

## 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.

## EC2 launch templates

Streamline, simplify and standardize instance launches

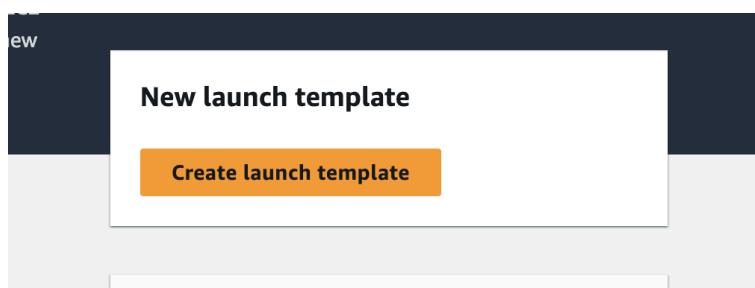
Use launch templates to automate instance launches, simplify permission policies, and enforce best practices across your organization. Save launch parameters in a template that can be used for on-demand launches and with managed services, including EC2 Auto Scaling and EC2 Fleet. Easily update your launch parameters by creating a new launch template version.

- **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

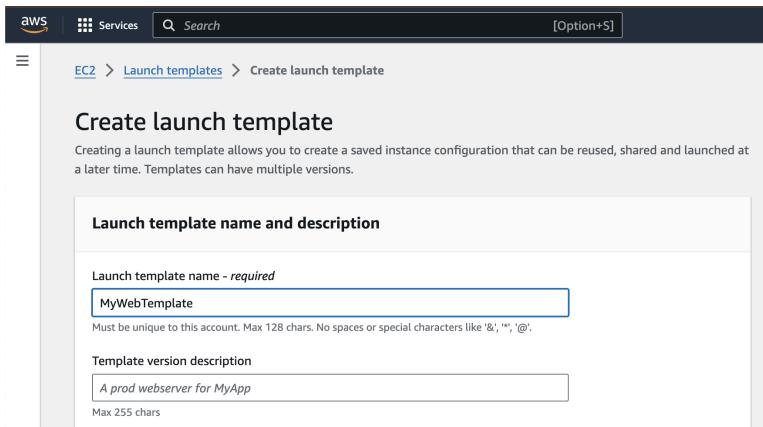
▼ Instances

- Instances
- Instance Types
- Launch Templates**
- Spot Requests
- Savings Plans
- Reserved Instances
- Dedicated Hosts
- Capacity Reservations

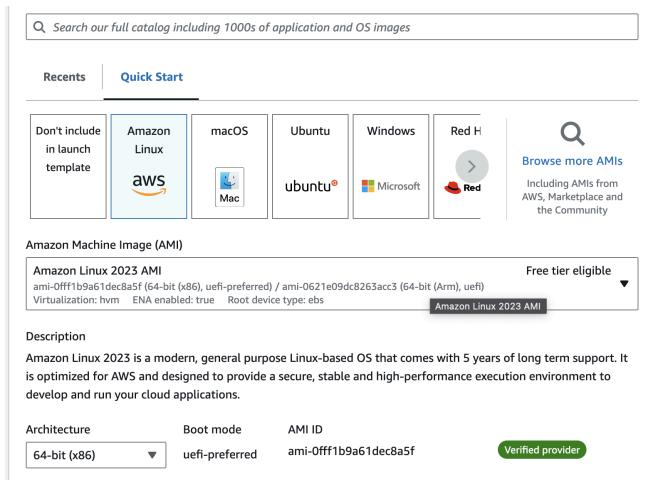
- After that click on the Create Launch template



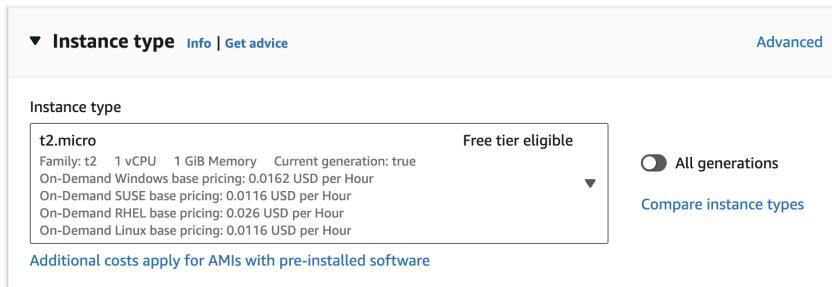
- After that give the name of a template



- 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.

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name

nama-devops

[Create new key pair](#)

- 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

Subnet [Info](#)

subnet-025f8d09570dd9d39

VPC: vpc-0391286238f54dc23 Owner: 058264258551

Availability Zone: us-east-1a Zone type: Availability Zone

IP addresses available: 4091 CIDR: 172.31.0.0/20

[Create new subnet](#)

When you specify a subnet, a network interface is automatically added to your template.

- 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).

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.

Select existing security group

Create security group

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

Security group name - *required*

MyWebSG-SSH-HTTP

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and \_-:/()\#,@[]+=;&;!\$\*

Description - *required* [Info](#)

Allow SSH & HTTP

- After that i allow the firewall rules

Inbound Security Group Rules

▼ Security group rule 1 (TCP, 22, 0.0.0.0/0, Allow SSH)

Type | Info      Protocol | Info      Port range | Info  
ssh      TCP      22

Source type | Info      Source | Info      Description - optional | Info  
Anywhere      Add CIDR, prefix list or security      Allow SSH  
0.0.0.0/0 X

▼ Security group rule 2 (TCP, 80, 0.0.0.0/0, Allow HTTP)

Type | Info      Protocol | Info      Port range | Info  
HTTP      TCP      80

Source type | Info      Source | Info      Description - optional | Info  
Anywhere      Add CIDR, prefix list or security      Allow HTTP  
0.0.0.0/0 X

⚠️ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only. X

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

▼ Advanced details | Info

IAM instance profile | Info  
MyEC2RoleToAllowAll  
arn:aws:iam::058264258551:instance-profile/MyEC2RoleToAllowAll

Create new IAM profile C

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

Instance auto-recovery | Info  
Don't include in launch template

- **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

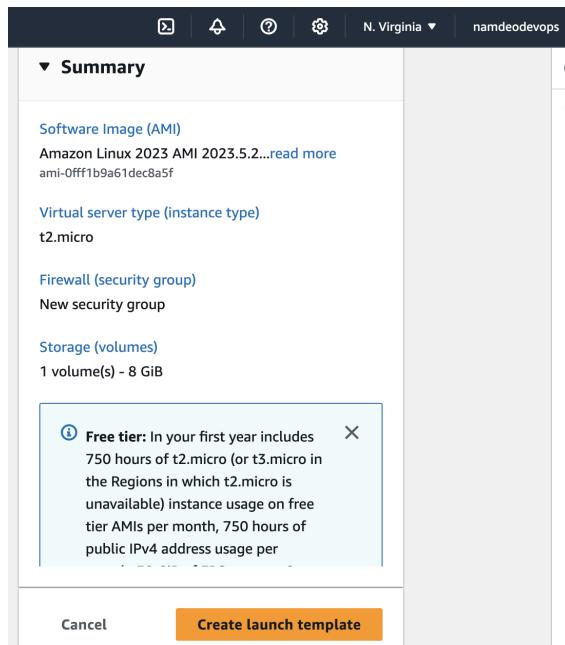
yum install ruby wget -y

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

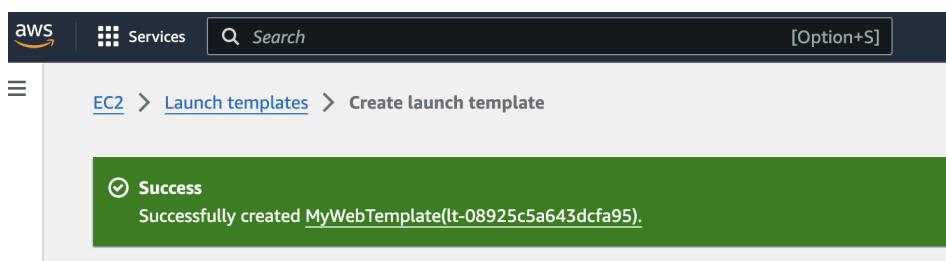
chmod +x ./install
./install auto
```

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

The screenshot shows the AWS Management Console navigation bar. The 'Services' button is selected, revealing a dropdown menu. The 'EC2' option is highlighted. Below it, the 'Launch templates' and 'Launch instance from template' options are visible. The URL in the address bar is also shown: EC2 > Launch templates > Launch instance from template.

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

The screenshot shows the 'Launch instance from template' configuration page. The title is 'Launch instance from template'. A note says: 'Launching from a template allows you to launch from an instance configuration that you would have saved in the past. These saved configurations can be reused and shared with other users to standardize launches across an organization.' The main section is titled 'Choose a launch template'. It shows two dropdown menus: 'Source template' containing 'MyWebTemplate' (ID: lt-08925c5a643dcfa95) and 'Version' containing '1 (Default)'.

- Then click on the launch instance

The screenshot shows the 'Launch instance' summary configuration page. The top navigation bar includes icons for refresh, alert, help, and settings, followed by 'N. Virginia' and 'namdeodevops'. The main area is titled 'Summary' and contains the following configuration details:

- Number of instances**: 1
- Software Image (AMI)**: Amazon Linux 2023 AMI 2023.5.2... [read more](#)
- Virtual server type (instance type)**: t2.micro
- Firewall (security group)**: MyWebSG-SSH-HTTP
- Storage (volumes)**: 1 volume(s) - 8 GiB

A callout box highlights the **Free tier** information: 'In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable) instance usage on free'.

At the bottom, there are 'Cancel', 'Launch instance' (in a blue button), and 'Review commands' links.

The screenshot shows a green success message box with the text "Success" and "Successfully initiated launch of instance (i-0dba3280ac31ad4cc)". Below this, there is a section titled "Launch log" with two entries: "Initializing requests" and "Launching instance from template", both marked as "Succeeded".

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

The screenshot shows the AWS EC2 Instances page with one instance listed. The instance ID is i-0dba3280ac31ad4cc, it is named "MyWebTemplate", its state is "Running", and its instance type is "t2.micro". It is currently initializing.

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

The screenshot shows the AWS Launch Templates page with one template listed. The template ID is lt-08925c5a643dcfa95, it is named "MyWebTemplate", and it is the latest version.

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

The screenshot shows the AWS Launch Templates page with the "Actions" dropdown menu open. The "Modify template (Create new version)" option is selected. Other options in the menu include "Launch instance from template", "Delete template", "Delete template version", "Set default version", "Manage tags", "Create Spot Fleet", "Create Auto Scaling group", and "View details".

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

**Resource tags**

Key	Value	Resource types
Name	MyWeb	Select resource ty...
Env	Prod	Select resource ty...

Add new tag

You can add up to 48 more tags.

- After that click on the Create template

**Success**

Successfully created [MyWebTemplate\(lt-08925c5a643dcfa95\)](#).

**Actions log**

Action	Status
Initializing requests	Succeeded
Create Launch Template	Succeeded

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

**Launch Templates (1/1)**

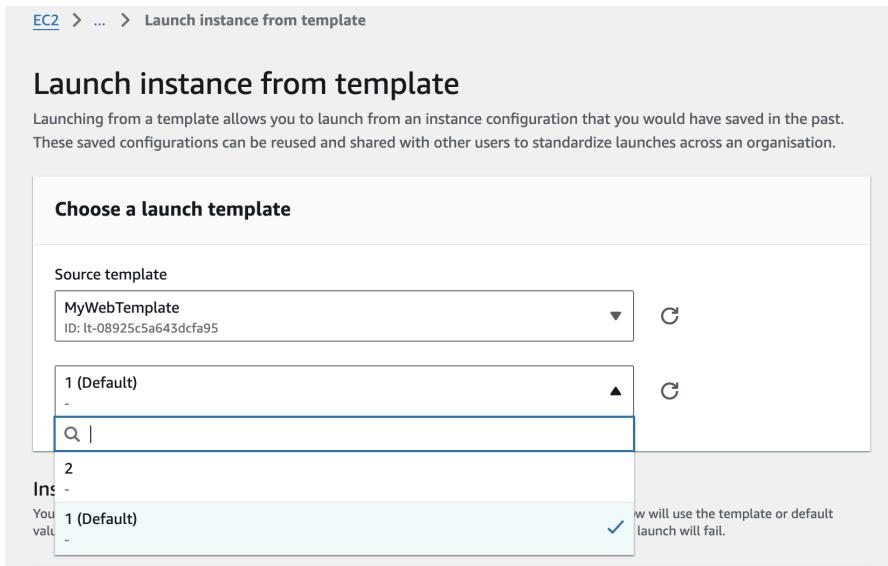
Launch Template ID	Launch Template Name	Default Version	Latest Version	Create Time
lt-08925c5a643dcfa95	MyWebTemplate	1	2	2024-10-06T08:19:44.000Z

- 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

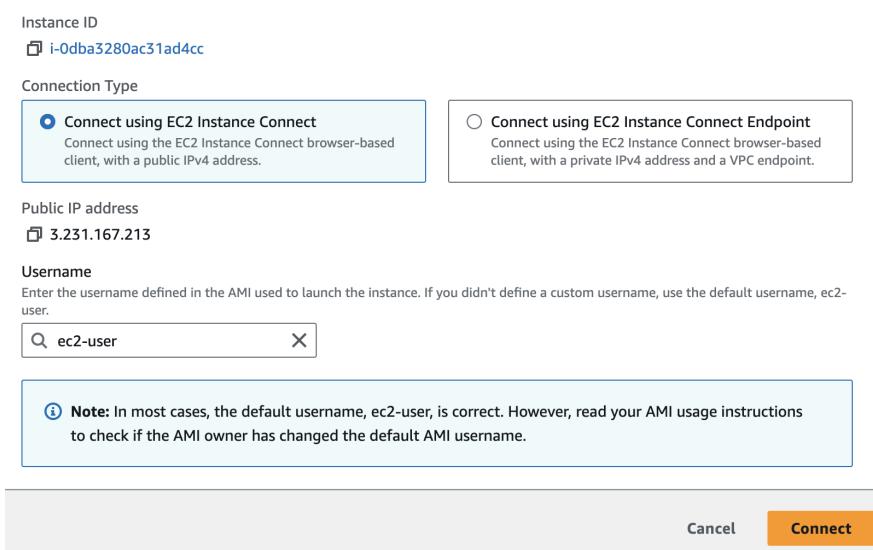
**Actions**

- Launch instance from template
- Modify template (Create new version)
- Delete template
- Delete template version
- Set default version
- Manage tags
- Create Spot Fleet
- Create Auto Scaling group
- View details

- 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

```

aws | Services | Search [Option+S]
#               #_
#_ \_###_      Amazon Linux 2023
-- \_###\_
-- \###|
--   \#/   https://aws.amazon.com/linux/amazon-linux-2023
--     V-.'->
---   /
~~.-./'
~/m/,'_
[ec2-user@ip-172-31-12-108 ~]$ sudo su -
[root@ip-172-31-12-108 ~]# rpm -q httpd
httpd-2.4.62-1.amzn2023.x86_64
[root@ip-172-31-12-108 ~]# cd /var/www/html/
[root@ip-172-31-12-108 html]# ls
index.html
[root@ip-172-31-12-108 html]# cat index.html
Welcome to My Blue-Green Deployment Project Version - 1
[root@ip-172-31-12-108 html]# 

```

```

aws | Services | Search [Option+S]
[root@ip-172-31-12-108 html]# systemctl status httpd
● httpd.service - The Apache HTTP Server
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; preset: disabled)
  Active: active (running) since Sun 2024-10-06 08:27:47 UTC; 17min ago
    Docs: man:httpd.service(8)
  Main PID: 3917 (httpd)
    Status: "Total requests: 1; Idle/Busy workers 100/0;Requests/sec: 0.000943; Bytes served/sec: 0 B/sec"
      Tasks: 177 (limit: 1112)
     Memory: 13.2M
        CPU: 677ms
       CGroup: /system.slice/httpd.service
           ├─3917 /usr/sbin/httpd -DFOREGROUND
           ├─3975 /usr/sbin/httpd -DFOREGROUND
           ├─3979 /usr/sbin/httpd -DFOREGROUND
           ├─3980 /usr/sbin/httpd -DFOREGROUND
           └─3981 /usr/sbin/httpd -DFOREGROUND

Oct 06 08:27:47 ip-172-31-12-108.ec2.internal systemd[1]: Starting httpd.service - The Apache HTTP Server...
Oct 06 08:27:47 ip-172-31-12-108.ec2.internal systemd[1]: Started httpd.service - The Apache HTTP Server.
Oct 06 08:27:47 ip-172-31-12-108.ec2.internal httpd[3917]: Server configured, listening on: port 80
[root@ip-172-31-12-108 html]# 

```

```

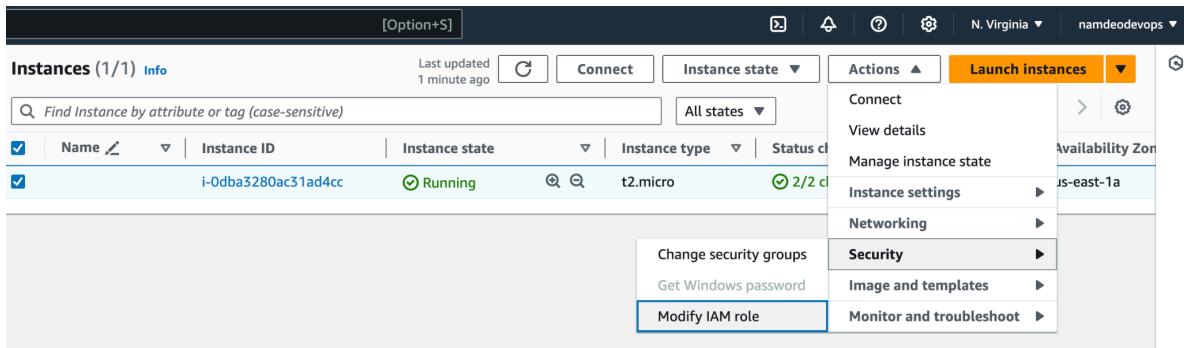
aws | Services | Search [Option+S]
[root@ip-172-31-12-108 html]# systemctl status codedeploy-agent.service
● codedeploy-agent.service - AWS CodeDeploy Host Agent
  Loaded: loaded (/usr/lib/systemd/system/codedeploy-agent.service; enabled; preset: disabled)
  Active: active (running) since Sun 2024-10-06 08:27:55 UTC; 18min ago
  Main PID: 13659 (ruby)
    Tasks: 3 (limit: 1112)
   Memory: 67.7M
      CPU: 1.093s
     CGroup: /system.slice/codedeploy-agent.service
             ├─13659 "codedeploy-agent: master 13659"
             └─13693 "codedeploy-agent: InstanceAgent::Plugins::CodeDeployPlugin::CommandPoller of master 13659"

Oct 06 08:27:54 ip-172-31-12-108.ec2.internal systemd[1]: Starting codedeploy-agent.service - AWS CodeDeploy Host Agent...
Oct 06 08:27:55 ip-172-31-12-108.ec2.internal systemd[1]: Started codedeploy-agent.service - AWS CodeDeploy Host Agent.
[root@ip-172-31-12-108 html]# 

```

- 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



[EC2](#) > [Instances](#) > [i-0dba3280ac31ad4cc](#) > Modify IAM role

### Modify IAM role [Info](#)

Attach an IAM role to your instance.

Instance ID  
i-0dba3280ac31ad4cc

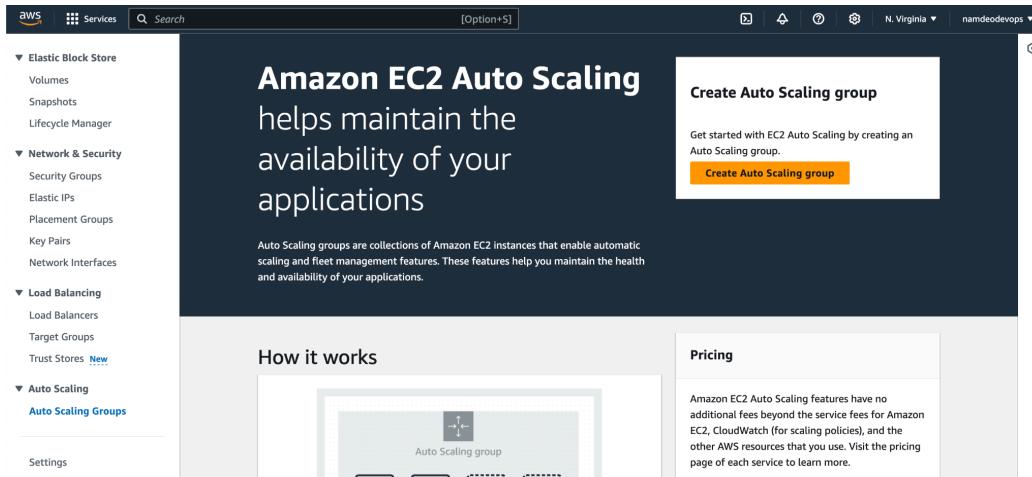
IAM role  
Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

MyEC2RoleToAllowAll [Create new IAM role](#)

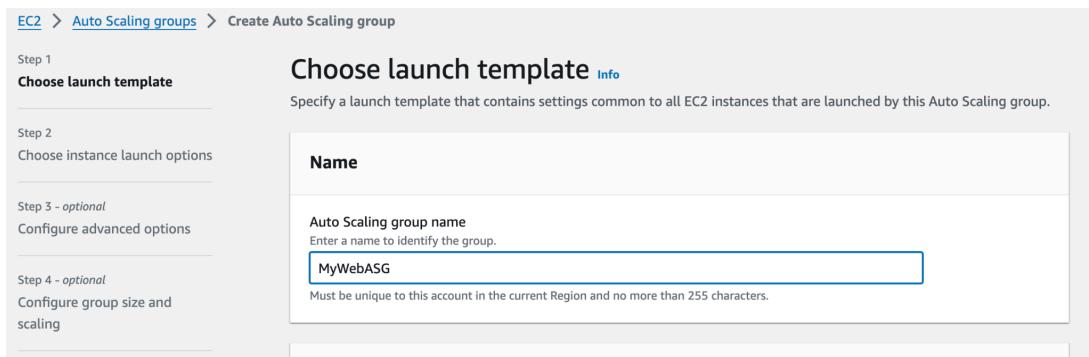
[Cancel](#) [Update IAM role](#)

- 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.

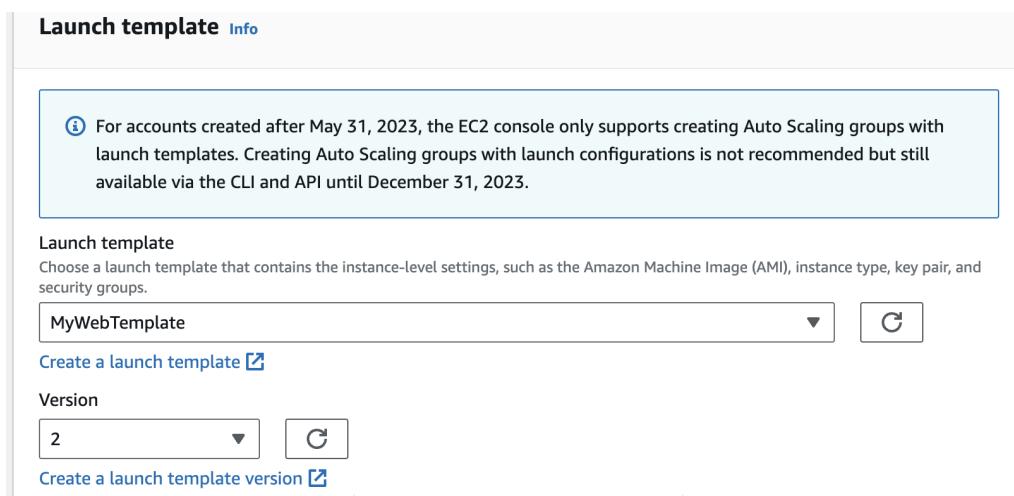
- To create an ASG go to the instance and scroll down then find Auto Scaling Group after that click on the Auto Scaling Group



- 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

**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-0391286238f54dc23 ▾ C

[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 ▾ C

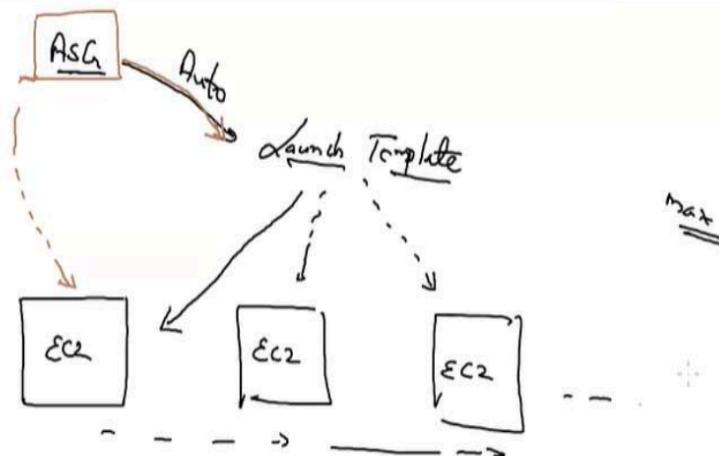
us-east-1a | subnet-025f8d09570dd9d39 X  
172.31.0.0/20 Default

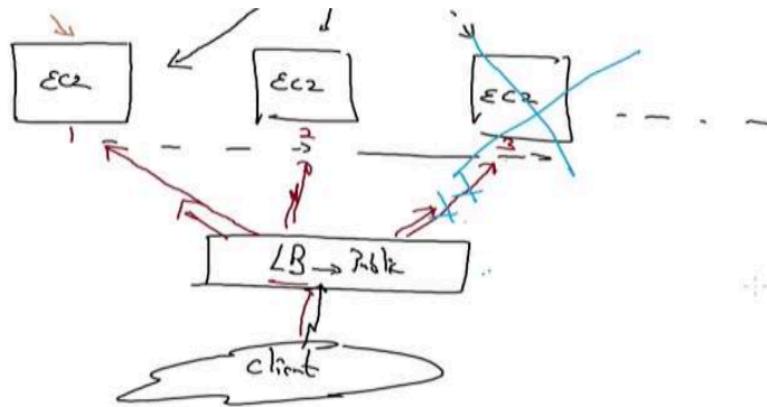
us-east-1b | subnet-0bdb81d88048effd9 X  
172.31.80.0/20 Default

us-east-1c | subnet-04f280d5e5b0865da X  
172.31.16.0/20 Default

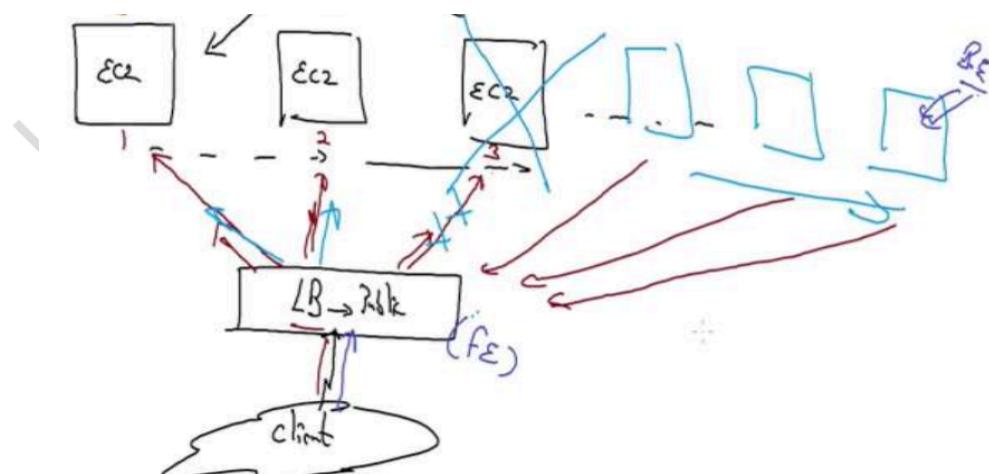
[Create a subnet](#)

- 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

**Load balancing Info**

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.

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

#### Load balancer type

Choose from the load balancer types offered below. Type selection cannot be changed after the load balancer is created. If you need a different type of load balancer than those offered here, [visit the Load Balancing console](#).

Application Load Balancer  
HTTP, HTTPS

Network Load Balancer  
TCP, UDP, TLS

#### Load balancer name

Name cannot be changed after the load balancer is created.

MyWebASG-1

#### Load balancer scheme

Scheme cannot be changed after the load balancer is created.

Internal

Internet-facing

#### Network mapping

Your new load balancer will be created using the same VPC and Availability Zone selections as your Auto Scaling group. You can select different subnets and add subnets from additional Availability Zones.

- After that select the subnet

#### Availability Zones and subnets

You must select a single subnet for each Availability Zone enabled. Only public subnets are available for selection to support DNS resolution.

us-east-1c

subnet-04f280d5e5b0865da

us-east-1a

subnet-025f8d09570dd9d39

us-east-1b

subnet-0bdb81d88048effd9

- After that i create a target group

#### Listeners and routing

If you require secure listeners, or multiple listeners, you can configure them from the [Load Balancing console](#) after your load balancer is created.

Protocol	Port	Default routing (forward to)
HTTP	80	<a href="#">Create a target group</a> <input type="button" value="New target group name"/> <small>An instance target group with default settings will be created.</small> <input type="text" value="MyWebASG-1"/>

#### Tags - optional

Consider adding tags to your load balancer. Tags enable you to categorize your AWS resources so you can more easily manage them.

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

#### EC2 health checks

[\(i\) Always enabled](#)

#### Additional health check types - optional

[Info](#)

[Turn on Elastic Load Balancing health checks](#) [Recommended](#)

Elastic Load Balancing monitors whether instances are available to handle requests. When it reports an unhealthy instance, EC2 Auto Scaling can replace it on its next periodic check.

[\(i\) EC2 Auto Scaling will start to detect and act on health checks performed by Elastic Load Balancing.](#)

To avoid unexpected terminations, first verify the settings of these health checks in the [Load Balancer console](#).

X

- After that i change the desired capacity

#### 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

- Now Give the Scaling

**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

2

Equal or less than desired capacity

Max desired capacity

5

Equal or greater than desired capacity

- After that click on the Create Auto Scaling Group

**Tag new instances**

No tags

**Create Auto Scaling group**

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

**EC2 > Auto Scaling groups**

Auto Scaling groups (1/1) <a href="#">Info</a>							<a href="#">Actions</a>	<a href="#">Create Auto Scaling group</a>
<input type="text"/> Search your Auto Scaling groups							< 1 >	
Name	Launch template/configuration	Instances	Status	Desired capacity	Min	Max		
<a href="#">MyWebASG</a>	<a href="#">MyWebTemplate   Version 2</a>	0	Updating capacity...	2	2	5	<a href="#">Edit</a>	

- Also i create a load balancer

**EC2 > Load balancers**

**Load balancers (1)**

Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

**Filter load balancers**

Name	DNS name	State	VPC ID	Availability Zones	
<a href="#">MyWebASG-1</a>	<a href="#">MyWebASG-1-518453280...</a>	Provisioning.	vpc-0391286238f54dc...	3 Availability Zones	<a href="#">Edit</a>

- As we can see two new instances are launch

Instances (3) Info		Last updated less than a minute ago	Connect	Instance state	Actions	Launch instances
		Find Instance by attribute or tag (case-sensitive)		All states		
Instance state = running		Clear filters				
	Name	Instance ID	Instance state	Instance type	Status check	Alarm status
<input type="checkbox"/>	MyWeb	i-014d7da636d7792de	Running	t2.micro	2/2 checks passed	<a href="#">View alarms +</a>
<input type="checkbox"/>		i-0dba3280ac31ad4cc	Running	t2.micro	2/2 checks passed	<a href="#">View alarms +</a>
<input type="checkbox"/>	MyWeb	i-02d88bae97b44477f	Running	t2.micro	2/2 checks passed	<a href="#">View alarms +</a>

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

Instances (1/3) Info

Last updated 1 minute ago

Find Instance by attribute or tag (case-sensitive)

All states

Instance state = running

Clear filters

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
<input checked="" type="checkbox"/>	MyWeb	i-014d7da636d7792de	Running	t2.micro	2/2 checks passed	<a href="#">View alarms +</a>	us-east-1a
<input type="checkbox"/>		i-0dba3280ac31ad4cc	Running	t2.micro	2/2 checks passed	<a href="#">View alarms +</a>	us-east-1a
<input type="checkbox"/>	MyWeb	i-02d88bae97b44477f	Running	t2.micro	2/2 checks passed	<a href="#">View alarms +</a>	us-east-1c

**i-014d7da636d7792de (MyWeb)**

Details Status and alarms Monitoring Security Networking Storage Tags

Instance summary

Instance ID: i-014d7da636d7792de (MyWeb)

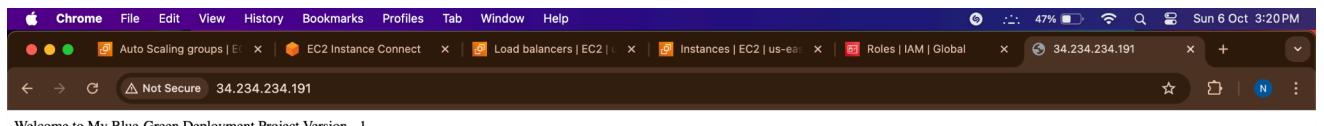
IPv6 address: -

Public IPv4 address copied: 34.234.234.191 | [open address](#)

Private IPv4 addresses: 172.31.15.133

Public IPv4 DNS: ec2-34-234-234-191.compute-1.amazonaws.com | [open address](#)

Instance state: Running



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

**Load balancer: MyWebASG-1**

Scheme	Hosted zone	Availability Zones	Date created
Internet-facing	Z355XDOTRQ7X7K	subnet-04f280d5e5b0865da (us-east-1c (use1-az4)) subnet-025f8d09570dd9d39 (us-east-1a (use1-az1)) subnet-0bdb81d88048effd9 (us-east-1b (use1-az2))	October 6, 2024, 15:13 (UTC+05:30)

Load balancer ARN: arn:aws:elasticloadbalancing:us-east-1:058264258551:loadbalancer/app/MyWebASG-1/bd35fe07cbf656a2

DNS name copied

MyWebASG-1-518453280.us-east-1.elb.amazonaws.com (A Record)

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

Inbound rules

Security group rule ID	Type	Protocol	Port range	Source	Description - optional
sgr-0faf6eccf15086686	All traffic	All	All	Custom	sg-0dc9d9bda2e6743d2
-	HTTP	TCP	80	Anywhere	0.0.0.0/0

Rules with source of 0.0.0.0/0 or ::/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only.

Add rule

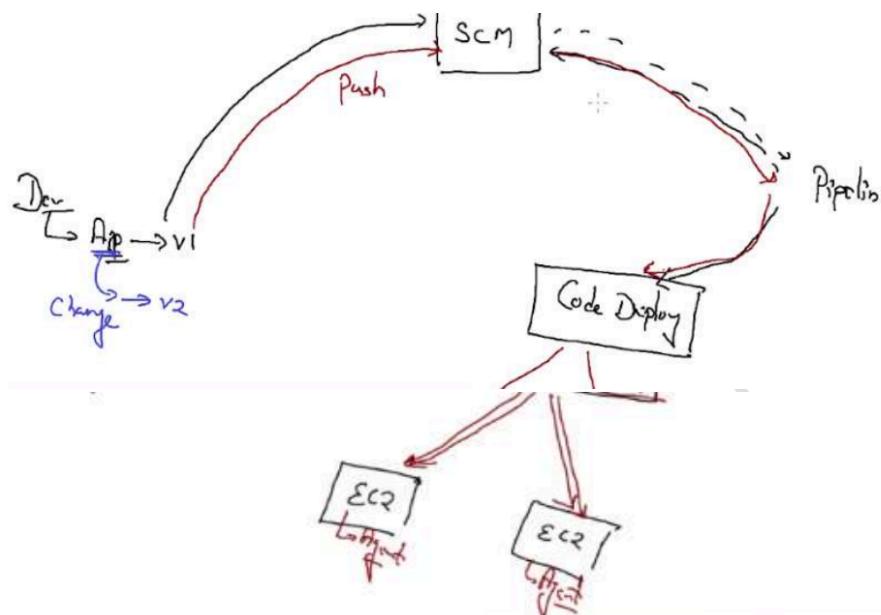
Cancel Preview changes Save rules

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

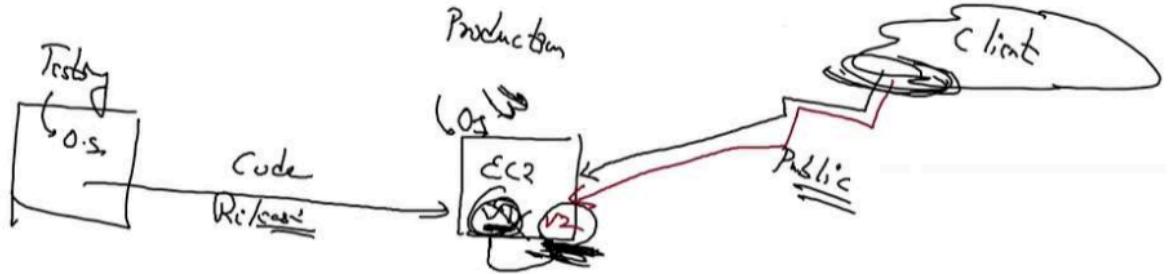
Welcome to My Blue-Green Deployment Project Version - 1

## Now we start our main concept of blue-green deployment

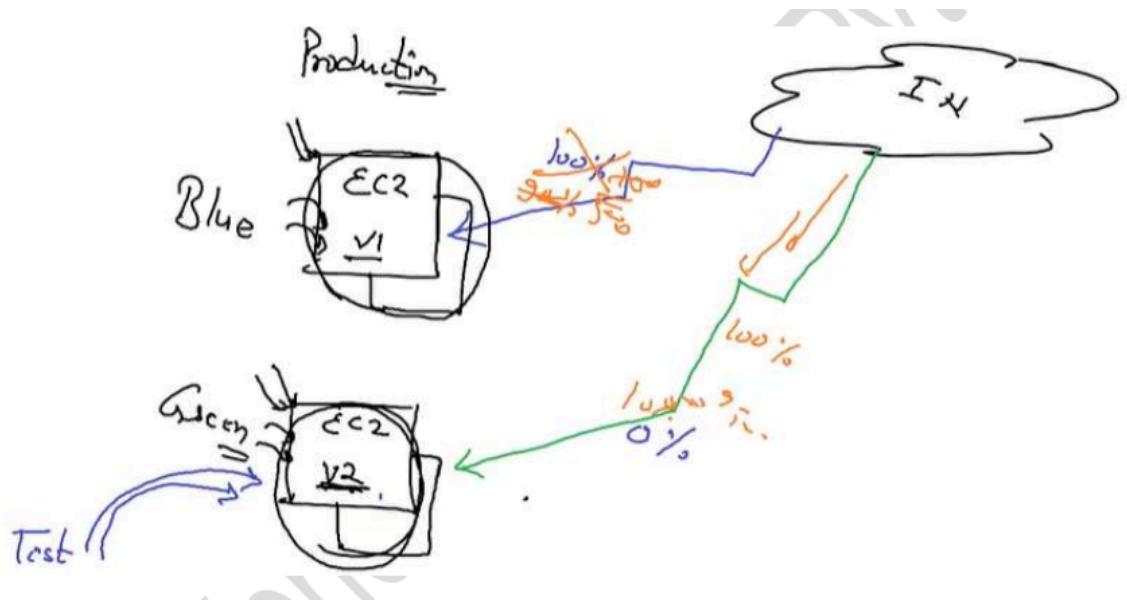
- First, we discuss a common scenario where a developer creates an app (in this case, an HTML page) and deploys it to EC2 instances using **CodeDeploy** as part of a CI/CD pipeline. When changes are made (new code versions), the pipeline fetches and deploys the new version to the EC2 instance
- The traditional 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

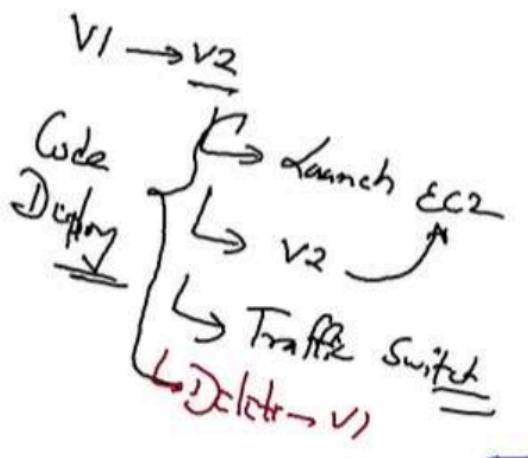


- **Risks of In-place Code Release:** Directly releasing new code to production (without careful traffic management) can lead to issues like session crashes, bugs, and problems with ongoing customer activities.
- **Blue-Green Deployment Strategy:**
  - The current running system (blue) handles all live 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

### Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository](#).

---

*Required fields are marked with an asterisk (\*)*.

<b>Owner *</b> <input style="border: 1px solid #ccc; width: 100%; height: 20px; border-radius: 5px; margin-bottom: 5px;" type="text" value="namdeopawar"/> <span style="font-size: small;">/</span>	<b>Repository name *</b> <input style="border: 1px solid #ccc; width: 100%; height: 20px; border-radius: 5px; margin-bottom: 5px;" type="text" value="AWS_CD_BG_Straterry"/> <span style="font-size: small;">✓ AWS_CD_BG_Straterry is available.</span>
Great repository names are short and memorable. Need inspiration? How about <a href="#">psychic-carnival</a> ?	
<b>Description (optional)</b> <input style="width: 100%; height: 40px; border: 1px solid #ccc; border-radius: 5px; margin-bottom: 5px;" type="text"/>	
<input checked="" type="radio"/> <b>Public</b> Anyone on the internet can see this repository. You choose who can commit.	
<input type="radio"/> <b>Private</b> You choose who can see and commit to this 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

```
Terminal Shell Edit View Window Help
code_deploy_blue_green --zsh
[nama@Pawar ~/Programs/AWS] 15:46 % pwd
/Users/nama/Programs/AWS
[nama@Pawar ~/Programs/AWS] 15:46 % mkdir code_deploy_blue_green
[nama@Pawar ~/Programs/AWS] 15:47 % cd code_deploy_blue_green
[nama@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:47 % 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
Initialized empty Git repository in /Users/nama/Programs/AWS/code_deploy_blue_green/.git/
[master (root-commit) b3c0504] first_commit
 1 file changed, 1 insertion(+)
 create mode 100644 README.md
Username for 'https://github.com/namdeopawar/AWS_CD_BG_Stratergy.git': namdeopawar
Password for 'https://namdeopawar@github.com/namdeopawar/AWS_CD_BG_Stratergy.git':
Enumerating objects: 3, done.
Counting objects: 100% (3/3), done.
Writing objects: 100% (3/3), 238 bytes | 238.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/namdeopawar/AWS_CD_BG_Stratergy.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
[nama@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:49 % ls
README.md
[nama@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:49 %
```

- After that write an index.html file

```
index.html
[nama@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:49 % vim index.html
[nama@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:50 % cat index.html
This is code change version -2
• After that write an index.html file
[nama@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:50 %
```

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

```
[namdeo@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:50 % ls
README.md index.html
[namdeo@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:52 % git add .
[namdeo@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:52 % git commit -m "Initial Commit"
[main cc05849] Initial Commit
 1 file changed, 1 insertion(+)
 create mode 100644 index.html
[namdeo@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:53 % git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 238 bytes | 238.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/namdeoapawar/AWS_CD_BG_Straterry.git
  * [new branch] main -> main
Branch 'main' set up to track 'origin/main'.
[namdeo@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:49 % ls
[namdeo@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:50 % git push
Enumerating objects: 4, done.
Counting objects: 100% (4/4), done.
Delta compression using up to 8 threads
Compressing objects: 100% (2/2), done.
Writing objects: 100% (3/3), 315 bytes | 315.00 KiB/s, done.
Total 3 (delta 0), reused 0 (delta 0), pack-reused 0 (from 0)
To https://github.com/namdeoapawar/AWS_CD_BG_Straterry.git
 b3c0504..cc05849 main -> main
[namdeo@Pawar ~/Programs/AWS/code_deploy_blue_green] 15:53 %
```

• After that write an index.html file

• After that we add, commit, and push the code on the GitHub

- As we can see I push code on GitHub

File	Commit Message	Time Ago
README.md	first commit	6 minutes ago
index.html	Initial Commit	1 minute ago

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

Application name	Compute platform	Created
MyWebBlueGreenDeploy		1 minute ago

- After that give the application name

Application name	Enter an application name MyWebBlueGreenDeploy
Compute platform	Choose a compute platform EC2/on-premises
Tags	Add tag

- After that I need to create a deployment group

The screenshot shows the AWS CodeDeploy console under the 'Deployment groups' tab. At the top, there are navigation tabs for 'Deployments', 'Deployment groups' (which is selected), and 'Revisions'. Below the tabs is a search bar and a set of buttons: 'View details', 'Edit', and a prominent orange 'Create deployment group' button. The main area displays a table with the following columns: Name, Status, Last attempted deploy..., Last successful deplo..., and Trigger count. There is one row visible in the table.

- After that give the deployment name

The screenshot shows a configuration step for creating a deployment group. The title is 'Deployment group name'. Below it is a text input field with the placeholder 'Enter a deployment group name'. The value 'BlueGreen\_CodeDeploy\_Group' is entered into the field. Below the input field is a note: '100 character limit'.

- After that select role

The screenshot shows a configuration step for selecting a service role. The title is 'Service role'. Below it is a text input field with the placeholder 'Enter a service role'. The value 'arn:aws:iam::058264258551:role/MyCodeDeployServiceRole' is entered into the field. To the right of the input field is a clear button represented by a 'X' symbol.

- After that click on the Blue-green deployment

The screenshot shows a configuration step for choosing a deployment type. The title is 'Deployment type'. Below it is a section titled 'Choose how to deploy your application'. It contains two options: 'In place' and 'Blue/green'. The 'Blue/green' option is selected, indicated by a blue circle and a checked radio button. A detailed description of the 'Blue/green' deployment type follows the radio button.

- Now we select automatically options

The screenshot shows a configuration step for environment setup. The title is 'Environment configuration'. Below it is a section titled 'Specify the Amazon EC2 Auto Scaling groups or Amazon EC2 instances where the current application revision is deployed.' It contains two options: 'Automatically copy Amazon EC2 Auto Scaling group' and 'Manually provision instances'. The 'Automatically copy Amazon EC2 Auto Scaling group' option is selected, indicated by a blue circle and a checked radio button. A detailed description of the 'Automatically copy' option follows the radio button.

- After that give deployment settings

**Deployment settings**

Traffic rerouting

Reroute traffic immediately  
 I will choose whether to reroute traffic

Days	Hours	Minutes
0	0	5

Choose whether instances in the original environment are terminated when the deployment succeeds, and how long to wait before termination.

Terminate the original instances in the deployment group  
 Keep the original instances in the deployment group running

Days	Hours	Minutes
0	1	0

**Deployment configuration**

Choose from a list of default and customised deployment configurations. A deployment configuration is a set of rules that determines how fast an application is deployed and the success or failure conditions for a deployment.

or

- After that select load balancer

**Load balancer**

Select a load balancer to manage incoming traffic during the deployment process. The load balancer blocks traffic from each instance while it's being deployed to and allows traffic to it again after the deployment succeeds.

Enable load balancing

Load Balancer type

Application Load Balancer or Network Load Balancer

Choose target groups

<input checked="" type="checkbox"/> MyWebASG-1	MyWebASG-1
<input type="checkbox"/> Classic Load Balancer	

- Then click on Create Deployment group

<input type="button" value="Cancel"/>	<input type="button" value="Create deployment group"/>
---------------------------------------	--

- After that we need to create a pipeline

**Choose pipeline settings** Info

Step 1 of 5

**Pipeline settings**

**Pipeline name**  
Enter the pipeline name. You cannot edit the pipeline name after it is created.  
 No more than 100 characters

**Pipeline type**  
① You can no longer create V1 pipelines through the console. We recommend you use the V2 pipeline type with improved release safety, pipeline triggers, parameterized pipelines, and a new billing model.

**Advanced settings**

**Artifact store**

**Default location**  
Create a default S3 bucket in your account.

**Custom location**  
Choose an existing S3 location from your account in the same region and account as your pipeline

**Encryption key**

**Default AWS Managed Key**  
Use the AWS managed customer master key for CodePipeline in your account to encrypt the data in the artifact store.

**Customer Managed Key**  
To encrypt the data in the artifact store under an AWS KMS customer managed key, specify the key ID, key ARN, or alias ARN.

**Cancel** **Next**

- After that select the source provider

**Source**

**Source provider**  
This is where you stored your input artifacts for your pipeline. Choose the provider and then provide the connection details.

**GitHub (Version 1)**

Grant AWS CodePipeline access to your GitHub repository. This allows AWS CodePipeline to upload commits from GitHub to your pipeline.

**Connect to GitHub**

**The GitHub (Version 1) action is not recommended**  
The selected action uses OAuth apps to access your GitHub repository. This is no longer the recommended method. Instead, choose the GitHub (Version 2) action to access your repository by creating a connection. Connections use GitHub Apps to manage authentication and can be shared with other resources. [Learn more](#)

**Change detection options**  
Choose a detection mode to automatically start your pipeline when a change occurs in the source code.

**GitHub webhooks (recommended)**  
Use webhooks in GitHub to automatically start my pipeline when a change occurs

**AWS CodePipeline**  
Use AWS CodePipeline to check periodically for changes

- After that give GitHub repo name

Repository  
namdeopawar/AWS\_CD\_BG\_Strategy

Branch  
main

Change detection options  
Choose a detection mode to automatically start your pipeline when a change occurs in the source code.

GitHub webhooks (recommended)  
Use webhooks in GitHub to automatically start my pipeline when a change occurs

AWS CodePipeline  
Use AWS CodePipeline to check periodically for changes

Cancel Previous Next

- After that skip the build step

Add build stage [Info](#)

Step 3 of 5

**Build - optional**

**Build provider**  
Choose the tool you want to use to run build commands and specify artifacts for your build action.

Commands     Other build providers

**Commands**  
Specify the shell commands to run with your compute action in CodePipeline. You do not need to create any resources in CodeBuild. Note: Using compute time for this action will incur separate charges in AWS CodeBuild.

```
ls
echo "Hello World"
```

**Input artifacts**  
Choose an input artifact for this action. [Learn more](#)

SourceArtifact [X](#)  
Defined by: Source

No more than 100 characters

Cancel Previous Skip build stage Next

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

**Deploy**

**Deploy provider**  
Choose how you deploy to instances. Choose the provider, and then provide the configuration details for that provider.

AWS CodeDeploy ▾

**Region**

US East (N. Virginia) ▾

**Input artifacts**  
Choose an input artifact for this action. [Learn more](#) ⓘ

No more than 100 characters ▾

**Application name**  
Choose an application that you have already created in the AWS CodeDeploy console. Or create an application in the AWS CodeDeploy console and then return to this task.

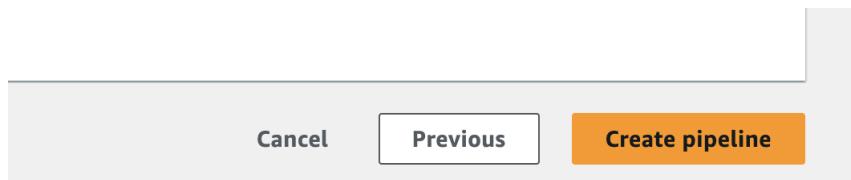
Q MyWebBlueGreenDeploy X

**Deployment group**  
Choose a deployment group that you have already created in the AWS CodeDeploy console. Or create a deployment group in the AWS CodeDeploy console and then return to this task.

Q BlueGreen\_CodeDeploy\_Group X

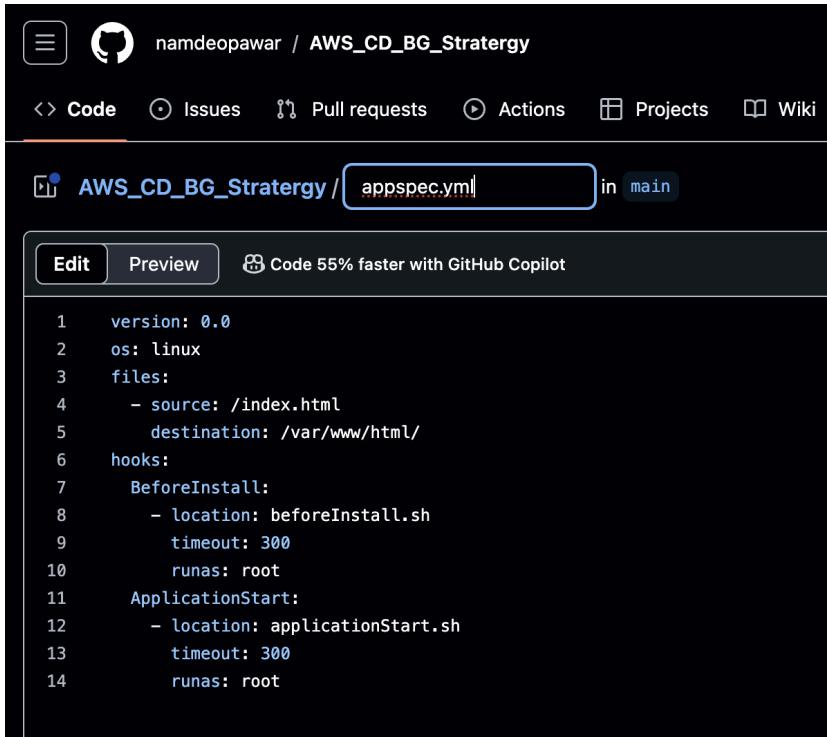
BlueGreen\_CodeDeploy\_Group

- 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



The screenshot shows a GitHub repository page for 'AWS\_CD\_BG\_Straterry'. The 'Code' tab is selected. A file named 'appspec.yml' is highlighted in the list. The code content is as follows:

```

1 version: 0.0
2 os: linux
3 files:
4   - source: /index.html
5     destination: /var/www/html/
6 hooks:
7   BeforeInstall:
8     - location: beforeInstall.sh
9       timeout: 300
10      runas: root
11 ApplicationStart:
12   - location: applicationStart.sh
13     timeout: 300
14     runas: root

```

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



The screenshot shows the 'applicationStart.sh' file in the 'AWS\_CD\_BG\_Straterry' repository. The 'Code' tab is selected. The code content is as follows:

```

#!/bin/bash
systemctl restart httpd

```



The screenshot shows the 'beforeInstall.sh' file in the 'AWS\_CD\_BG\_Straterry' repository. The 'Code' tab is selected. The code content is as follows:

```

#!/bin/bash
rm -rf /var/www/html/*

```

- Now i have all the files

The screenshot shows a GitHub repository page for 'AWS\_CD\_BG\_Straterry'. The repository has 1 branch and 0 tags. It contains several files:

- README.md: first commit, 1 hour ago
- applicationStart.sh: Create applicationStart.sh, 2 minutes ago
- appspec.yml: Create appspec.yml, 3 minutes ago
- beforeInstall.sh: Create beforeInstall.sh, 1 minute ago
- index.html: Initial Commit, 53 minutes ago

- Now as i see our pipeline start automatically

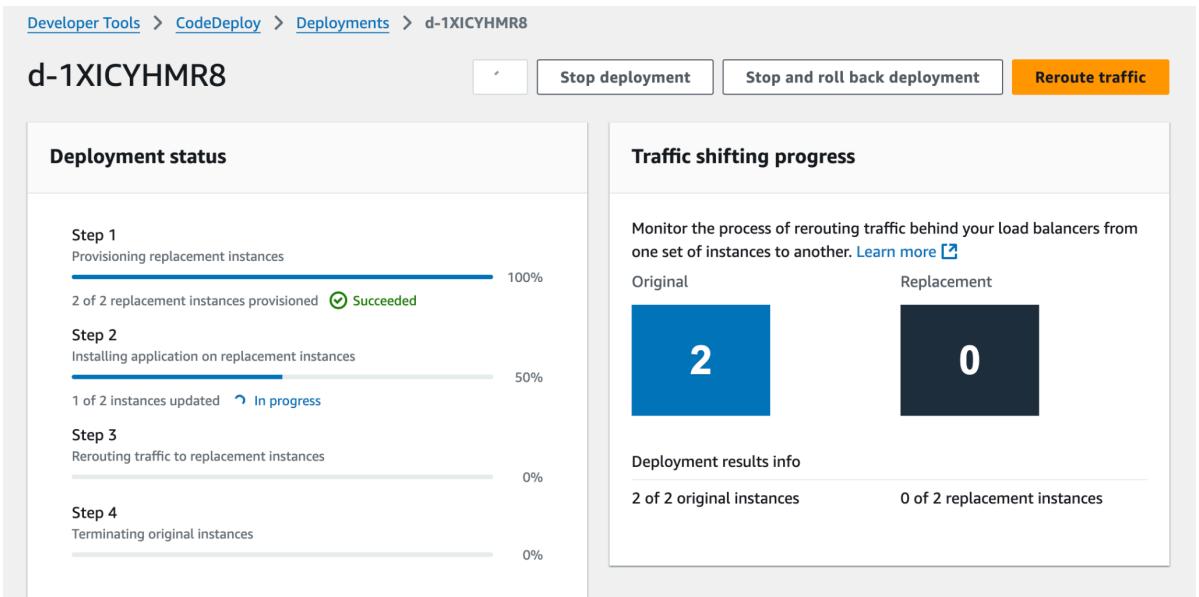
The screenshot shows the AWS CodePipeline console. A blue banner at the top introduces the new V2 pipeline type. The main interface shows a single pipeline named 'BlueGreenDeploymentPipeline' in the 'Info' tab. The pipeline is currently in 'In progress' status, triggered by 'Source - f81acbe7' (Update index.html) and started 'Just now'. There is a link to 'View details'.

- As i can see the launch of two new instances

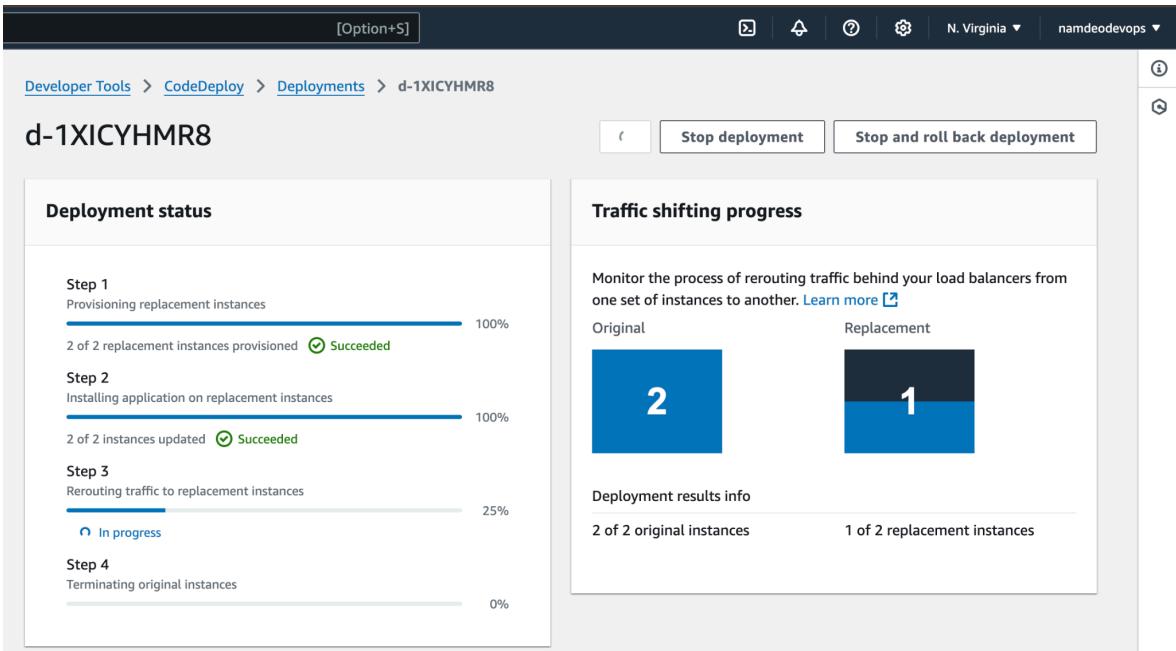
The screenshot shows the AWS Lambda console. It lists four instances of the function 'MyWeb' with the following details:

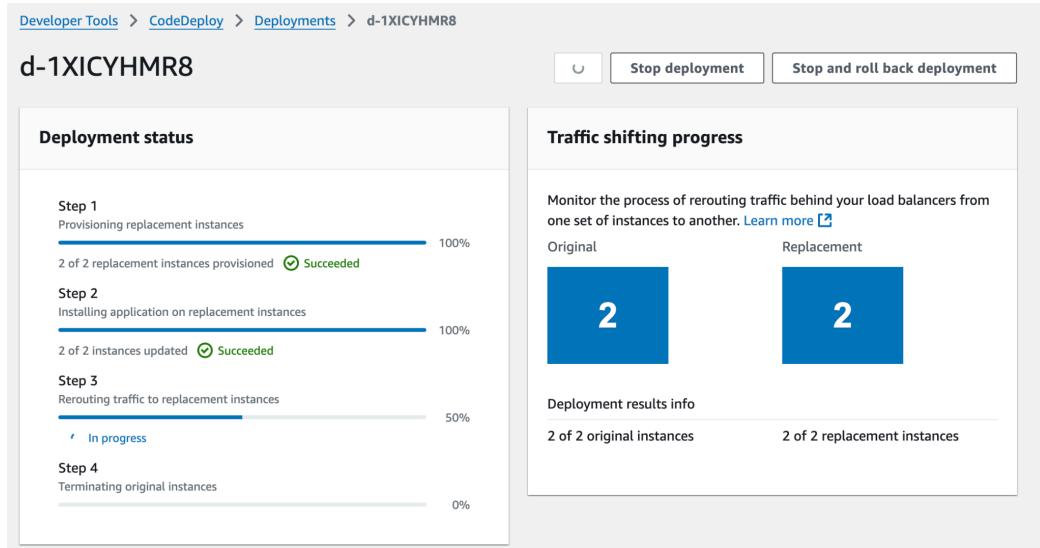
Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
MyWeb	i-0e9d86c0016b38c5e	Running	t2.micro	Initializing	View alarms +	us-east-1b
MyWeb	i-0797b970eb10d834c	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1a
MyWeb	i-01d2a8aab0e835e86	Running	t2.micro	Initializing	View alarms +	us-east-1c
MyWeb	i-058545739d3ce554b	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1c

- Now we see they slowly replace the instance

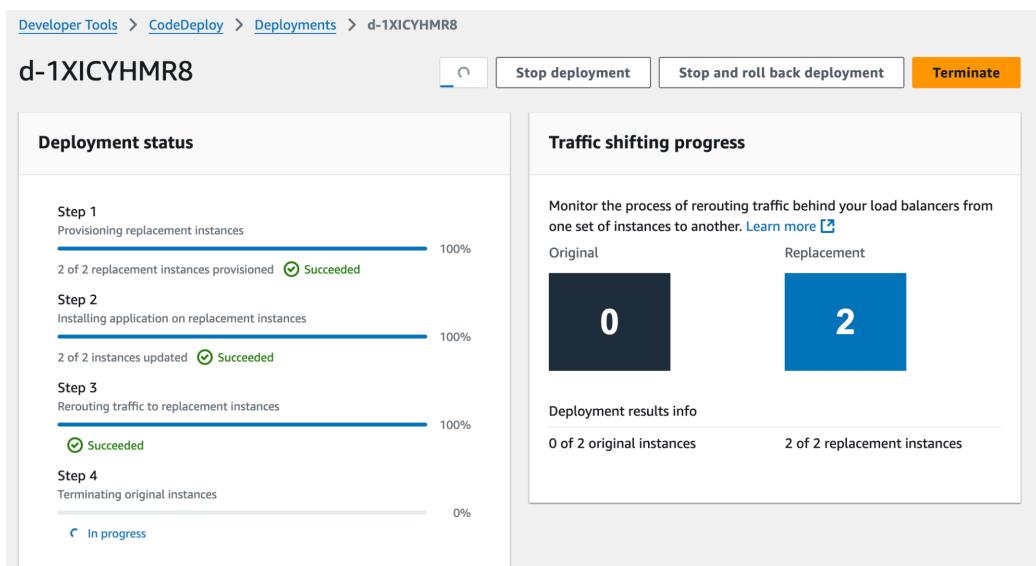
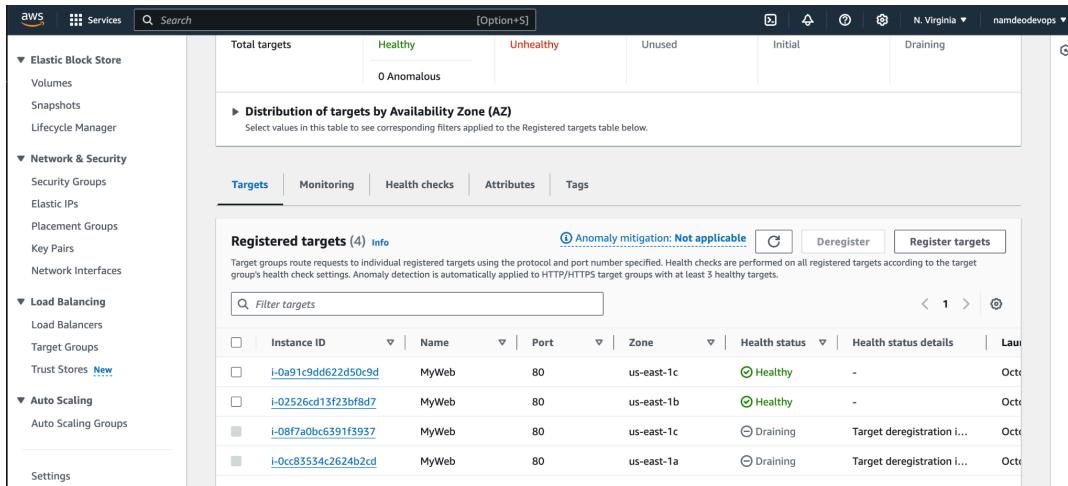


- As we can see they change the traffic from the blue to green





- 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 <a href="#">[?]</a>	Replacement	Yes	4 minutes 34 seconds	Succeeded	AfterAllowTraffic	<a href="#">View events</a>	Oct 6, 2022
i-08f7a0bc6391f3937 <a href="#">[?]</a>	Original	No	5 minutes 11 seconds	Succeeded	AfterBlockTraffic	<a href="#">View events</a>	Oct 6, 2022
i-0a91c9dd622d50c9d <a href="#">[?]</a>	Replacement	Yes	7 minutes 4 seconds	Succeeded	AfterAllowTraffic	<a href="#">View events</a>	Oct 6, 2022
i-0cc83534c2624b2cd <a href="#">[?]</a>	Original	No	5 minutes 11 seconds	Succeeded	AfterBlockTraffic	<a href="#">View events</a>	Oct 6, 2022

Developer Tools > [CodeDeploy](#) > [Deployments](#) > d-1XICYHMR8

## d-1XICYHMR8

[C](#)
[Copy deployment](#)
[Retry deployment](#)

### Deployment status

Step 1  
Provisioning replacement instances

2 of 2 replacement instances provisioned

100%

(Succeeded)

Step 2  
Installing application on replacement instances

2 of 2 instances updated

100%

(Succeeded)

Step 3  
Rerouting traffic to replacement instances

(Succeeded)

100%

Step 4  
Terminating original instances

2 of 2 original instances terminated

100%

(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
---------------------------	------------------------------