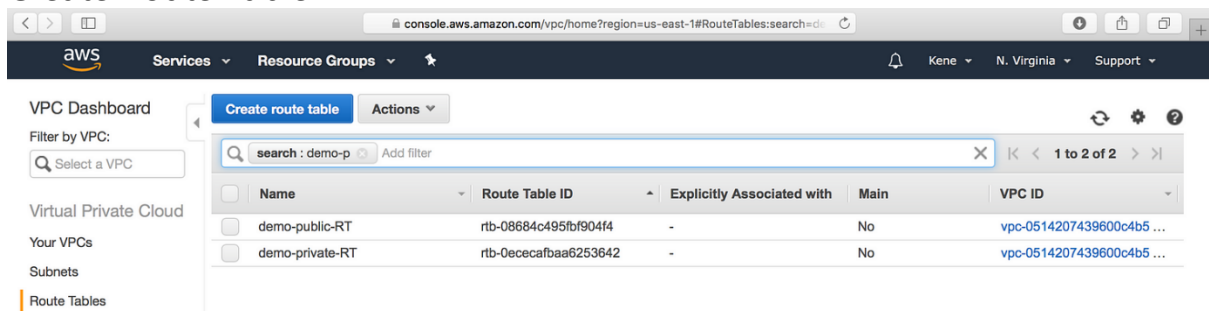
Name tag: demo-private-RT ⓘ

VPC*: vpc-0514207439600c4b5 ⓘ

* Required

Cancel Create

Create Route Table



VPC Dashboard

Filter by VPC: Select a 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-08684c495fb904f4	-	No	vpc-0514207439600c4b5 ...
<input checked="" 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.

The screenshot shows the AWS Management Console interface. On the left is the 'VPC Dashboard' sidebar with various navigation links. The main content area displays a list of route tables. The 'demo-public-RT' route table is selected. Below the list, the 'Subnet Associations' tab is active, showing a table with columns for Subnet ID, IPv4 CIDR, and IPv6 CIDR. The message 'You do not have any subnet associations.' is displayed at the bottom of the tab.

Name	Route Table ID	Explicitly Associated with	Main	VPC ID
demo-public-RT	rtb-08684c495fb904f4	-	No	vpc-0514207439600c4b5...
demo-private-RT	rtb-0ececfaabaa6253642	-	No	vpc-0514207439600c4b5...

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

You do not have any subnet associations.

Subnet Associations

The screenshot shows the 'Edit subnet associations' page in the AWS Management Console. It displays the 'Route table' as 'rtb-08684c495fb904f4 (demo-public-RT)'. Below this, there are two 'Associated subnets' dropdown menus. A table below shows a list of subnets with their IDs, names, IPv4 CIDR, IPv6 CIDR, and 'Current Route'. The 'demo-public-subnet-2' and 'demo-public-subnet-1' are highlighted. At the bottom, there are 'Cancel' and 'Save' buttons.

Route table: rtb-08684c495fb904f4 (demo-public-RT)

Associated subnets: subnet-0955b5c3444d27dd8, subnet-0a7c26278766da22f

Subnet ID	Subnet Name	IPv4 CIDR	IPv6 CIDR	Current Route
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

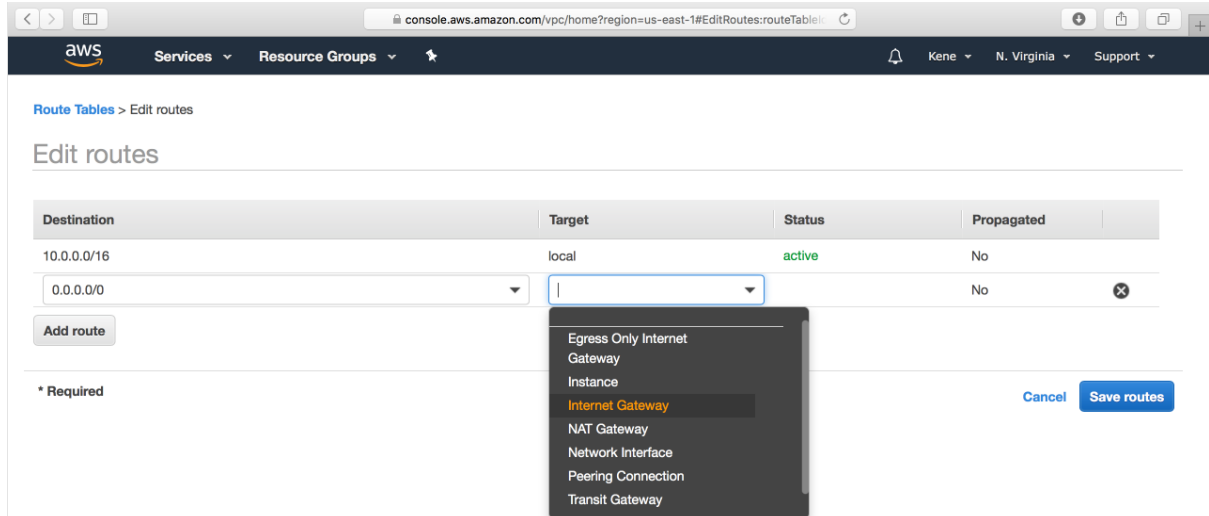
* Required

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.

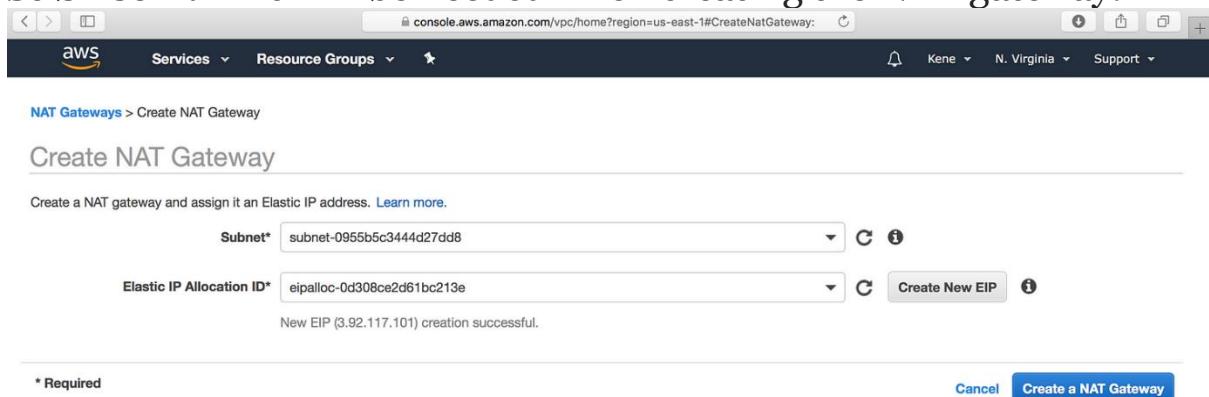
To do that we select the public route table and then choose the **Routes** tab. The rule should be similar to the one shown below:



Edit Route for the public route table

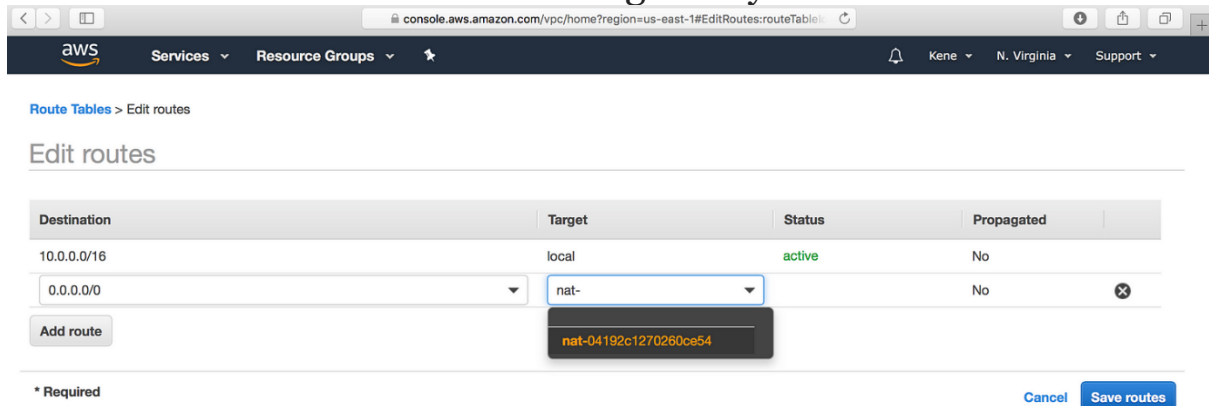
5. Create the NAT Gateway: The NAT gateway enables the EC2 instances in the private subnet to access the internet. The NAT Gateway is an AWS managed service for the NAT instance. To create the NAT gateway, navigate to the NAT Gateways page, and then click on the **Create NAT Gateway**.

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

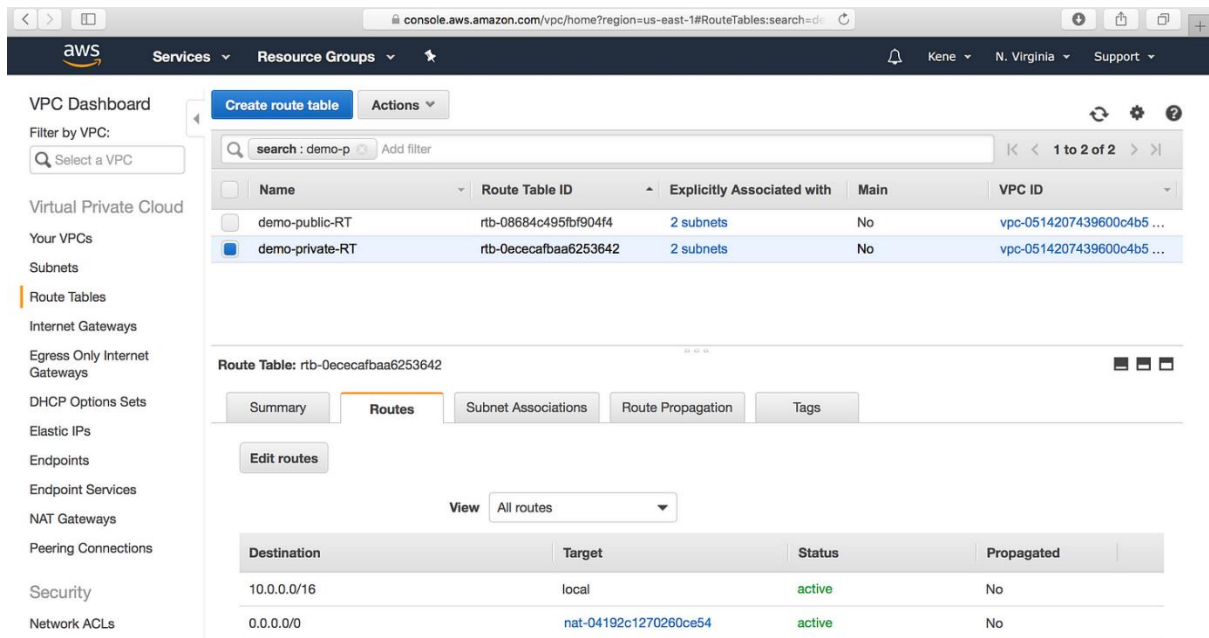


Create NAT Gateway

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.



Edit the Private Route Table

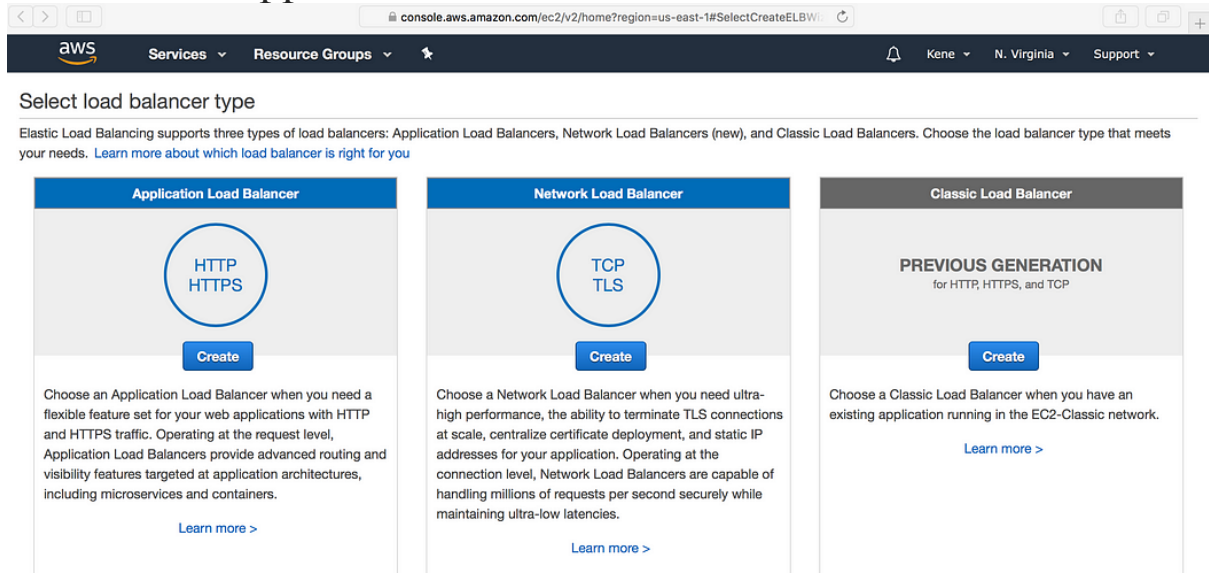


Edit Private Route Table to use NAT Gateway for private EC2 instances

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.**

aws

Services

Resource Groups

Kene

N. Virginia

Support

1. Configure Load Balancer

2. Configure Security Settings

3. Configure Security Groups

4. Configure Routing

5. Register Targets

6. Review

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

Demo-ELB

Scheme

☒ internet-facing
 ☐ internal

IP address type

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

aws

Services

Resource Groups

Kene

N. Virginia

Support

1. Configure Load Balancer

2. Configure Security Settings

3. Configure Security Groups

4. Configure Routing

5. Register Targets

6. Review

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

Demo-internal-ELB

Scheme

☐ internet-facing
 ☒ internal

IP address type

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**

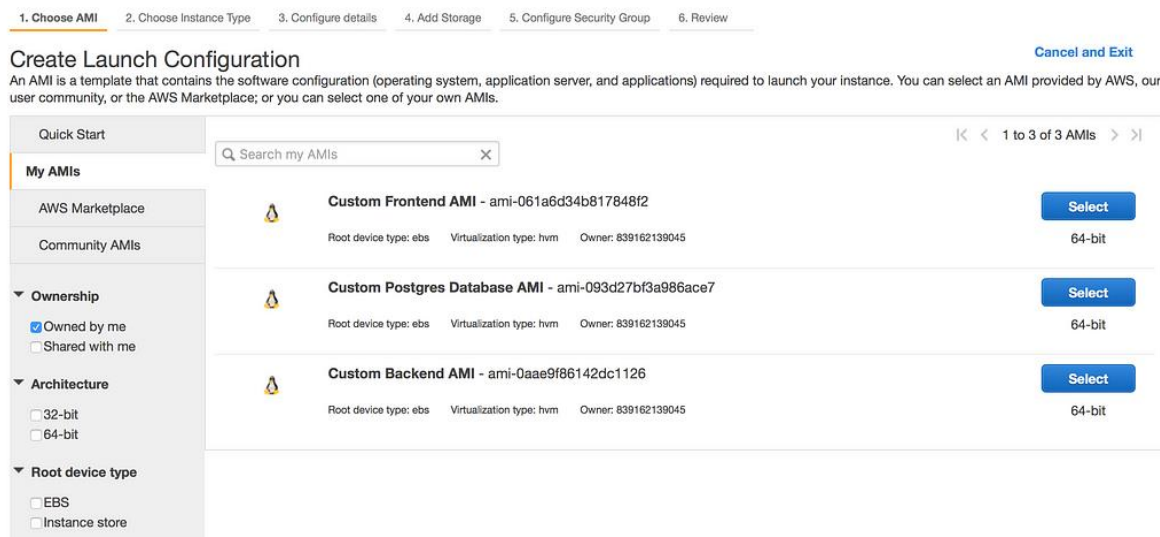
Skip the Register Targets and then go ahead and review the configuration and then click on the **Create** button.

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.



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.

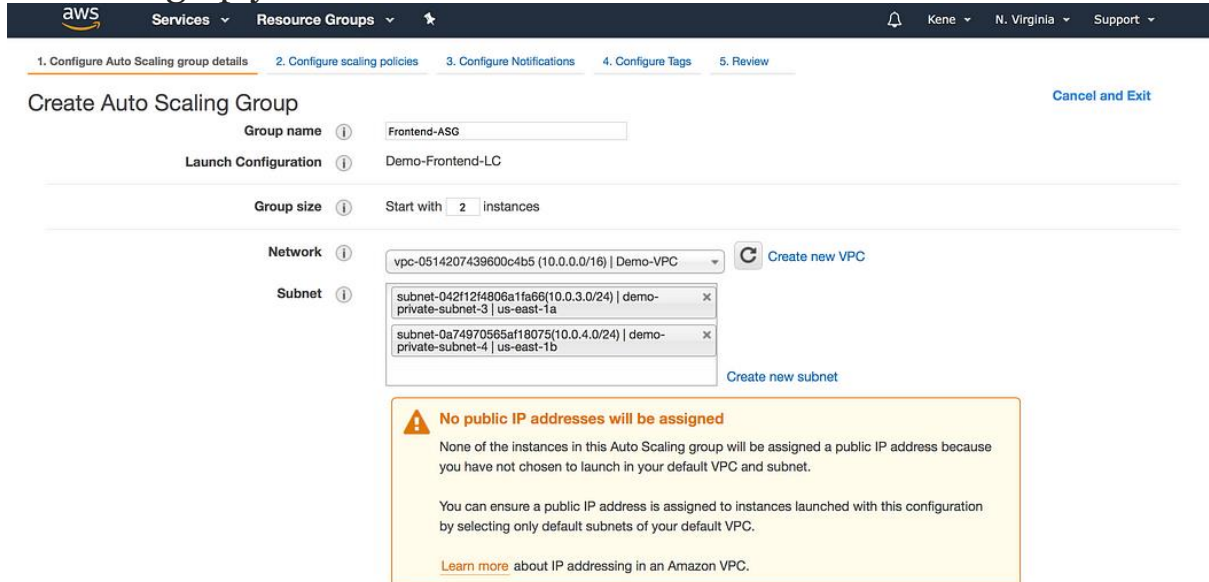
c. Under the Configure details, give the Launch Configuration a name, eg **Demo-Frontend-LC**. Also, under the **Advance Details** dropdown, the **User data** is provided for you to type in a command that is needed to install dependencies and start the application.

The screenshot shows the AWS Management Console interface for creating a Launch Configuration. The breadcrumb trail indicates the path: console.aws.amazon.com/ec2/autoscaling/home?region=us-east-1#CreateLaunchConfiguration. The top navigation bar shows the AWS logo, 'Services', 'Resource Groups', and user information (Kene, N. Virginia, Support). The main content area has a progress bar with six steps: 1. Choose AMI, 2. Choose Instance Type, 3. Configure details (active), 4. Add Storage, 5. Configure Security Group, and 6. Review. The 'Create Launch Configuration' form includes the following fields: 'Name' (Demo-Frontend-LC), 'Purchasing option' (Request Spot Instances), 'IAM role' (None), 'Monitoring' (Enable CloudWatch detailed monitoring), and an expanded 'Advanced Details' section. The 'Advanced Details' section contains 'Kernel ID' (Use default), 'RAM Disk ID' (Use default), 'User data' (As text, with a text area containing 'npm start'), and 'IP Address Type' (Assign a public IP address to instances launched in the default VPC and subnet (default)). At the bottom, there are buttons for 'Cancel', 'Previous', 'Skip Storage', and 'Next: Add Storage'.

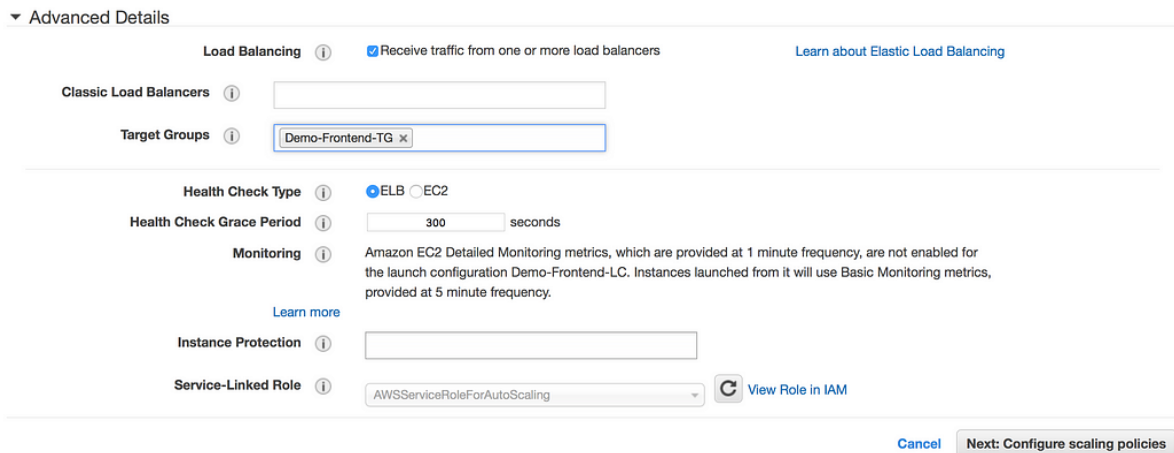
d. Again under the security group, we want to only allow the ports that are necessary for our application.

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.



Auto Scaling Group 1



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:

Increase Group Size

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:

Add instances when 80 <= CPUUtilization < +infinity

Add step

Instances need:

300 seconds to warm up after each step

Create a simple scaling policy

Scale-up

Decrease Group Size

Name:

Decrease Group Size

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:

Remove instances when 50 >= 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.

Step 3: Configure Instance Details

Number of instances: 1 [Launch into Auto Scaling Group](#)

Purchasing option: ☐ Request Spot instances

Network: vpc-0514207439600c4b5 | Demo-VPC [Create new VPC](#)
No default VPC found. [Create a new default VPC.](#)

Subnet: subnet-0a7c26278766da22f | demo-public-subnet-1 | us-east-1 [Create new subnet](#)
250 IP Addresses available

Auto-assign Public IP: Enable

Placement group: ☐ Add instance to placement group

Capacity Reservation: Open [Create new Capacity Reservation](#)

IAM role: None [Create new IAM role](#)

Shutdown behavior: Stop

Enable termination protection: ☐ Protect against accidental termination

Monitoring: ☐ Enable CloudWatch detailed monitoring
[Additional charges apply.](#)

Tenancy: Shared - Run a shared hardware instance

[Cancel](#) [Previous](#) [Review and Launch](#) [Next: Add Storage](#)

Bastion Host EC2 instance in public subnet

Step 6: Configure Security Group

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

[Add Rule](#)

Security Group of the Bastion Host

We also need to allow SSH from our private instances from the Bastion Host.

Conclusion

There were lots of clicking and configurations when using the console to set up a three-tier architecture in AWS. It is, however, necessary that a beginner goes through this procedure before moving towards automation.