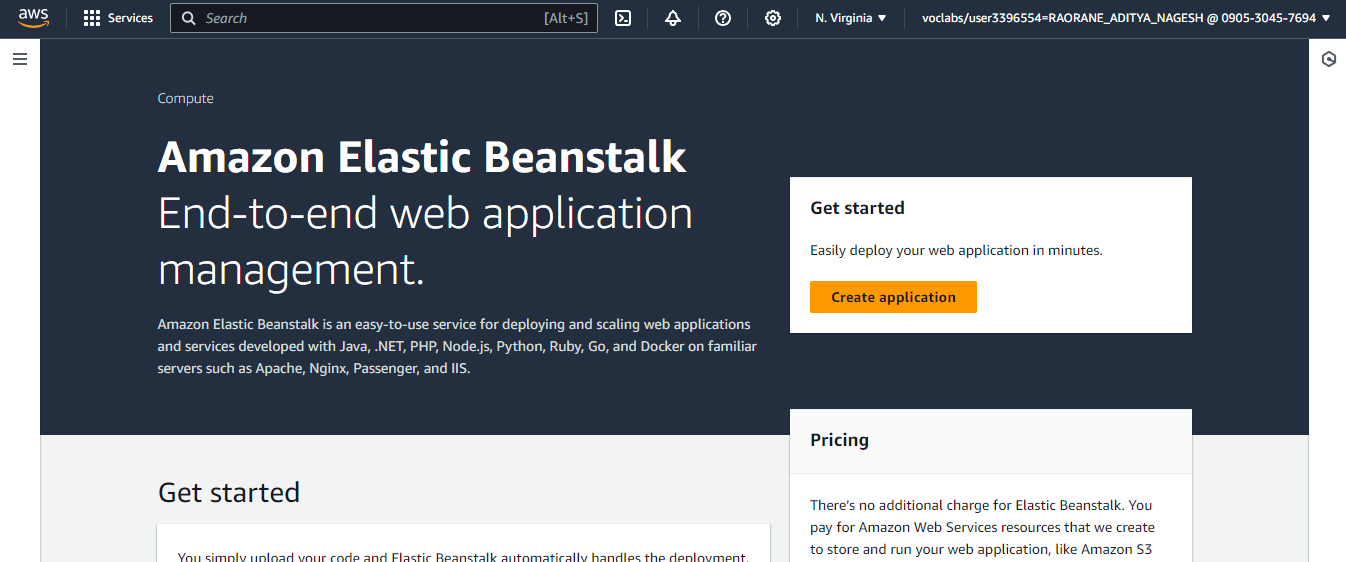
**2]Application Deployment with AWS Elastic Beanstalk and AWS CodePipeline**

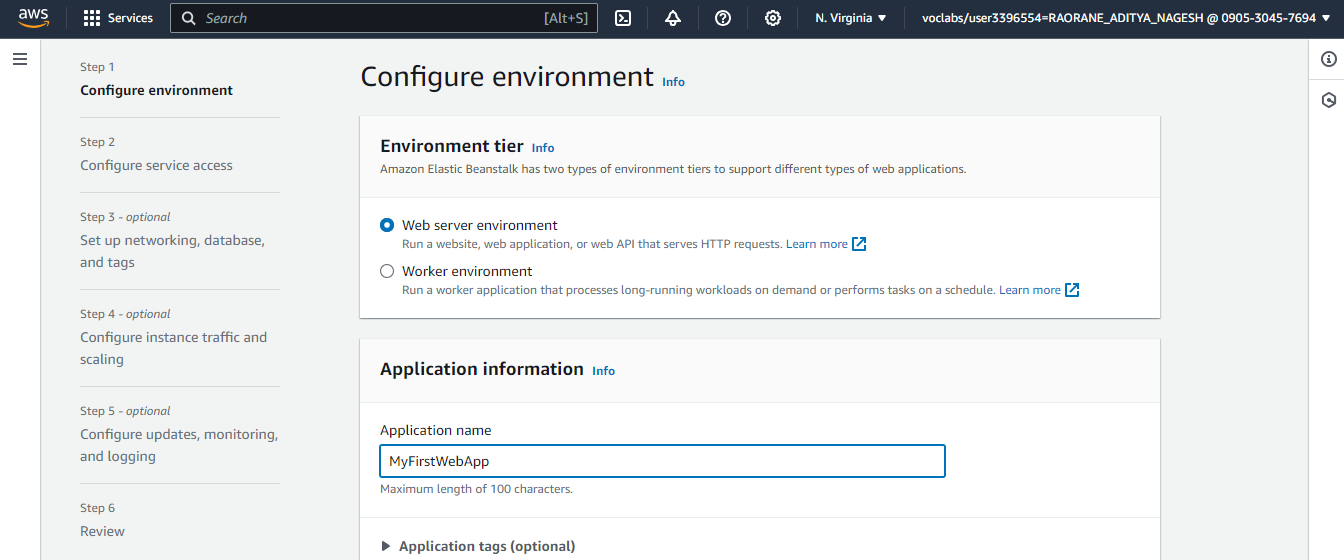
Step 1:To open AWS Elastic Beanstalk, log in to the AWS Management Console, navigate to Elastic Beanstalk under "**Compute**," and click “**Create Application**” to start a new project  


2]After clicking "Create Application" in Elastic Beanstalk, you'll choose between two environment tiers:

**Web Server Environment**: Handles HTTP requests for web applications with built-in load balancing, auto-scaling, and a pre-configured web server.

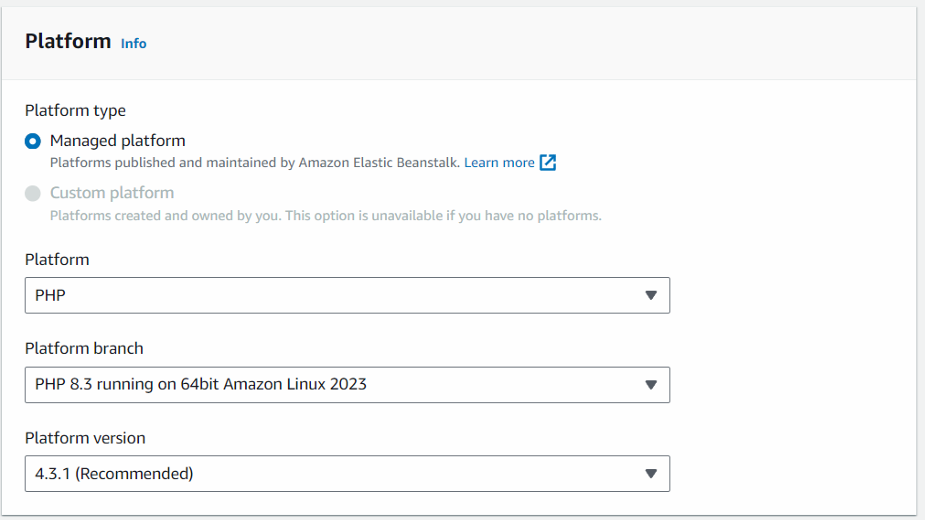
**Worker Environment**: Manages background tasks asynchronously, processing jobs from an SQS queue without direct user interaction.

Click on “Web server Environment”.

  
  
Step 3:Under Platform, you can choose between Managed Platform and Custom Platform. Selected Managed Platform and configured it with PHP for ease of use and automatic updates.

**Managed Platform**: AWS manages and updates the environment, providing pre-configured platforms like PHP, making deployment easier and maintenance automated.

**Custom Platform**: Allows you to create and manage your own platform, giving full control over the environment configuration and software stack.



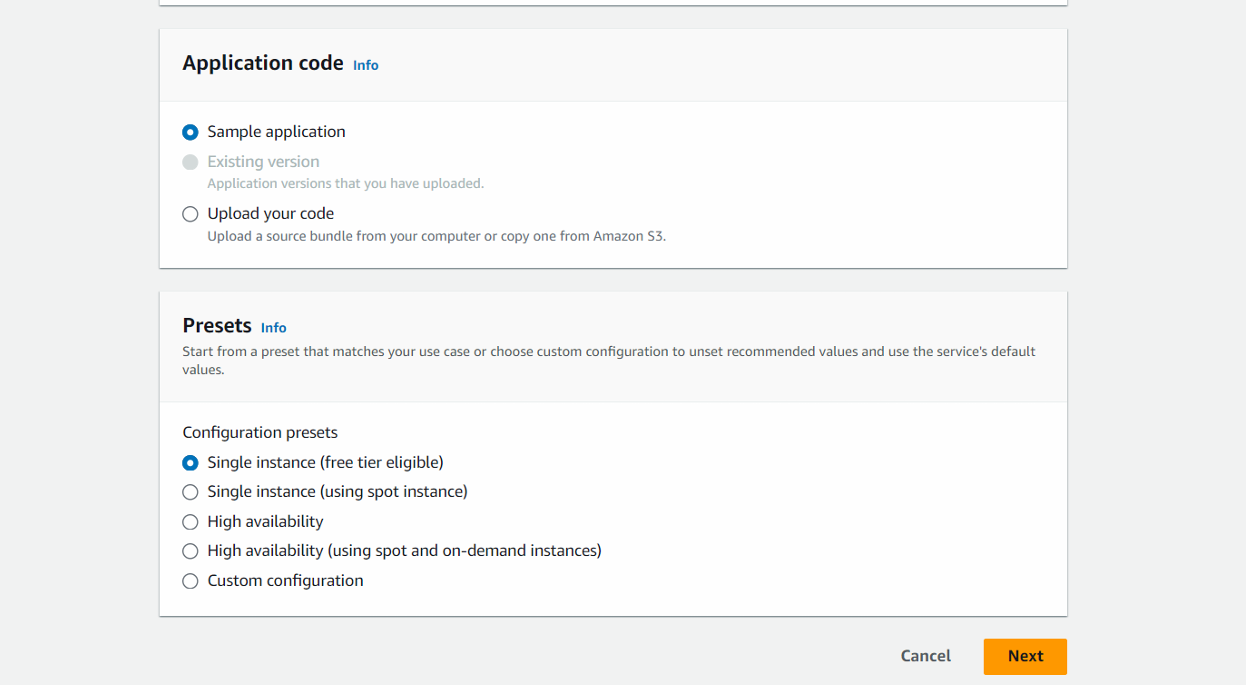
Step 4:**Presets** offer different deployment configurations to match your needs.

**Single Instance (Free Tier Eligible)** runs your application on one EC2 instance, suitable for development or low-traffic scenarios and eligible for the AWS Free Tier.

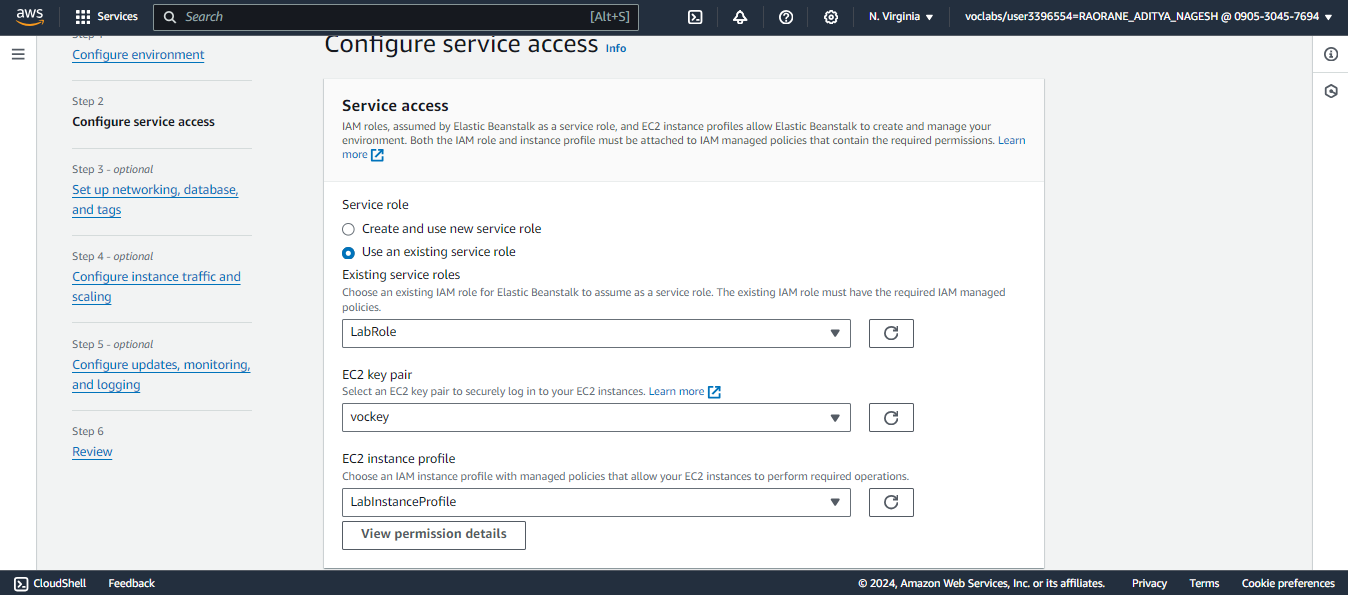
**Single Instance** deploys on one EC2 instance but without Free Tier limitations, ideal for small-scale projects.

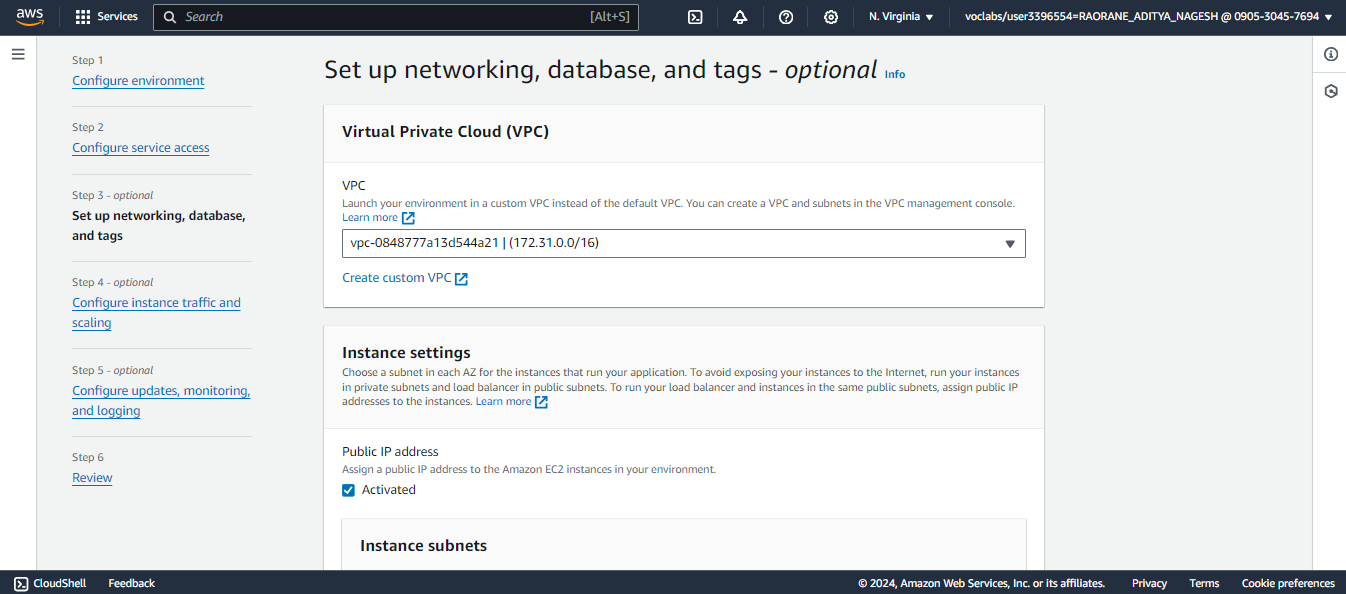
**High Availability** uses multiple instances and a load balancer to ensure fault tolerance and reliability for production environments.

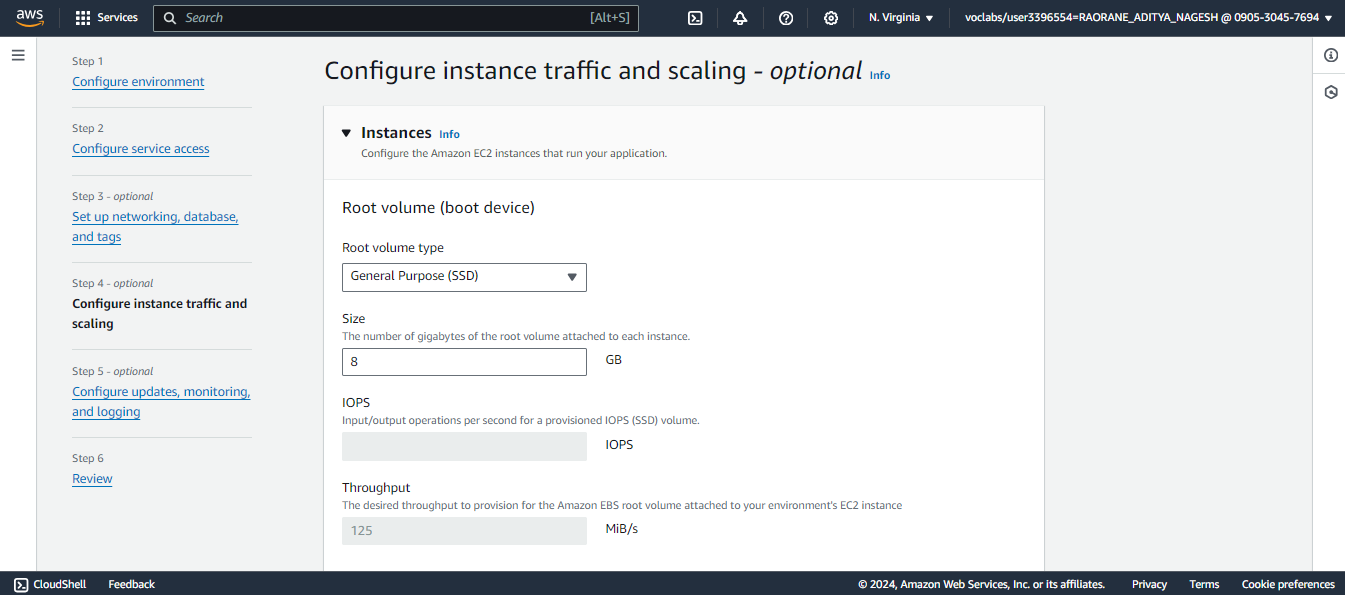
**Using Spot Instances** leverages unused EC2 capacity at reduced costs, suitable for applications that can handle interruptions and require cost savings.



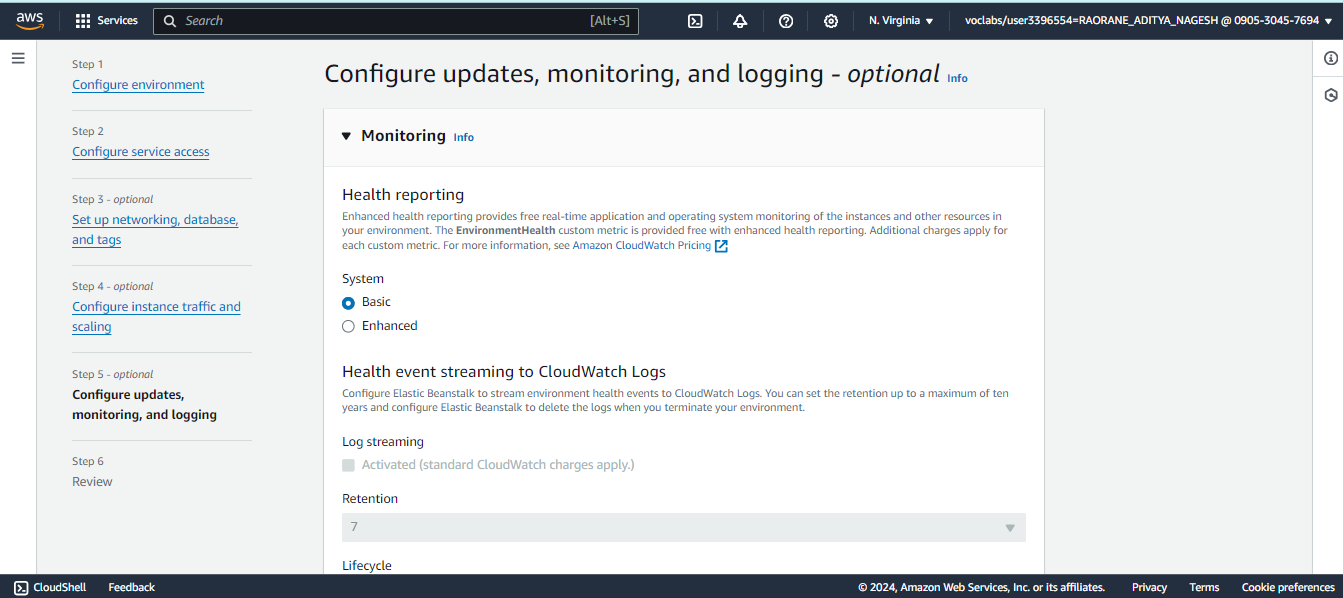
Step 5:Define your application's service configuration, including settings for load balancing, auto-scaling, and health checks.Each EC2 instance can be accessed using its corresponding key pair.Choosing an EC2 key pair as a "vockey" (or a "virtual key") is typically done to enable secure access to your EC2 instances; allowing you to securely connect to your instances using SSH (Linux), ensuring that only authorized users can access the server.

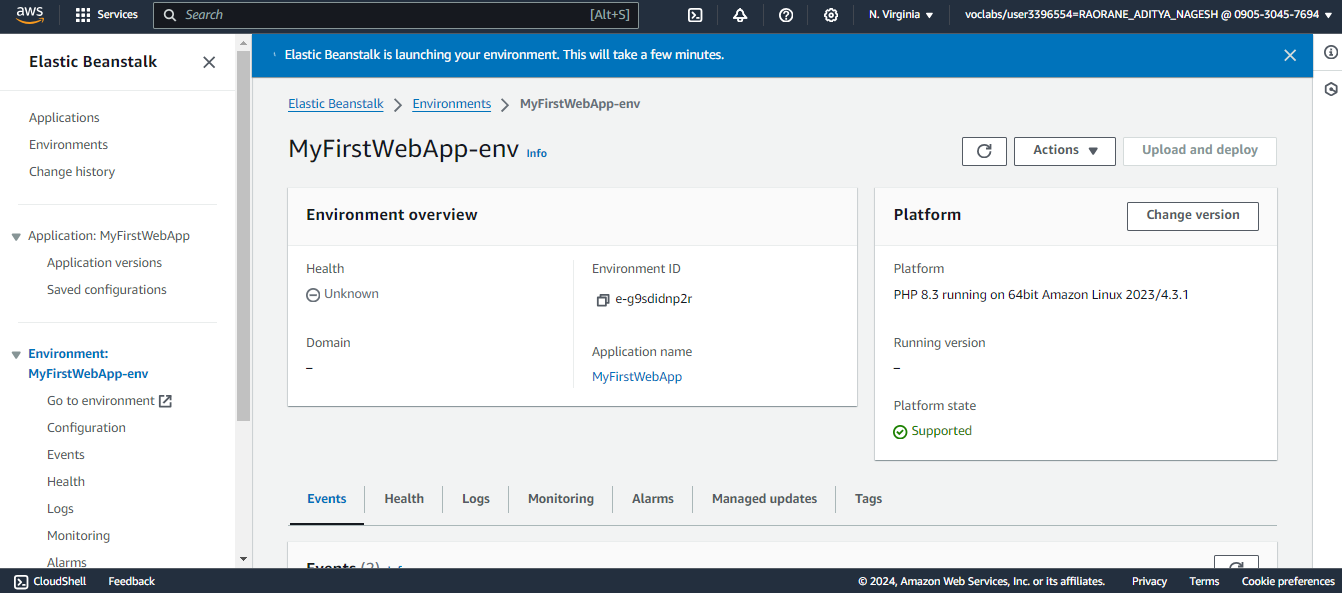


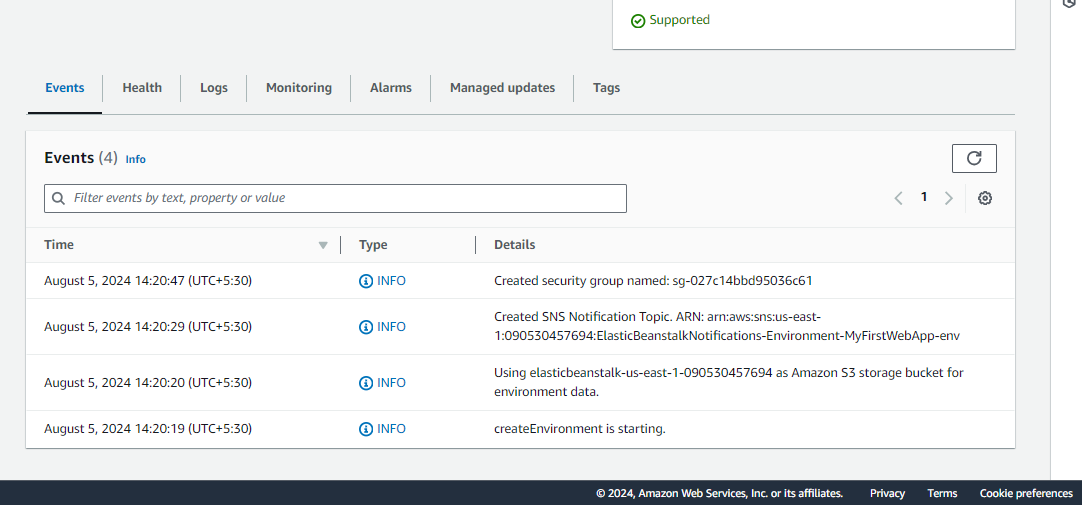
Step 6: Configure VPC, subnets, and security groups for network access. Set up your environment’s database if needed and apply tags for resource management.  


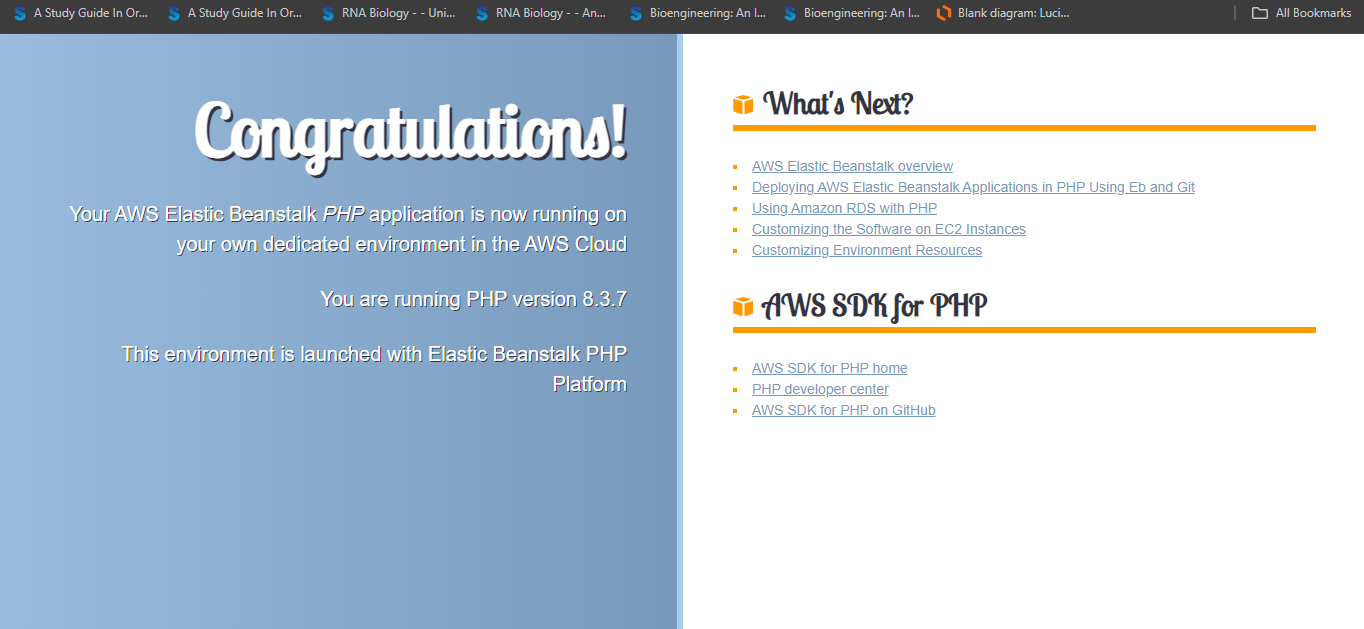
Step 7:Adjust load balancer settings to manage incoming traffic and configure auto-scaling rules to handle changes in traffic volume.The **root volume type** is the primary storage for the OS, and selecting **General Purpose (SSD)** offers balanced performance and durability for most applications.

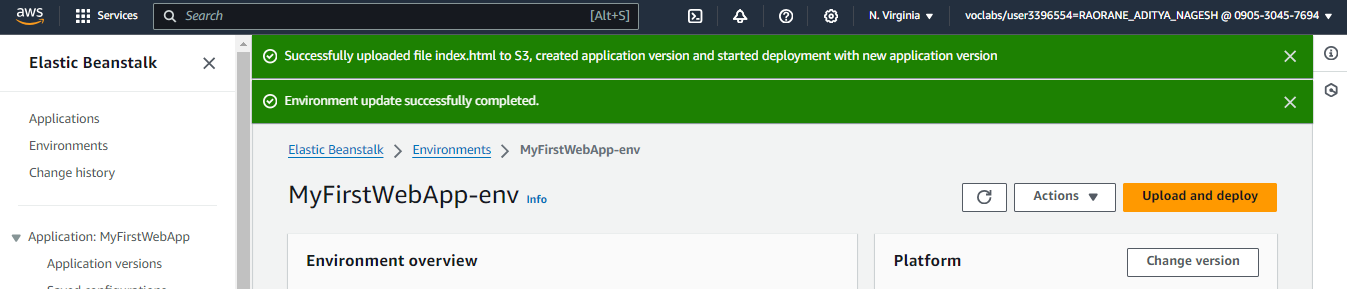
Step 8:Set up deployment policies for updates, enable monitoring for performance metrics, and configure logging to capture and analyze application logs.



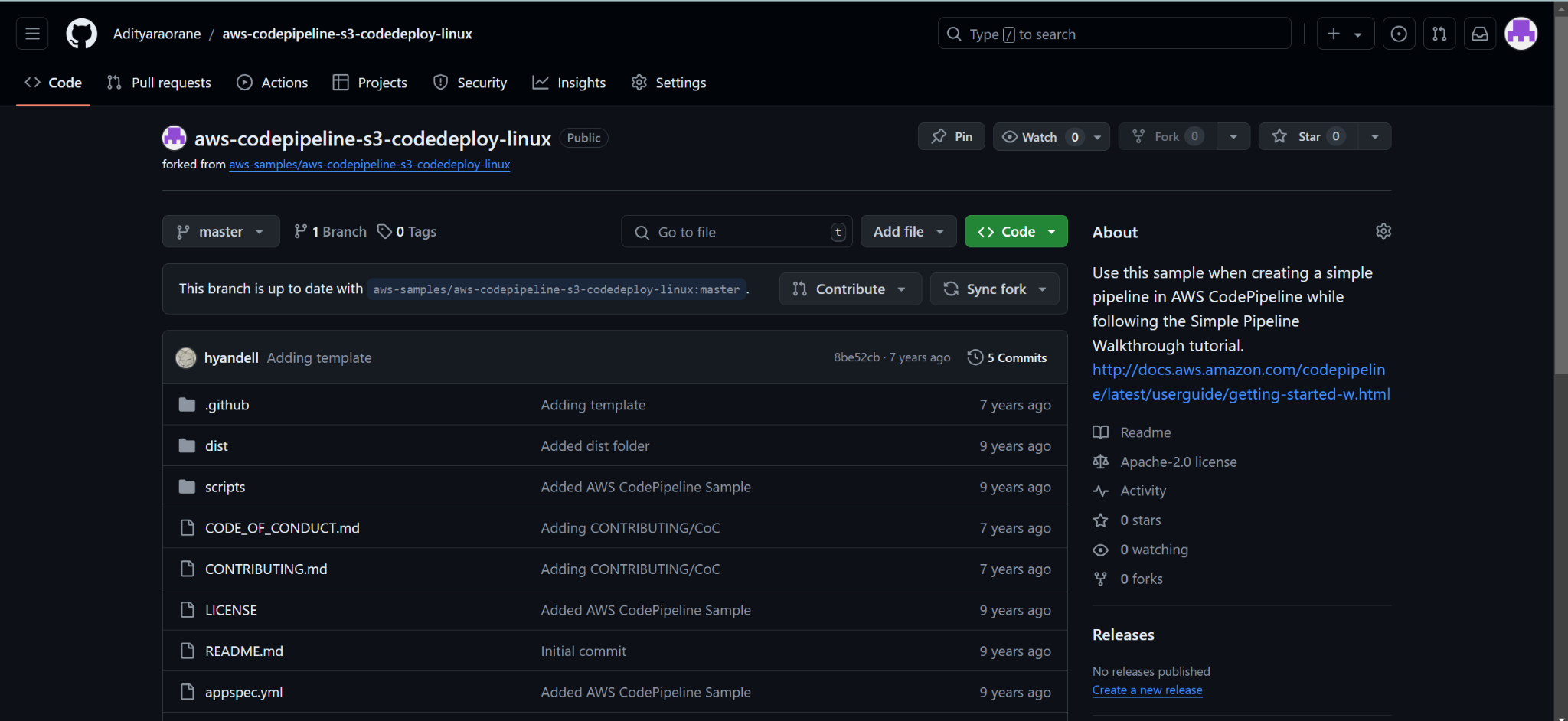
Step 9:This creates our environment .We can now check our sample application uploaded if our environment is successfully deployed.  


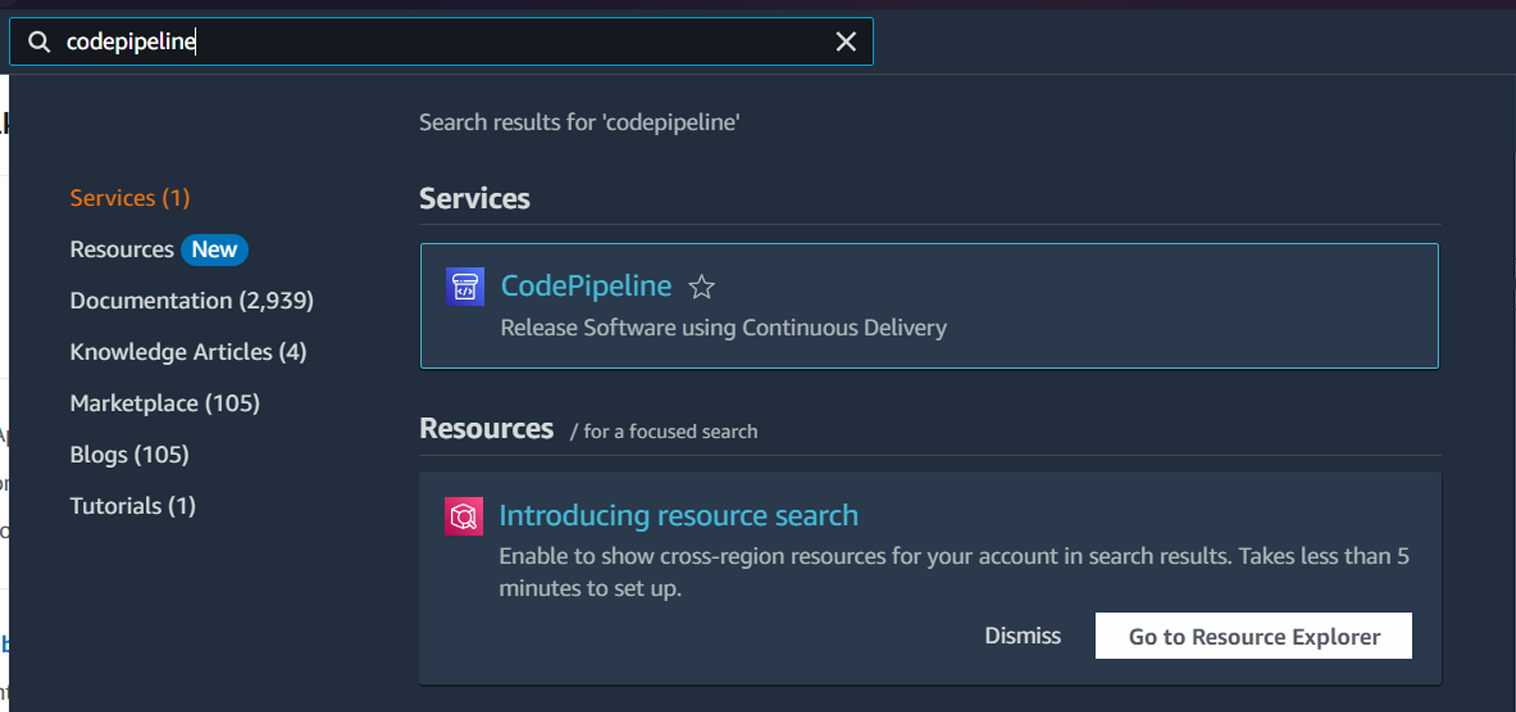


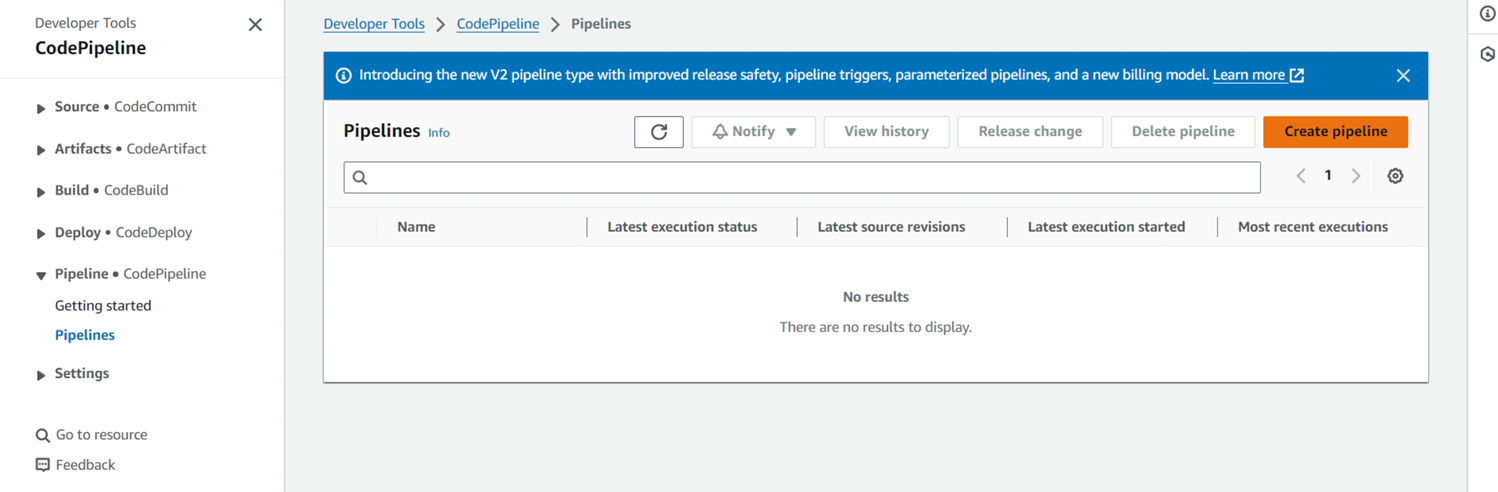




Step 10: Go to the github link below. This is a github with a sample code for deploying a file on AWSCodePipeline. Fork this repository into your personal github. <https://github.com/aws-samples/aws-codepipeline-s3-codedeploy-linux>



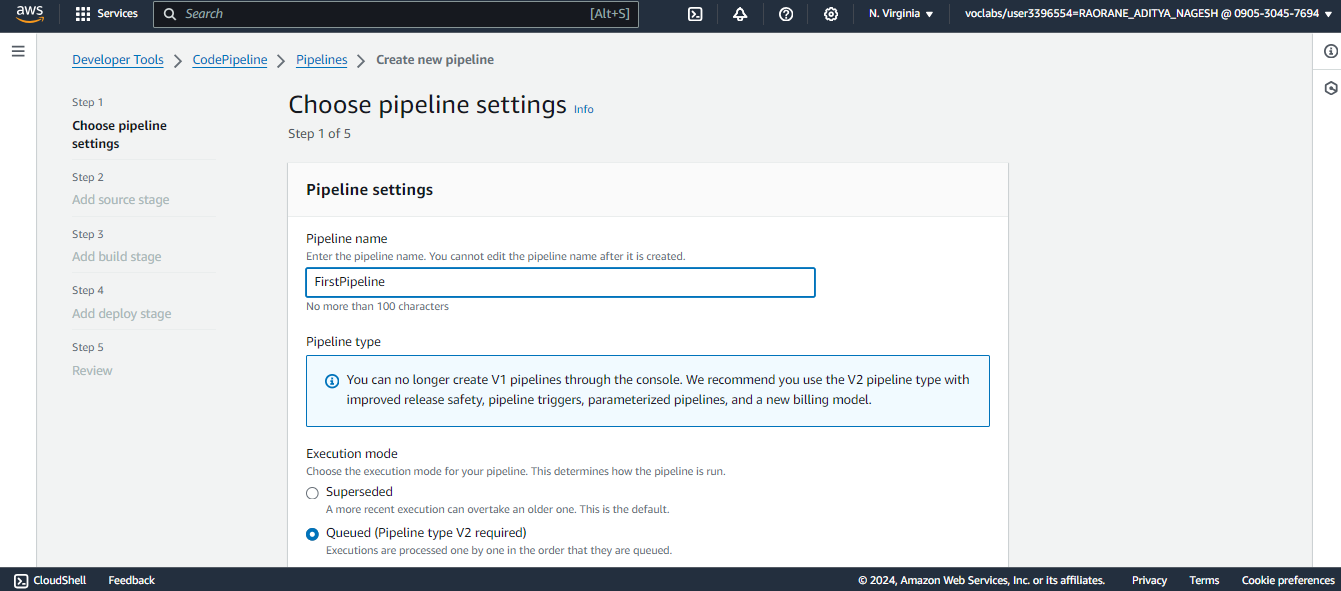
Step 11:Search **CodePipeline** in the “services” tab and click on it.

Step 12: After navigation to CodePipeline under “Developer Tools,” and click “Create pipeline” to start setting up your CI/CD.

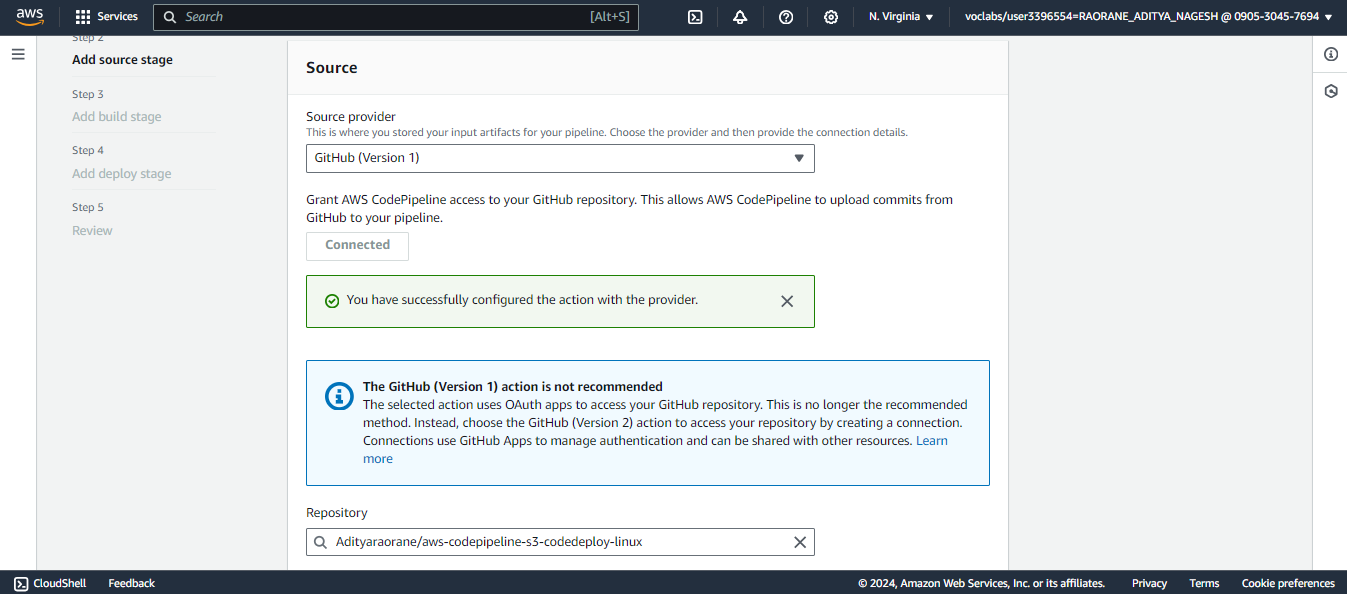
Step 13: Give a name to your Pipeline. A new service role would be created with the name of the pipeline. The **pipeline type** options are:

* **Suspended**: Pauses the pipeline, stopping all execution of actions and deployments until resumed.
* **Queued**: Holds pipeline executions in a queue, executing them sequentially. This is useful for managing limited resources and preventing overload.
* **Parallel**: Runs multiple pipeline executions concurrently, allowing for faster processing but potentially higher resource usage.

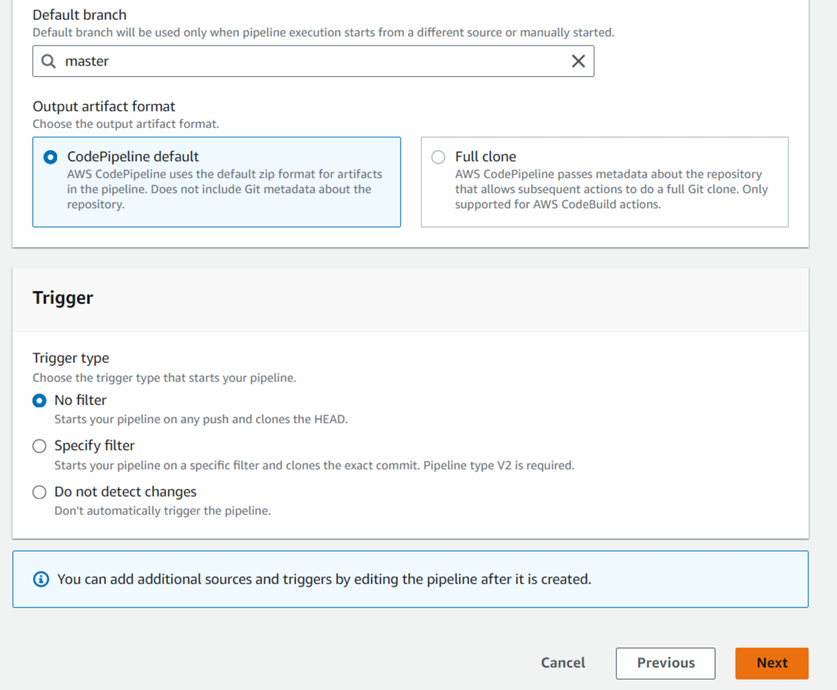
Selecting **Queued** helps manage resource allocation by ensuring that pipelines run one at a time, avoiding conflicts and optimizing resource usage.Finally, click on “**Next**” at the bottom.



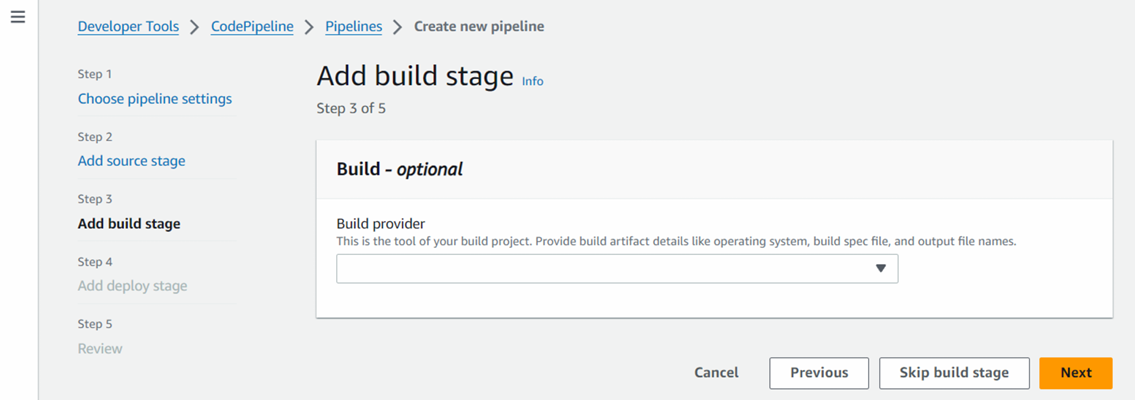
Step 14: In the Source stage of AWS CodePipeline, select GitHub(Version 1) as your source provider, then connect your GitHub account and choose the repository and branch you want to use for the pipeline(by default the branch is **master**). Then click on “**Next**”.



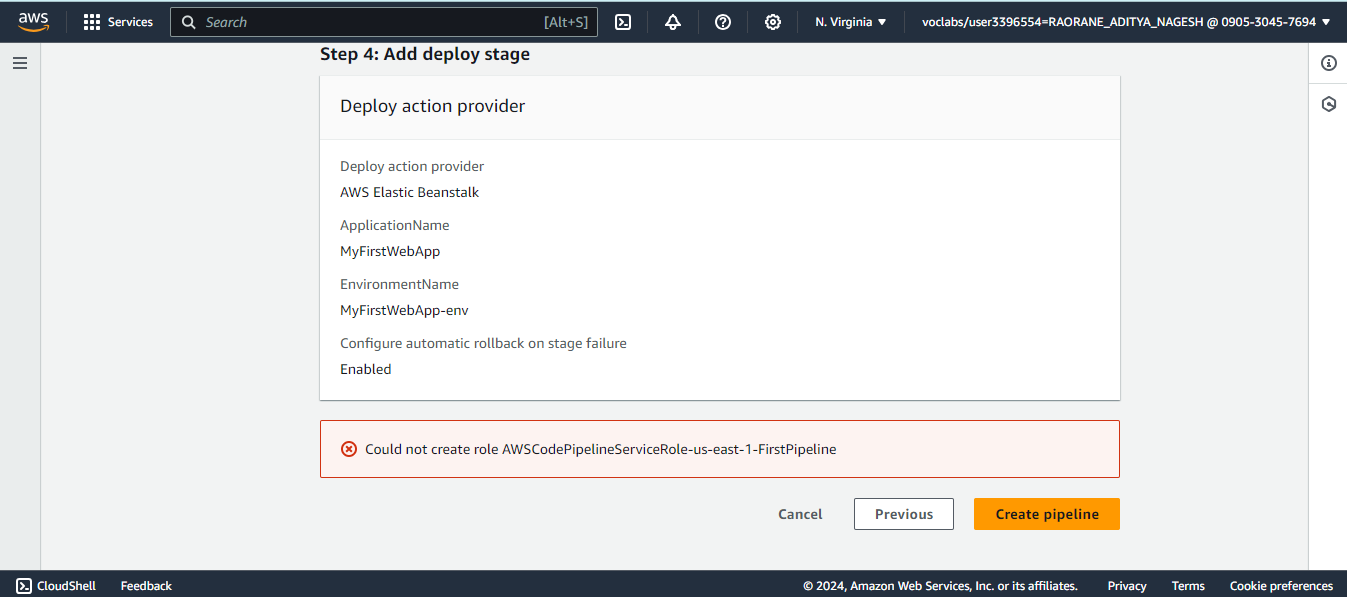
Step 15: Set the Trigger type as no filter. This would allow it to the website to update as soon as some change is made in the github.

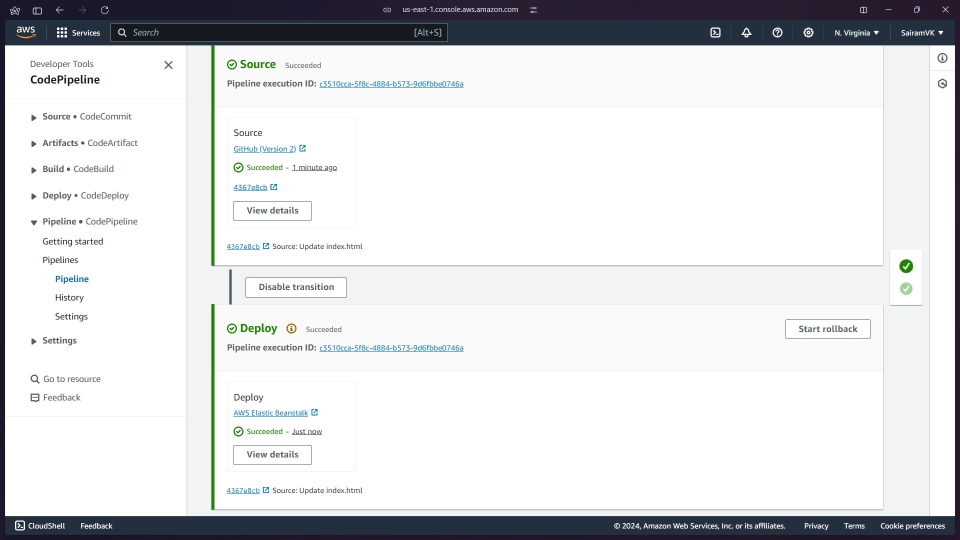


Step 16: Skip the build stage and go to Deploy. Select the deploy provider as AWS Elastic Beanstalk and Input Artifact as SourceArtifact. The application name would be the name of your Elastic Beanstalk. Then click on next.

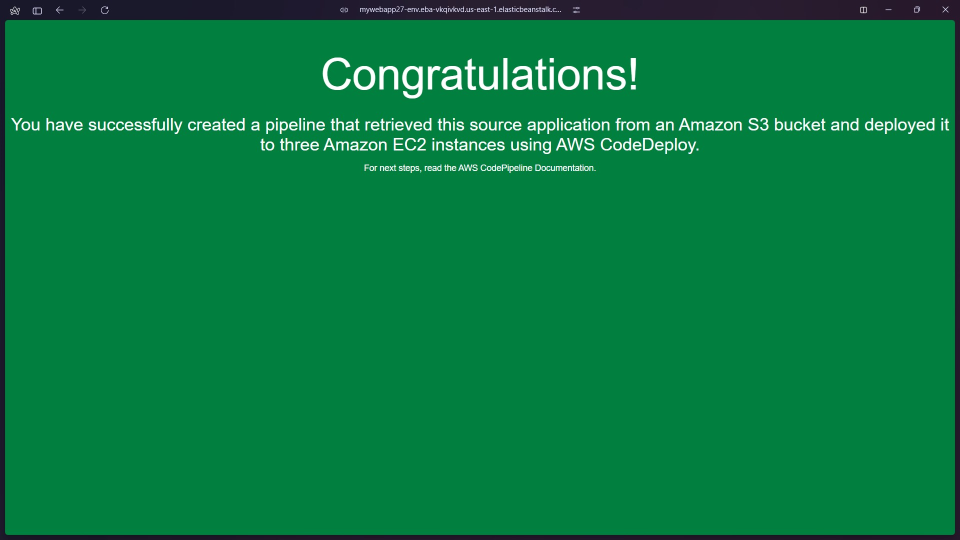


Step 17:The error occurred during the deployment stage because AWS Academy accounts often have **restricted IAM permissions**, preventing the creation of roles needed for the pipeline. This limitation can cause issues when configuring pipelines that require specific roles for execution. To resolve the error, deploy the pipeline using your personal AWS account, which typically has broader permissions and can create the necessary roles for successful deployment.



11]Once the source and deploy stages are successfully completed in AWS CodePipeline, you will see a message indicating "Review" to verify and confirm the pipeline setup before finalizing the deployment.  


12]This will successfully show the sample website hosted.



**Note**:We can make some changes to the index.html file in the github.Once the changes are committed, when the website is refreshed, the changes can be seen.