

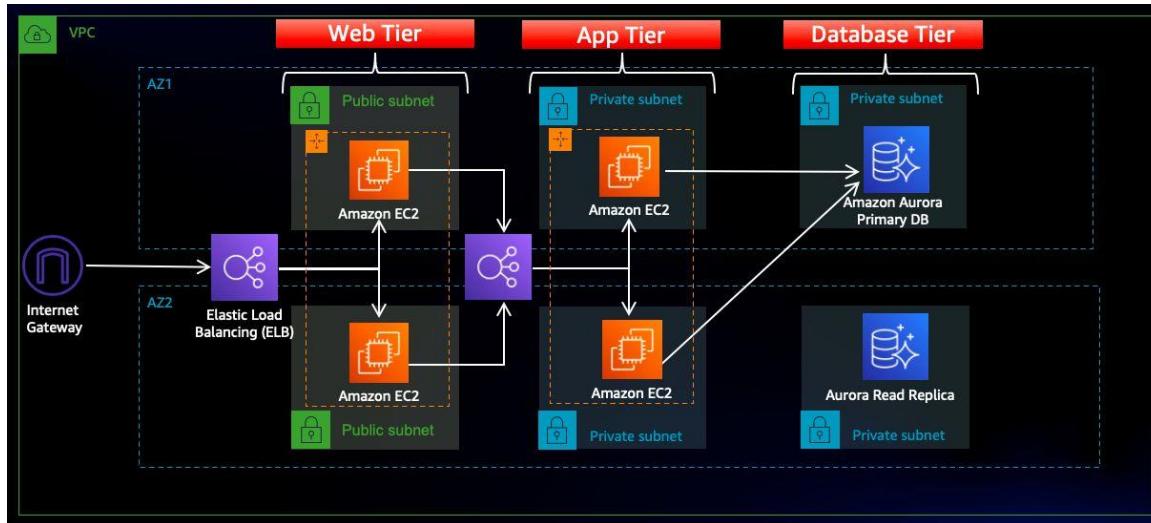
AWS Three Tier Web Architecture

Project hands on

by P.Lokesh(mech)

Introduction :

Architecture Overview :



In this architecture, a public-facing Application Load Balancer forwards client traffic to our web tier EC2 instances. The web tier is running Nginx webservers that are configured to serve a React.js website and redirects our API calls to the application tier's internal facing load balancer. The internal facing load balancer then forwards that traffic to the application tier, which is written in Node.js. The application tier manipulates data in an Aurora MySQL multi-AZ database and returns it to our web tier. Load balancing, health checks and autoscaling groups are created at each layer to maintain the availability of this architecture.

It contains total 6parts as below :

Part 0 : Set up

Part 1 : Networking & security

Part 2 : Database Deployment

Part 3 : [App Tier Instance Deployment](#)

Part 4 : [Internal Load Balancing and Auto Scaling](#)

Part 5 : [Web Tier Instance Deployment](#)

Part 6 : [External Load Balancer and Auto Scaling](#)

Part 0 : Set up

For this workshop, we will be downloading the code from Github and upload it to S3 so our instances can access it. We will also create an AWS Identity and Access Management EC2 role so we can use AWS Systems Manager Session Manager to connect to our instances securely and without needing to create SSH key pairs.

Learning Objectives:

- S3 Bucket Creation.
- IAM EC2 Instance Role Creation.
- Download Code from Github Repository.

Download Code from Github :

Download the code from [this repository](#) into your local environment by running the command below. If you don't have git installed, you can just download the zip. Save it somewhere you can easily access.
git clone <https://github.com/aws-samples/aws-three-tier-web-architecture-workshop.git>

S3 Bucket Creation:

- 1) Navigate to the S3 service in the AWS console and create a new S3 bucket.

The screenshot shows the AWS S3 console with the 'General purpose buckets' tab selected. A table lists one bucket: 'codepipeline-us-east-1' located in 'US East (N. Virginia)' region 'us-east-1'. The bucket was created on 'February 4, 2024, 08:33:34 (UTC+05:30)'. The left sidebar includes links for Buckets, Access Grants, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Block Public Access settings for this account, Storage Lens, Dashboards, and Storage Lens groups.

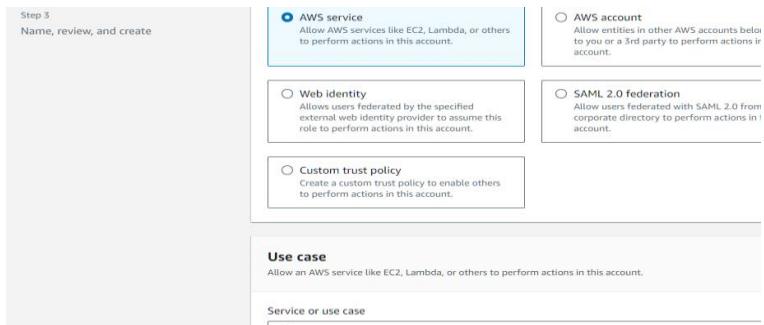
- 2) Give it a unique name, and then leave all the defaults as in. Make sure to select the region that you intend to run this whole lab in. This bucket is where we will upload our code later.

The screenshot shows the 'Create bucket' wizard. In the 'General configuration' step, the 'AWS Region' is set to 'US East (N. Virginia) us-east-1'. The 'Bucket type' section shows 'General purpose' selected (recommended for most use cases), and 'Directory - New' is also listed. The left sidebar has the same navigation as the previous screenshot.

IAM EC2 Instance Role Creation:

- 1) Navigate to the IAM dashboard in the AWS console and create an EC2 role.

The screenshot shows the IAM Roles list. It displays four roles: 'AWSCodePipelineServiceRole-us-east-1-git2app', 'AWSCodePipelineServiceRole-us-east-1-gitapp', and 'AWSServiceRoleForAutoScaling', all created yesterday. The left sidebar includes Identity and Access Management (IAM), Dashboard, and Access management (User groups, Users, Roles).



- 2) Select EC2 as the trusted entity .
- 3) When adding permissions, include the following AWS managed policies. You can search for them and select them. These policies will allow our instances to download our code from S3 and use Systems Manager Session Manager to securely connect to our instances without SSH keys through the AWS console.
 - **AmazonSSMManagedInstanceCore**
 - **AmazonS3ReadOnlyAccess.**

Add permissions

Permissions policies (1/915)

Choose one or more policies to attach to your new role.

Filter by Type: All types | 1 match

Policy name	Type	Description
AmazonSSMManagedInst...	AWS managed	The p...

- 4) Give your role a name, and then click **Create Role**.

Part 1: Networking and Security

In this section we will be building out the VPC networking components as well as security groups that will add a layer of protection around our EC2 instances, Aurora databases, and Elastic Load Balancers

Learning Objectives:

- Create an isolated network with the following components:
 - VPC
 - Subnets
 - Route Tables
 - Internet Gateway
 - NAT gateway
 - Security Groups

VPC and Subnets

1) VPC Creation :

Navigate to the VPC dashboard in the AWS console and navigate to **Your VPCs** on the left hand side

Name	VPC ID	State	IPv4 CIDR	IPv6 CIDR
awsdemoworkshop	vpc-04a1ecf88ab7e6c3b	Available	10.0.0.0/16	-
three-tier-app	vpc-0506ded4ddab6035e	Available	10.0.0.0/16	-
-	vpc-07067d9c6e6de2bb5	Available	172.31.0.0/16	-

2) Make sure VPC only is selected, and fill out the VPC Settings with a Name tag and a CIDR range of your choice.

Create VPC [Info](#)

A VPC is an isolated portion of the AWS Cloud populated by AWS objects, such as Amazon

VPC settings

Resources to create [Info](#)
Create only the VPC resource or the VPC and other networking resources.

VPC only VPC and more

Name tag - optional
Creates a tag with a key of 'Name' and a value that you specify.
my-vpc-01

IPv4 CIDR block [Info](#)
 IPv4 CIDR manual input IPAM-allocated IPv4 CIDR block
IPv4 CIDR
10.0.0.0/24
CIDR block size must be between /16 and /28.

IPv6 CIDR block [Info](#)

NOTE: Make sure you pay attention to the region you're deploying all your resources in. You'll want to stay consistent for this workshop.

NOTE: Choose a CIDR range that will allow you to create at least 6 subnets.

Subnet Creation:

1. Next, create your subnets by navigating to **Subnets** on the left side of the dashboard and clicking **Create subnet**.

Subnets (11) [Info](#)

Name	Subnet ID	State	VPC
private-db-subnet-az2	subnet-0417c59905b7fe8c0	Available	vpc-04a1ecf88ab7e6c3b
app-tier-private-1a	subnet-0e65ab6a42801b273	Available	vpc-0506ded4ddab6035e
private-db-subnet-az1	subnet-0551af335066ae8ce	Available	vpc-04a1ecf88ab7e6c3b
web-tier-public-1b	subnet-00b95fa63611f8c0d	Available	vpc-0506ded4ddab6035e

Create subnet [Info](#)

VPC

VPC ID
Create subnets in this VPC.
vpc-04a1ecf88ab7e6c3b (awsdemoworkshop)

Associated VPC CIDRs

IPv4 CIDRs
10.0.0.0/16

Subnet settings
Specify the CIDR blocks and Availability Zone for the subnet.

Subnet 1 of 1

Your final subnet setup should be similar to this. Verify that you have 3 subnets across 2 different availability zones.

The screenshot shows the AWS VPC Subnets page. At the top, there's a search bar and a filter section. Below that, a table lists six subnets:

Name	Subnet ID	State	VPC	IPv4 CIDR	Availability Zone
Public-Subnet-AZ-1	subnet-042290143e243f9a7	Available	vpc-0d2a254...	10.0.0.0/24	us-west-1b
Private-Subnet-AZ1	subnet-0f2b746796c554aaa	Available	vpc-0d2a254...	10.0.1.0/24	us-west-1b
Private-DB-Subnet-AZ1	subnet-04dbbe1cf38fb22	Available	vpc-0d2a254...	10.0.2.0/24	us-west-1b
Public-Subnet-AZ2	subnet-00511a4ff95e0335d	Available	vpc-0d2a254...	10.0.3.0/24	us-west-1c

Internet Connectivity:

Internet Gateway

- In order to give the public subnets in our VPC internet access we will have to create and attach an Internet Gateway. On the left hand side of the VPC dashboard, select **Internet Gateway**.

The screenshot shows the AWS VPC Internet Gateways page. It displays three existing internet gateways:

Name	Internet gateway ID	State	VPC ID
three-tier-igw	igw-0159be8ccc2637403	Attached	vpc-0506ded4ddab6035e three-tier
three-tier-internetgateway	igw-02657a8fd6a56c14d	Attached	vpc-04a1ecf8ab7e6c3b awsdem
-	igw-05718a8d8508ca7d4	Attached	vpc-07067d9c6e6de2bb5

- Create your internet gateway by simply giving it a name and clicking **Create internet gateway**.

The screenshot shows the "Create internet gateway" wizard. Step 1: Internet gateway settings. It asks for a name tag:

Name tag
Creates a tag with a key of 'Name' and a value that you specify.

Tags - optional
A tag is a label that you assign to an AWS resource. Each tag consists of a key and an optional value. You can use tags to search and filter your resources or track your AWS costs.

No tags associated with the resource.
[Add new tag](#)
You can add 50 more tags.

[Cancel](#) [Create internet gateway](#)

- After creating the internet gateway, attach it to your VPC that you create in the **VPC and Subnet Creation** step of the workshop. You have a couple options on how to do this, either with the creation success message or the **Actions** drop down.

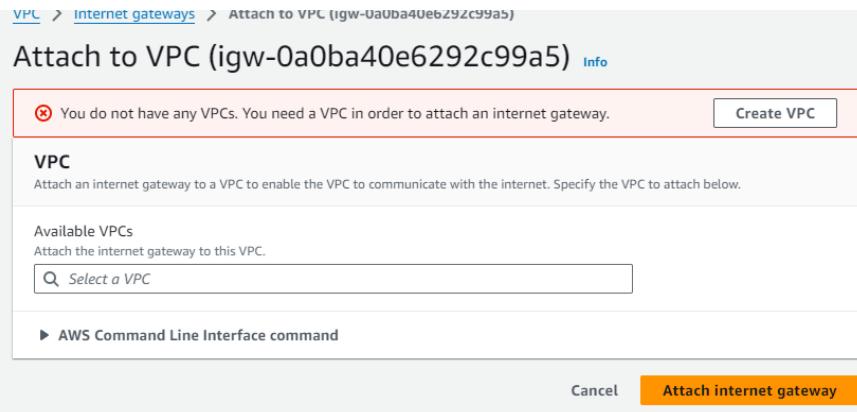
The screenshot shows the AWS VPC Internet Gateways page. It displays a single internet gateway:

Internet gateway ID	State	VPC ID	Owner
igw-0a0ba40e6292c99a5	Detached	-	97504

The Actions menu for this gateway includes:

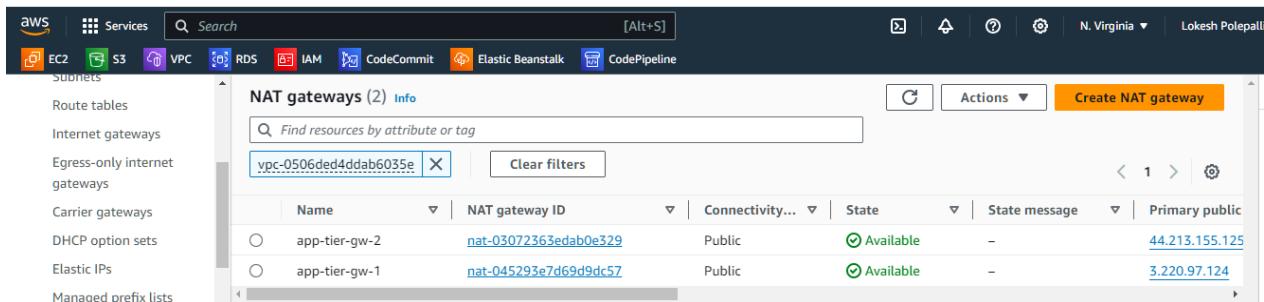
- Attach to VPC
- Detach from VPC
- Manage tags
- Delete

Then, select the correct VPC and click **Attach internet gateway**.

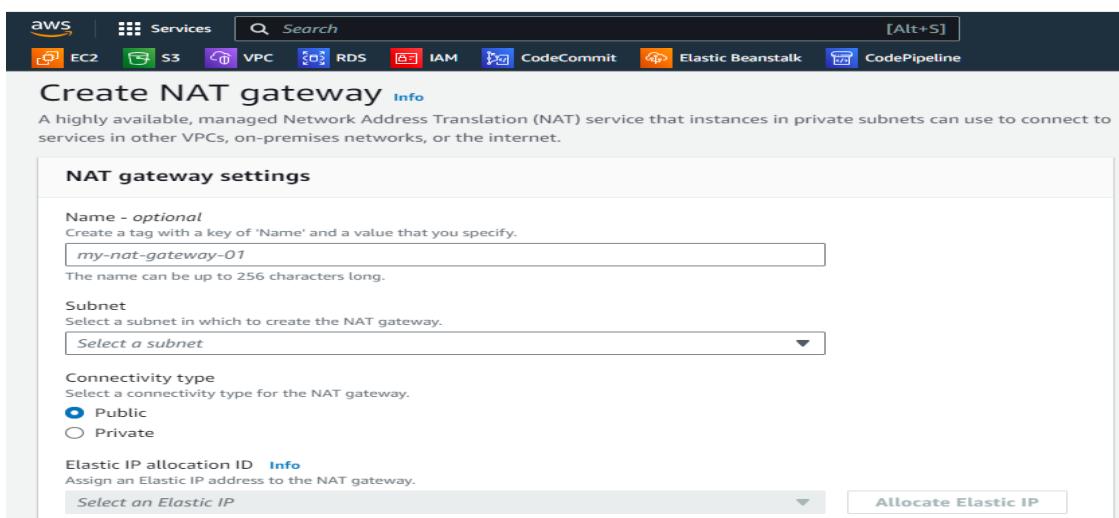


NAT Gateway

1. In order for our instances in the app layer private subnet to be able to access the internet they will need to go through a NAT Gateway. For high availability, you'll deploy one NAT gateway in each of your **public subnets**. Navigate to **NAT Gateways** on the left side of the current dashboard and click **Create NAT Gateway**.



2. Fill in the **Name**, choose one of the **public subnets** you created in part 2, and then allocate an Elastic IP. Click **Create NAT gateway**.



3. Repeat step 1 and 2 for the other subnet.

Routing Configuration:

1. Navigate to Route Tables on the left side of the VPC dashboard and click Create route table First, let's create one route table for the web layer **public subnets** and name it accordingly.

The screenshot shows the AWS VPC Route Tables page. At the top, there is a search bar and a 'Create route table' button. Below the search bar, there is a filter section for 'Route tables' and a table listing existing route tables. The table columns include Name, Route table ID, Explicit subnet associations, Edge associations, and Main. Two route tables are listed: 'db-tier-1' and 'rt-public2'. The 'db-tier-1' table has two explicit subnet associations: 'subnet-00b95fa63611f8c0d' and 'subnet-0af6d9be932204cad'. The 'rt-public2' table has one explicit subnet association: 'subnet-00b95fa63611f8c0d'. Below the table, there is a 'Route table settings' modal. It contains fields for 'Name - optional' (set to 'my-route-table-01'), 'VPC' (set to 'Select a VPC'), and 'Tags' (empty). At the bottom of the modal is a 'Create route table' button.

2. After creating the route table, you'll automatically be taken to the details page. Scroll down and click on the **Routes tab** and **Edit routes**.

The screenshot shows the 'rtb-0af6d9be932204cad / rt-public2' route table details page. The 'Routes' tab is selected, showing two routes. The first route has a destination of '0.0.0.0/0' and a target of 'igw-0159be8ccc2637403'. The second route has a destination of '10.0.0.0/16' and a target of 'local'. Both routes are marked as 'Active' and 'Propagated'.

3. Add a route that directs traffic from the VPC to the internet gateway. In other words, for all traffic **destined** for IPs outside the VPC CIDR range, add an entry that directs it to the internet gateway as a **target**. Save the changes.

The screenshot shows the 'Edit routes' page for the 'rtb-0af6d9be932204cad' route table. A new route is being added with a destination of '10.0.0.0/16' and a target of 'Internet Gateway'. The target dropdown also lists 'igw-0159be8ccc2637403'. The status is set to 'Active' and 'Propagated' is set to 'No'. At the bottom right are 'Cancel', 'Preview', and 'Save changes' buttons.

4. Edit the *Explicit Subnet Associations* of the route table by navigating to the route table details again. Select **Subnet Associations** and click **Edit subnet associations**.

The screenshot shows the 'Subnet associations' tab selected in the top navigation bar. Below it is a table titled 'Explicit subnet associations (1)'. The table has columns for Name, Subnet ID, IPv4 CIDR, and IPv6 CIDR. One row is listed: 'Name' is 'web-tier-public-1b', 'Subnet ID' is 'subnet-00b95fa63611f8c0d', 'IPv4 CIDR' is '10.0.1.0/24', and 'IPv6 CIDR' is '-'. A 'Find subnet association' search bar is at the top of the table.

Select the two web layer public subnets you created earlier and click **Save associations**.

This screenshot shows the 'Edit subnet associations' dialog. At the top left, it says 'Available subnets (1/6)'. Below is a table with columns: Name, Subnet ID, IPv4 CIDR, IPv6 CIDR, and Route table ID. Several subnets are listed, including 'app-tier-private-1a', 'web-tier-public-1b' (which is checked), 'app-tier-private-1b', 'web-tier-public-1a', 'db-private-1a', and 'db-private-1b'. In the 'Selected subnets' section below, 'subnet-00b95fa63611f8c0d / web-tier-public-1b' is selected. At the bottom right are 'Cancel' and 'Save associations' buttons.

4. Now create 2 more route tables, one for each app layer private subnet in each availability zone. These route tables will route app layer traffic destined for outside the VPC to the NAT gateway in the respective availability zone, so add the appropriate routes for that.

The top part of the screenshot shows the 'Route table settings' dialog. It includes fields for 'Name - optional' (containing 'my-route-table-01'), 'VPC' (set to 'Select a VPC'), and 'Tags' (with a note about adding new tags). At the bottom are 'Cancel' and 'Create route table' buttons.

The bottom part shows the 'Edit routes' interface. The URL is 'VPC > Route tables > rtb-0af6d9be932204cad > Edit routes'. It lists two routes:

Destination	Target	Status	Propagated
10.0.0.0/16	local	Active	No
0.0.0.0/0	Internet Gateway	Active	No

Buttons at the bottom include 'Add route', 'Remove', 'Cancel', 'Preview', and 'Save changes'.

Once the route tables are created and routes added, add the appropriate subnet associations for each of the app layer private subnets.

Available subnets (1/6)

Name	Subnet ID	IPv4 CIDR	IPv6 CIDR	Route table ID
app-tier-private-1a	subnet-0e63ab6a42801b273	10.0.16.0/20	-	rtb-0567a66f04f85e600 / app-tier-
web-tier-public-1b	subnet-00b95fa63611f8c0d	10.0.1.0/24	-	rtb-0af6d9be932204cad / rt-public
app-tier-private-1b	subnet-031cedb6b48869e3	10.0.32.0/20	-	rtb-0567a66f04f85e600 / app-tier-
web-tier-public-1a	subnet-010192cf02bb90b8c	10.0.0.0/24	-	rtb-09a9d552f1743e689 / rt-public
<input checked="" type="checkbox"/> db-private-1a	subnet-0ac4928c5f947564c	10.0.48.0/24	-	rtb-0822a0b3c4dfc2e88 / db-tier-1
db-private-1b	subnet-0e46c1debe8d7694d	10.0.49.0/24	-	rtb-0822a0b3c4dfc2e88 / db-tier-1

Selected subnets

subnet-0ac4928c5f947564c / db-private-1a X

[Cancel](#) [Save associations](#)

Security Groups:

1. Security groups will tighten the rules around which traffic will be allowed to our Elastic Load Balancers and EC2 instances. Navigate to **Security Groups** on the left side of the VPC dashboard, under **Security**.

Endpoint services
NAT gateways
Peering connections

Security

- Network ACLs
- Security groups**
- DNS firewall
- Rule groups
- Domain lists

Network Firewall

Name	Security group ID	Security group name	VPC ID
-	sg-03bd3421cdf1a362a	launch-wizard-4	vpc-07067d9c6e6de2bb5
-	sg-00989e029acf35f12	default	vpc-04a1ecf88ab7e6c3b
-	sg-0063bfc09b414fb9e	launch-wizard-3	vpc-07067d9c6e6de2bb5

2. The first security group you'll create is for the public, **internet facing** load balancer. After typing a name and description, add an inbound rule to allow **HTTP** type traffic for your **IP**.

Create security group [Info](#)

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To

Basic details

Security group name [Info](#)

Name cannot be edited after creation.

Description [Info](#)

VPC Info

Inbound rules [Info](#)

Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	My IP	110.235.236.164/32 X

[Add rule](#)

3. The second security group you'll create is for the public instances in the web tier. After typing a name and description, add an inbound rule that allows **HTTP** type traffic from your internet facing load balancer security group you created in the previous step. This will allow traffic from your public facing load balancer to hit your instances. Then, add an additional rule that will allow HTTP type traffic for your IP. This will allow you to access your instance when we test.

Basic details

Security group name [Info](#)
webtiersg
Name cannot be edited after creation.

Description [Info](#)
sg for webtier

VPC [Info](#)
vpc-07067d9c6e6de2bb5

Inbound rules [Info](#)

Security group rule ID	Type Info	Protocol Info	Port range Info	Source Info	Description - optional Info
sgr-057447dc0dbd7d80f	HTTP	TCP	80	Cus... ▾	Q_ 110.235.236.205/32 X
sgr-016744770043fb4e	Custom TCP	TCP	0	Cus... ▾	Q_ sg-019a484f8011a8de9 X

- The third security group will be for our internal load balancer. Create this new security group and add an inbound rule that allows **HTTP** type traffic from your public instance security group. This will allow traffic from your web tier instances to hit your internal load balancer.

Create security group [Info](#)

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. To...

Basic details

Security group name [Info](#)
internal-lb
Name cannot be edited after creation.

Description [Info](#)
sg for internal lb

VPC [Info](#)
vpc-0506ded4ddab6035e (three-tier-app)

Inbound rules [Info](#)

Type Info	Protocol Info	Port range Info	Source Info
HTTP	TCP	80	Cus... ▾

CIDR blocks

Security Groups
web-tier-sg | sg-0cdd2adfff74d9219 ! - optional [Info](#)

Prefix lists
Q web- X
web- X

- The fourth security group we'll configure is for our private instances. After typing a name and description, add an inbound rule that will allow **TCP** type traffic on port **4000** from the **internal load balancer security group** you created in the previous step. This is the port our app tier application is running on and allows our internal load balancer to forward traffic on this port to our private instances. You should also add another route for port **4000** that allows **your IP** for testing.

Basic details

Security group name [Info](#)
privateinstances-sg
Name cannot be edited after creation.

Description [Info](#)
sg for privateinstances

VPC [Info](#)
vpc-0506ded4ddab6035e (three-tier-app)

Inbound rules [Info](#)

Type Info	Protocol Info	Port range Info	Source Info	Description - optional Info
Custom TCP	TCP	4000	Cus... ▾	Q_intel X

CIDR blocks

Security Groups
internal-lb-sg | sg-02d6c1ac0c3c36ae3

Prefix lists

- The fifth security group we'll configure protects our private database instances. For this security group, add an inbound rule that will allow traffic from the private instance security group to the MYSQL/Aurora port (3306).

Basic details

Security group name [Info](#)
dbsg
Name cannot be edited after creation.

Description [Info](#)
sg for db

VPC [Info](#)
vpc-0506ded4ddab6035e (three-tier-app)

Inbound rules [Info](#)

Type Info	Protocol Info	Port range Info	Source Info	Description Info
MYSQL/Aurora	TCP	3306	Cus... ▾	Q_db-s X

CIDR blocks

Security Groups
db-sg | sg-0dc62dcedbdff887c0

Prefix lists

Part 2: Database Deployment

This section of the workshop will walk you through deploying the database layer of the three tier architecture.

Learning Objectives:

- Deploy Database Layer
 - Subnet Groups
 - Multi-AZ Database

Subnet Groups:

1. Navigate to the RDS dashboard in the AWS console and click on **Subnet groups** on the left hand side. Click **Create DB subnet group**.

The screenshot shows the 'Subnet groups' page in the Amazon RDS console. On the left, there's a sidebar with links like Dashboard, Databases, Query Editor, etc. The main area shows a table titled 'Subnet groups (2)'. The table has columns for Name, Description, Status, and VPC. There are two entries:

- db-3tier-app-sng: subnet group for db 3tier, Status Complete, VPC vpc-0506ded4ddab6035e
- three-tier-db-subnet-group: three-tier-db-subnet-group, Status Complete, VPC vpc-04a1ecf88ab7e6c3b

A prominent orange button at the top right says 'Create DB subnet group'.

2. Give your subnet group a name, description, and choose the VPC we created.

The screenshot shows the 'Create DB subnet group' dialog. It has a title 'Create DB subnet group' and a sub-section 'Subnet group details'.

- Name:** three-tier-db (disabled)
- Description:** threetier
- VPC:** three-tier-app (vpc-0506ded4ddab6035e)

3. When adding subnets, make sure to add the subnets we created in each availability zone specifically for our database layer. You may have to navigate back to the VPC dashboard and check to make sure you're selecting the correct subnet IDs.

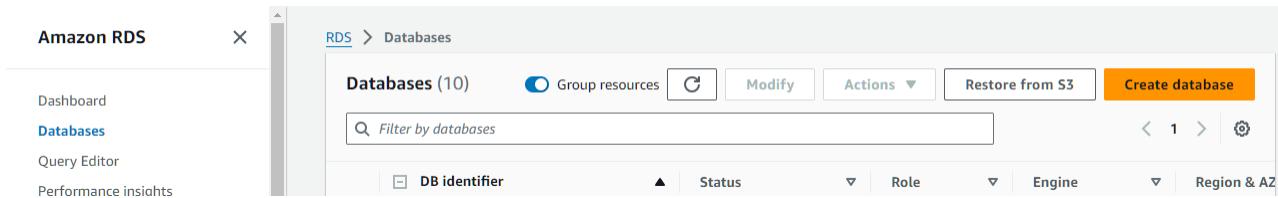
The screenshot shows the 'Add subnets' dialog. It has sections for 'Availability Zones' and 'Subnets'.

- Availability Zones:** A dropdown menu labeled 'Choose an availability zone' with options 'us-east-1a' and 'us-east-1b' selected.
- Subnets:** A dropdown menu labeled 'Select subnets' with two subnets selected: 'subnet-0e46c1debe8d7694d (10.0.49.0/24)' and 'subnet-0ac4928c5f947564c (10.0.48.0/24)'.

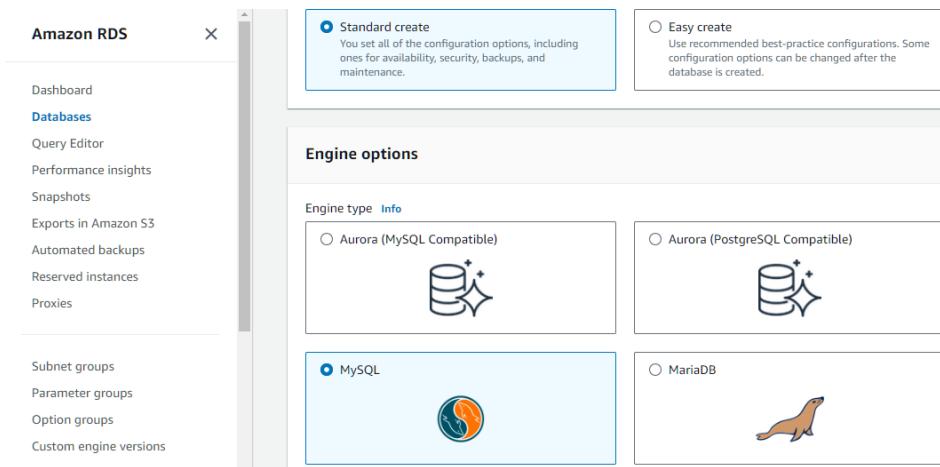
A note at the bottom says: 'For Multi-AZ DB clusters, you must select 3 subnets in 3 different Availability Zones.'

Database Deployment:

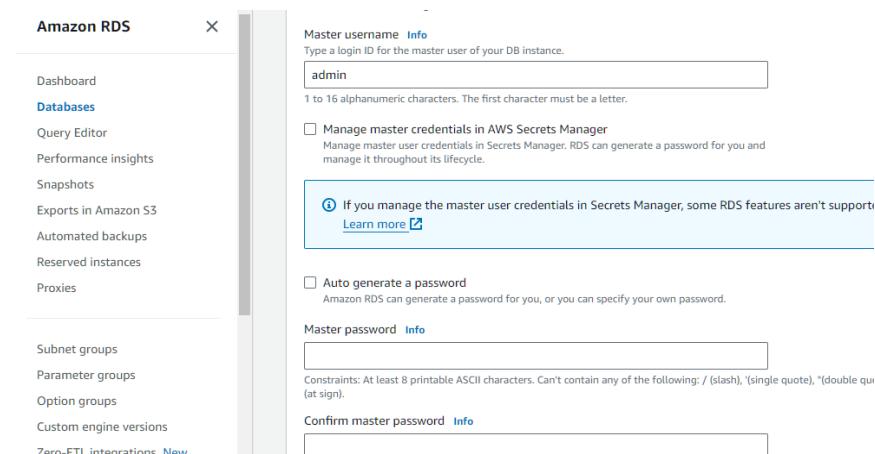
1. Navigate to **Databases** on the left hand side of the RDS dashboard and click **Create database**.



2. We'll now go through several configuration steps. Start with a **Standard create** for this **MySQL (FREE TIER)** database. Leave the rest of the defaults in the **Engine options** as default.



3. Under the **Templates** section choose **FREE TIER** since this isn't being used for production at the moment. Under **Settings** set a username and password of your choice and note them down since we'll be using password authentication to access our database.



4. Under **Connectivity**, set the VPC, choose the subnet group we created earlier, and select no for public access

Compute resource
Choose whether to set up a connection to a compute resource for this database. Setting up a connection will automatically change connectivity settings so that the compute resource can connect to this database.
 Don't connect to an EC2 compute resource
Don't set up a connection to a compute resource for this database. You can manually set up a connection to a compute resource later.

Virtual private cloud (VPC) [Info](#)
Choose the VPC. The VPC defines the virtual networking environment for this DB instance.
three-tier-app (vpc-0506ded4ddab6035e)
6 Subnets, 2 Availability Zones

DB subnet group [Info](#)
Choose the DB subnet group. The DB subnet group defines which subnets and IP ranges the DB instance can use in the VPC that you selected.
db-3tier-app-sng
2 Subnets, 2 Availability Zones

Only VPCs with a corresponding DB subnet group are listed.

After a database is created, you can't change its VPC.

Set the security group we created for the database layer, make sure **password authentication** is selected as our authentication choice, and create the database.

Database authentication

Database authentication options [Info](#)

- Password authentication**
Authenticates using database passwords.
- Password and IAM database authentication**
Authenticates using the database password and user credentials through AWS IAM users and roles.
- Password and Kerberos authentication**
Choose a directory in which you want to allow authorized users to authenticate with this DB instance using Kerberos Authentication.

3. When your database is provisioned, you should see your database available. Note down the writer endpoint for your database for later use.

threetier-webapp Available Instance MySQL Community us-east-1a

Connectivity & security		
Endpoint threetier-webapp.c7w8g2e2o4j7.us-east-1.rds.amazonaws.com	Networking Availability Zone us-east-1a VPC three-tier-app (vpc-0506ded4ddab6035e)	Security VPC security groups db-sg (sg-0dc62dcedbdf887c0) <input checked="" type="radio"/> Active
Port 3306		Publicly accessible No

Part 3: App Tier Instance Deployment

In this section of our workshop we will create an EC2 instance for our app layer and make all necessary software configurations so that the app can run. The app layer consists of a Node.js application that will run on port 4000. We will also configure our database with some data and tables.

Learning Objectives:

- Create App Tier Instance
- Configure Software Stack
- Configure Database Schema
- Test DB connectivity

App Instance Deployment:

1. Navigate to the EC2 service dashboard and click on **Instances** on the left hand side. Then, click **Launch Instances**.

The screenshot shows the AWS EC2 Instances page. On the left sidebar, under the 'Instances' section, 'Instances' is selected. At the top right, there is a 'Launch instances' button. Below it, a table lists three instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
repository files	i-07e4ea165e938ed73	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1c
	i-0db0b17fe2630e7bb	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b
	i-0eabd8943e3247f62	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b

2. Select the first **Amazon Linux 2 AMI**

The screenshot shows the 'Amazon Machine Image (AMI)' selection screen. It displays the 'Amazon Linux 2 AMI (HVM) - Kernel 5.10, SSD Volume Type' as the selected option. To the right, it indicates 'Free tier eligible'. Below the main title, it provides details: 'ami-0cf10cdf9fcfd62d37 (64-bit (x86)) / ami-023c9d904d7c3bf72 (64-bit (Arm))', 'Virtualization: hvm', 'ENA enabled: true', and 'Root device type: ebs'.

Description

3. We'll be using the free tier eligible **T.2 micro** instance type. Select that and click **Next: Configure Instance Details**.

The screenshot shows the 'Configure Instance Details' step. Under 'Instance type', 't2.micro' is selected. A tooltip provides detailed information: 'Family: t2 1 vCPU 1 GiB Memory Current generation: true', 'On-Demand Windows base pricing: 0.0162 USD per Hour', 'On-Demand SUSE base pricing: 0.0116 USD per Hour', 'On-Demand RHEL base pricing: 0.0716 USD per Hour', and 'On-Demand Linux base pricing: 0.0116 USD per Hour'. To the right, there is a 'Free tier eligible' badge, a 'All generations' toggle switch (which is turned on), and a 'Compare instance types' link. A note at the bottom states 'Additional costs apply for AMIs with pre-installed software'.

4. When configuring the instance details, make sure to select to correct **Network, subnet, and select existing group** we created. Note that this is the app layer, so use one of the private subnets we created for this layer.

VPC - required [Info](#)
 vpc-0506ded4ddab6035e (three-tier-app)
 10.0.0.0/16

Subnet [Info](#)
 subnet-0e63ab6a42801b273 app-tier-private-1a
 VPC: vpc-0506ded4ddab6035e Owner: 975049895916 Availability Zone: us-east-1a IP addresses available: 4088 CIDR: 10.0.16.0/20

Create new subnet [\[x\]](#)

Auto-assign public IP [Info](#)
 Disable

Firewall (security groups) [Info](#)
 A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

Create security group Select existing security group

Common security groups [Info](#)
 Select security groups

Compare security group rules [\[x\]](#)

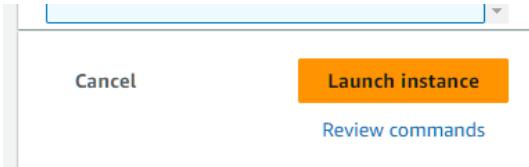
5. We have to configure **IAM INSTANCE PROFILEROLE** in advanced settings & then hit create Instance.

Advanced details [Info](#)

Domain join directory [Info](#)
 Select [\[x\]](#) Create new directory [\[x\]](#)

IAM instance profile [Info](#)
 demoEC2-role [arn:aws:iam::975049895916:instance-profile/demoEC2-role](#) [\[x\]](#) Create new IAM profile [\[x\]](#)

You'll be taken to a page where you can click launch instance, and you'll see the instance you just launched.



Connect to Instance:

1. Navigate to your list of running EC2 Instances by clicking on **Instances** on the left hand side of the EC2 dashboard. When the instance state is running, connect to your instance by clicking the checkmark box to the left of the instance, and click the connect button on the top right corner of the dashboard. Select the Session Manager tab, and click connect. This will open a new browser tab for you.

NOTE: If you get a message saying that you cannot connect via session manager, then check that your instances can route to your NAT gateways and verify that you gave the necessary permissions on the IAM role for the Ec2 instance.

- When you first connect to your instance like this, you will be logged in as ssm-user which is the default user. Switch to ec2-user by executing the following command in the browser terminal:

sudo -su ec2-user

- Let's take this moment to make sure that we are able to reach the internet via our NAT gateways. If your network is configured correctly up till this point, you should be able to ping the google DNS servers:

ping 8.8.8.8

You should see a transmission of packets. Stop it by pressing cntrl c.

NOTE: If you can't reach the internet then you need to double check your route tables and subnet associations to verify if traffic is being routed to your NAT gateway!

Session ID: root-087f6065d30da15d9 Instance ID: i-00e85beb0c592beec

```
sh-5.2$ ping 8.8.8.8
PING 8.8.8.8 (8.8.8.8) 56(84) bytes of data.
64 bytes from 8.8.8.8: icmp_seq=1 ttl=113 time=0.924 ms
64 bytes from 8.8.8.8: icmp_seq=2 ttl=113 time=0.916 ms
64 bytes from 8.8.8.8: icmp_seq=3 ttl=113 time=0.911 ms
64 bytes from 8.8.8.8: icmp_seq=4 ttl=113 time=0.957 ms
^C
--- 8.8.8.8 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 3138ms
rtt min/avg/max/mdev = 0.911/0.927/0.957/0.017 ms
sh-5.2$
```

Configure Database:

- To configure the database of **Mysql**, firstly have to download package by using following **wget & rpm** commands :

steps for installing mysql :

sudo wget https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm

sudo rpm --import https://repo.mysql.com/RPM-GPG-KEY-mysql-2022

sudo yum install https://dev.mysql.com/get/mysql57-community-release-el7-11.noarch.rpm

sudo yum install mysql

```

Total                                         65 MB/s | 35 MB   00:00
Running transaction check
Transaction check succeeded.
Running transaction test
Transaction test succeeded.
Running transaction
Preparing      :
Installing    : mysql-community-common-5.7.44-1.el7.x86_64          1/1
Installing    : mysql-community-libs-5.7.44-1.el7.x86_64          1/4
Running scriptlet: mysql-community-libs-5.7.44-1.el7.x86_64          2/4
Installing    : ncurses-compat-libs-6.2-4.20200222.amzn2023.0.5.x86_64 2/4
Installing    : mysql-community-client-5.7.44-1.el7.x86_64          3/4
Running scriptlet: mysql-community-client-5.7.44-1.el7.x86_64          4/4
Verifying     : ncurses-compat-libs-6.2-4.20200222.amzn2023.0.5.x86_64 4/4
Verifying     : mysql-community-client-5.7.44-1.el7.x86_64          1/4
Verifying     : mysql-community-common-5.7.44-1.el7.x86_64          2/4
Verifying     : mysql-community-libs-5.7.44-1.el7.x86_64          3/4
Verifying     : mysql-community-libs-5.7.44-1.el7.x86_64          4/4

Installed:
  mysql-community-client-5.7.44-1.el7.x86_64      mysql-community-common-5.7.44-1.el7.x86_64      mysql-community-libs-5.7.44-1.el7.x86_64
  ncurses-compat-libs-6.2-4.20200222.amzn2023.0.5.x86_64

Complete!

```

1. Start by downloading the MySQL CLI:

sudo yum install mysql -y

2. Initiate your DB connection with your Aurora RDS writer endpoint. In the following command, replace the RDS writer endpoint and the username, and then execute it in the browser terminal:

mysql -h CHANGE-TO-YOUR-RDS-ENDPOINT -u CHANGE-TO-USER-NAME -p

You will then be prompted to type in your password. Once you input the password and hit enter, you should now be connected to your database.

NOTE: If you cannot reach your database, check your credentials and security groups.

3. Create a database called **webappdb** with the following command using the MySQL CLI:

CREATE DATABASE webappdb;

You can verify that it was created correctly with the following command:

SHOW DATABASES;

4. Create a data table by first navigating to the database we just created:

USE webappdb;

Then, create the following **transactions** table by executing this create table command:

CREATE TABLE IF NOT EXISTS transactions(id INT NOT NULL

AUTO_INCREMENT, amount DECIMAL(10,2), description

VARCHAR(100), PRIMARY KEY(id));

Verify the table was created:

SHOW TABLES;

5. Insert data into table for use/testing later:

INSERT INTO transactions (amount,description) VALUES ('400','groceries');

Verify that your data was added by executing the following command:

SELECT * FROM transactions;

6. When finished, just type **exit** and hit enter to exit the MySQL client.

Configure App Instance:

1. The first thing we will do is update our database credentials for the app tier. To do this, open the **application-code/app-tier/DbConfig.js** file from the github repo in your favorite text editor on your computer. You'll see empty strings for the hostname, user, password and database. Fill this in with the credentials you configured for your database, the **writer** endpoint of your database as the hostname, and **webappdb** for the database. Save the file.

NOTE: This is NOT considered a best practice, and is done for the simplicity of the lab. Moving these credentials to a more suitable place like Secrets Manager is left as an extension for this workshop.

The screenshot shows the AWS S3 console interface. The path in the top navigation bar is: Amazon S3 > Buckets > demoweb-app-loke-1 > app-tier/. The main area displays the contents of the 'app-tier/' folder. There are six objects listed:

Name	Type	Last modified	Size	Storage class
index.js	js	February 6, 2024, 10:51:03 (UTC+05:30)	3.2 KB	Standard
package-lock.json	json	February 6, 2024, 10:51:04 (UTC+05:30)	42.9 KB	Standard
package.json	json	February 6, 2024, 10:51:03 (UTC+05:30)	682.0 B	Standard
README.md	md	February 6, 2024, 10:51:03 (UTC+05:30)	14.0 B	Standard
TransactionService.js	js	February 6, 2024, 10:51:03 (UTC+05:30)	1.8 KB	Standard

2. Upload the **app-tier** folder to the S3 bucket that you created in part 0.

3. Go back to your SSM session. Now we need to install all of the necessary components to run our backend application. Start by installing NVM (node version manager).

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
```

```
source ~/.bashrc
```

4. Next, install a compatible version of Node.js and make sure it's being used

```
nvm install 16
```

```
nvm use 16
```

5. PM2 is a daemon process manager that will keep our node.js app running when we exit the instance or if it is rebooted. Install that as well.

```
npm install -g pm2
```

6. Now we need to download our code from our s3 buckets onto our instance. In the command below, replace BUCKET_NAME with the name of the bucket you uploaded the **app-tier** folder to:

```
cd ~/
```

```
aws s3 cp s3://BUCKET_NAME/app-tier/ app-tier --recursive
```

7. Navigate to the app directory, install dependencies, and start the app with pm2.

```
cd ~/app-tier
```

```
npm install
```

```
pm2 start index.js
```

```
Run `npm audit` for details.
[ec2-user@ip-10-0-0-79 app-tier]$ pm2 start index.js
[PM2] Starting /home/ec2-user/app-tier/index.js in fork_mode (1 instance)
[PM2] Done.



| id | name  | namespace | version | mode | pid   | uptime | ø | status | cpu | mem    | user     | watching |
|----|-------|-----------|---------|------|-------|--------|---|--------|-----|--------|----------|----------|
| 0  | index | default   | 1.0.0   | fork | 20302 | 0s     | 0 | online | 0%  | 24.4mb | ec2-user | disabled |


```

To make sure the app is running correctly run the following:

```
pm2 list
```

If you see a status of online, the app is running. If you see errored, then you need to do some troubleshooting. To look at the latest errors, use this command:

pm2 logs

NOTE: If you're having issues, check your configuration file for any typos, and double check that you have followed all installation commands till now.

8. Right now, pm2 is just making sure our app stays running when we leave the SSM session. However, if the server is interrupted for some reason, we still want the app to start and keep running. This is also important for the AMI we will create:

pm2 startup

Test App Tier:

Now let's run a couple tests to see if our app is configured correctly and can retrieve data from the database.

To hit out health check endpoint, copy this command into your SSM terminal. This is our simple health check endpoint that tells us if the app is simply running.

curl <http://localhost:4000/health>

The response should looks like the following:

"This is the health check"

Next, test your database connection. You can do that by hitting the following endpoint locally:

curl <http://localhost:4000/transaction>

If you see both of these responses, then your networking, security, database and app configurations are correct.

7. Congrats! Your app layer is fully configured and ready to go.

Part 4: Internal Load Balancing and Auto Scaling

In this section of the workshop we will create an Amazon Machine Image (AMI) of the app tier instance we just created, and use that to set up autoscaling with a load balancer in order to make this tier highly available.

Learning Objectives:

- Create an AMI of our App Tier
- Create a Launch Template
- Configure Autoscaling
- Deploy Internal Load Balancer

App Tier AMI

1. Navigate to **Instances** on the left hand side of the EC2 dashboard. Select the app tier instance we created and under **Actions** select **Image and templates**. Click **Create Image**.

The screenshot shows the AWS EC2 Instances page with 10 instances listed. A search bar at the top has 'app' entered. The 'threeapptier' instance is selected and highlighted with a blue border. A context menu is open over this instance, with 'Image and templates' being the highlighted option. Other visible options in the menu include 'View details', 'Manage instance state', 'Instance settings', 'Networking', 'Security', 'Monitor and troubleshoot', and 'View alarms'.

2. Give the image a name and description and then click **Create image**. This will take a few minutes, but if you want to monitor the status of image creation you can see it by clicking **AMIs** under **Images** on the left hand navigation panel of the EC2 dashboard.

The screenshot shows the 'Create image' dialog box. At the top, it says 'Create image Info'. Below that is a note: 'An image (also referred to as an AMI) defines the programs and settings that are applied when you launch an EC2 instance. You can create an image from the console or from the command line.' The form fields are:

- Instance ID: i-06630b81d70ee6970 (threeapptier)
- Image name:
- Image description - optional:
- Maximum 255 characters
- No reboot:

Target Group

1. While the AMI is being created, we can go ahead and create our target group to use with the load balancer. On the EC2 dashboard navigate to **Target Groups** under **Load Balancing** on the left hand side. Click on **Create Target Group**.

The screenshot shows the AWS EC2 Target Groups page. On the left sidebar, under 'Load Balancing', 'Target Groups' is selected. The main area displays a table titled 'Target groups (3)'. One row is visible, showing a target group named 'threapp-tier' with ARN 'arn:aws:elasticloadbalancing:us-east-1:123456789012:targetgroup/threapp-tier/1234567890123456', port 4000, protocol HTTP, and target type Instance.

2. The purpose of forming this target group is to use with our load balancer so it may balance traffic across our private app tier instances. Select Instances as the target type and give it a name.

The screenshot shows the 'Basic configuration' step of the 'Create Target Group' wizard. It includes sections for 'Choose a target type' (selected 'Instances'), 'Protocol & Port' (HTTP, port 80), and 'IP address type' (IPv4 selected). The 'Protocol & Port' section also shows a note about using Auto Scaling.

Then, set the protocol to **HTTP** and the port to **4000**. Remember that this is the port our Node.js app is running on. Select the VPC we've been using thus far, and then change the health check path to be **/health**. This is the health check endpoint of our app. Click **Next**.

The screenshot shows the 'Protocol & Port' configuration step. It lists 'Protocol' (HTTP, port 4000), 'IP address type' (IPv4 selected), 'VPC' (three-tier-app selected), and 'Protocol version' (HTTP1).

3. We are **NOT** going to register any targets for now, so just skip that step and create the target group.

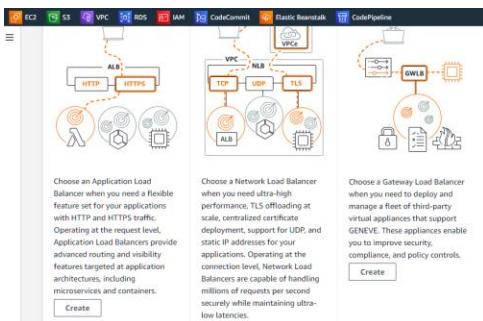
The screenshot shows the 'Register targets' and 'Review targets' steps. The 'Register targets' step indicates 'No instances added yet' and 'Specify instances above, or leave the group empty if you prefer to add targets later.' The 'Review targets' step shows a table with columns: Targets (0), Instance ID, Name, Port, State, Security groups, Zone, Private IPv4 address, and Subnet. A note at the bottom says '0 pending'.

Internal Load Balancer:

1. On the left hand side of the EC2 dashboard select **Load Balancers** under **Load Balancing** and click **Create Load Balancer**.

The screenshot shows the AWS Lambda service dashboard. On the left sidebar, under 'Network & Security', 'Load Balancers' is selected. At the top right, there is a large orange 'Create load balancer' button. Below it, the 'Load balancers' section shows a table with one entry: 'internallb' (DNS name: internalinternallb.209.16, State: Active, VPC ID: vpc-0506ed0d6b6035e). A message at the bottom says '0 load balancers selected'.

2. We'll be using an **Application Load Balancer** for our **HTTP** traffic so click the create button for that option.



3. After giving the load balancer a name, be sure to select **internal** since this one will not be public facing, but rather it will route traffic from our web tier to the app tier.

The screenshot shows the 'Basic configuration' step. It includes fields for 'Load balancer name' (app-tier-internal-lb), 'Scheme' (set to 'Internal'), and 'IP address type' (set to 'IPv4'). There are also sections for 'Network mapping' and 'Mappings' where specific subnets and availability zones are selected.

Select the correct network configuration for VPC and private subnets

The screenshot shows the 'Network mapping' step. It lists a VPC ('three-tier-app') and its subnets ('subnet-065ab6a42801b273' and 'subnet-0316cedb6b48869e5'). Under 'Mappings', two subnets are selected: 'us-east-1a (use1-az1)' and 'us-east-1b (use1-az2)'. Each subnet is associated with a private IP range ('app-tier-private-1a' and 'app-tier-private-1b').

Select the security group we created for this internal ALB. Now, this ALB will be listening for HTTP traffic on port 80. It will be forwarding the traffic to our **target group** that we just created, so select it from the dropdown, and create the load balancer

The screenshot shows the 'Listeners and routing' section of the Load Balancer configuration. Under 'Listener: HTTP:80', the 'Protocol' is set to 'HTTP' and the 'Port' is set to '80'. The 'Default action' dropdown is set to 'Select a target group' and has a placeholder 'Create target group'. A security group named 'internal-lb-sg' is selected.

Launch Template

- Before we configure Auto Scaling, we need to create a Launch template with the AMI we created earlier. On the left side of the EC2 dashboard navigate to **Launch Template** under **Instances** and click **Create Launch Template**.

Launch Template ID	Launch Template Name	Default Version	Latest Version	Create Time
lt-0a6c47d84866420fa	threetier-launch-template	1	2	2024-02-06T04:56:24.0
lt-0308c360ba66624a9	webtier	1	1	2024-02-06T07:29:22.0

- Name the Launch Template, and then under **Application and OS Images** include the app tier AMI you created.

The screenshot shows the 'My AMIs' tab selected. A search bar is at the top. Below it are filters: 'Don't include in launch template' (unchecked), 'Owned by me' (checked), and 'Shared with me' (unchecked). A 'Browse more AMIs' link is on the right. At the bottom, the 'webtier-image' AMI is selected, showing its details: AMI ID 'ami-031512dd8351d224d', creation date '2024-02-06T06:37:47.000Z', Virtualization type 'hvm', ENA enabled status 'true', and Root device type 'ebs'.

Under **Instance Type** select t2.micro. For **Key pair** and **Network Settings** don't include it in the template. We don't need a key pair to access our instances and we'll be setting the network information in the autoscaling group.

Instance type [Info](#) | [Get advice](#)

t2.micro Family: t2 1 vCPU 1 GiB Memory Current generation: true On-Demand Windows base pricing: 0.0162 USD per Hour On-Demand SUSE base pricing: 0.0116 USD per Hour On-Demand RHEL base pricing: 0.0716 USD per Hour On-Demand Linux base pricing: 0.0116 USD per Hour

Free tier eligible

[All generations](#) [Compare instance types](#)

Additional costs apply for AMIs with pre-installed software

Set the correct security group for our app tier, and then under **Advanced details** use the same IAM instance profile we have been using for our EC2 instances.

When you specify a source, a network interface is automatically added to your compute.

Firewall (security groups) [Info](#)
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach instance.

Select existing security group Create security group

Common security groups [Info](#)
[Select security groups](#)

privateinstances-sg sg-0bc5869f4229fc08 X
VPC: vpc-04a1ecf8ab7e6c3b

Compare sec group rules

Security groups that you add or remove here will be added to or removed from all your network interfaces.

Advanced details [Info](#)

IAM instance profile [Info](#)
demoEC2-role arn:aws:iam::975049859516:instance-profile/demoEC2-role

Hostname type [Info](#)
Don't include in launch template

DNS Hostname [Info](#)
 Enable resource-based IPv4 (A record) DNS requests
 Enable resource-based IPv6 (AAAA record) DNS requests

Auto Scaling

1. We will now create the Auto Scaling Group for our app instances. On the left side of the EC2 dashboard navigate to **Auto Scaling Groups** under **Auto Scaling** and click **Create Auto Scaling group**.

Network & Security

- Security Groups
- Elastic IPs
- Placement Groups
- Key Pairs
- Network Interfaces

Load Balancing

- Load Balancers
- Target Groups
- Trust Stores [New](#)

Auto Scaling

- Auto Scaling Groups**

Create Auto Scaling group

Name	Launch template/configuration	Instances	Status
sppwebtier	webtier Version 1	2	-
webapp	webtier Version 1	2	-
webtier	webtier-launch-template Version 3	0	Update
threetierapp-sg	threetier-launch-template Version 2	2	-

0 Auto Scaling groups selected

2. Give your Auto Scaling group a name, and then select the Launch Template we just created and click next.

Choose launch template [Info](#)
Specify a launch template that contains settings common to all EC2 instances that are launched by this Auto Scaling group.

Name

Auto Scaling group name
Enter a name to identify the group.
app-tier-asg

Must be unique to this account in the current Region and no more than 255 characters.

Launch template [Info](#)

For accounts created after May 31, 2023, the EC2 console only supports creating Auto Scaling groups with launch templates. Creating Auto Scaling groups with launch configurations is not recommended but still available via the CLI and API until December 31, 2025.

Launch template
Choose a launch template that contains the instance-level settings, such as the Amazon Machine Image (AMI), instance type, key pair, and security groups.
webtier

[Create a launch template](#)

3. On the **Choose instance launch options** page set your VPC, and the private instance subnets for the app tier and continue to step 3.

Network Info

For most applications, you can use multiple Availability Zones and let EC2 Auto Scaling balance your instances across the zones. The default VPC and default subnets are suitable for getting started quickly.

VPC
Choose the VPC that defines the virtual network for your Auto Scaling group.
vpc-0506ded4ddab6035e (three-tier-app)
10.0.0.0/16

Create a VPC [Create a VPC](#)

Availability Zones and subnets
Define which Availability Zones and subnets your Auto Scaling group can use in the chosen VPC.
Select Availability Zones and subnets [Select Availability Zones and subnets](#)

- us-east-1a | subnet-0e63ab6a42801b273 (app-tier-private-1a)
10.0.16.0/20
- us-east-1b | subnet-0316cedb6b48869e3 (app-tier-private-1b)
10.0.32.0/20

Create a subnet [Create a subnet](#)

For this next step, attach this Auto Scaling Group to the Load Balancer we just created by selecting the existing load balancer's target group from the dropdown. Then, click next

Use the options below to attach your Auto Scaling group to an existing load balancer, or to a new load balancer that you define.

No load balancer
Traffic to your Auto Scaling group will not be fronted by a load balancer.

Attach to an existing load balancer
Choose from your existing load balancers.

Attach to a new load balancer
Quickly create a basic load balancer to attach to your Auto Scaling group.

Attach to an existing load balancer
Select the load balancers that you want to attach to your Auto Scaling group.

Choose from your load balancer target groups
This option allows you to attach Application, Network, or Gateway Load Balancers.

Choose from Classic Load Balancers

Existing load balancer target groups
Only instance target groups that belong to the same VPC as your Auto Scaling group are available for selection.
Select target groups [Select target groups](#)

threapp-tier | HTTP
Application Load Balancer: internallb

5. For **Configure group size and scaling policies**, set desired, minimum and maximum capacity to 2. Click skip to review and then Create Auto Scaling Group.

Group size Info
Set the initial size of the Auto Scaling group. After creating the group, you can change its size to meet demand, either manually or by using automatic scaling.

Desired capacity type
Choose the unit of measurement for the desired capacity value. vCPUs and Memory(GiB) are only supported for mixed instances groups configured with a set of instance attributes.

Units (number of instances) [Select](#)

Desired capacity
Specify your group size.
2

Scaling Info
You can resize your Auto Scaling group manually or automatically to meet changes in demand.

Scaling limits
Set limits on how much your desired capacity can be increased or decreased.

Min desired capacity	Max desired capacity
2	2
Equal or less than	Equal or greater than

You should now have your internal load balancer and autoscaling group configured correctly. You should see the autoscaling group spinning up 2 new app tier instances. If you wanted to test if this is working correctly, you can delete one of your new instances manually and wait to see if a new instance is booted up to replace it.

NOTE: Your original app tier instance is excluded from the ASG so you will see 3 instances in the EC2 dashboard. You can delete your original instance that you used to generate the app tier AMI but it's recommended to keep it around for troubleshooting purposes.

Part 5: Web Tier Instance Deployment

In this section we will deploy an EC2 instance for the web tier and make all necessary software configurations for the NGINX web server and React.js website.

Learning Objectives

- Update NGINX Configuration Files
- Create Web Tier Instance
- Configure Software Stack

Update Config File

Before we create and configure the web instances, open up the **application-code/nginx.conf** file from the repo we downloaded. Scroll down to **line 58** and replace [INTERNAL-LOADBALANCER-DNS] with your internal load balancer's DNS entry. You can find this by navigating to your internal load balancer's details page.

```
#proxy for internal lb
location /api/ {
    proxy_pass http://internal-internallb-209461046.us-east-1.elb.amazonaws.com:80;
}

ings for a TLS enabled server.

server {
    listen      443 ssl http2;
    listen      [::]:443 ssl http2;
    server_name _;
    root        /usr/share/nginx/html;

    ssl_certificate "/etc/pki/nginx/server.crt";
    ssl_certificate_key "/etc/pki/nginx/private/server.key";
```

Then, upload this file **and** the **application-code/web-tier** folder to the s3 bucket you created for this lab.

Web Instance Deployment

- Follow the same instance creation instructions we used for the App Tier instance in **Part 3: App Tier Instance Deployment**, with the exception of the subnet. We will be provisioning this instance in one of our **public subnets**. Make sure to select the correct network components, security group, and IAM role. **This time, auto-assign a public ip** on the **Configure Instance Details** page. Remember to tag the instance with a name so we can identify it more easily.

The screenshot shows the AWS CloudWatch Metrics Insights interface. A query is being run against the CloudWatch Metrics Insights metrics. The results are displayed in two main sections: 'Metrics' and 'Logs'.

Metric Results:

- Series: CloudWatch Metrics Insights Metrics
- Dimensions: None
- Time Range: 1 hour ago to now
- Metrics:
 - CloudWatch Metrics Insights Metrics: Value 1, Last 1 hour
 - CloudWatch Metrics Insights Metrics: Value 2, Last 1 hour

Log Results:

- Log Stream: /aws/lambda/lambda-function
- Time Range: 1 hour ago to now
- Logs:
 - CloudWatch Metrics Insights Metrics: Value 1, Last 1 hour
 - CloudWatch Metrics Insights Metrics: Value 2, Last 1 hour

Connect to Instance

- Follow the same steps you used to connect to the app instance and change the user to **ec2-user**. Test connectivity here via ping as well since this instance should have internet connectivity:

```
sudo -su ec2-user
```

```
ping 8.8.8.8
```

Note: If you don't see a transfer of packets then you'll need to verify your route tables attached to the subnet that your instance is deployed in.

Configure Web Instance

- We now need to install all of the necessary components needed to run our front-end application. Again, start by installing NVM and node :

```
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash
source ~/.bashrc
nvm install 16
nvm use 16
```

- Now we need to download our web tier code from our s3 bucket:

```
cd ~/
aws s3 cp s3://BUCKET_NAME/web-tier/ web-tier --recursive
```

Navigate to the web-layer folder and create the build folder for the react app so we can serve our code:

```
cd ~/web-tier  
npm install  
npm run build
```

3. NGINX can be used for different use cases like load balancing, content caching etc, but we will be using it as a web server that we will configure to serve our application on port 80, as well as help direct our API calls to the internal load balancer.

```
sudo amazon-linux-extras install nginx1 -y
```

Installation instructions for Amazon Linux 2023

- Create the /etc/ssl/nginx/ directory:
 - `sudo mkdir -p /etc/ssl/nginx`
- Install prerequisite packages:
 - `sudo yum install ca-certificates`
- Copy the above two files to the Amazon Linux server into `/etc/ssl/nginx/` directory. Use your SCP client or other secure file transfer tools.
- Add NGINX Plus repository by downloading the file [plus-amazonlinux2023.repo](#) to `/etc/yum.repos.d`:
 - `sudo wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/plus-amazonlinux2023.repo`
- If you have modsecurity subscription, add modsecurity repository by downloading the file [modsecurity-amazonlinux2023.repo](#) to `/etc/yum.repos.d`:
 - `sudo wget -P /etc/yum.repos.d https://cs.nginx.com/static/files/modsecurity-amazonlinux2023.repo`

Update the Package Repository

Before installing Nginx, it's a good practice to update the package repository on your Amazon Linux instance. To do this, run the following command:

```
sudo dnf update -y
```

Install Nginx

Next, we need to install Nginx on Amazon Linux 2023. To do this, run the following command:

Copy Code`sudo dnf install nginx -y`

Start Nginx Once the installation is complete, start the Nginx service using the following command:

```
sudo systemctl status nginx
```

If Nginx is running correctly, you will see output similar to the following:

```
[ec2-user@ip-10-0-0-79 nginx]$ sudo systemctl status nginx
● nginx.service - The nginx HTTP and reverse proxy server
   Loaded: loaded (/usr/lib/systemd/system/nginx.service; enabled; preset: disabled)
   Active: active (running) since Wed 2024-02-07 05:41:07 UTC; 29min ago
     Process: 2132 ExecStartPre=/usr/bin/rm -f /run/nginx.pid (code=exited, status=0/SUCCESS)
     Process: 2144 ExecStartPre=/usr/sbin/nginx -t (code=exited, status=0/SUCCESS)
     Process: 2149 ExecStart=/usr/sbin/nginx (code=exited, status=0/SUCCESS)
   Main PID: 2150 (nginx)
      Tasks: 2 (limit: 1114)
        Memory: 4.8M
         CPU: 62ms
        CGroup: /system.slice/nginx.service
                  └─2150 "nginx: master process /usr/sbin/nginx"
                     ├─2151 "nginx: worker process"

Feb 07 05:41:07 ip-10-0-0-79.ec2.internal systemd[1]: Starting nginx.service - The nginx HTTP and reverse proxy server...
Feb 07 05:41:07 ip-10-0-0-79.ec2.internal nginx[2144]: nginx: the configuration file /etc/nginx/nginx.conf syntax is ok
Feb 07 05:41:07 ip-10-0-0-79.ec2.internal nginx[2144]: nginx: configuration file /etc/nginx/nginx.conf test is successful
Feb 07 05:41:07 ip-10-0-0-79.ec2.internal systemd[1]: Started nginx.service - The nginx HTTP and reverse proxy server.
```

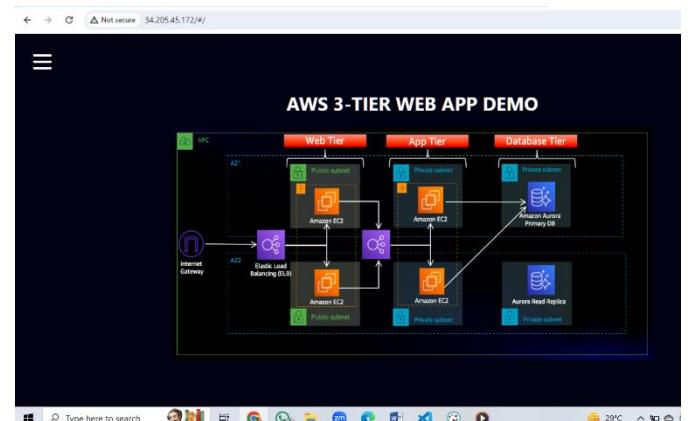
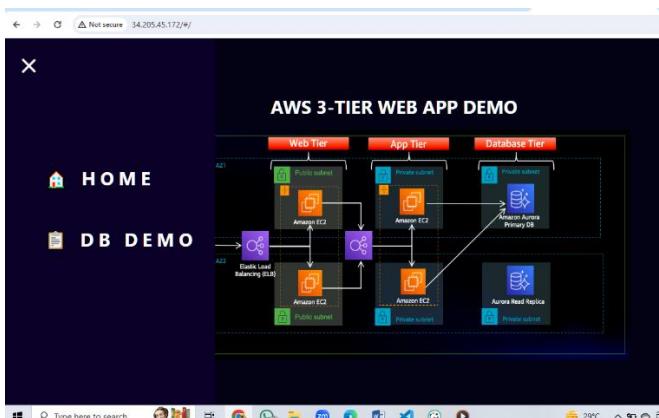
To make sure Nginx has permission to access our files execute this command:

```
chmod -R 755 /home/ec2-user
```

And then to make sure the service starts on boot, run this command:

```
sudo chkconfig nginx on
```

- Now when you plug in the **public IP** of your web tier instance, you should see your website, which you can find on the Instance details page on the EC2 dashboard. If you have the database connected and working correctly, then you will also see the database working. You'll be able to add data. Careful with the delete button, that will clear all the entries in your database.



The screenshot shows the 'AURORA DATABASE DEMO PAGE'. At the top, there is a header with three horizontal bars. Below the header, there is a table with three columns: 'ID', 'AMOUNT', and 'DESC'. There is a 'DEL' button in the top right corner of the table. At the bottom left, there is an 'ADD' button. The table currently contains one row with empty fields.

ID	AMOUNT	DESC
<input type="text"/>	<input type="text"/>	<input type="text"/>

Part 6: External Load Balancer and Auto Scaling

In this section of the workshop we will create an Amazon Machine Image (AMI) of the web tier instance we just created, and use that to set up autoscaling with an external facing load balancer in order to make this tier highly available.

Learning Objectives:

- Create an AMI of our Web Tier
- Create a Launch Template
- Configure Auto Scaling
- Deploy External Load Balancer

Web Tier AMI

1. Navigate to **Instances** on the left hand side of the EC2 dashboard. Select the web tier instance we created and under **Actions** select **Image and templates**. Click **Create Image**.
2. we created and under **Actions** select **Image and templates**. Click **Create Image**.

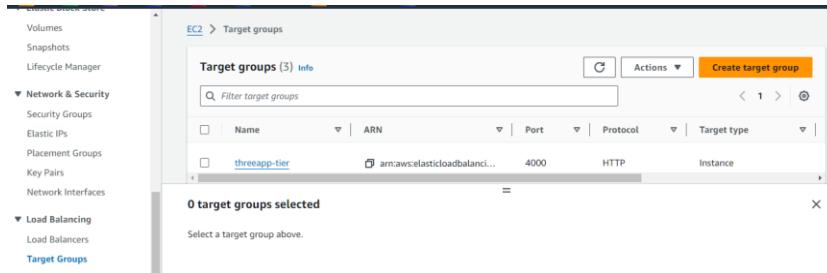
The screenshot shows the AWS EC2 Instances page with 10 instances listed. One instance, 'threapptier', is selected. The Actions menu is open, showing options like 'Create image' and 'Create template from instance'. The 'Image and templates' option is highlighted.

3. Give the image a name and description and then click **Create image**. This will take a few minutes, but if you want to monitor the status of image creation you can see it by clicking **AMIs** under **Images** on the left hand navigation panel of the EC2 dashboard.

The screenshot shows the 'Create image' dialog box. It includes fields for 'Image name' (placeholder: 'Enter image name'), 'Image description - optional' (placeholder: 'Image description'), and checkboxes for 'No reboot' and 'Create ami directly'.

Target Group

2. While the AMI is being created, we can go ahead and create our target group to use with the load balancer. On the EC2 dashboard navigate to **Target Groups** under **Load Balancing** on the left hand side. Click on **Create Target Group**.



3. The purpose of forming this target group is to use with our load balancer so it may balance traffic across our private app tier instances. Select Instances as the target type and give it a name.

A screenshot of the 'Basic configuration' step of the 'Create Target Group' wizard. It shows two tabs: 'Instances' (selected) and 'IP addresses'. Under 'Instances', it says 'Supports load balancing to instances within a specific VPC.' Under 'IP addresses', it says 'Supports load balancing to VPC and on-premises resources.' The 'Protocol' section shows 'HTTP' selected with port '80'. The 'IP address type' section shows 'IPv4' selected.

Then, set the protocol to **HTTP** and the port to 4000. Remember that this is the port our Node.js app is running on. Select the VPC we've been using thus far, and then change the health check path to be **/health**. This is the health check endpoint of our app. Click **Next**.

A screenshot of the 'Protocol & Port' step of the 'Create Target Group' wizard. It shows the 'Protocol' dropdown set to 'HTTP' and the 'Port' dropdown set to '4000'. Below that is the 'IP address type' section with 'IPv4' selected. Further down are 'VPC' and 'Protocol version' sections, both currently set to their defaults.

4. We are **NOT** going to register any targets for now, so just skip that step and create the target group.

Register targets

This is an optional step to create a target group. However, to ensure that your load balancer routes traffic to this target group you must register your targets.

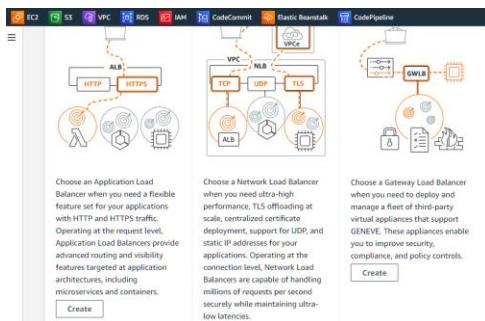
A screenshot of the 'Available instances' step of the 'Register targets' wizard. It shows a table with six rows of instance details. The columns are Instance ID, Name, State, and Security groups. The first three rows have 'Running' status and 'app-tier-sg' security group. The last three rows have 'Running' status and 'web-tier-sg' security group.A screenshot of the 'Review targets' step of the 'Register targets' wizard. It shows a table with zero rows. A message at the bottom says 'No instances added yet. Specify instances above, or leave the group empty if you prefer to add targets later.' At the bottom right are 'Cancel', 'Previous', and 'Create target group' buttons.

Internal Load Balancer:

2. On the left hand side of the EC2 dashboard select **Load Balancers** under **Load Balancing** and click **Create Load Balancer**.

The screenshot shows the AWS EC2 dashboard with the 'Load Balancers' section selected. The table lists a single load balancer named 'internallb'. The 'Name' column shows 'internallb', the 'DNS name' column shows 'internal-internallb-20216', the 'State' column shows 'Active', the 'VPC ID' column shows 'vpc-0506defdd6b6035e', and the 'Availability Zones' column shows 'us-east-1a, us-east-1b'. A search bar at the top is labeled 'Filter load balancers'.

3. We'll be using an **Application Load Balancer** for our **HTTP** traffic so click the create button for that option.



4. After giving the load balancer a name, be sure to select **internal** since this one will not be public facing, but rather it will route traffic from our web tier to the app tier.

The screenshot shows the 'Basic configuration' step of the AWS Load Balancer creation wizard. It includes fields for 'Load balancer name' (set to 'app-tier-internal-lb'), 'Scheme' (radio button selected for 'Internal'), and 'IP address type' (radio button selected for 'IPv4').

Select the correct network configuration for VPC and private subnets

The screenshot shows the 'Network mapping' step of the AWS Load Balancer creation wizard. It includes a dropdown for 'VPC' (selected 'three-tier-app') and two sections for 'Mappings'. The first section, 'us-east-1a (use1-a1)', has a checked checkbox and a 'Subnet' dropdown set to 'subnet-0e65ab6a42801b273'. The second section, 'us-east-1b (use1-a2)', also has a checked checkbox and a 'Subnet' dropdown set to 'subnet-0316cedb6b48869e3'. Both sections have dropdowns for 'app-tier-private-1a' and 'app-tier-private-1b'.

Select the security group we created for this internal ALB. Now, this ALB will be listening for HTTP traffic on port 80. It will be forwarding the traffic to our **target group** that we just created, so select it from the dropdown, and create the load balancer

The screenshot shows the 'Listeners and routing' section of the Load Balancer configuration. Under 'Listener HTTP:80', a new rule is being defined. The 'Protocol' is set to 'HTTP' and the 'Port' is set to '80'. The 'Default action' dropdown is set to 'Select a target group', which is currently empty. A 'Create target group' button is visible below the dropdown.

Launch Template

- Before we configure Auto Scaling, we need to create a Launch template with the AMI we created earlier. On the left side of the EC2 dashboard navigate to **Launch Template** under **Instances** and click **Create Launch Template**.

The screenshot shows the 'Launch Templates' page with two existing templates listed: 'threetier-launch-template' and 'webtier'. A 'Create launch template' button is located at the top right. The left sidebar shows navigation options like 'EC2 Dashboard', 'EC2 Global View', 'Events', 'Console-to-Code', and 'Instances' (with 'Launch Templates' selected).

- Name the Launch Template, and then under **Application and OS Images** include the app tier AMI you created.

The screenshot shows the 'Application and OS Images (Amazon Machine Image)' page. It includes a search bar, filter options ('Recent', 'My AMIs', 'Quick Start'), and a 'Browse more AMIs' section. The 'My AMIs' tab is selected, and the 'Owned by me' filter is applied. A specific AMI, 'webtier-image', is selected, showing its details: 'ami-031512dd8351d224d', '2024-02-06T06:37:47.000Z', 'Virtualization: hvm', 'ENA enabled: true', and 'Root device type: ebs'.

Under **Instance Type** select t2.micro. For **Key pair** and **Network Settings** don't include it in the template. We don't need a key pair to access our instances and we'll be setting the network information in the autoscaling group.

The screenshot shows two side-by-side sections of the AWS EC2 instance configuration interface.

Instance type: Shows the selected instance type as t2.micro, which is Free tier eligible. It provides details about the family (t2), vCPU count (1), memory (1 GiB), and current generation (true). Pricing information for On-Demand Windows, SUSE, RHEL, and Linux is listed. A note at the bottom states: "Additional costs apply for AMIs with pre-installed software".

Network settings: Shows the selected subnet as subnet-0417c59905b7fe8c0, located in VPC vpc-04a1ecf88ab7e6c3b, owned by user 975049895916, and in Availability Zone us-east-1b. It also lists IP addresses available (248) and CIDR (10.0.6.0/24).

Set the correct security group for our app tier, and then under **Advanced details** use the same IAM instance profile we have been using for our EC2 instances.

The screenshot shows the security groups configuration for the instance.

Firewall (security groups): Includes options to "Select existing security group" (radio button selected) or "Create security group".

Common security groups: A dropdown menu titled "Select security groups" contains the entry "privateinstances-sg_sg-0bcb5869f4229fc08" with a delete icon (X).

Compare sec group rules: A link to compare security group rules.

Security groups: A note stating "Security groups that you add or remove here will be added to or removed from all your network interfaces."

The screenshot shows the advanced details configuration for the instance.

IAM instance profile: Set to "demoEC2-role" (radio button selected). A note indicates the ARN: "arn:aws:iam::975049895916:instance-profile/demoEC2-role".

Hostname type: Set to "Don't include in launch template".

DNS Hostname: Options to "Enable resource-based IPv4 (A record) DNS requests" and "Enable resource-based IPv6 (AAAA record) DNS requests".

Auto Scaling

2. We will now create the Auto Scaling Group for our app instances. On the left side of the EC2 dashboard navigate to **Auto Scaling Groups** under **Auto Scaling** and click **Create Auto Scaling group**.

The screenshot shows the Auto Scaling Groups section of the EC2 dashboard.

Network & Security: Includes links for Security Groups, Elastic IPs, Placement Groups, Key Pairs, and Network Interfaces.

Load Balancing: Includes links for Load Balancers, Target Groups, and Trust Stores.

Auto Scaling: Includes a link for "Auto Scaling Groups".

Launch configurations: A tab showing a list of existing launch configurations: "appwebtier", "webapp", "webtier", and "threetierapp-sg".

Actions: A dropdown menu with options like "Create Auto Scaling group", "Launch configuration", "Launch template", and "Actions".

Create Auto Scaling group: A prominent orange button.

3. Give your Auto Scaling group a name, and then select the Launch Template we just created and click next.

The screenshot shows the "Choose launch template" step of the Auto Scaling group creation wizard.

Name: A text input field where "apptierasg" is entered.

Auto Scaling group name: A note stating "Enter a name to identify the group." Below is a dropdown menu with the value "apptierasg".

Must be unique to this account in the current Region and no more than 255 characters.

The screenshot shows the "Launch template" step of the Auto Scaling group creation wizard.

Launch template: A note stating "For accounts created after May 31, 2023, the EC2 console only supports creating Auto Scaling groups with launch templates. Creating Auto Scaling groups with launch configurations is not recommended but still available via the CLI and API until December 31, 2023."

Launch template: A dropdown menu set to "webtier".

Create a launch template: A blue "Create" button.

5. On the Choose instance launch options page set your VPC, and the public instance subnets for the web tier and continue to step 3.

Network Info

For most applications, you can use multiple Availability Zones and let EC2 Auto Scaling balance your instances across the zones. The default VPC and default subnets are suitable for getting started quickly.

VPC
Choose the VPC that defines the virtual network for your Auto Scaling group.
vpc-0506ded4ddab6035e (three-tier-app) ▾

Create a VPC

Availability Zones and subnets
Define which Availability Zones and subnets your Auto Scaling group can use in the chosen VPC.

Select Availability Zones and subnets ▾

us-east-1a | subnet-0e63ab6a42801b273 (app-tier-private-1a)
10.0.16.0/20

us-east-1b | subnet-0316cedb6b48869e3 (app-tier-private-1b)
10.0.32.0/20

Create a subnet

For this next step, attach this Auto Scaling Group to the Load Balancer we just created by selecting the existing load balancer's target group from the dropdown. Then, click next

Use the options below to attach your Auto Scaling group to an existing load balancer, or to a new load balancer that you define.

No load balancer
Traffic to your Auto Scaling group will not be fronted by a load balancer.

Attach to an existing load balancer
Choose from your existing load balancers.

Attach to a new load balancer
Quickly create a basic load balancer to attach to your Auto Scaling group.

Attach to an existing load balancer

Select the load balancers that you want to attach to your Auto Scaling group.

Choose from your load balancer target groups
This option allows you to attach Application, Network, or Gateway Load Balancers.

Choose from Classic Load Balancers

Existing load balancer target groups

Only instance target groups that belong to the same VPC as your Auto Scaling group are available for selection.

Select target groups ▾

threeapp-tier | HTTP
Application Load Balancer: internallb

6. For Configure group size and scaling policies, set desired, minimum and maximum capacity to 2. Click skip to review and then Create Auto Scaling Group.

Group size Info

Set the initial size of the Auto Scaling group. After creating the group, you can change its size to meet demand, either manually or by using automatic scaling.

Desired capacity type

Choose the unit of measurement for the desired capacity value. vCPUs and Memory(GiB) are only supported for mixed instances groups configured with a set of instance attributes.

Units (number of instances) ▾

Desired capacity

Specify your group size.

2

Scaling Info

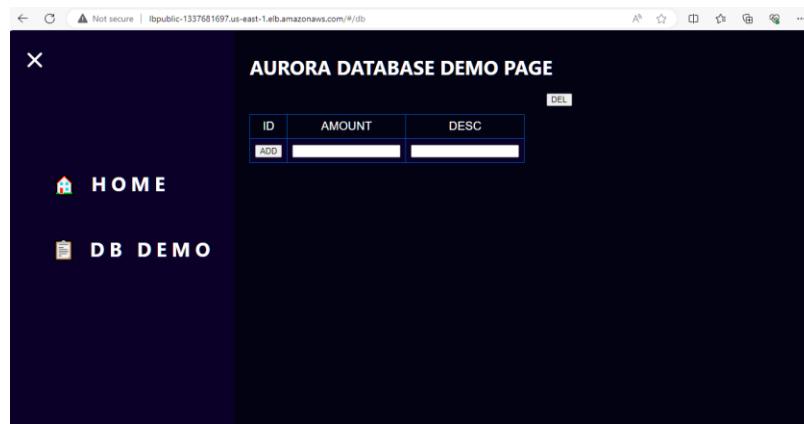
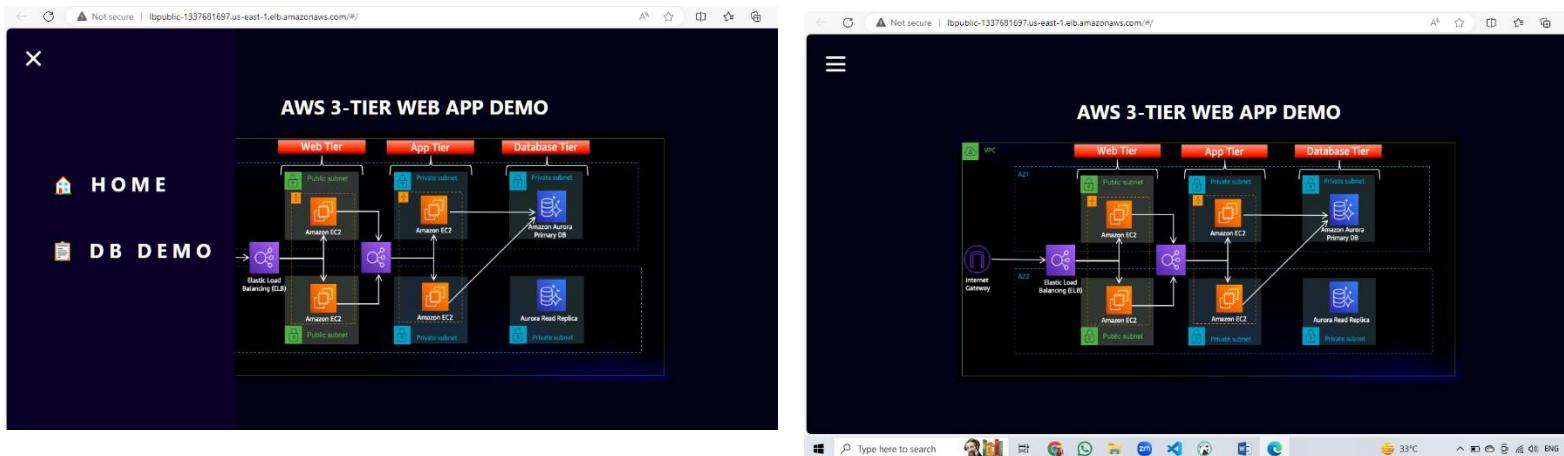
You can resize your Auto Scaling group manually or automatically to meet changes in demand.

Scaling limits

Set limits on how much your desired capacity can be increased or decreased.

Min desired capacity	Max desired capacity
2 <input type="button" value="Change"/>	2 <input type="button" value="Change"/>
Equal or less than	Equal or greater than

You should now have your external load balancer and autoscaling group configured correctly. You should see the autoscaling group spinning up 2 new web tier instances. If you wanted to test if this is working correctly, you can delete one of your new instances manually and wait to see if a new instance is booted up to replace it. To test if your entire architecture is working, navigate to your external facing loadbalancer, and plug in the DNS name into your browser.



NOTE: Again, your original web tier instance is excluded from the ASG so you will see 3 instances in the EC2 dashboard. You can delete your original instance that you used to generate the web tier AMI but it's recommended to keep it around for troubleshooting purposes.

Congrats! You've Implemented a 3 Tier Web Architecture!

Appendix

DOCUMENTATION:

- NVM (Node Version Manager)
 - <https://github.com/nvm-sh/nvm>
- Node.js
 - <https://nodejs.org>
- PM2
 - <https://pm2.keymetrics.io/docs/usage/pm2-doc-single-page/>
- NGINX
 - https://docs.nginx.com/?_ga=2.4975283.1227355032.1619285306-501395053.1619109877
- Amazon Linux Extras
 - <https://aws.amazon.com/amazon-linux-2/faqs/>