**AWS Project**

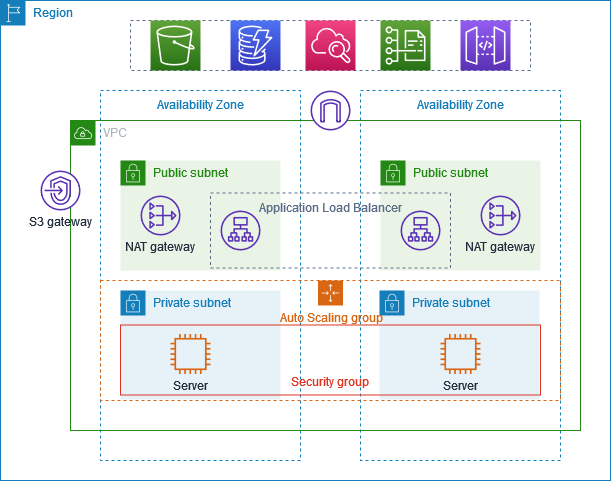
**VPC with public-private subnet in Production**

**About this Project:**

* This example demonstrates how to create a VPC that you can use for servers in a production environment.
* To improve resiliency, you deploy the servers in two Availability Zones, by using an Auto Scaling group and an Application Load Balancer. For additional security, you deploy the servers in private subnets. The servers receive requests through the load balancer. The servers can connect to the internet by using a NAT gateway. To improve resiliency, you deploy the NAT gateway in both Availability Zones.

**OVERVIEW:**

* The VPC has public subnets and private subnets in two Availability Zones.
* Each public subnet contains a NAT gateway and a load balancer node.
* The servers run in the private subnets are launched and terminated by using an Auto Scaling group, and receive traffic from the load balancer.
* The servers can connect to the internet by using the NAT gateway.

****

**Before we start:**

* **Auto Scaling Group =>** An Auto Scaling Group (ASG) is a collection of Amazon EC2 instances that are treated as a logical grouping for the purposes of automatic scaling and management. It is a key component of Amazon EC2 Auto Scaling, which automatically adjusts the number of EC2 instances in a group based on predefined policies or metrics.
* **Amazon EC2 Auto Scaling helps you ensure that you have the correct number of Amazon EC2 instances available to handle the load for your application. You create collections of EC2 instances, called Auto Scaling groups.** You can specify the minimum number of instances in each Auto Scaling group, and Amazon EC2 Auto Scaling ensures that your group never goes below this size. You can specify the maximum number of instances in each Auto Scaling group, and Amazon EC2 Auto Scaling ensures that your group never goes above this size. If you specify the desired capacity, either when you create the group or at any time thereafter, Amazon EC2 Auto Scaling ensures that your group has these many instances. If you specify scaling policies, then Amazon EC2 Auto Scaling can launch or terminate instances as demand on your application increases or decreases.
* **Load Balancer =>** In Amazon Web Services (AWS), a Load Balancer is a service that distributes incoming application traffic across multiple targets, such as EC2 instances, containers, and IP addresses, in one or more Availability Zones (AZs). This ensures high availability, scalability, and fault tolerance for applications.
* AWS offers four types of Load Balancers:

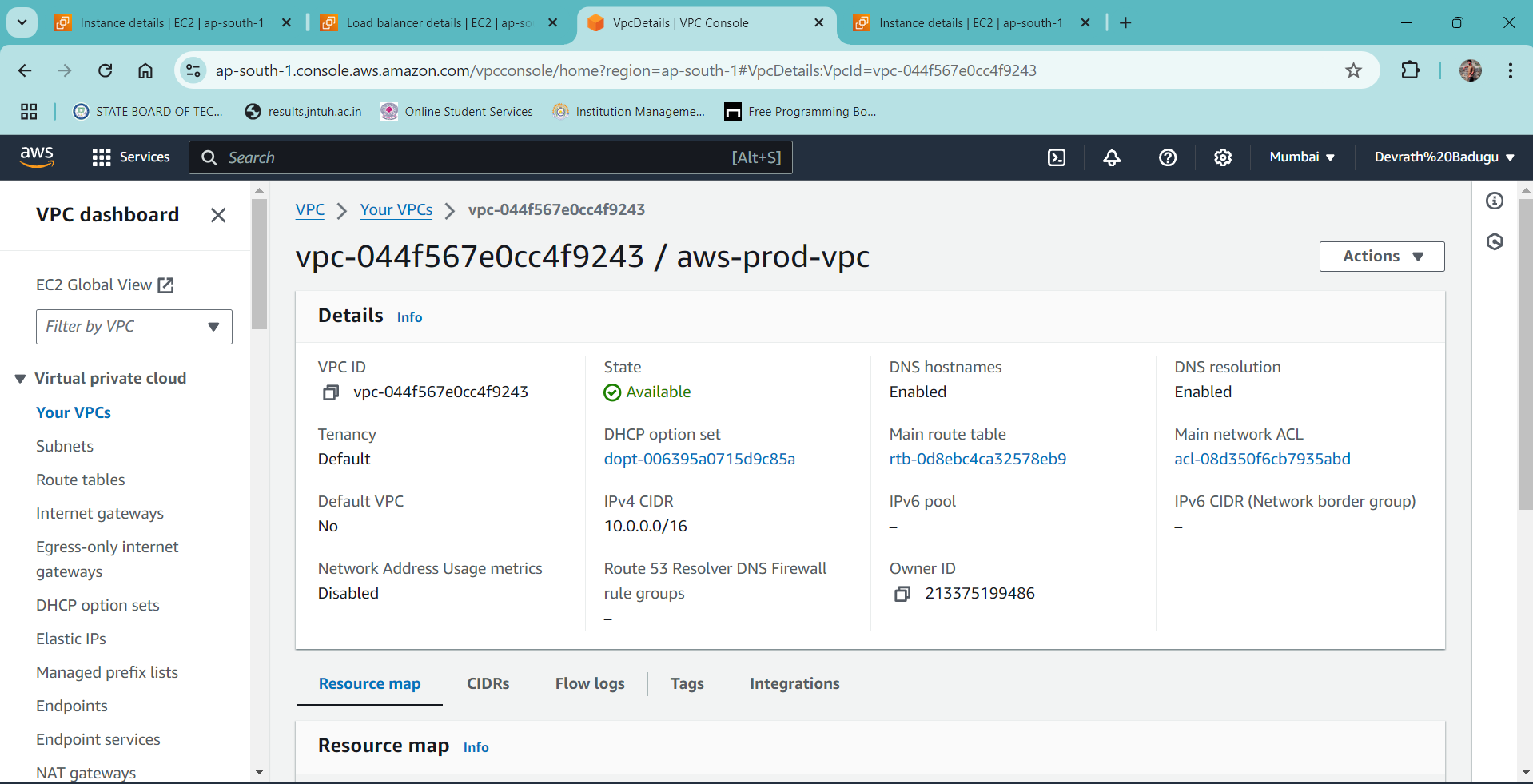
1. **Application Load Balancer (ALB)**: Designed for HTTP/HTTPS traffic, ALB is ideal for web applications and microservices. It provides features like path-based routing, URL rewriting, and support for WebSocket and HTTP/2 protocols.
2. **Network Load Balancer (NLB)**: Optimized for TCP traffic, NLB is suitable for applications that require low-latency, high-throughput, and high-availability. It provides features like session persistence, connection draining, and support for IPv4 and IPv6.
3. **Gateway Load Balancer (GLB)**: A newer type of Load Balancer, GLB is designed for hybrid cloud and multi-cloud environments. It provides features like support for multiple protocols (TCP, UDP, and HTTP), multiple Availability Zones, and integration with AWS Transit Gateway.
4. **Classic Load Balancer (CLB)**: An older type of Load Balancer, CLB is still available but not recommended for new deployments. It’s designed for legacy applications and provides basic load balancing features.

* **Target Group =>** A Target Group in AWS is **a logical grouping of targets, such as EC2 instances, containers, Lambda functions, Application Load Balancers, or Kubernetes Pods, that receive traffic from a single load balancer**. Target Groups enable customers to:
* Configure multiple targets to receive traffic from a single load balancer
* Monitor the health of registered targets and take action when necessary
* Route traffic to specific application endpoints
* In AWS, Target Groups are **used with Application Load Balancers (ALBs), Network Load Balancers (NLBs), and Gateway Load Balancers (GLBs) to distribute traffic to registered targets**. Each load balancer requires at least one listener, which defines the protocol and port combination for incoming requests. The listener then forwards requests to one or more Target Groups, based on the rules defined.
* **Bastion Host or Jump Server =>** A bastion host or jump server is **a server that provides a single point of access from an external network to the resources located in a private network**. In AWS, a bastion host is a server exposed to the internet, which poses a potential security risk for unauthorized access. To mitigate this risk, bastion hosts typically run a minimum amount of services and are commonly used to proxy and log communications, such as SSH sessions.
* In AWS, a bastion host **can be used to manage access to internal or private networks from an external network**. It acts as a secure entry point, allowing authorized users to access internal resources while restricting direct access to those resources from the internet.
* Bastion hosts (also called **“jump servers”**) are often used as a best practice for **accessing privately accessible hosts within a system environment**. For example, your system might include an application host that is not intended to be publicly accessible. To access it for product updates or managing system patches, you typically log in to a bastion host and then access (or “jump to”) the application host from there.

**=======LET’S BEGIN THE IMPLEMENTATION=======**

* **Create a VPC:**
* Use the following procedure to create a VPC with a public subnet and a private subnet in two Availability Zones, and a NAT gateway in each Availability Zone.
* **To create the VPC**

1. Open the Amazon VPC console at <https://console.aws.amazon.com/vpc/>.
2. On the dashboard, choose **Create VPC**.
3. For **Resources to create**, choose **VPC and more**.
4. **Configure the VPC**
   1. For **Name tag auto-generation**, enter a name for the VPC (**aws-prod**).
   2. For **IPv4 CIDR block**, you can keep the default suggestion, or alternatively you can enter the CIDR block required by your application or network.
5. **Configure the subnets**
   1. For **Number of Availability Zones**, choose **2**, so that you can launch instances in multiple Availability Zones to improve resiliency.
   2. For **Number of public subnets**, choose **2**.
   3. For **Number of private subnets**, choose **2**.
   4. You can keep the default CIDR block for the public subnet, or alternatively you can expand **Customize subnet CIDR blocks** and enter a CIDR block. For more information, see [Subnet CIDR blocks](https://docs.aws.amazon.com/vpc/latest/userguide/subnet-sizing.html).
6. For **NAT gateways**, choose **1 per AZ** to improve resiliency.
7. For **VPC endpoints**, if your instances must access an S3 bucket, keep the **S3 Gateway** default. Otherwise, instances in your private subnet can't access Amazon S3. There is no cost for this option, so you can keep the default if you might use an S3 bucket in the future. **=>**If you choose **None**, you can always add a gateway VPC endpoint later on. **<=**
8. For **DNS options**, clear **Enable DNS hostnames**.
9. Choose **Create VPC**.



* **Create a launch template for an Auto Scaling group:**

Before you can create an Auto Scaling group using a launch template, you must create a launch template that contains the configuration information to launch an instance, including the ID of the Amazon Machine Image (AMI).

To create new launch templates, use the following procedures.

* **Create your launch template (console)**

The following steps describe how to configure a basic launch template:

* Specify the Amazon Machine Image (AMI) from which to launch the instances.
* Choose an instance type that is compatible with the AMI that you specify.
* Specify the key pair to use when connecting to instances, for example, using SSH.
* Add one or more security groups to allow network access to the instances.
* Specify whether to attach additional volumes to each instance.
* Add custom tags (key-value pairs) to the instances and volumes.

**To create a launch template**

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>.
2. On the navigation pane, under **Instances**, choose **Launch Templates**.
3. Choose **Create launch template**. Enter a name (**aws-prod**) and provide a description (**proof of concept for app deploy in aws private subnet**) for the initial version of the launch template.
4. Under **Launch template contents**, fill out each required field and any optional fields as needed.
   1. **Application and OS Images (Amazon Machine Image)**: (**Ubuntu**) Choose the ID of the AMI for your instances. You can search through all available AMIs, or select an AMI from the **Recents** or **Quick Start** list.
   2. For **Instance, type:** t2.micro [free tier eligible]
   3. **Key pair (login)**: For **Key pair name**, choose an existing key pair, or choose **Create new key pair** to create a new one.
   4. **Network settings**: For **Firewall (security groups)**, use one or more security groups, or keep this blank and configure one or more security groups as part of the network interface.
   5. Do one of the following:
      * Change the default network interface settings. For example, you can enable or disable the public IPv4 addressing feature, which overrides the auto-assign public IPv4 addresses setting on the subnet. For more information, see [Change the default network interface settings (console)](https://docs.aws.amazon.com/autoscaling/ec2/userguide/create-launch-template.html#change-network-interface).
      * Skip this step to keep the default network interface settings.
   6. Do one of the following:
      * Modify the storage configuration. For more information, see [Modify the storage configuration (console)](https://docs.aws.amazon.com/autoscaling/ec2/userguide/create-launch-template.html#modify-storage-configuration).
      * Skip this step to keep the default storage configuration.
   7. For **Resource tags**, specify tags by providing key and value combinations. If you specify instance tags in your launch template and then you choose to propagate your Auto Scaling group's tags to its instances, all the tags are merged. If the same tag key is specified for a tag in your launch template and a tag in your Auto Scaling group, then the tag value from the group takes precedence.
5. When you are ready to create the launch template, choose **Create launch template**.
6. To create an Auto Scaling group, choose **Create Auto Scaling group** from the confirmation page.

* **Create an Auto Scaling group using a launch template:**
* When you create an Auto Scaling group, you must specify the necessary information to configure the Amazon EC2 instances, the Availability Zones and VPC subnets for the instances, the desired capacity, and the minimum and maximum capacity limits.
* To configure Amazon EC2 instances that are launched by your Auto Scaling group, you can specify a launch template or a launch configuration. The following procedure demonstrates how to create an Auto Scaling group using a launch template.
* **Prerequisites**
* You must have created a launch template.

**To create an Auto Scaling group using a launch template (console)**

1. Open the Amazon EC2 console at <https://console.aws.amazon.com/ec2/>, and choose **Auto Scaling Groups** from the navigation pane.
2. On the navigation bar at the top of the screen, choose the same AWS Region that you used when you created the launch template.
3. Choose **Create an Auto Scaling group**.
4. On the **Choose launch template or configuration** page, do the following:
   1. For **Auto Scaling group name**, enter a name (**aws-prod**) for your Auto Scaling group.
   2. For **Launch template**, choose an existing launch template.
   3. For **Launch template version**, choose whether the Auto Scaling group uses the default, the latest, or a specific version of the launch template when scaling out.
   4. Verify that your launch template supports all of the options that you are planning to use, and then choose **Next**.
5. On the **Choose instance launch options** page, if you're not using multiple instance types, you can skip the **Instance type requirements** section to use the EC2 instance type that is specified in the launch template.
6. Under **Network**, for **VPC**, choose a VPC. The Auto Scaling group must be created in the same VPC as the security group you specified in your launch template (**aws-prod**).
7. For **Availability Zones and subnets**, choose one or more subnets in the specified VPC. Use **private subnets 1 and 2** in multiple Availability Zones for high availability.
8. If you created a launch template with an instance type specified, then you can continue to the next step to create an Auto Scaling group that uses the instance type in the launch template.
9. Choose **Next** to continue to the next step.

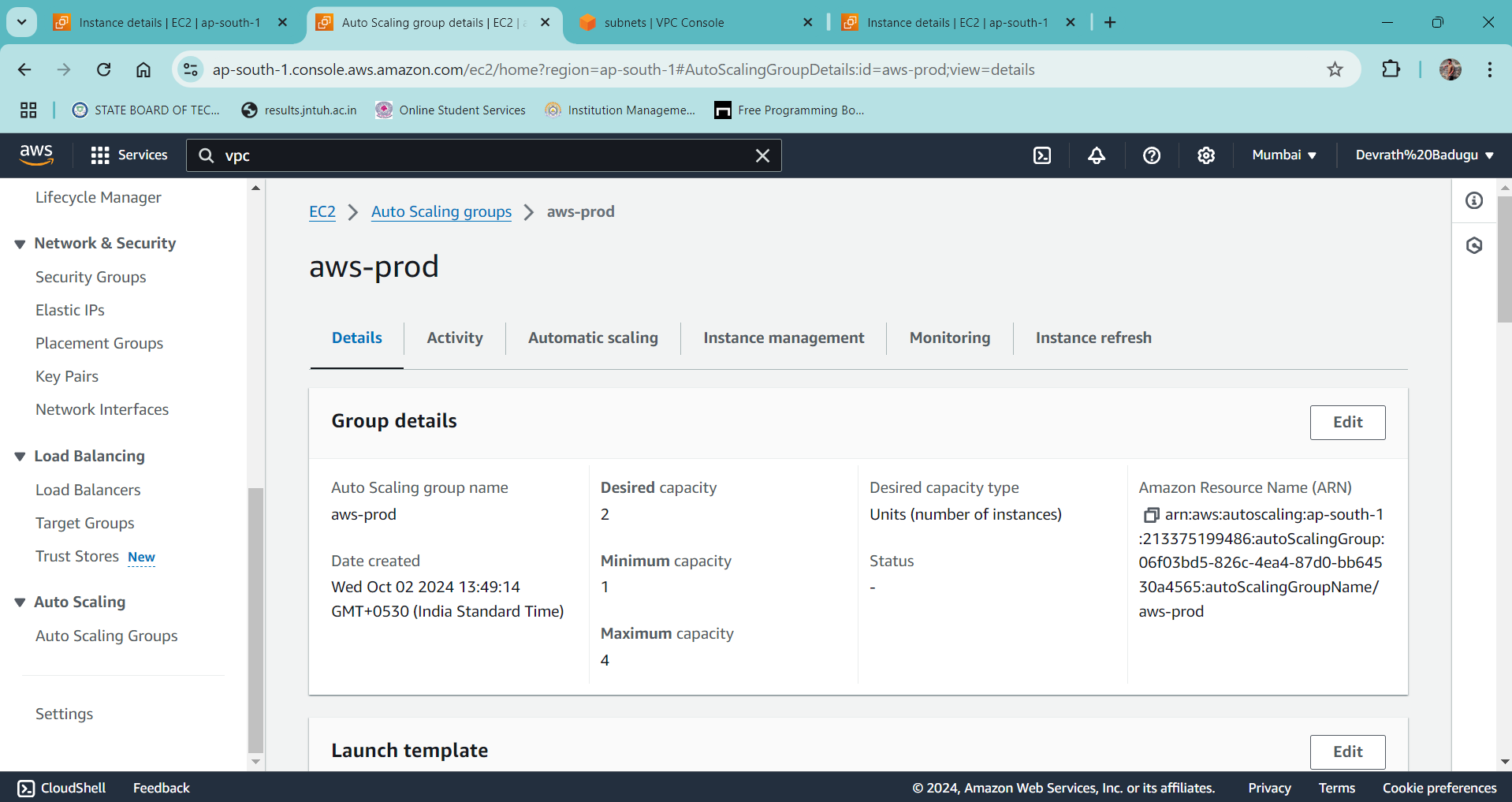
Or, you can accept the rest of the defaults, and choose **Skip to review**.

1. (Optional) On the **Configure advanced options** page, configure the following options, and then choose **Next**:

**Load balancing:** No Load Balancer.

* 1. (Optional) For **Health checks**, **Additional health check types**, select **Turn on Amazon EBS health checks**.
  2. (Optional) For **Health check grace period**, enter the amount of time, in seconds. This amount of time is how long Amazon EC2 Auto Scaling needs to wait before checking the health status of an instance after it enters the In-Service state.
  3. Under **Additional settings**, **Monitoring**, choose whether to enable CloudWatch group metrics collection.
  4. For **Enable default instance warmup**, select this option and choose the warmup time for your application. If you are creating an Auto Scaling group that has a scaling policy, the default instance warmup feature improves the Amazon CloudWatch metrics used for dynamic scaling

1. (Optional) On the **Configure group size and scaling policies** page, configure the following options, and then choose **Next**:
   1. Under **Group size**, for **Desired capacity**, enter the initial number of instances to launch.
   2. In the **Scaling** section, under **Scaling limits**, if your new value 1for **Desired capacity** **= 2** is greater than **Min desired capacity = 1** and **Max desired capacity** **= 4**, the **Max desired capacity** is automatically increased to the new desired capacity value. You can change these limits as needed.
   3. For **Automatic scaling**, choose whether you want to create a target tracking scaling policy. You can also create this policy after you create your Auto Scaling group.
2. (Optional) To receive notifications, for **Add notification**, configure the notification, and then choose **Next**.
3. (Optional) To add tags, choose **Add tag**, provide a tag key and value for each tag, and then choose **Next**.
4. On the **Review** page, choose **Create Auto Scaling group**.

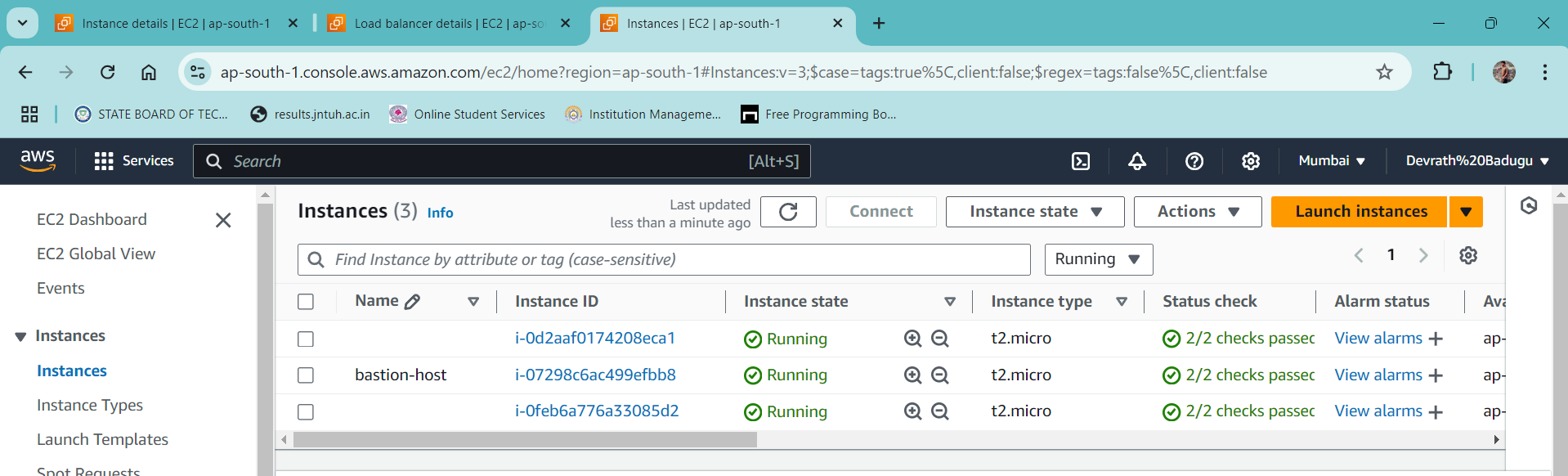


* **CHECK THE EC2 INSTANCES**
* **Create a Bastion host in ec2 instance**
* Click on Launch Instances
* Name: bastion-host
* AMI: Ubuntu
* Instances type: t2.micro [free eligible]
* Keypair: deva-keypair.pem
* Network settings: create security group

 Allow SSH traffic from: Anywhere [0.0.0.0/0]

**VPC required:** aws-prod (same)

* Auto assign public IP: **Enable LAUNCH INTANCE**



**===============TERMINAL===============**

* Bastion-host public IPv4 address: 3.110.221.93
* First instance private IPv4 address: 10.0.132.110
* Second instance private IPv4 address: 10.0.154.117

scp -i deva-keypair.pem deva-keypair.pem [ubuntu@ec2-3-110-221-93.ap-south-1.compute.amazonaws.com:/home/ubuntu/](mailto:ubuntu@ec2-3-110-221-93.ap-south-1.compute.amazonaws.com:/home/ubuntu/)

or

scp -i C:/Users/DEVRATH/Downloads/deva-keypair.pem C:/Users/DEVRATH/Downloads/deva-keypair.pem [ubuntu@ 3.110.221.93:/home/ubuntu](mailto:ubuntu@3.110.158.240:/home/ubuntu)

**yes**

ssh -i deva-keypair.pem [ubuntu@3.110.221.93](mailto:ubuntu@3.110.221.93)

* ls
* deva-keypair.pem

**first instance**

* ssh -i deva-keypair.pem [ubuntu@10.0.132.110](mailto:ubuntu@10.0.132.110) **yes**
* vim index.html

i [INSERT]

<!DOCTYPE html>

<html>

<body>

<h1>My First AWS PROJECT demonstrate apps in private subnet</h1>

<p>Congratulations on first instance!!!</p>

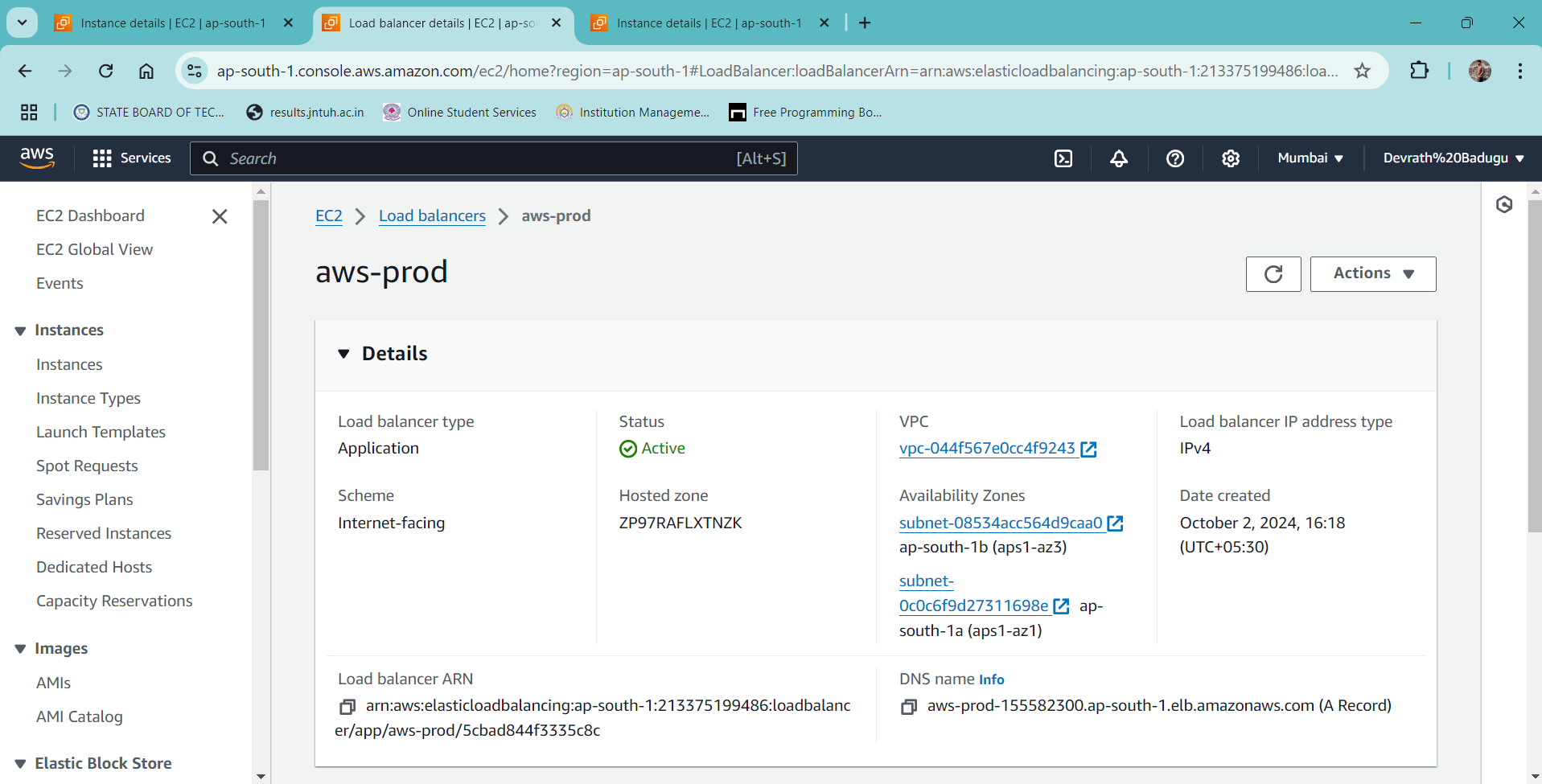
</body>

</html>

:wq! => To SAVE the file

* python3 -m http.server 8000
* **Create Load balancer and attach these instances as target group**
* Open Load balancer in aws console
* Click on create a **Load Balancer**
* Load balancer type: **ALB**
* Name: aws-prod
* Scheme: **Internet-facing**
* IP address type: **IPv4**
* Network mapping:
  + **VPC: aws-prod-vpc**
  + Mappings: Select both the availability zones and it should be with the PUBLIC subnets.
* Security group: aws-prod
* Listeners and routing:
  + Protocol: HTTP
  + Port: 80
  + Default actions: create target group (add target group which created now)
* Create a target group Create load balancer
  + Target type: **Instances**
  + Name: **aws-prod**
  + Protocol: **HTTP**
  + Port: **8000**
  + VPC: **aws-prod-vpc**
  + Health check: **default value** Next
  + Register targets:
    - Available instances:
    -  **first instance**
    - **Bastion host instance**
    -  **Second instance**
      * Port: 8000
      * Include as pending below

Create target group



* Open the Load balancer which is just created
  + Scroll down, now in **Listeners there is an error.** To resolve this error, click on **security** and click on edit inbound rules and **add rule**
  + **HTTP: Anywhere IPv4 [0.0.0.0/0]**
  + Save the **Rules**

Now the final part!

* Copy the DNS name in load balancer
* Now open it in new tap of the chrome and you can see….
* You have demonstrated apps in private subnet 1 and 2!

