Blue-Green Deployment on AWS

Overview:

Blue-Green Deployment is a widely-used strategy for ensuring smooth application updates with zero downtime. It involves two environments: **Blue** (current production) and **Green** (new version). After testing the updates in the Green environment, traffic is shifted from Blue to Green, ensuring no interruption to the application's availability.

AWS Services Used for Blue-Green Deployment:

- **CodeCommit or GitHub**: These are version control services where the code is stored.
- **CodeDeploy**: A service that automates the deployment of your applications to services like EC2 or Lambda.
- **CodePipeline**: Used to automate the entire CI/CD pipeline, from code changes to deployment.
- **Elastic Load Balancer (ELB)**: Distributes incoming traffic across multiple targets (like EC2 instances), ensuring your app stays available.
- **Auto Scaling Group (ASG)**: Automatically adjusts the number of EC2 instances to handle the load efficiently.
- **IAM**: Manages access and permissions for your AWS resources.

Additional Setup for Blue-Green Deployment:

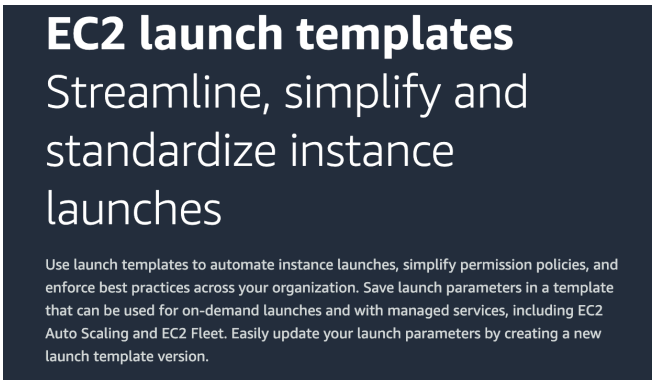
- You will also use **Auto Scaling Groups**, which help maintain the right number of EC2 instances based on your application's needs.
- Templates will be used to automate the launch of EC2 instances.
- **Load Balancers** will help direct traffic to the correct set of instances (Blue or Green).

Before we go deeper into Blue-Green deployment, let's quickly know a couple of concepts:

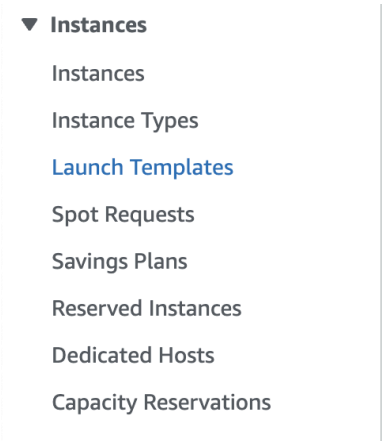
- **Load Balancer**: A Load Balancer distributes network traffic across multiple instances to ensure no single server gets overwhelmed. It helps keep your application available and responsive.
- **Auto Scaling Group**: An Auto Scaling Group helps you automatically launch or terminate EC2 instances based on the traffic your application receives. This helps you save costs and ensures you always have enough resources available.
- In the real world We usually launch EC2 instances manually by going through several steps, such as selecting an **Amazon Machine Image (AMI)**, instance type, network settings, security group, etc. While this method works, it can be repetitive.
- Means when you want to launch an EC2 instance (a virtual server in AWS), you normally fill in all the details like the operating system, instance type, and security

settings manually every time. However, if you plan to launch the same kind of instance multiple times, doing this process again and again can be time-consuming.

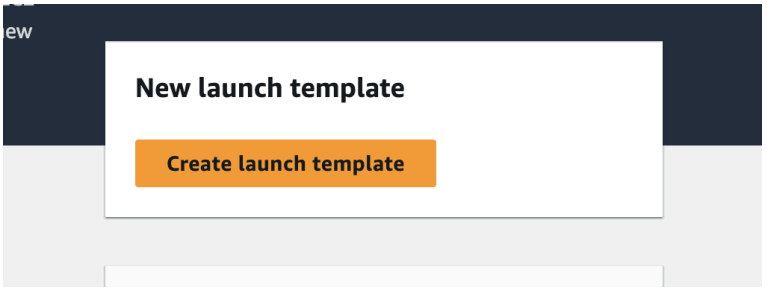
- To avoid this repetition, we can create a **launch template** or use **Auto Scaling Groups** to automatically handle the process of launching instances based on predefined conditions.



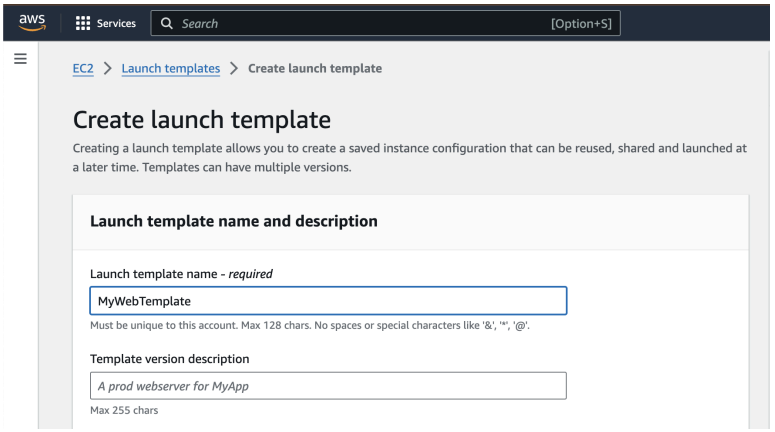
- **Using a Launch Template:** Instead of manually entering the details every time, you can create a *Launch Template*. A Launch Template is like a reusable blueprint where you define everything once, and then whenever you need to launch an instance, you can simply select this template and AWS will use it to launch the instance automatically.
- To create a **Launch Template** for that go to the EC2 instance and scroll down and find the launch Template after that click on the launch Template



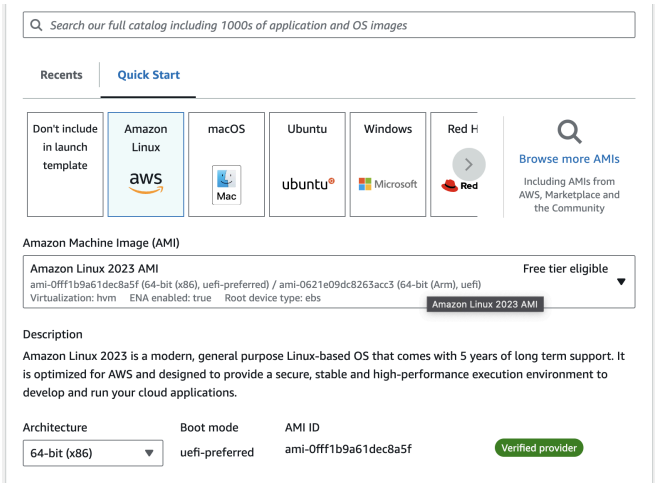
- After that click on the Create Launch template



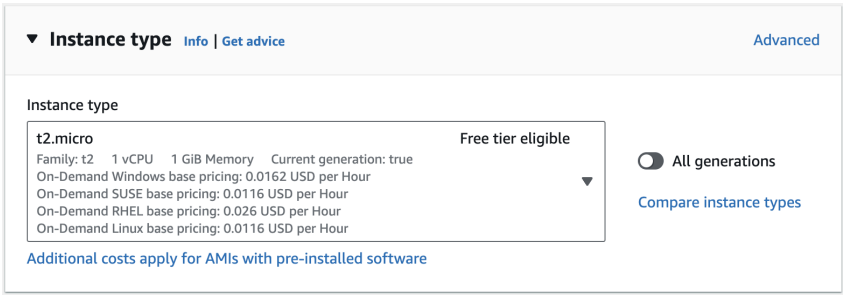
- After that give the name of a template



- When creating a launch template, you'll specify several details, including AMI (Amazon Machine Image, Instance Type, Key Pair, Network Settings, Security Group (Firewall) Etc
- After that select the AMI
- **AMI (Amazon Machine Image):** The operating system you want to use (like Amazon Linux).

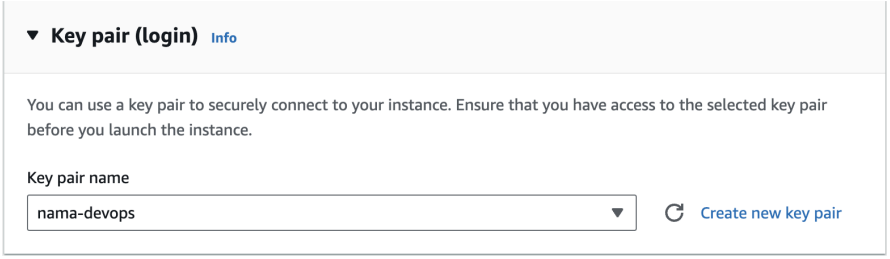


- After that select the instance type
- **Instance Type:** The type of virtual machine, like T2.micro, which defines how much computing power and memory your instance has

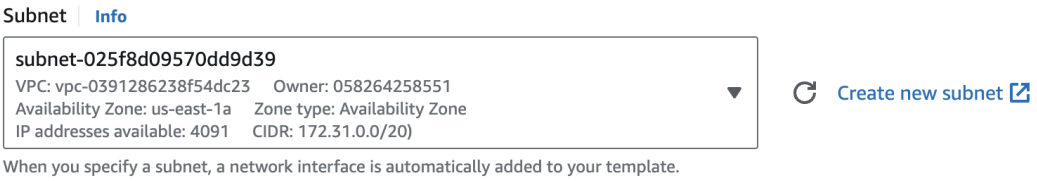


- After that select keypair

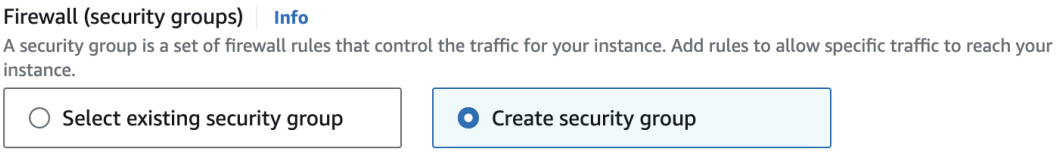
- **Key Pair:** This allows you to log in to your instance securely via SSH.



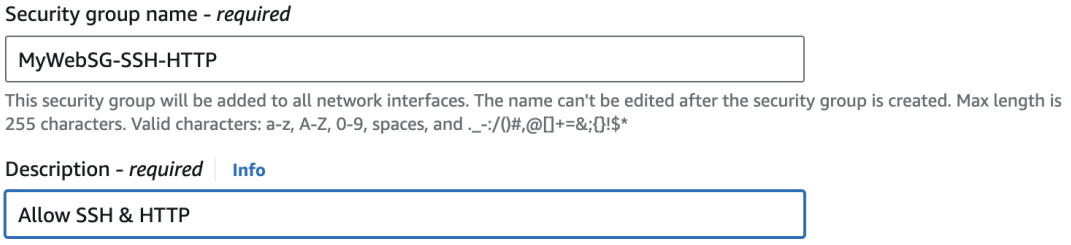
- After that select the subnet
- **Network Settings:** You can choose a network (subnet) for your instance, which is like placing it in a specific part of your AWS cloud



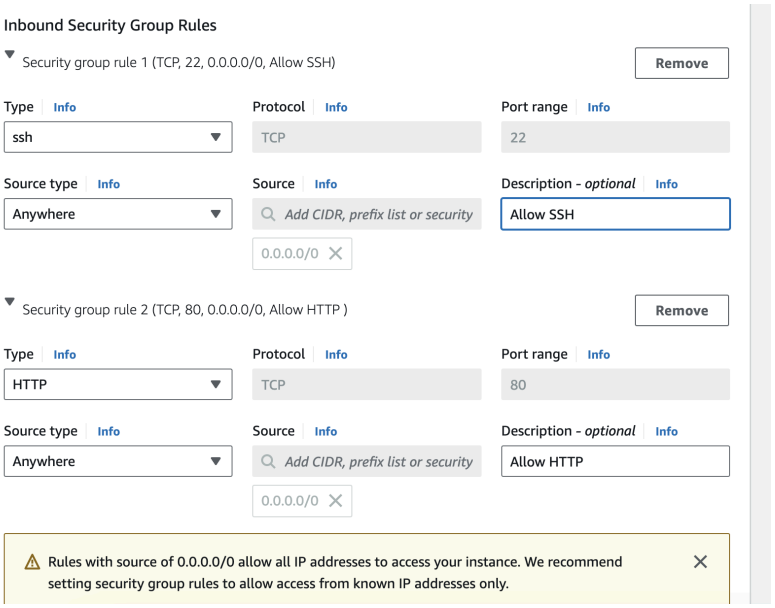
- After that click on the Create Security Group
- **Security Group (Firewall):** You can set rules for what kind of traffic is allowed to reach your instance. For example, you might allow SSH (for remote login) and HTTP (for web traffic).



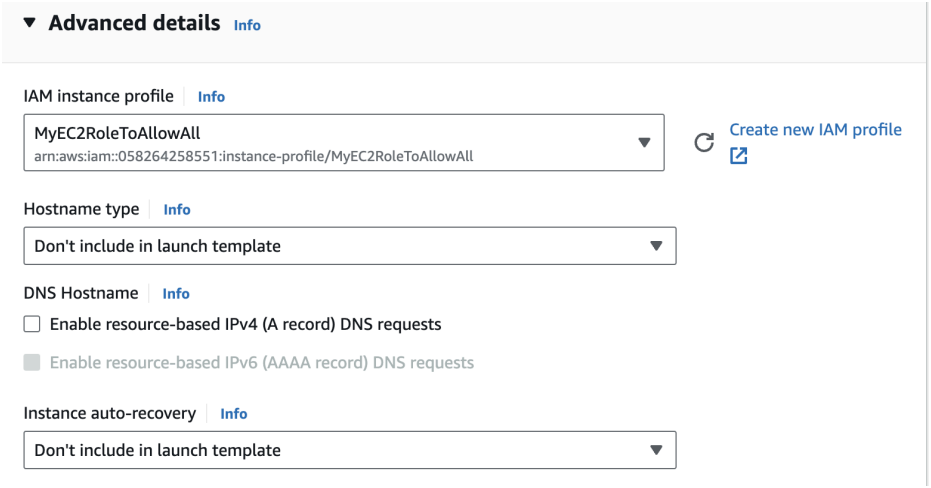
- After that give the name of the Security Group and add the description



- After that i allow the firewall rules



- After that select the IAM role for that EC2 instance if you don't have this role you can't access the EC2



- **IAM Role:** If you want your instance to interact with other AWS services (like S3 or CodeDeploy), you need to assign an *IAM Role*. This role gives the instance permission to perform specific tasks, like accessing files in an S3 bucket or deploying code
- **User Data Script:** You can also automate the setup of your instance further by adding a *User Data Script*. This script runs automatically when the instance starts. For example, you could use it to install a web server (like Apache) so that your website is ready as soon as the instance is launched.

- Also in that Script i Setup the code deploy agent

User data - *optional* | Info

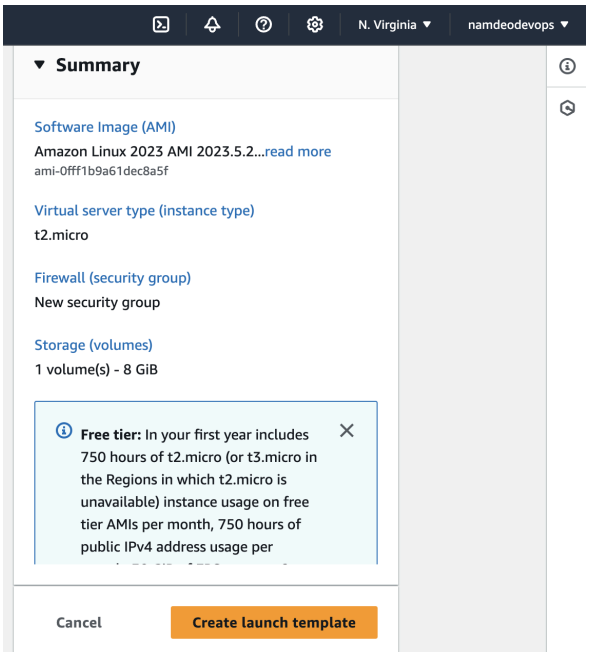
Upload a file with your user data or enter it in the field.

Choose file

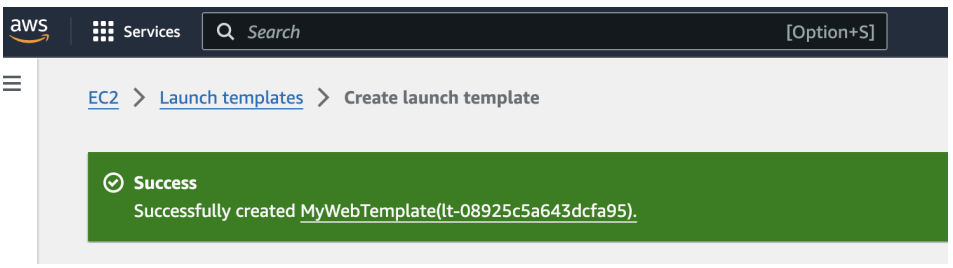
```
#!/bin/bash

yum install httpd -y

echo "Welcome to My Blue-Green Deployment Project Version - 1" >
/var/www/html/index.html

systemctl enable httpd --now

yum install ruby wget -y

wget https://aws-codedeploy-us-east-2.s3.us-east-2.amazonaws.com/latest/install

chmod +x ./install
./install auto
```

☐ User data has already been base64 encoded

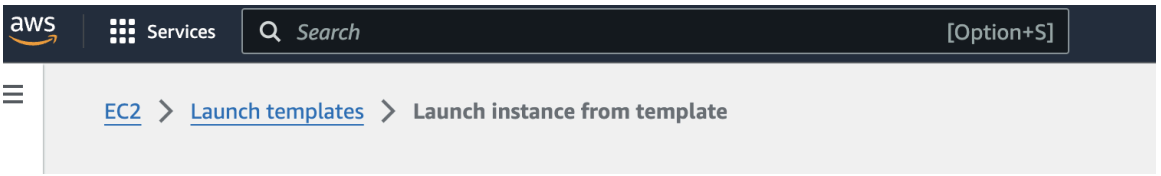
- After that click on the Create Launch Template



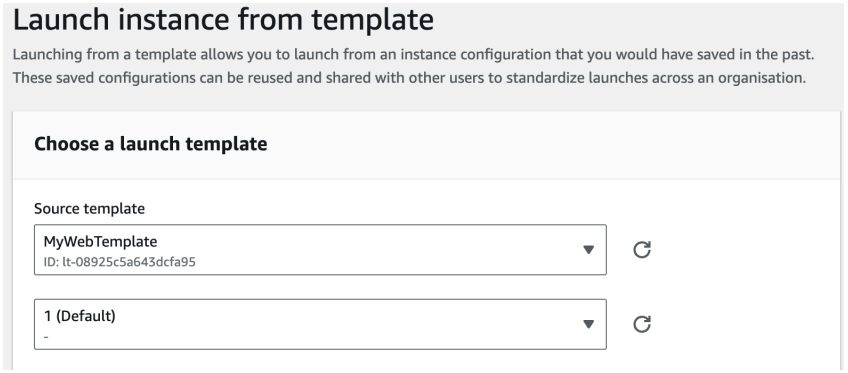
- Now we create a launch template



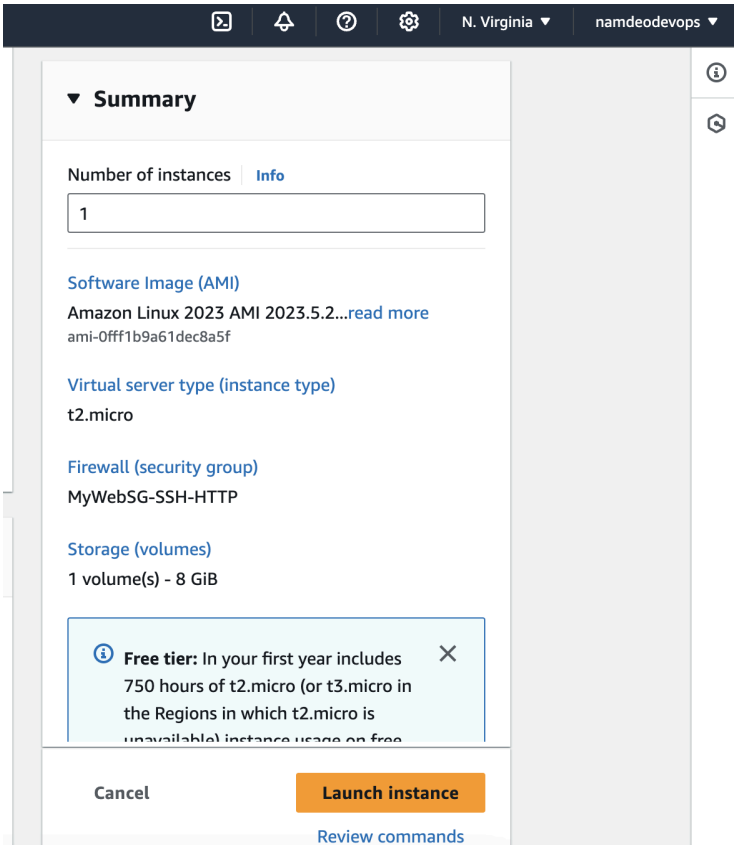
- By using the launch template, you can automate the entire process of launching and configuring your instance, saving you a lot of time. Once the template is set up, you just click to launch, and AWS handles everything from creating the server to installing software according to the instructions in your template.
- To launch the template for that go into the below path

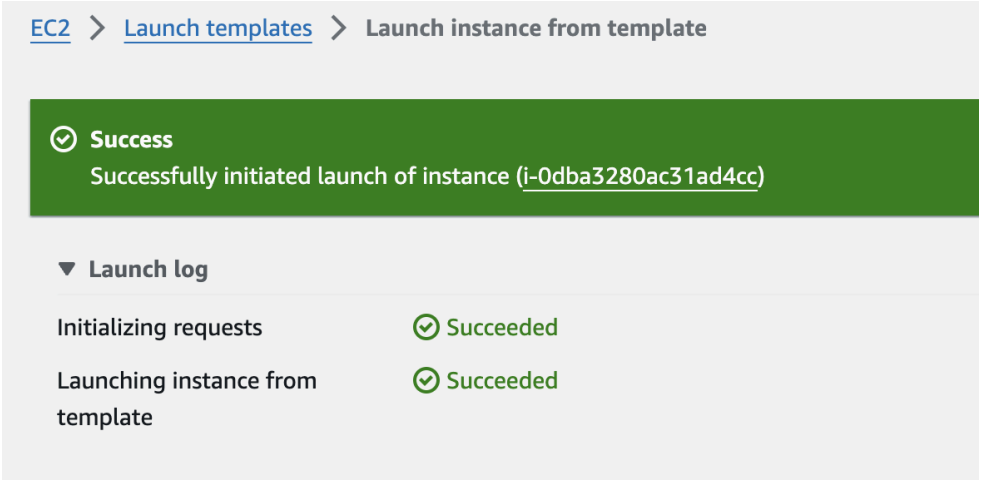


- After that select the template name and select the version default they select 1

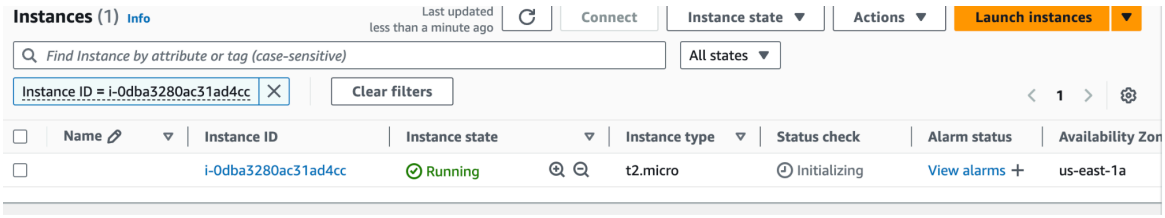


- Then click on the launch instance

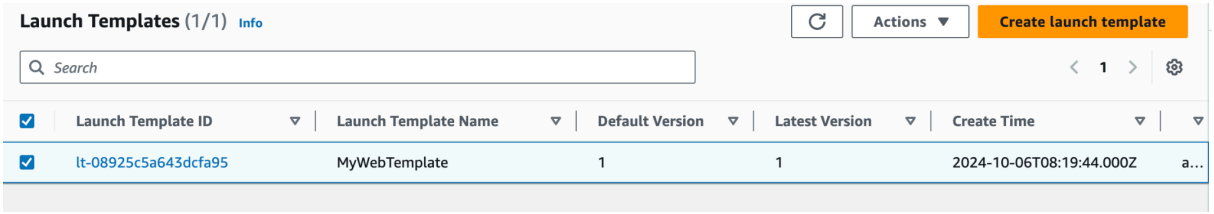




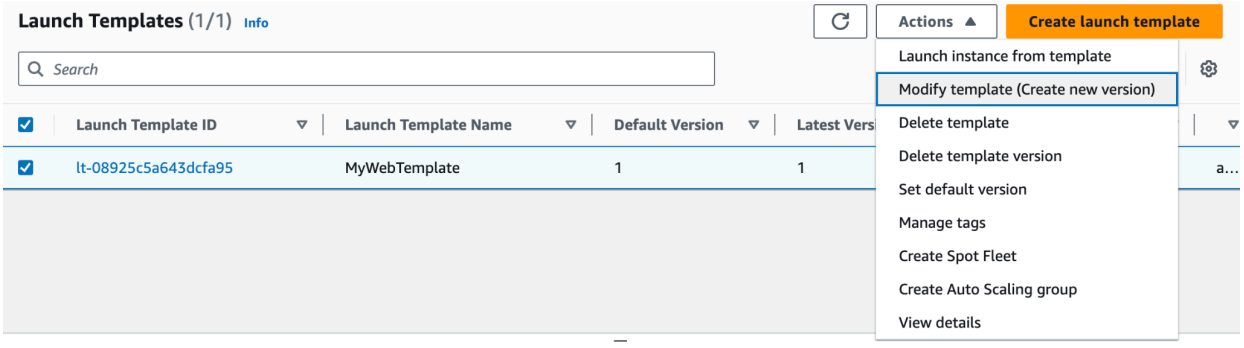
- Now I can see without setting up an Ec2 instance i successfully launch the instances



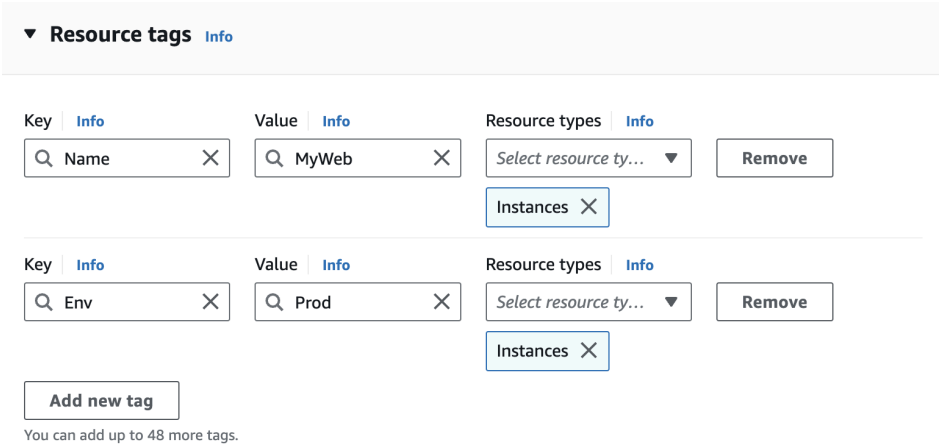
- If you want to modify the template you can modify it for that Click on the template name



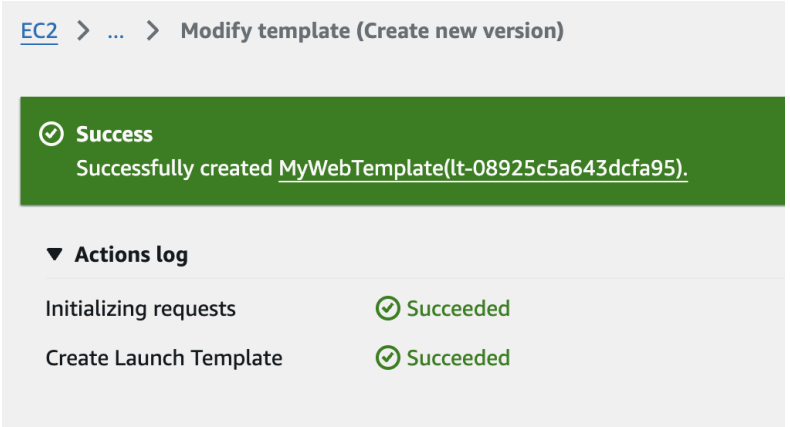
- After that click on the action button then click on the modify template



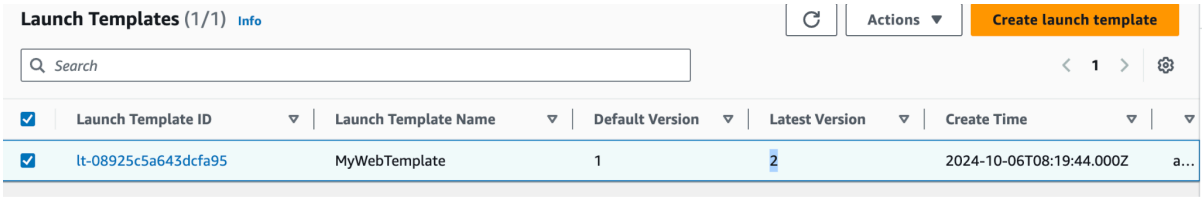
- In that template we change the tag of the instance good practice is always to give the tags



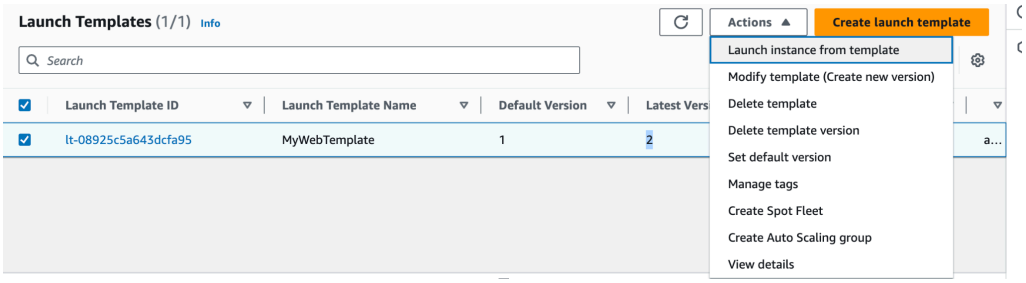
- After that click on the Create template



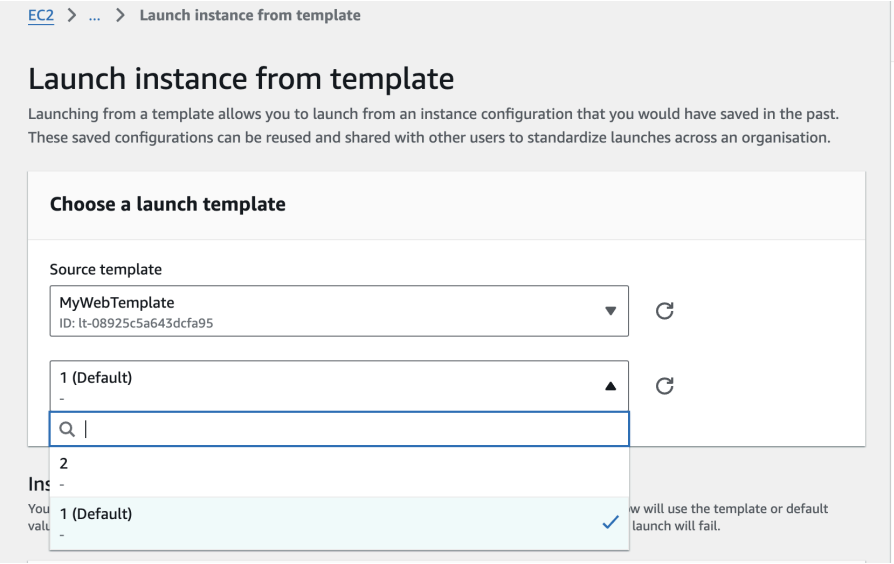
- As we can see I change the version of the template



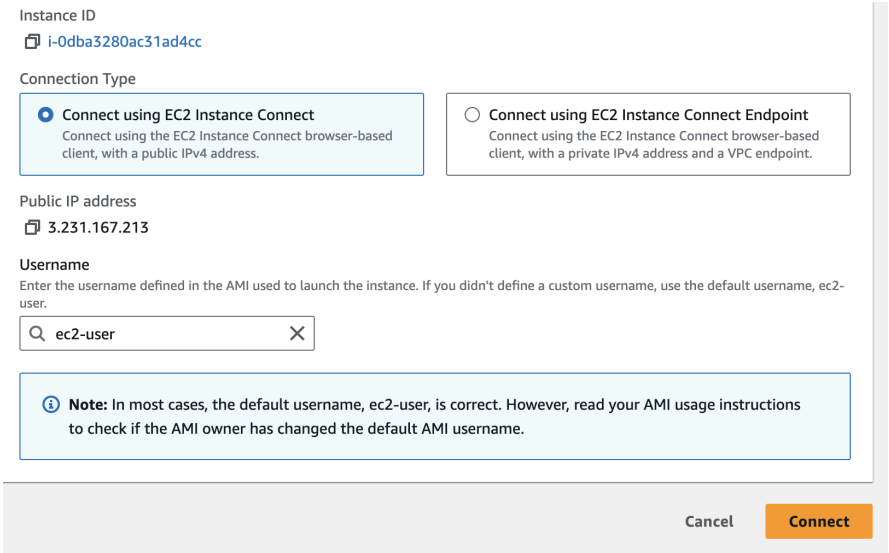
- If you want to launch which version they give the options for that click on the action and then click on the launch instance from the template



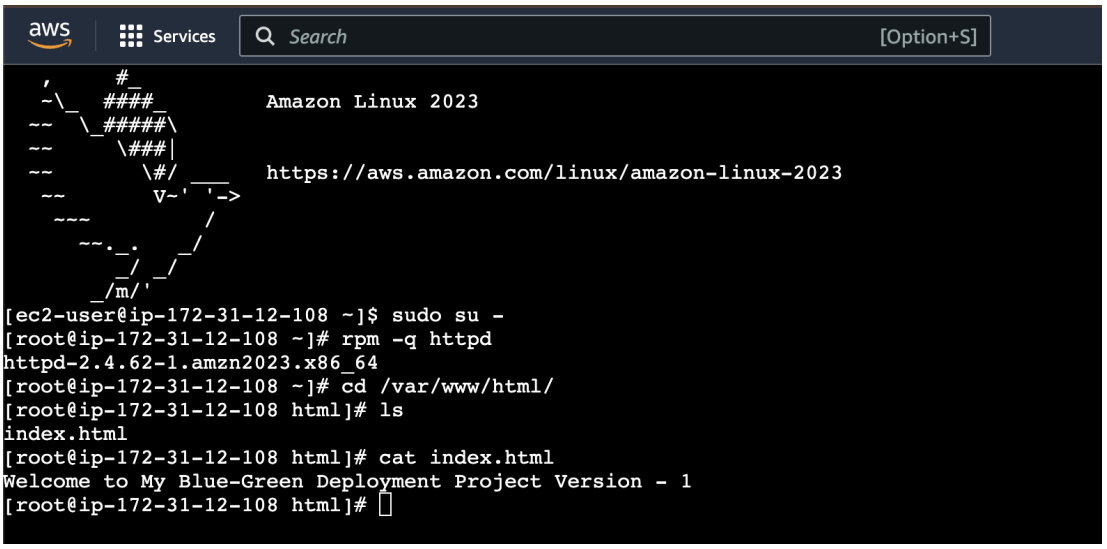
- As we can see i select the version of the template you want to use

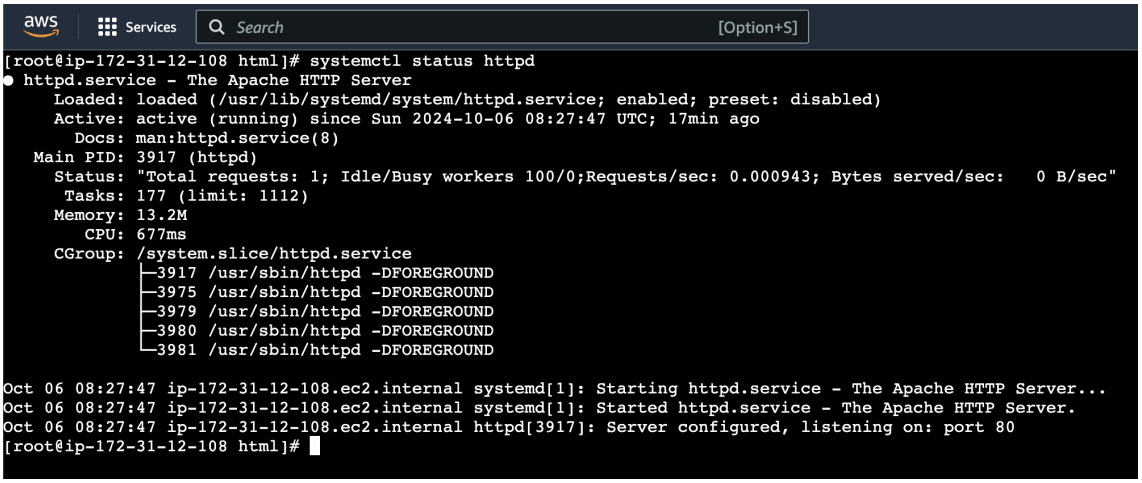


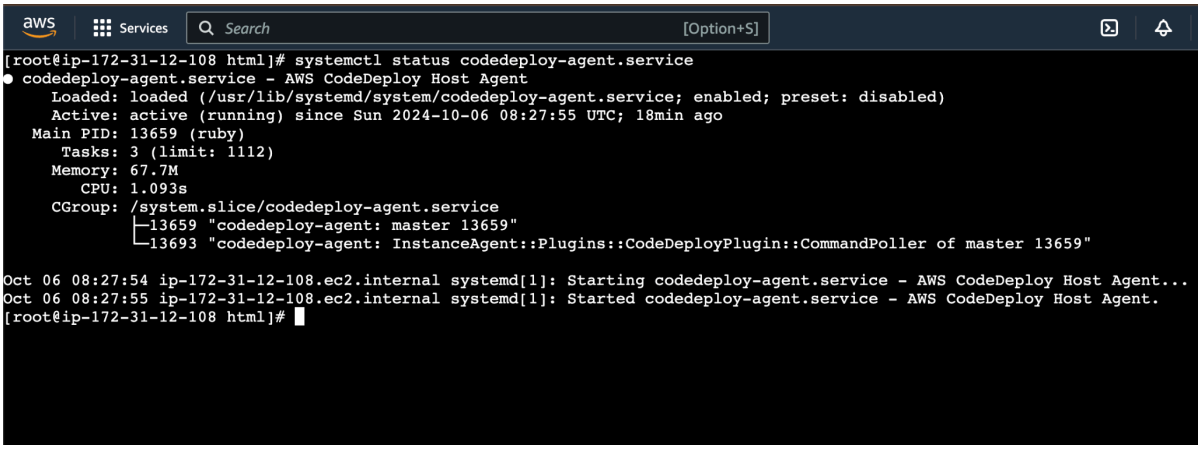
- The instance is set up to automatically install and configure a web server upon launch using a script (user data).
- The web server will host a homepage with specific content, such as "Welcome to My Blue-Green Deployment Project Version - 1".
- To ensure the instance can be managed in the future using AWS CodeDeploy, he installs the CodeDeploy agent during the instance launch process. This eliminates the need for manual installation later.
- Instead of manually configuring each instance, a launch template allows for reusable, automated configurations. Each template can have different versions, which can be updated and modified based on new requirements. He highlights the ease of deploying instances with a single click after the template is set up.
- For quick check instance is configured or not
- For that click on the Ec2 instance and connect with Ec2 terminal



- After that check the httpd software and HTML file also check the code deploy the agent

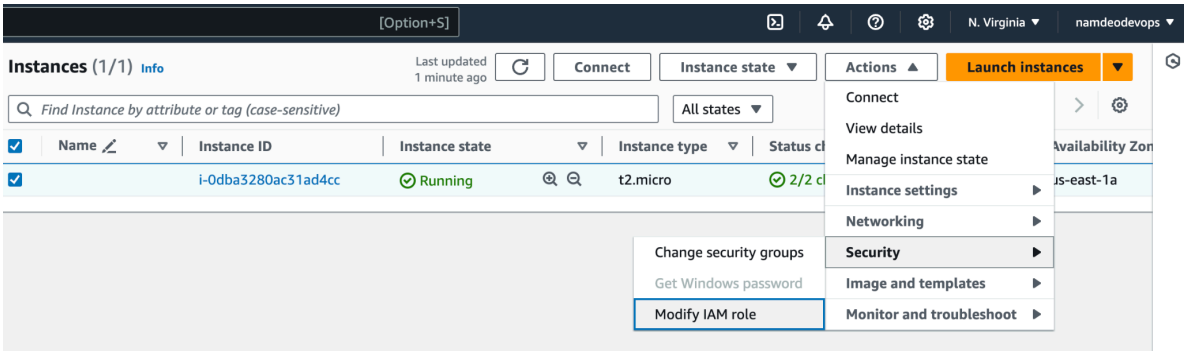


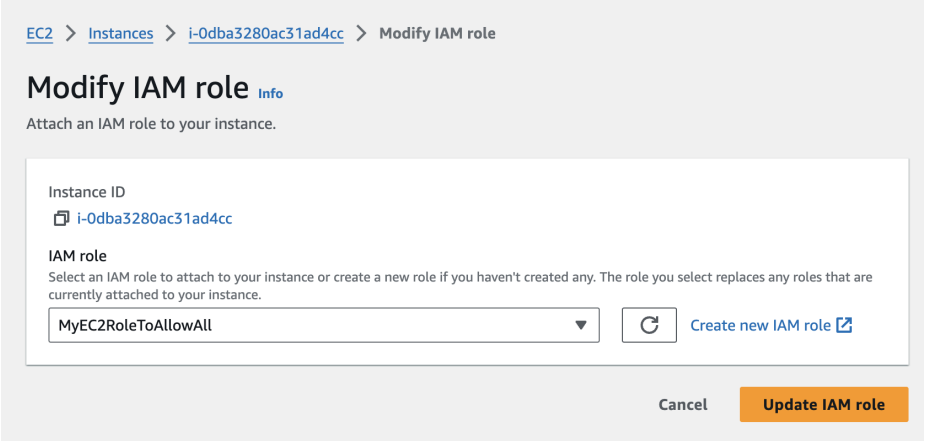




- As we know two access two services in the AWS we need to attach the IAM role

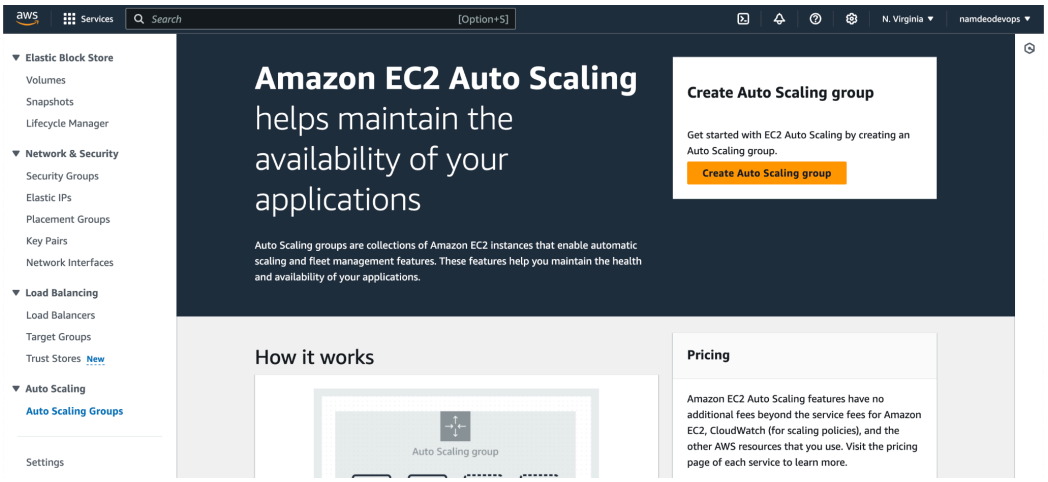
- But in our case i automatically attach the IAM role to check and follow the steps



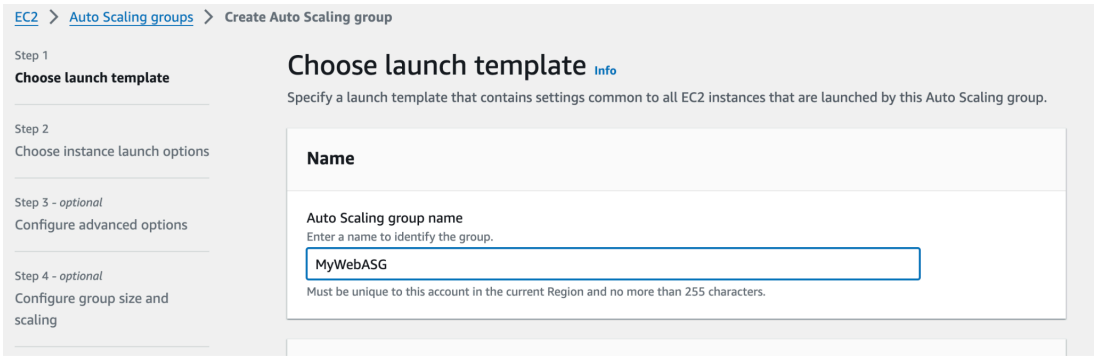


- We have two ways to launch the template
    - First is the direct launch template
    - Second is the Auto Scaling Group
- Auto Scaling Group (ASG) in AWS is a service that helps maintain the availability and scalability of your applications by dynamically adjusting the number of EC2 instances in response to changing demands.
- **Automatic Monitoring & Scaling**: ASG monitors the performance of instances, such as CPU utilization or network traffic. Based on load, it can launch or terminate instances according to predefined scaling policies
- **Scale up**: When demand increases (e.g., more traffic to a website), ASG can add more instances to handle the load.
- **Scale down**: When demand decreases, ASG can terminate unneeded instances, optimizing resource costs.
- You can set a desired number of instances to always be running (e.g., 10 instances). If any instance fails or becomes unhealthy, ASG automatically replaces it to maintain the desired state.
- ASG works with an Elastic Load Balancer (ELB) to distribute incoming traffic evenly across multiple instances, ensuring optimal performance.
- ASG uses launch templates to create instances, ensuring all have consistent configurations (e.g., same instance type, security settings).
- By using ASG, businesses can automate their instance scaling needs, handle unpredictable traffic spikes, and ensure application reliability.

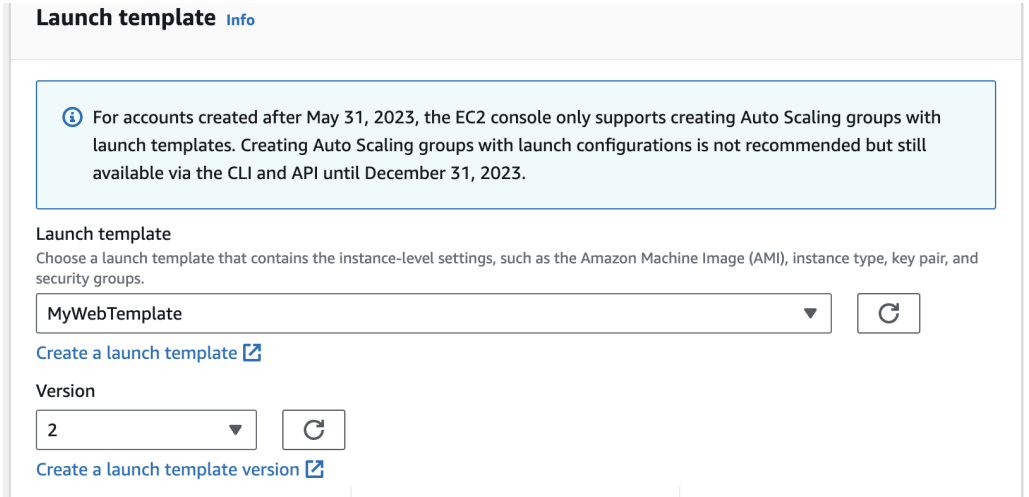
- To create an ASG go to the instance and scroll down then find Auto Scaling Group after that click on the Auto Scaling Group



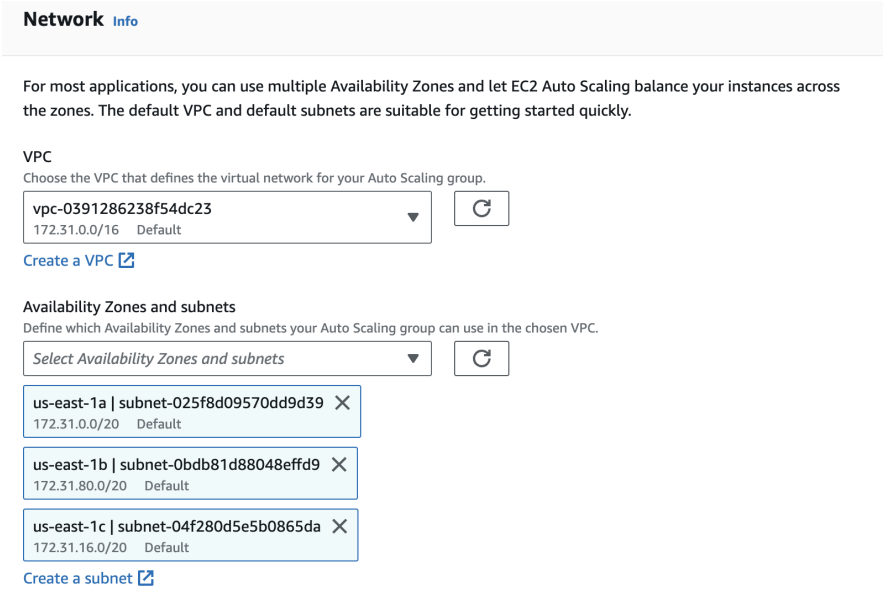
- After that give the name



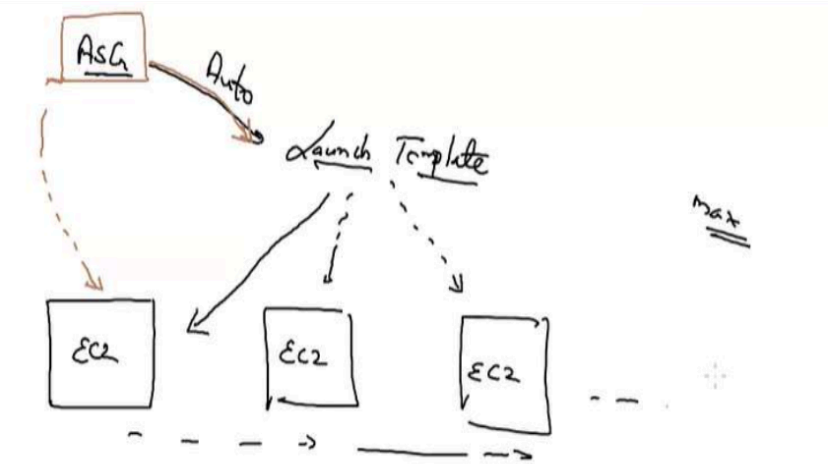
- After that select the template name also select the version of the template then click on the next

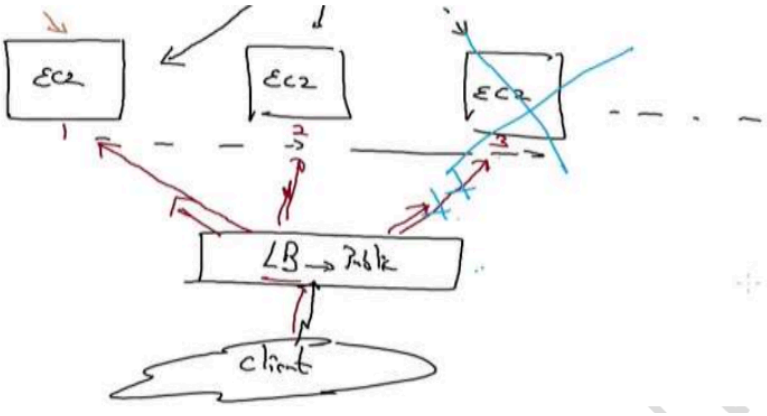


- After that select the Subnet then click on the next

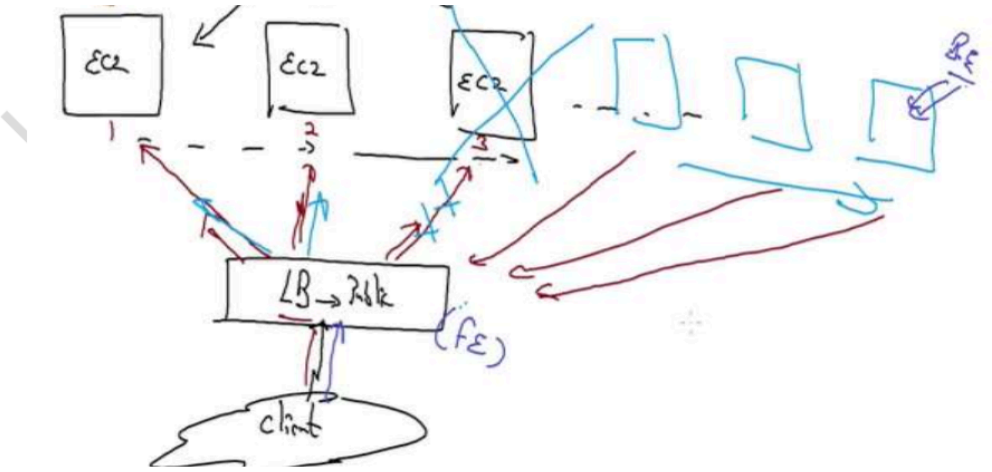


- When we set up a system with multiple instances (servers), it is common practice to avoid giving clients direct access to the instances' IP addresses. This is because instances can increase or decrease in number, causing their IPs to change dynamically
- Instead, we use a **load balancer**. The load balancer has a fixed public IP address, which clients can always use to connect.
- The load balancer's job is to distribute the incoming traffic evenly across the available instances
- If we have three instances, the load balancer will send one client request to the first instance, the next to the second instance, and so on. This is called **load balancing**.

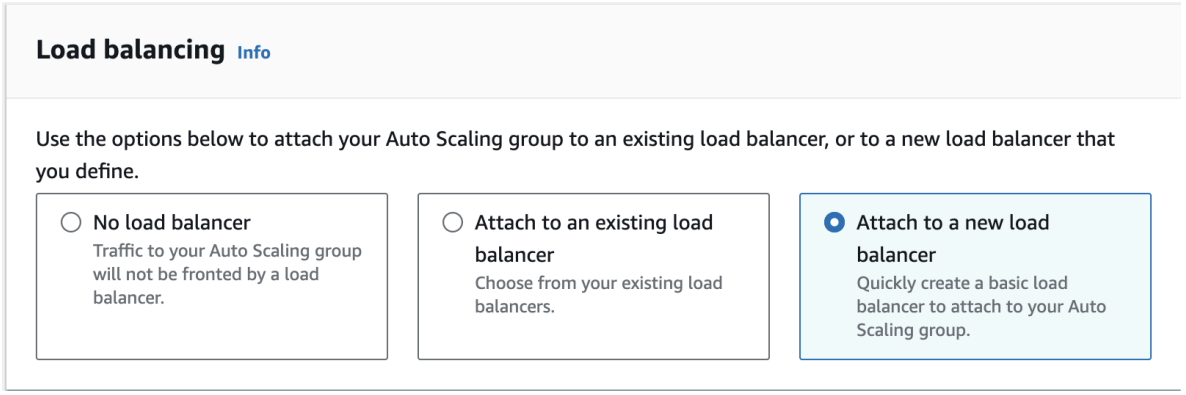




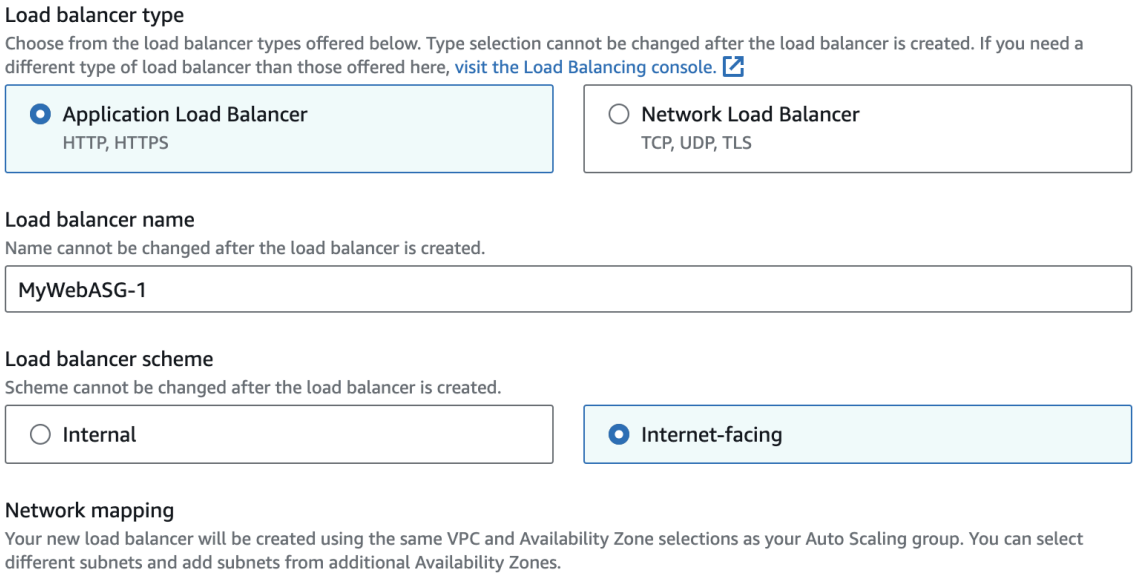
- Key Functions of Load Balancers:
    - **Distributes Traffic**: It sends client requests to different instances, ensuring no single server is overwhelmed.
    - **Health Check**: The load balancer monitors the health of instances. If one instance stops working, it automatically stops sending traffic to that instance
    - **Auto Scaling Integration**: When the load increases and new instances are launched by the **Auto Scaling Group (ASG)**, the load balancer automatically registers the new instances to handle more traffic.
- SG helps us maintain the right number of instances. I define a desired capacity (e.g., always having two running instances). If one instance fails, ASG will automatically launch a new one to replace it. ASG can also scale up (add more instances) when the load increases and scale down (terminate instances) when the load decreases.
- a load balancer works as the front end for clients, while ASG ensures that the backend (instances) always meets the required number, scaling up and down as needed based on the load



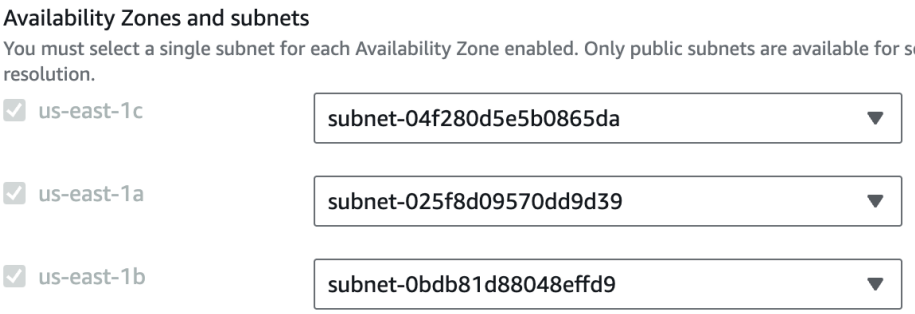
- After that click on the attach a new load balancer means they create a new load balancer for you



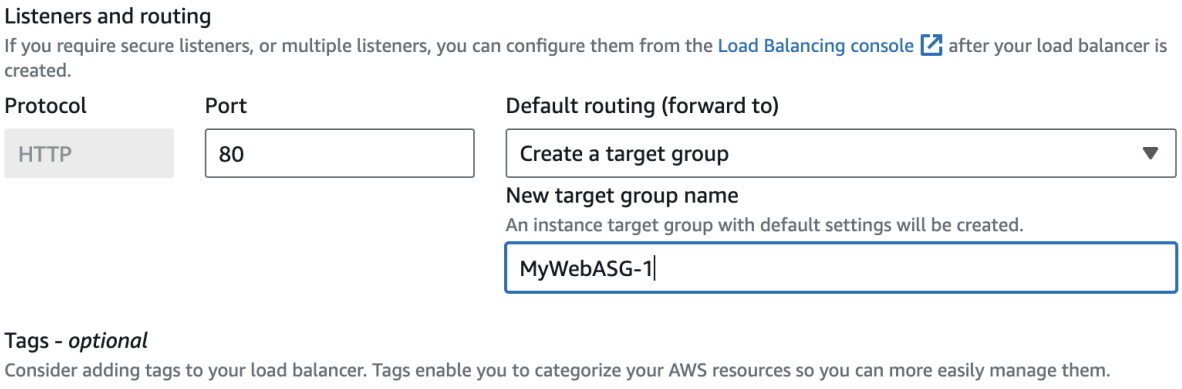
- After that we use an application load balancer and give the name of the load balancer also click



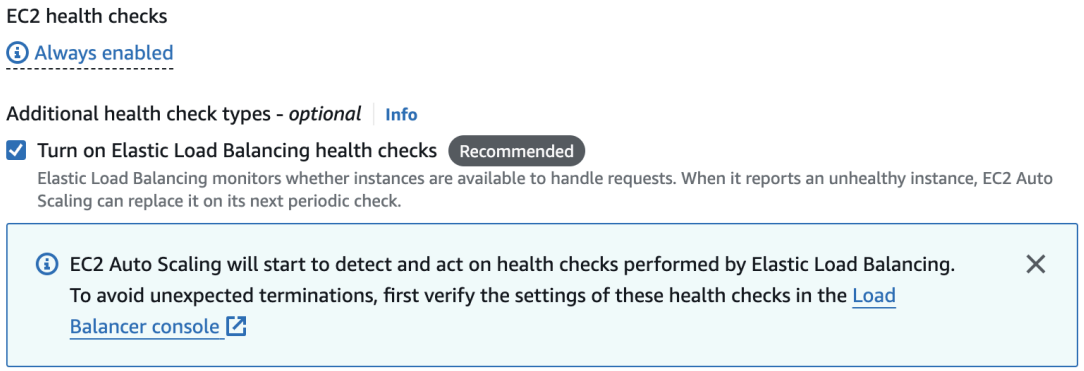
- After that select the subnet



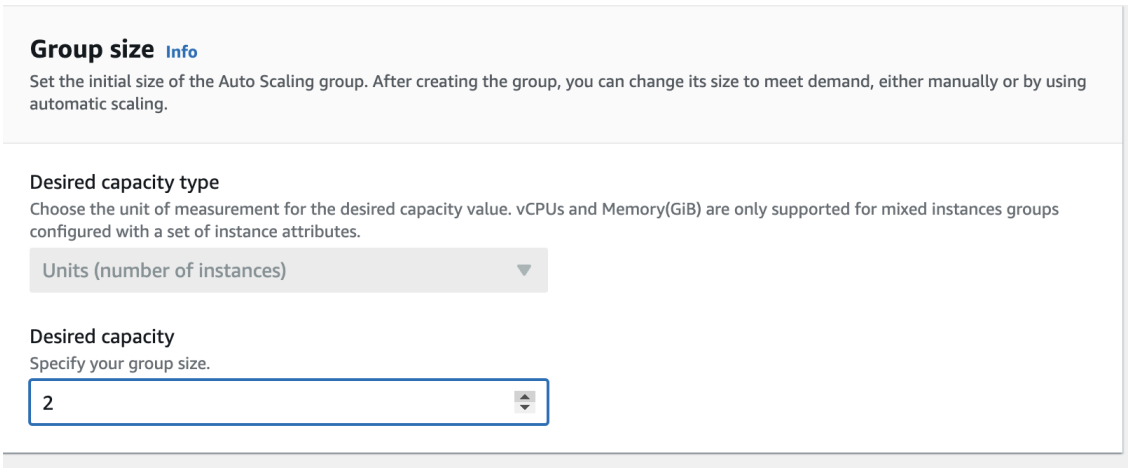
- After that i create a target group



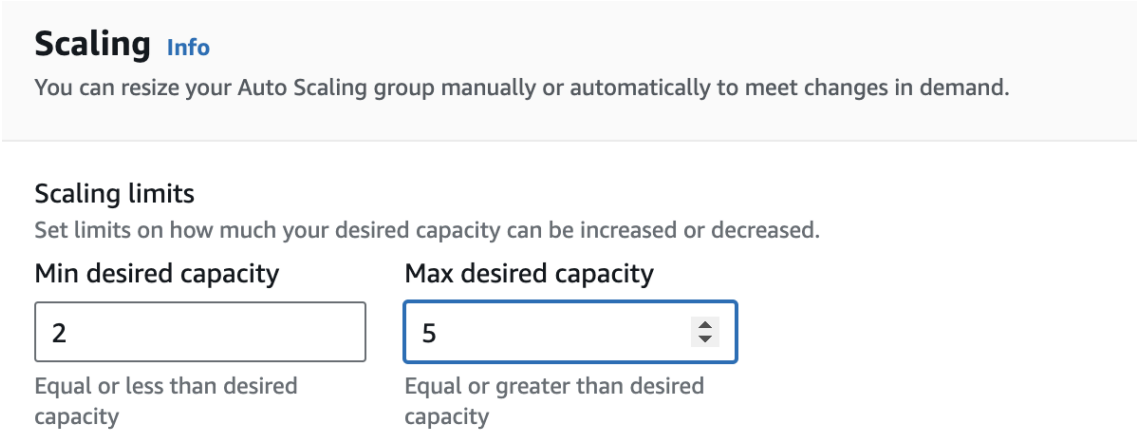
- Now i click on the turn-on ELB health checks



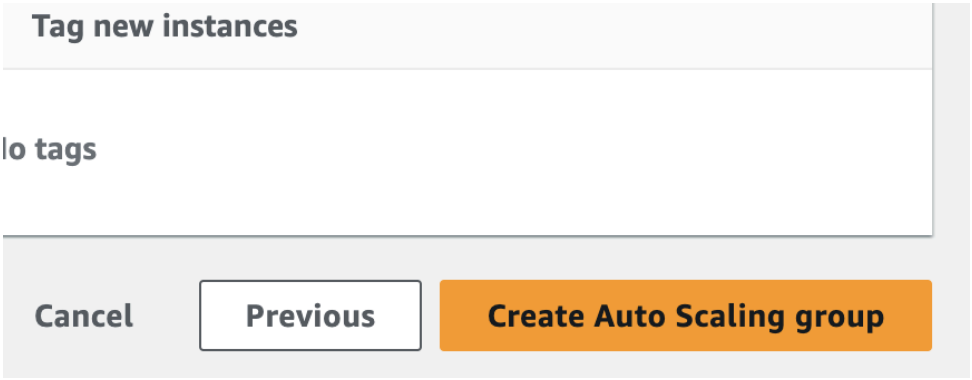
- After that i change the desired capacity



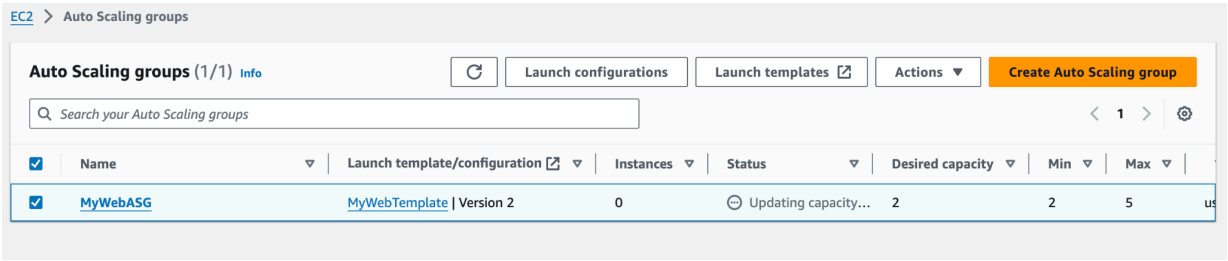
- Now Give the Scaling



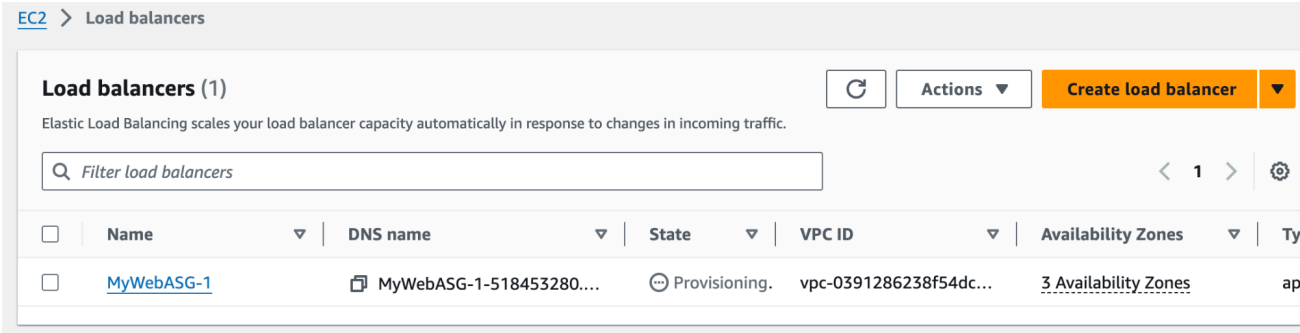
- After that click on the Create Auto Scaling Group



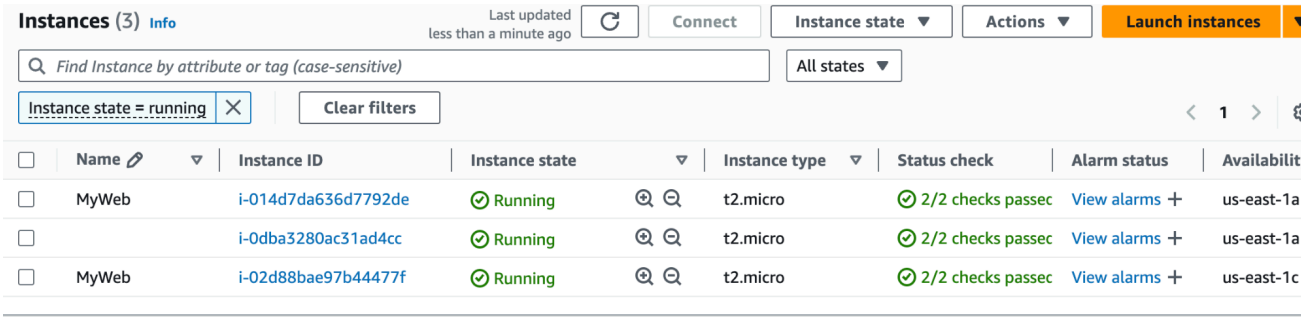
- As we can see i create a new Auto Scaling Group



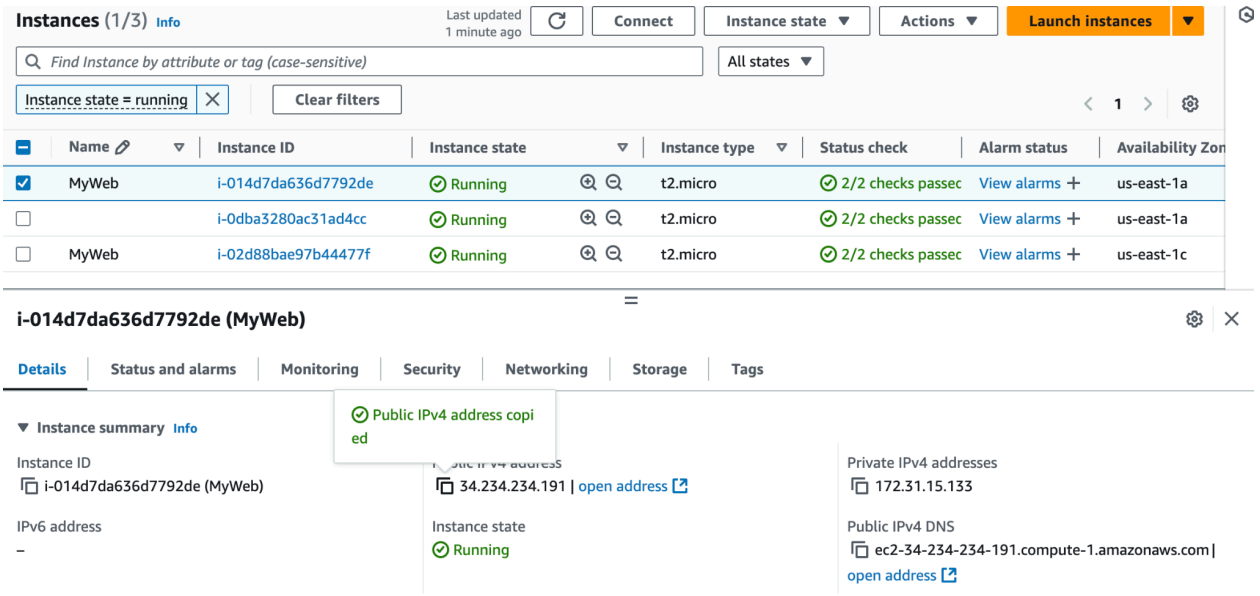
- Also i create a load balancer

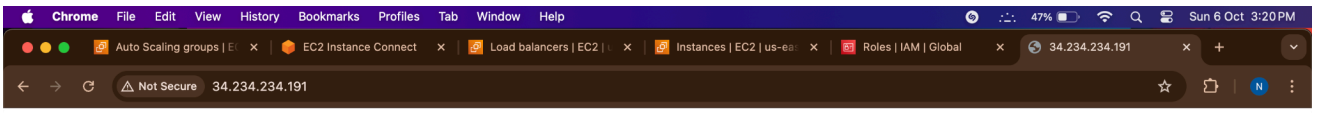


- As we can see two new instances are launch



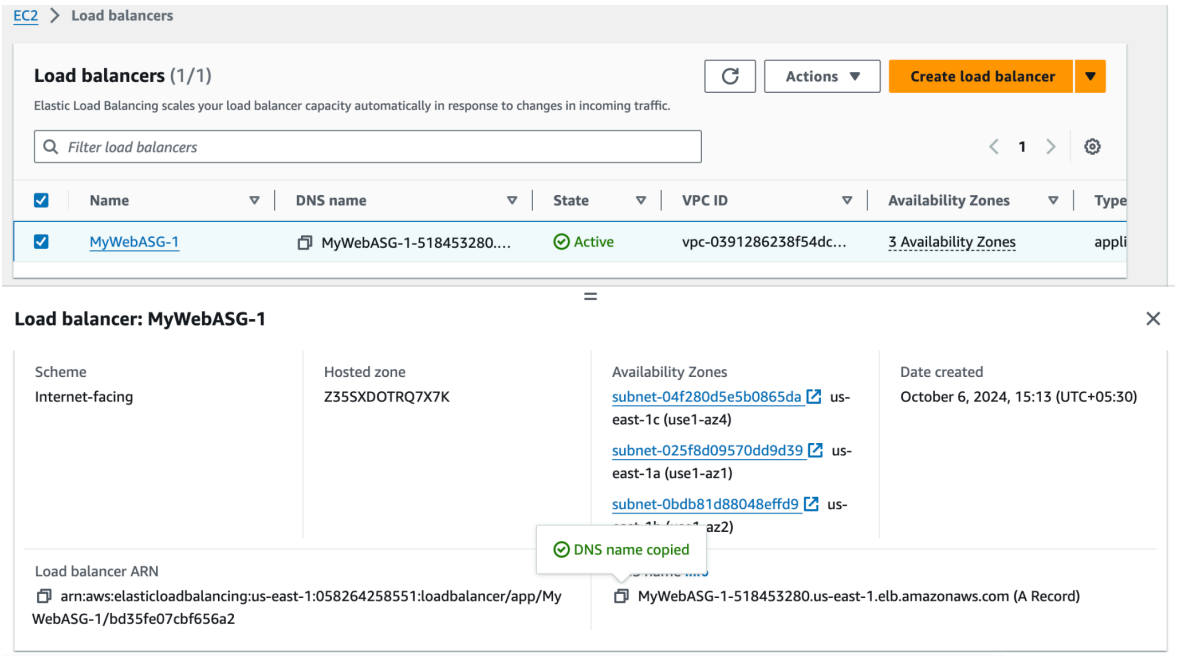
- Now if you want to check the web server then copy the Public IP then paste it into the browser



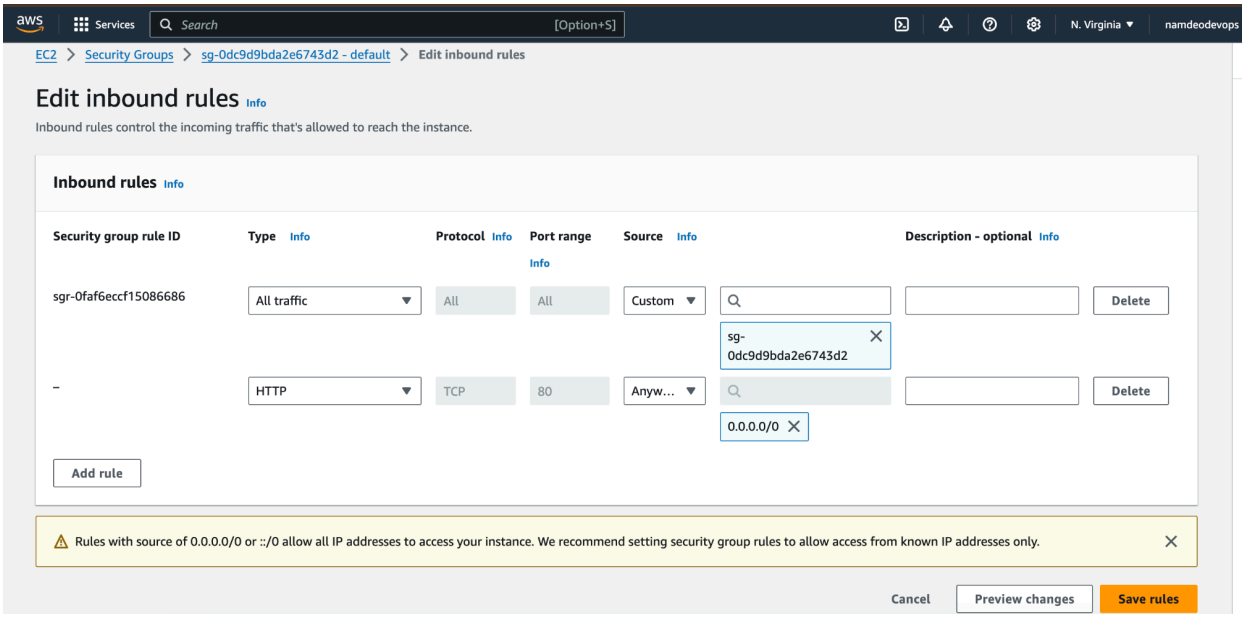


Welcome to My Blue-Green Deployment Project Version - 1

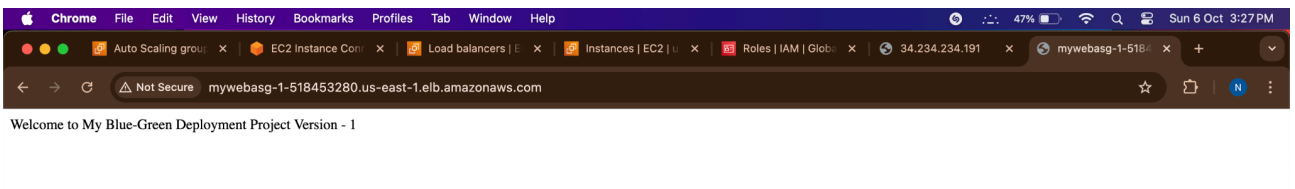
- But in the real world we do not share the public IP with clients for that we give a load balancer DNS name



- But first we need to allow the firewall to HTTP traffic

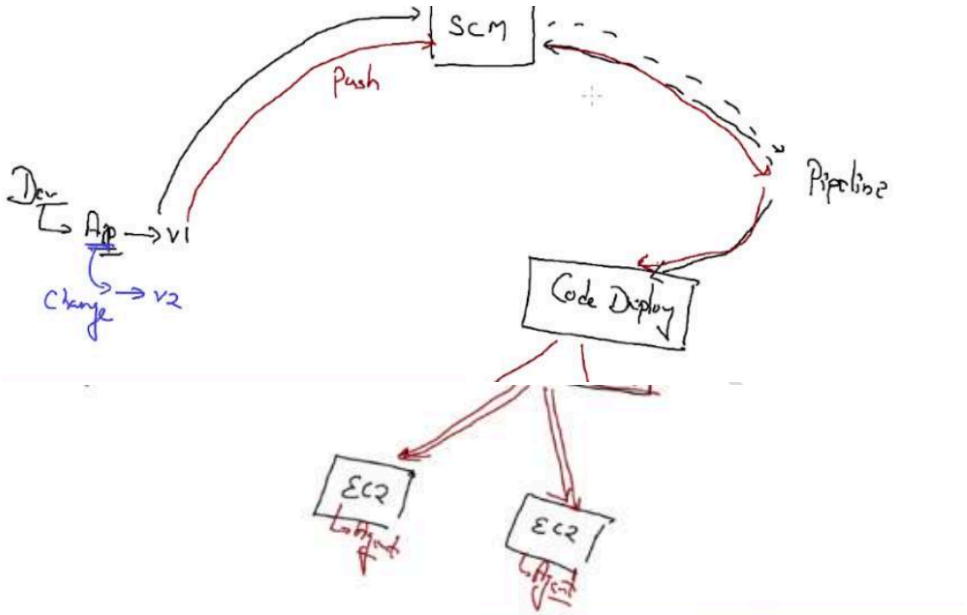


- After that copy the DNS name link and paste it on the browser

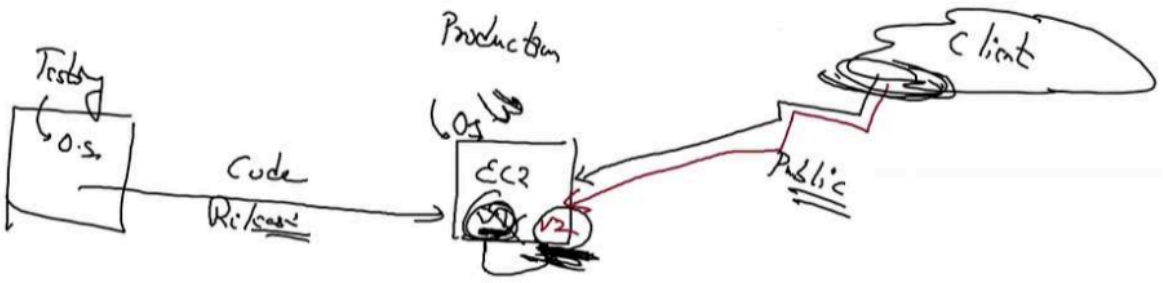


Now we start our main concept of blue-green deployment

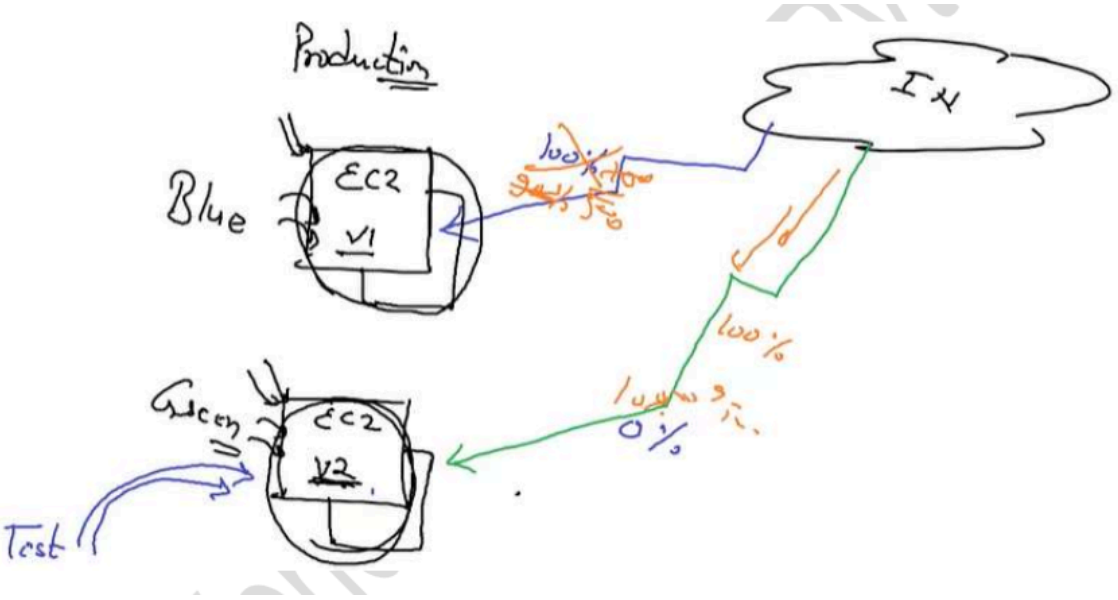
- First, we discuss a common scenario where a developer creates an app (in this case, an HTML page) and deploys it to EC2 instances using **CodeDeploy** as part of a CI/CD pipeline. When changes are made (new code versions), the pipeline fetches and deploys the new version to the EC2 instance
- The traditional way of deploying (without blue-green) is simply replacing the existing application code with the new version in the same environment
- This method works, but if there's an issue with the new version, the entire system could face downtime, as the deployment occurs directly in the existing instances.
- The **blue-green deployment** approach solves this issue
- Instead of deploying new code to the same environment, it deploys the new version to a separate set of instances (the "green" environment) while keeping the old version (the "blue" environment) running. Once the deployment is verified, traffic is shifted to the new version



- If any issues arise, it's easy to revert to the previous version by shifting traffic back to the "blue" environment.
- This strategy ensures smooth, low-risk updates without service disruption, as traffic can be switched between the environments instantly.
- The main idea is to ensure that new code releases are handled in a safe, low- risk manner by gradually switching traffic from the old system (blue) to the new system (green)
- **No Guarantee in Production**: Even if code is thoroughly tested before deployment, there's no guarantee it will work perfectly in production due to differences in system environments such as OS versions, Python, PHP versions, etc

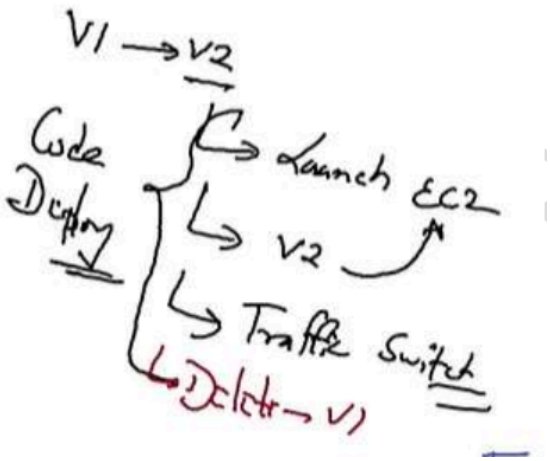


- **Risks of In-place Code Release**: Directly releasing new code to production (without careful traffic management) can lead to issues like session crashes, bugs, and problems with ongoing customer activities.
- **Blue-Green Deployment Strategy**:
    - ➢ The current running system (blue) handles all live traffic.
    - ➢ A new version of the application (green) is deployed on a separate set of instances.
    - ➢ Initially, all traffic goes to the blue environment, while the green environment is tested.
    - ➢ After successful testing, traffic is slowly switched from the blue to the green environment, either in phases (e.g., 10% green, 90% blue, then gradually shifting to 100% green) or all at once.

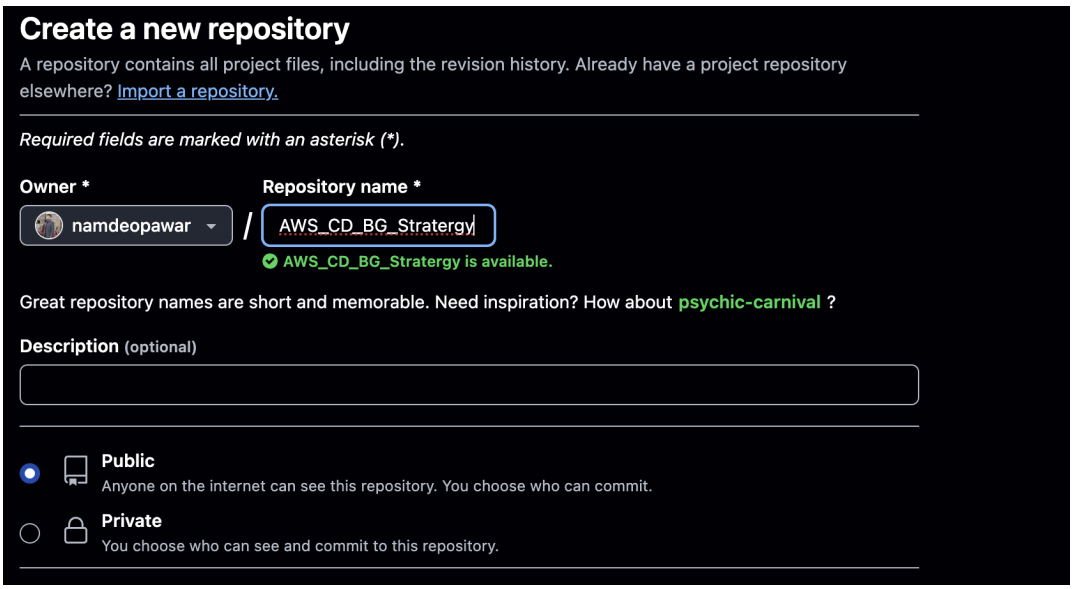


- This approach minimizes the risk of deploying faulty code to production, as it allows for gradual migration and the ability to switch back to the previous version if issues arise
- The process of switching traffic between the two environments can be automated and managed using tools that control network traffic, ensuring a smooth transition
- This strategy is particularly useful for minimizing downtime and reducing the risk of errors when deploying new code.

- In our case who does this for you
- For that we use code deploy service
- implement a **Blue-Green Deployment** strategy using AWS services like CodeDeploy, Auto Scaling, and Load Balancers
- Initially, deployment involved manually launching instances, deploying new code (version 2), and then switching traffic from the old instance (version 1) to the new instance using traffic redirection or switching
- **AWS CodeDeploy** automates this process. When a new code version is available, it automatically launches a new (green) instance, deploys the code, switches the traffic to the new instance, and terminates the old instance (blue) to save costs.



- **Auto Scaling** is used to launch new instances, based on launch templates
- **Load Balancer** (ALB - Application Load Balancer) is responsible for switching traffic between the blue and green instances.
- Now we implement the full End-to-End automation setup
- First I create a GitHub repository

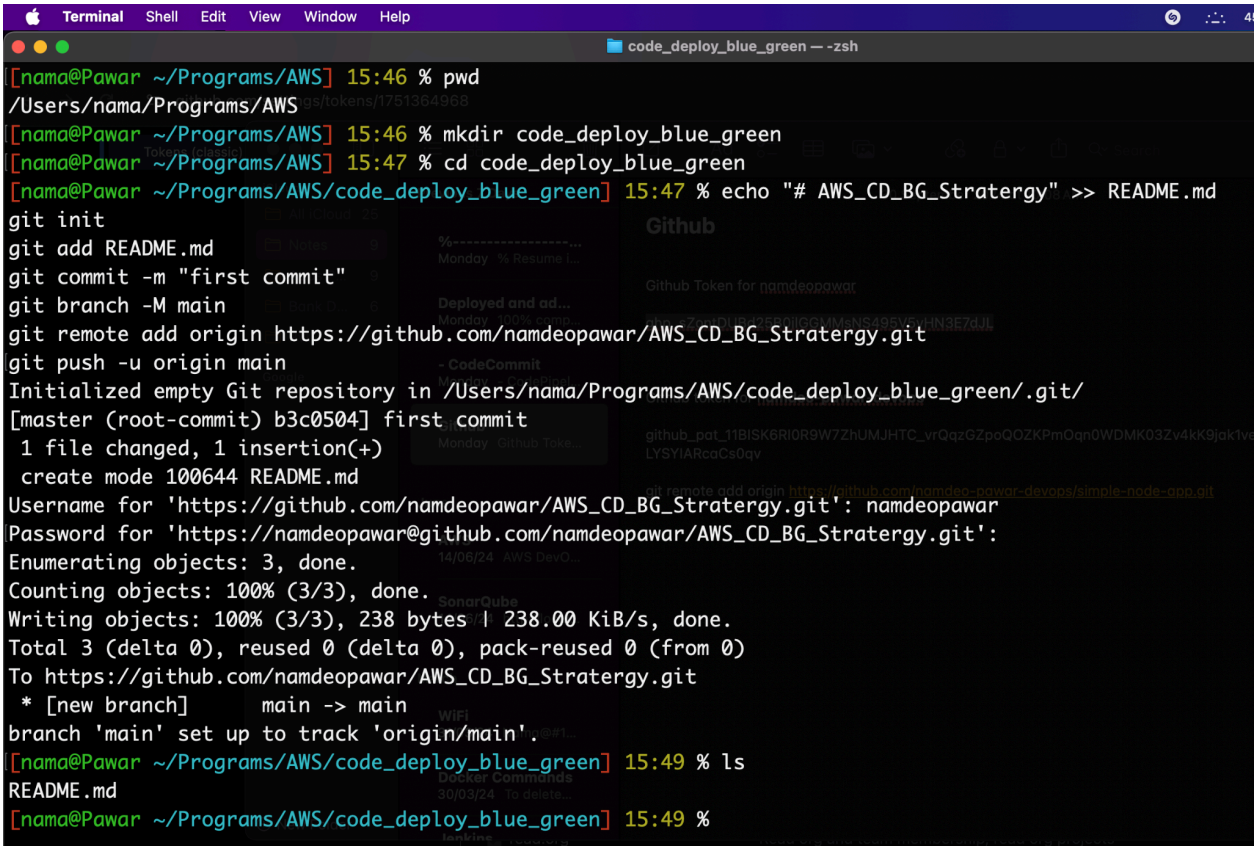


- After that copy the below command

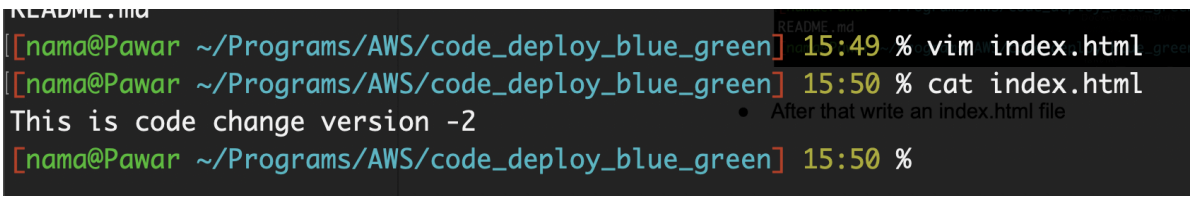
...or create a new repository on the command line

```
echo "# AWS_CD_BG_Stratergy" >> README.md
git init
git add README.md
git commit -m "first commit"
git branch -M main
git remote add origin https://github.com/namdeopawar/AWS_CD_BG_Stratergy.git
git push -u origin main
```

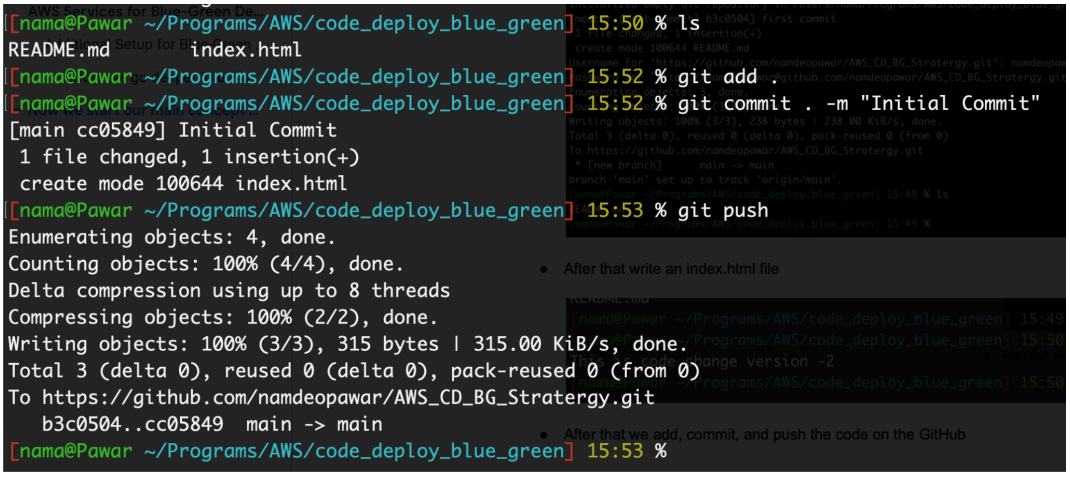
- Then we create a folder for the blue-green deployment project then paste all the commands



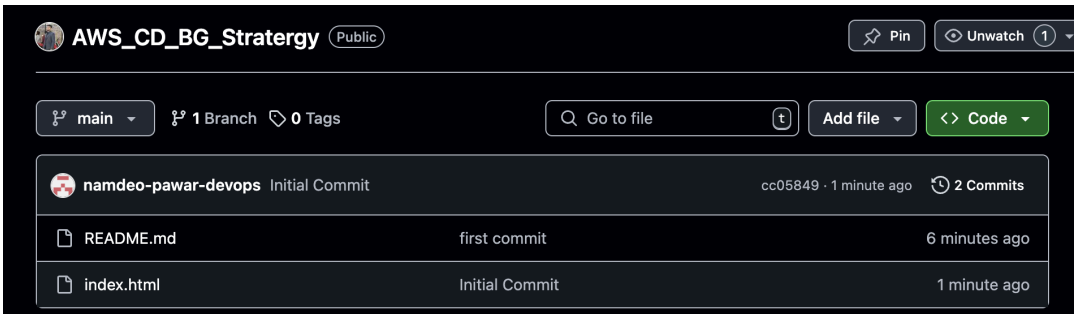
- After that write an index.html file



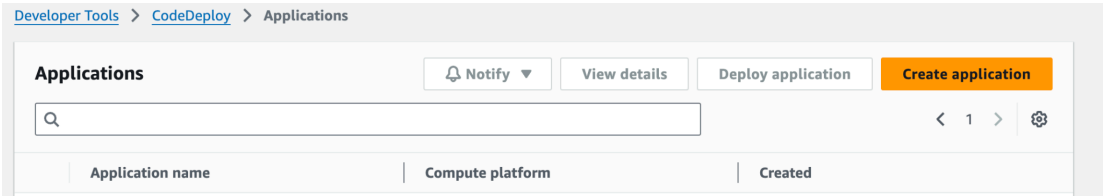
- After that I add, commit, and push the code on the GitHub



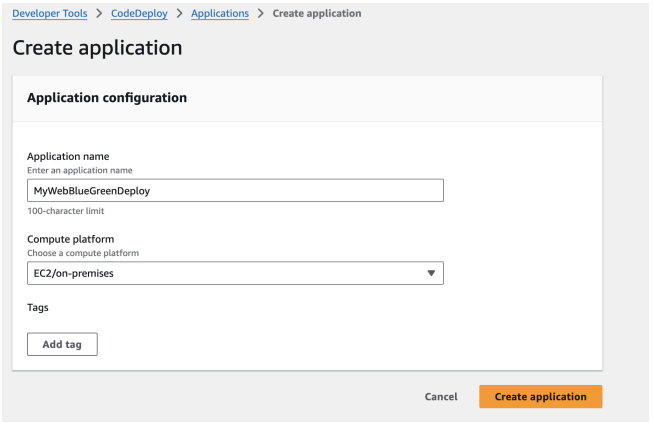
- As we can see I push code on GitHub



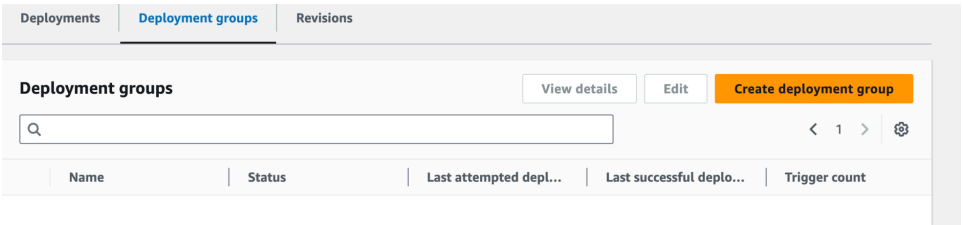
- Now we need to create a code deployment for that to go into the code deploy service
- Then create an application



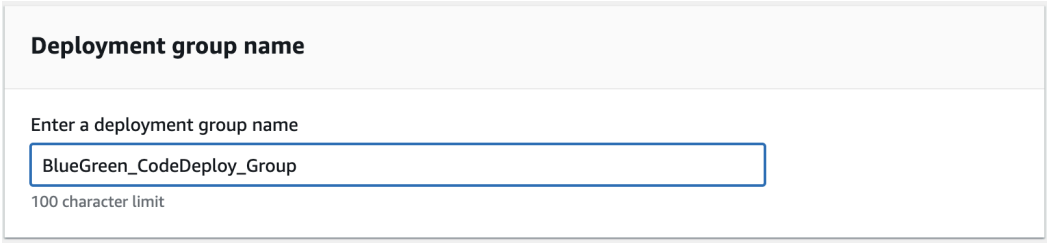
- After that give the application name



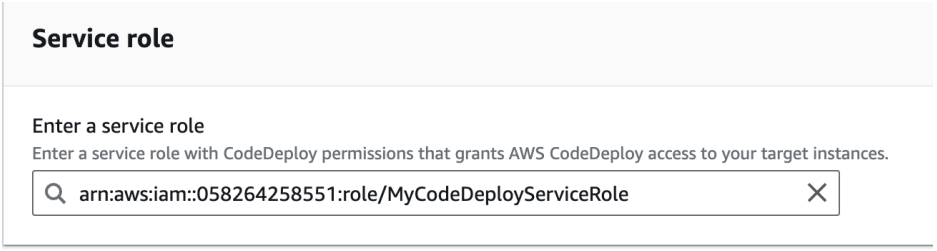
- After that I need to create a deployment group



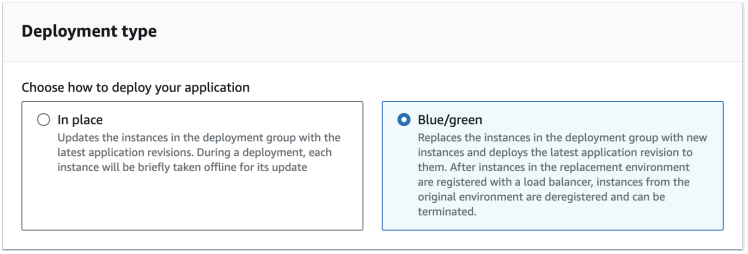
- After that give the deployment name



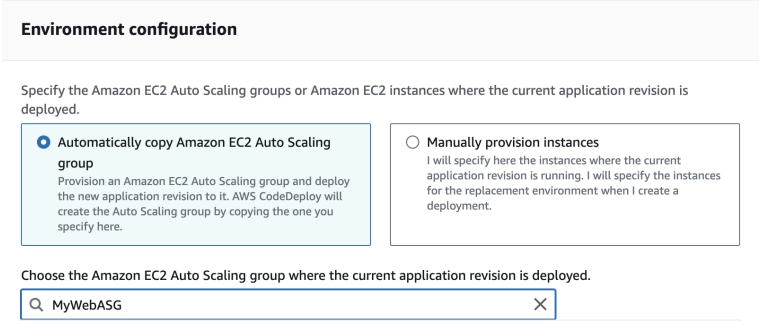
- After that select role



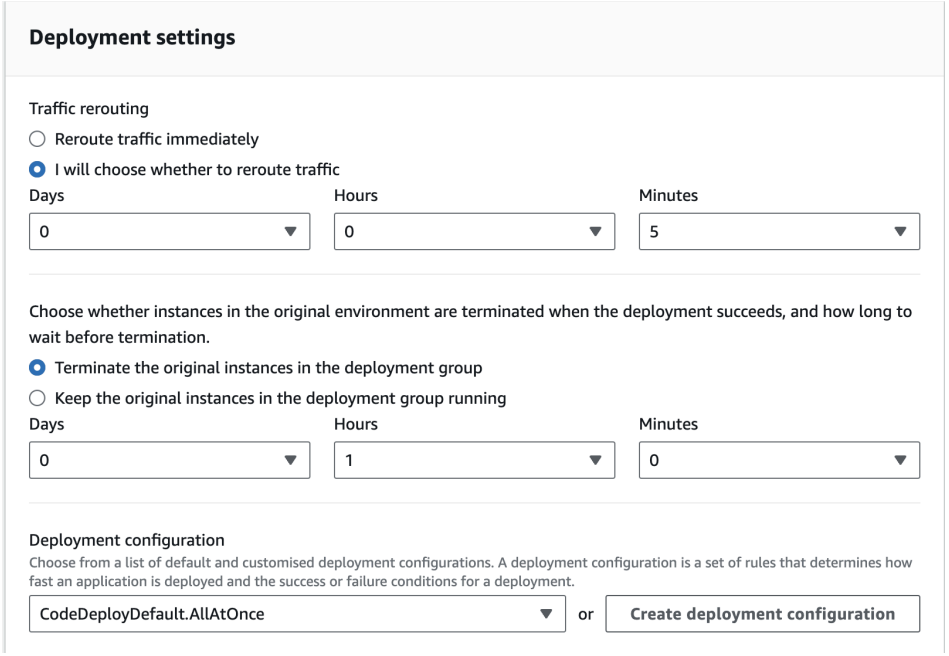
- After that click on the Blue-green deployment



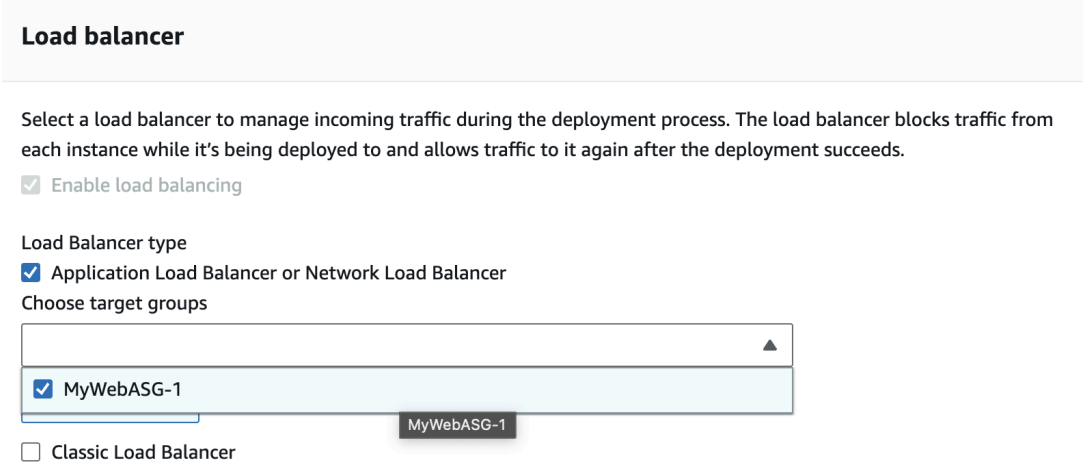
- Now we select automatically options



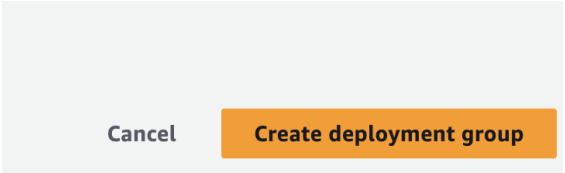
- After that give deployment settings



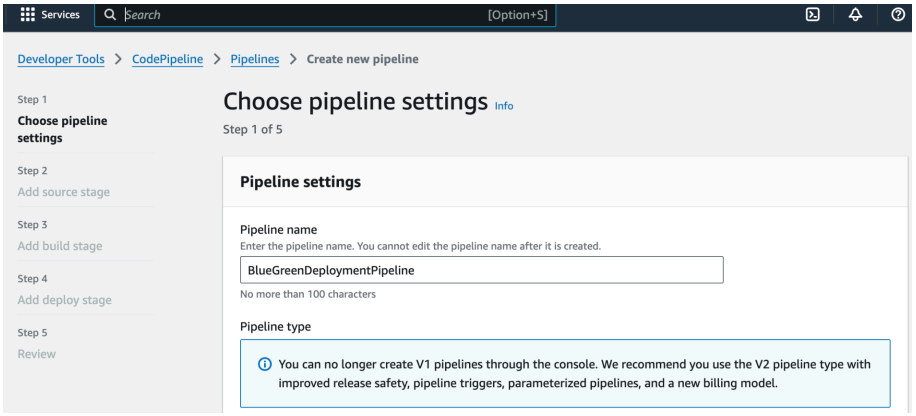
- After that select load balancer

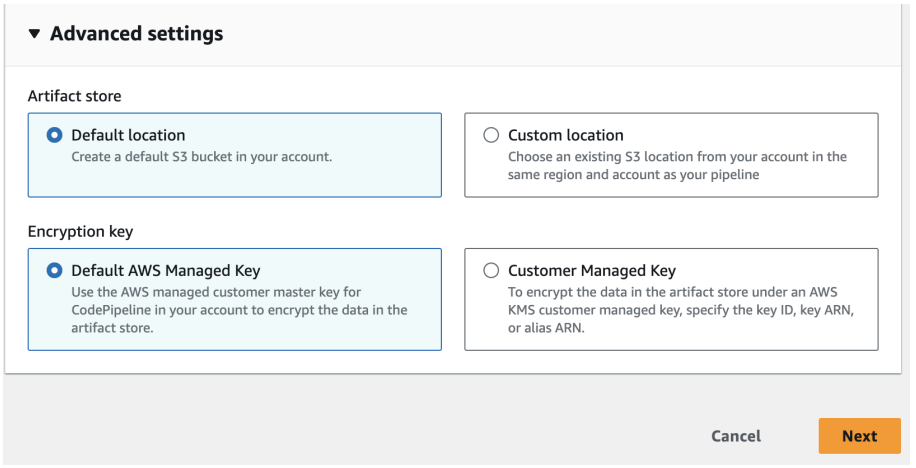


- Then click on Create Deployment group

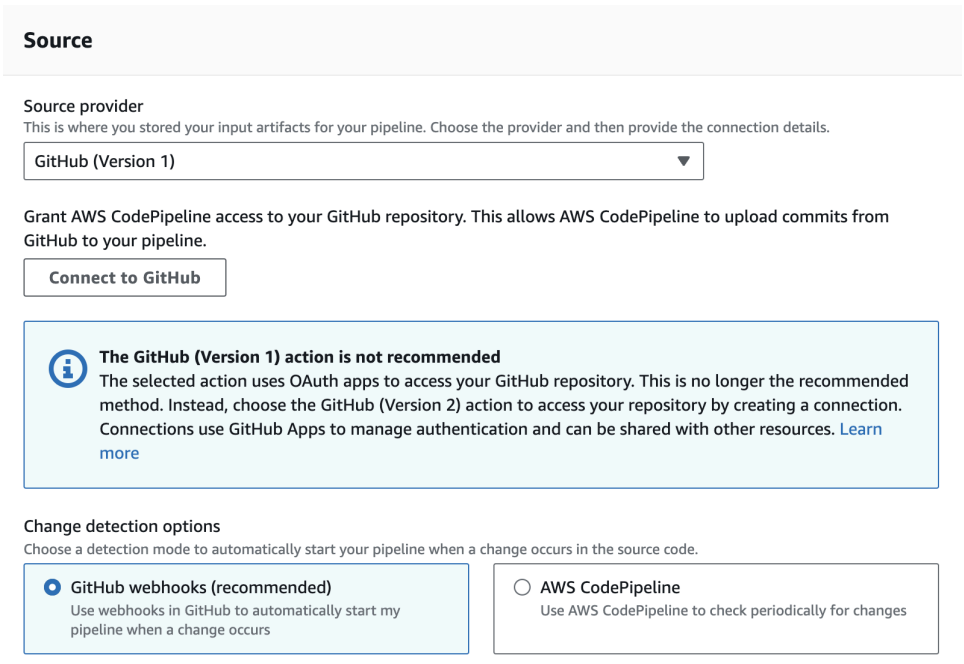


- After that we need to create a pipeline

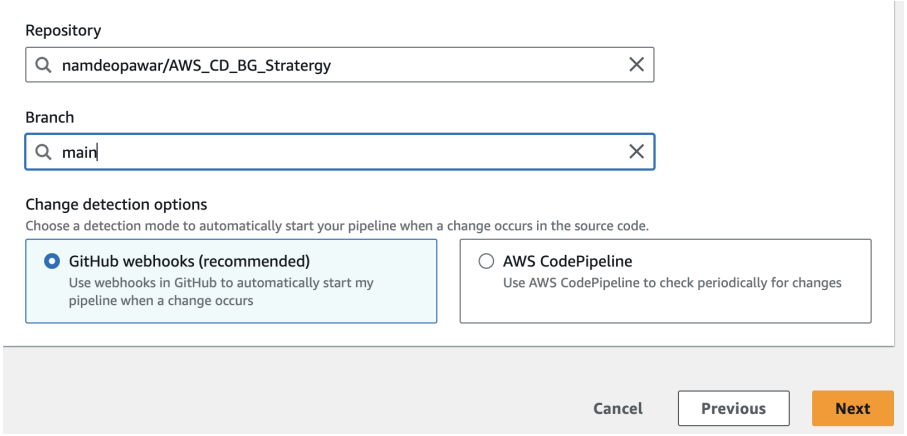




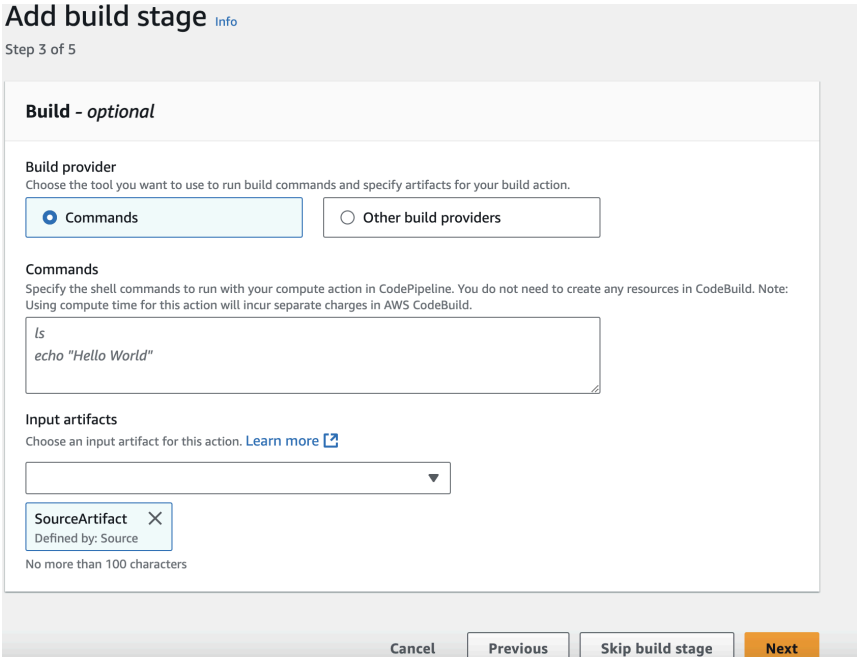
- After that select the source provider



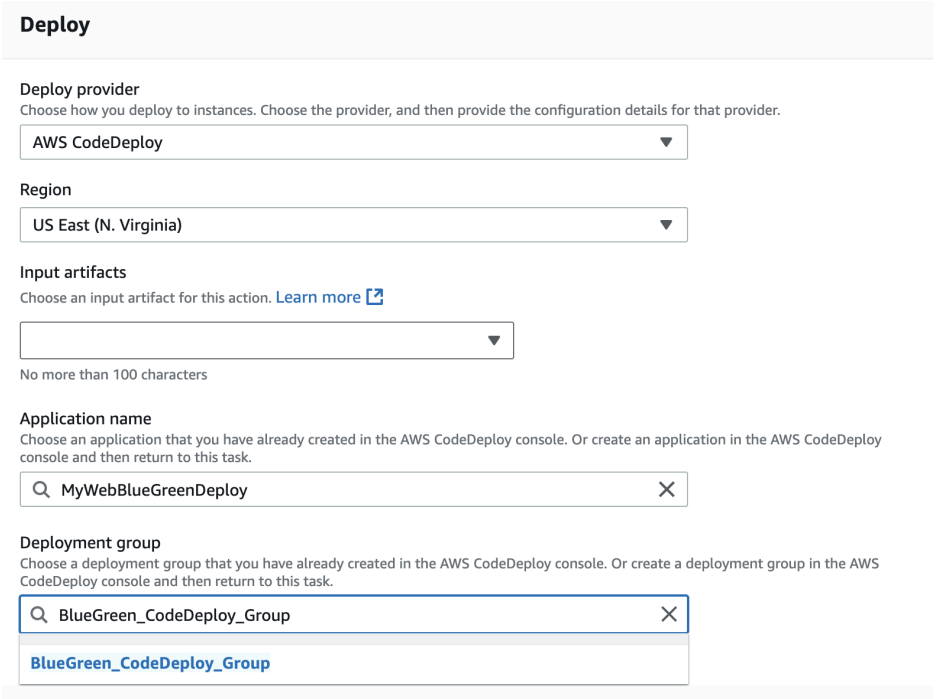
- After that give GitHub repo name



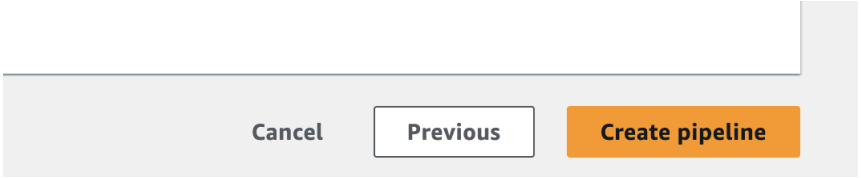
- After that skip the build step



- After that give the deploy provider, application name and deployment group name

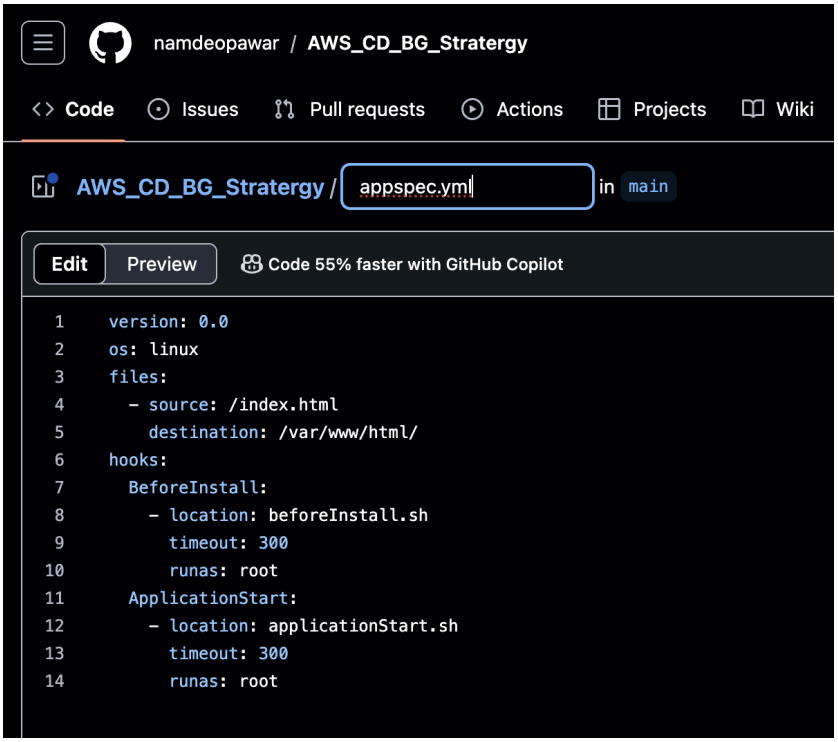


- After that click on the Create pipeline

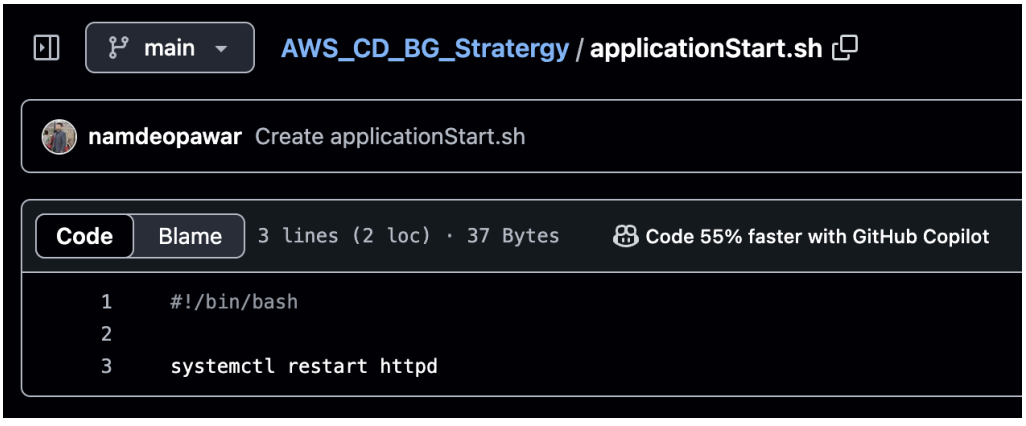


- After that you run the pipeline they fail because we don’t have the build spec and app spec files in the GitHub repo
- Note:- Building spec and app spec files is a very important thing

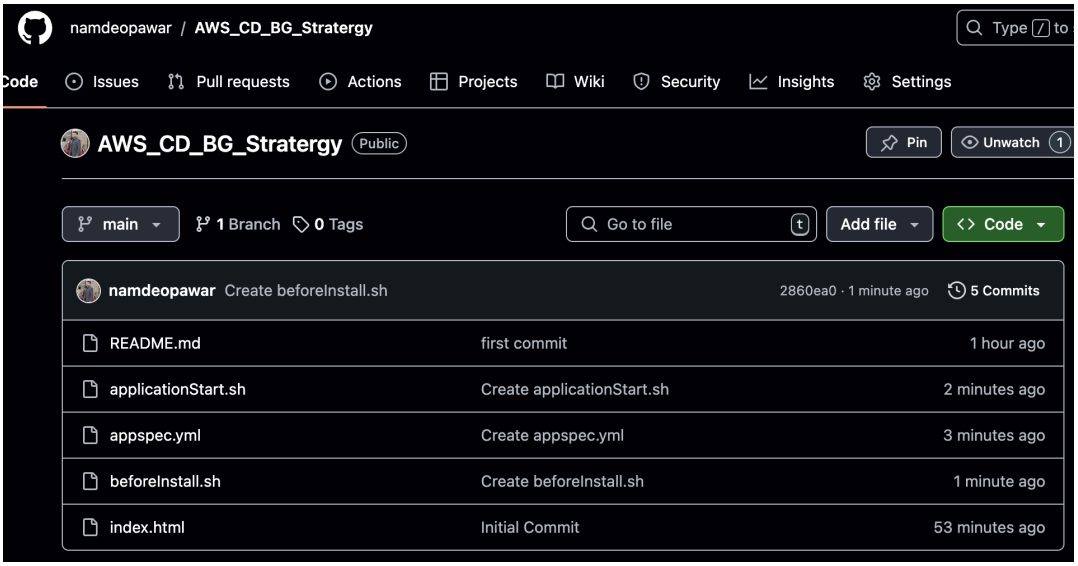
- Now i create an app spec file and upload on GitHub



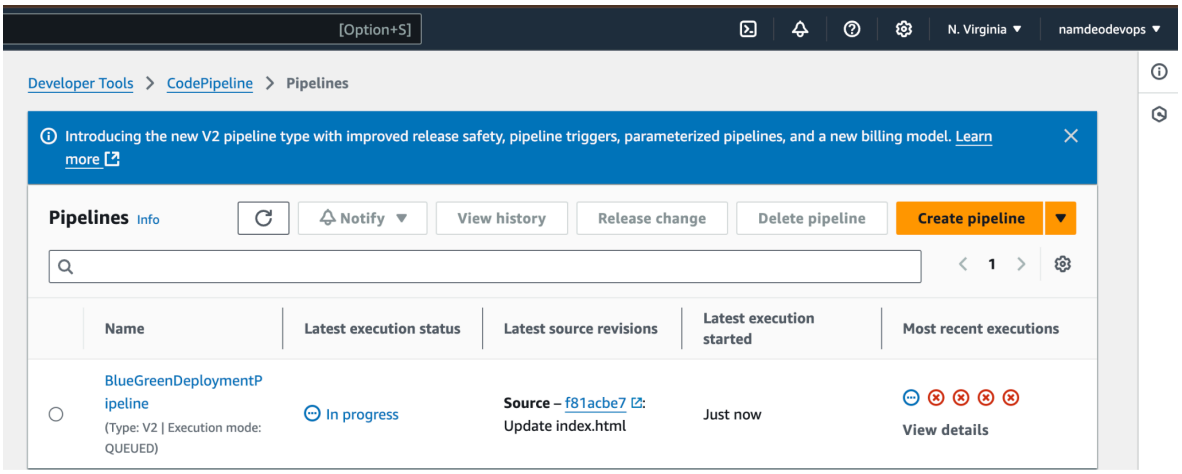
- But we need to create two more files in Git Hub one is before install.sh and application start.sh



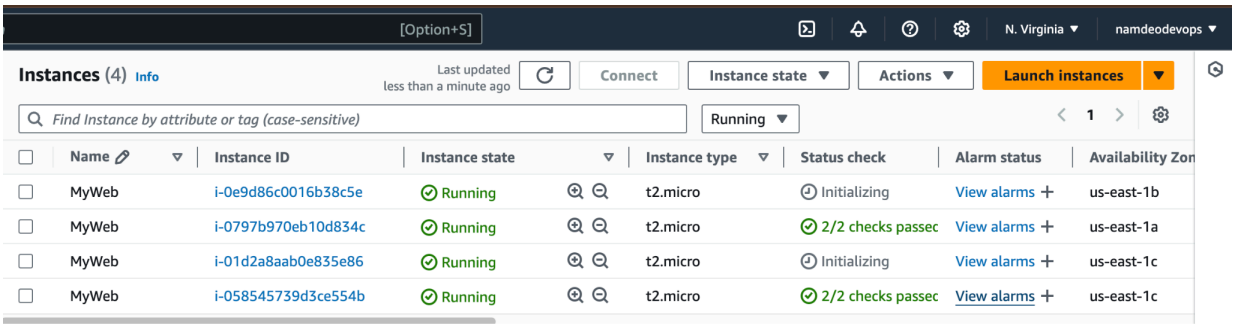
- Now i have all the files



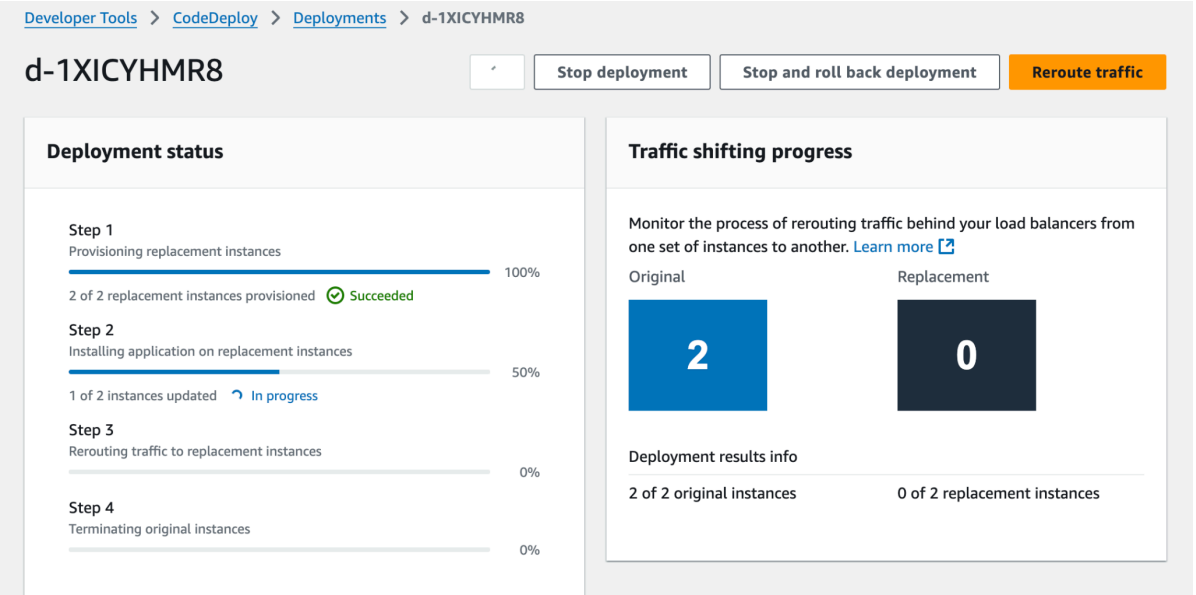
- Now as i see our pipeline start automatically



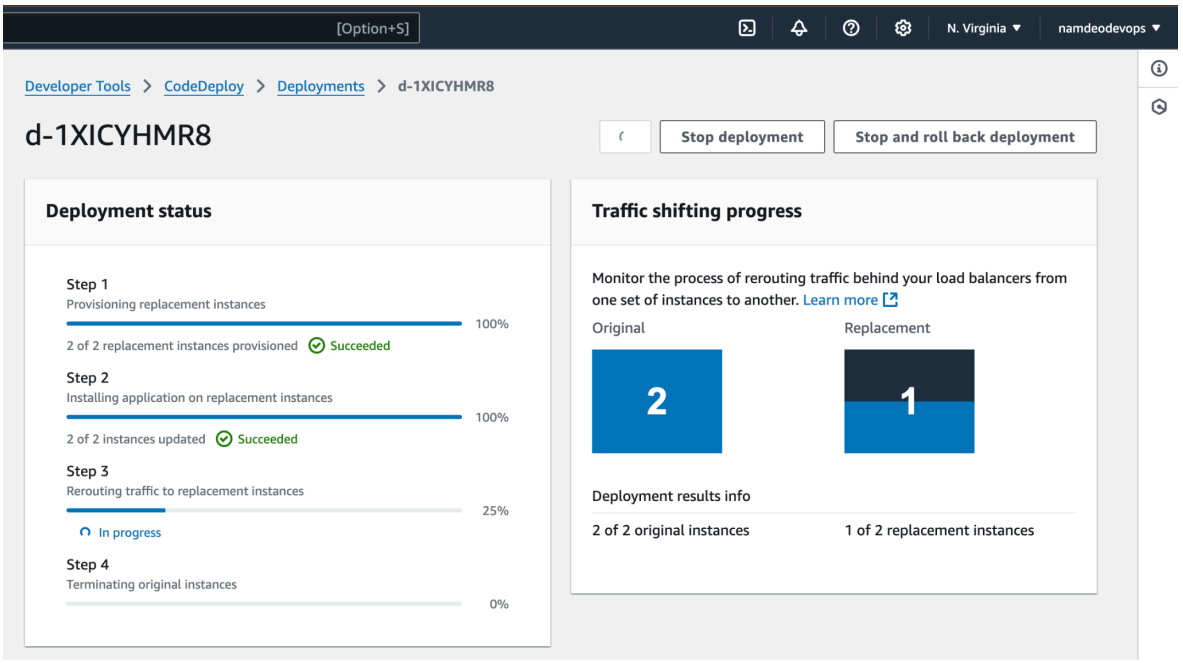
- As i can see the launch of two new instances

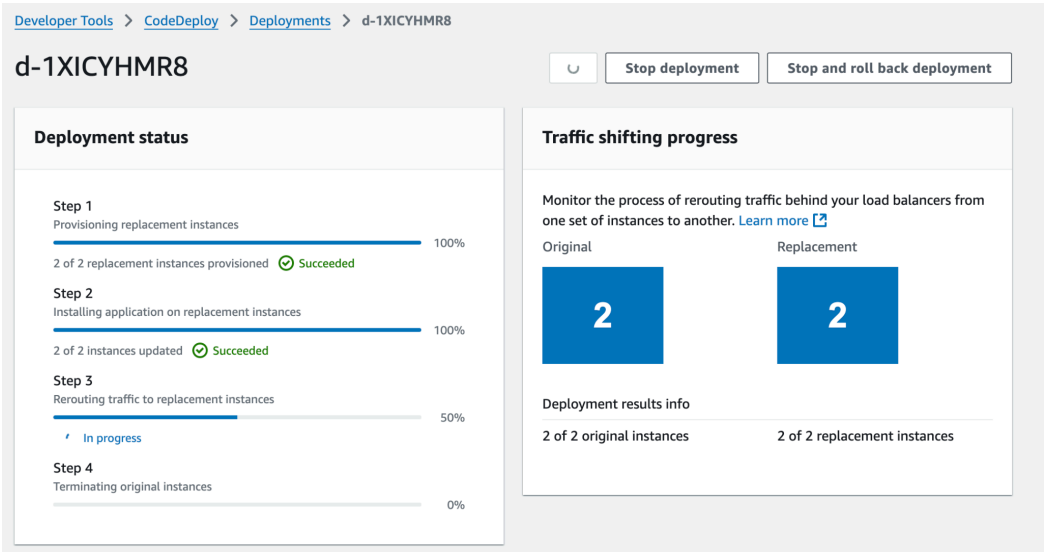


- Now we see they slowly replace the instance

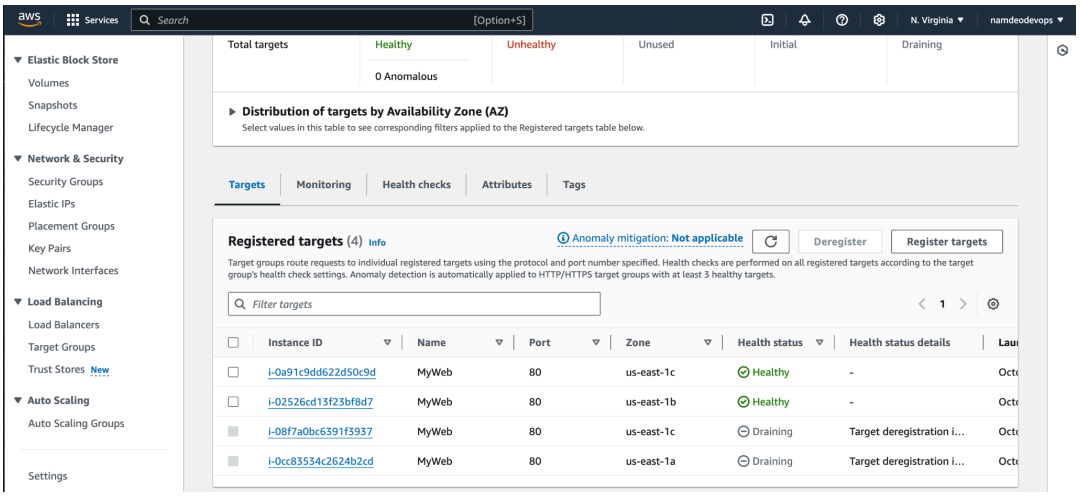


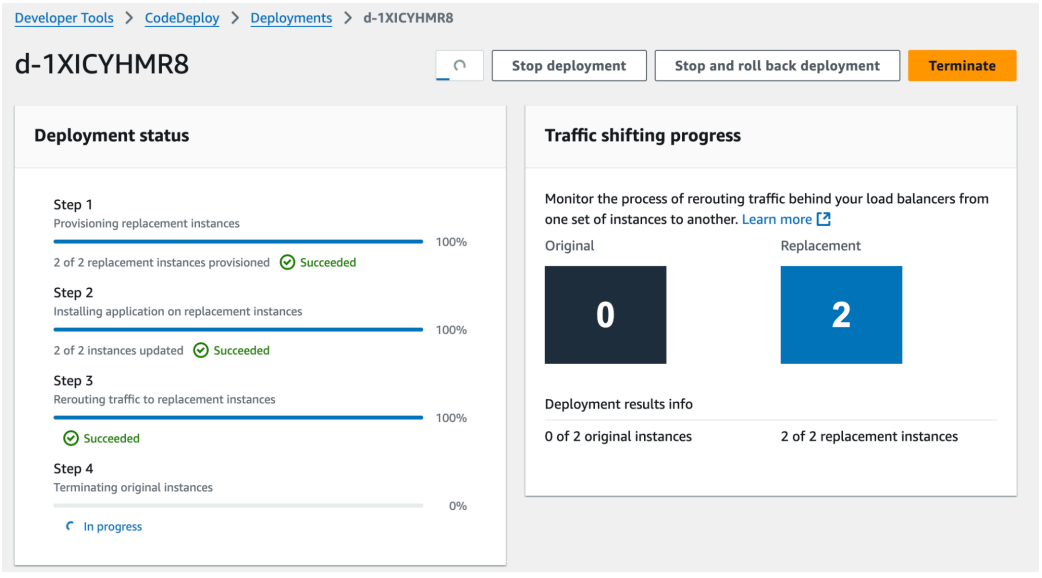
- As we can see they change the traffic from the blue to green





- After some time they stop the old instances and shift all traffic Into the new instances





Deployment lifecycle events

Instance ID	Environment	Traffic	Duration	Status	Most recent event	Events	Start time
i-02526cd13f23bf8d7	Replacement	Yes	4 minutes 34 seconds	Succeeded	AfterAllowTraffic	View events	Oct 6, 202
i-08f7a0bc6391f3937	Original	No	5 minutes 11 seconds	Succeeded	AfterBlockTraffic	View events	Oct 6, 202
i-0a91c9dd622d50c9d	Replacement	Yes	7 minutes 4 seconds	Succeeded	AfterAllowTraffic	View events	Oct 6, 202
i-0cc83534c2624b2cd	Original	No	5 minutes 11 seconds	Succeeded	AfterBlockTraffic	View events	Oct 6, 202

d-1XICYHMR8

Copy deployment | Retry deployment

Deployment status

Step 1
Provisioning replacement instances
100%
2 of 2 replacement instances provisioned Succeeded

Step 2
Installing application on replacement instances
100%
2 of 2 instances updated Succeeded

Step 3
Rerouting traffic to replacement instances
100%
Succeeded

Step 4
Terminating original instances
100%
2 of 2 original instances terminated Succeeded

Traffic shifting progress

Monitor the process of rerouting traffic behind your load balancers from one set of instances to another. Learn more

Original	Replacement
0	2

Deployment results info

0 of 2 original instances	2 of 2 replacement instances