

Automated Deployment with AWS Auto Scaling and Jenkins

Introduction

This project focuses on automating the deployment process using AWS Auto Scaling and Jenkins, ensuring that both new and existing servers are always running the latest version of your application code.

This project involves two main deployment pipelines:

1. **Initial Deployment Pipeline:** Automatically deploys the latest application code to any new server launched by AWS Auto Scaling. This pipeline is triggered by an AWS Lambda function, which responds to events generated by Amazon EventBridge whenever a new instance is added to the Auto Scaling group.
2. **Continuous Deployment Pipeline:** Ensures that all current servers managed under a Load Balancer (LB) target group receive the latest updates. This pipeline will be triggered using Github webhook when any commit or changes to the code, keeping the application consistent across all servers.

AWS Infrastructure Setup

To enable seamless auto-deployment, we must set up AWS resources that can manage server scaling and balance traffic:

Auto Scaling Group (ASG)

An Auto Scaling Group automatically adjusts the number of Amazon EC2 instances in response to demand or other conditions (e.g., CPU usage).

- Automatically launch and terminate instances to match the desired number of healthy instances at any time.
- Configuration:
 - Define a launch template or configuration that specifies the AMI (Amazon Machine Image), instance type, security groups, key pairs.

- Set scaling policies to determine when and how instances should be added or removed from the group.
- In Configure advanced options select attach to existing load balancer target group. So the instance will be automatically registered to a target group under load balancer.

EC2 > Auto Scaling groups

Auto Scaling groups (1/1) Info

Launch configurations Launch templates Actions Create Auto Scaling group

Search your Auto Scaling groups

Name	Launch template/configuration	Instances	Status	Desired capacity	Min	Max	Avai...
Ci-Cd-ASG	CiCd-LT Version Default	1	-	1	1	2	eu-nor...

Auto Scaling group: Ci-Cd-ASG

Details Activity Automatic scaling Instance management Monitoring Instance refresh

Group details

Auto Scaling group name Ci-Cd-ASG	Desired capacity 1	Desired capacity type Units (number of instances)	Amazon Resource Name (ARN) arn:aws:autoscaling:eu-north-1:756803849938:autoScalingGroup:30639660-4e8f-4ff6-9eec-7a83a69e8078:autoScalingGroupName/Ci-Cd-ASG
Date created Fri Aug 16 2024 23:28:28 GMT+0530 (India Standard Time)	Minimum capacity 1	Status -	
	Maximum capacity 2		

Load Balancer (LB)

An Application Load Balancer (ALB) distributes incoming application traffic across multiple targets, such as EC2 instances, in multiple Availability Zones. This enhances the availability and fault tolerance of your application.

- Manage incoming traffic and distribute it across the EC2 instances within the target group.
- Configuration:
 - Define target groups, specifying how the load balancer should route requests to one or more registered targets.
 - Register the instances from the Auto Scaling group as targets in the target group.

EC2 > Load balancers

Load balancers (1/1)

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

Actions

Create load balancer

<input checked="" type="checkbox"/>	Name	DNS name	State	VPC ID	Availability Zones	Type	Date created
<input checked="" type="checkbox"/>	CI-CD-ALB	CI-CD-ALB-1014251906.e...	Active	vpc-4db77e24	3 Availability Zones	application	August 16, 2024, 22:4..

Load balancer: CI-CD-ALB

Details

Listeners and rules

Network mapping

Resource map - new

Security

Monitoring

Integrations

Attributes

Tags

Details

Load balancer type

Application

Status

Active

VPC

[vpc-4db77e24](#)

Load balancer IP address type

IPv4

Scheme

Internet-facing

Hosted zone

Z23TAZ6LKFMNIO

Availability Zones

[subnet-8d8e67f6](#) eu-north-1b (eun1-az2)

[subnet-716ea118](#) eu-north-1a (eun1-az1)

[subnet-71ede53b](#) eu-north-1c (eun1-az3)

Date created

August 16, 2024, 22:42 (UTC+05:30)

AWS Lambda and EventBridge Setup to trigger Jenkins

Trigger Jenkins to automate the deployment to new instances, AWS Lambda and Amazon EventBridge are used:

Amazon EventBridge Rule

Amazon EventBridge is a serverless event bus service that enables you to connect your applications with data from various sources. It can be configured to respond to AWS events and route them to Lambda functions or other AWS services.

- Detects when a new EC2 instance is launched within the Auto Scaling group.
- Configuration:
 - Create a rule in EventBridge that listens for the EC2 Instance Launch Successful event.
 - Set the rule to trigger a Lambda function whenever a new instance is added to the Auto Scaling group.

Amazon EventBridge > Rules > Demo-CI-CD

Demo-CI-CD

Edit Disable Delete CloudFormation Template ▼

Rule details Info

Rule name Demo-CI-CD	Status Enabled	Event bus name default	Type Standard
Description	Rule ARN arn:aws:events:eu-north-1:756803849:rule/Demo-CI-CD	Event bus ARN arn:aws:events:eu-north-1:756803849:event-bus/default	

Event pattern Targets Monitoring Tags

Event pattern Info Edit

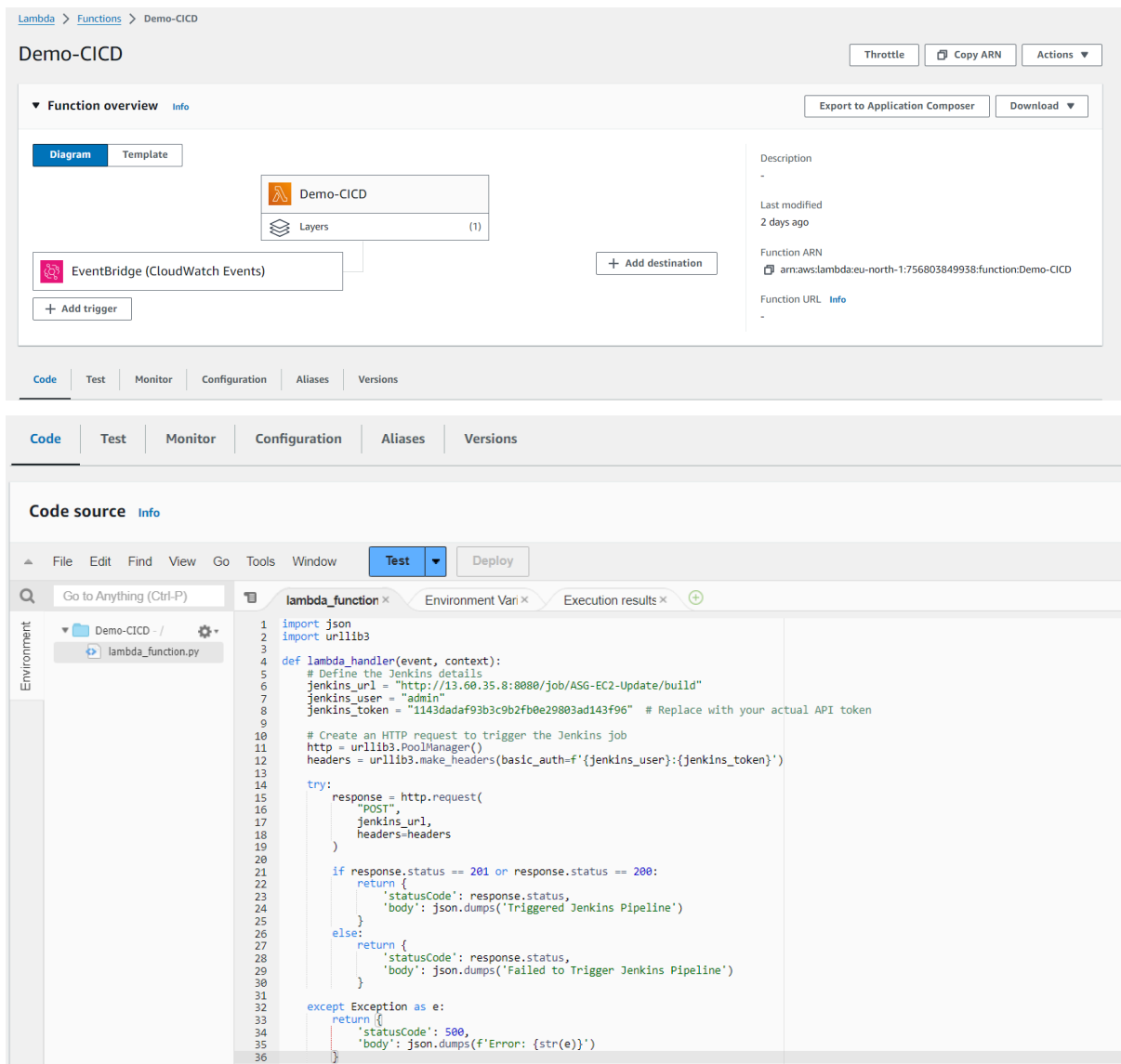
```
1 {
2   "source": ["aws.autoscaling"],
3   "detail-type": ["EC2 Instance Launch Successful"],
4   "detail": {
5     "AutoScalingGroupName": ["Ci-Cd-ASG"]
6   }
7 }
```

Copy

AWS Lambda Function

AWS Lambda allows you to run code without provisioning or managing servers. It is used here to trigger the Jenkins pipeline for initial deployment when a new instance is launched.

- Trigger the Jenkins pipeline automatically upon detecting a new instance launch in Auto Scaling.
- Configuration:
 - Creating a Lambda function code using Python will interact with Jenkins pipeline using Jenkins API and Jenkins Job url.
 - Configure the Lambda function to be triggered by the EventBridge rule.



The screenshot displays the AWS Lambda console for a function named "Demo-CICD". The top navigation bar shows "Lambda > Functions > Demo-CICD". The function overview section includes tabs for "Diagram" and "Template", a "Diagram" showing the function connected to "EventBridge (CloudWatch Events)", and a "Description" panel on the right with details like "Function ARN" and "Function URL". Below the overview, a horizontal menu contains "Code", "Test", "Monitor", "Configuration", "Aliases", and "Versions". The "Code source" section is active, showing a code editor with a file explorer on the left containing "Demo-CICD -/" and "lambda_function.py". The code editor displays the following Python code:

```
1 import json
2 import urllib3
3
4 def lambda_handler(event, context):
5     # Define the Jenkins details
6     jenkins_url = "http://13.60.35.8:8080/job/ASG-EC2-Update/build"
7     jenkins_user = "admin"
8     jenkins_token = "1143dadaf93b3c9b2fb0e29803ad143f96" # Replace with your actual API token
9
10    # Create an HTTP request to trigger the Jenkins job
11    http = urllib3.PoolManager()
12    headers = urllib3.make_headers(basic_auth=f'{jenkins_user}:{jenkins_token}')
13
14    try:
15        response = http.request(
16            "POST",
17            jenkins_url,
18            headers=headers
19        )
20
21        if response.status == 201 or response.status == 200:
22            return {
23                'statusCode': response.status,
24                'body': json.dumps('Triggered Jenkins Pipeline')
25            }
26        else:
27            return {
28                'statusCode': response.status,
29                'body': json.dumps('Failed to Trigger Jenkins Pipeline')
30            }
31
32    except Exception as e:
33        return {
34            'statusCode': 500,
35            'body': json.dumps(f'Error: {str(e)}')
36        }
```

Initial Deployment Pipeline

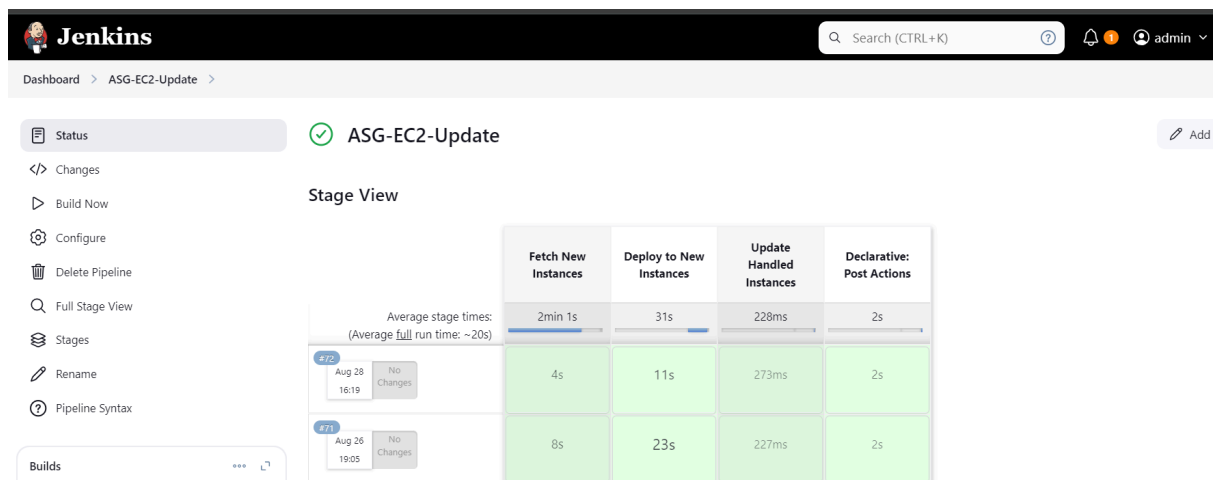
The initial deployment pipeline is triggered whenever a new instance is launched in the Auto Scaling group. This pipeline ensures that new instances are immediately updated with the latest application code from GitHub.

Configuring the Jenkins Pipeline

The Jenkins pipeline will deploy the latest code to the new instances identified by the Auto Scaling group.

Creating the Jenkins Pipeline

- **Open Jenkins Dashboard:**
 - Log in to your Jenkins instance.
 - Click on **New Item** to create a new pipeline job.
- **Pipeline Configuration:**
 - Provide a name for your pipeline and select **Pipeline** as the job type.
 - In the **Pipeline** section, set the definition to **Pipeline script** and paste your pipeline code.
- **Save and Apply:**
 - After configuring the pipeline, click **Save**.

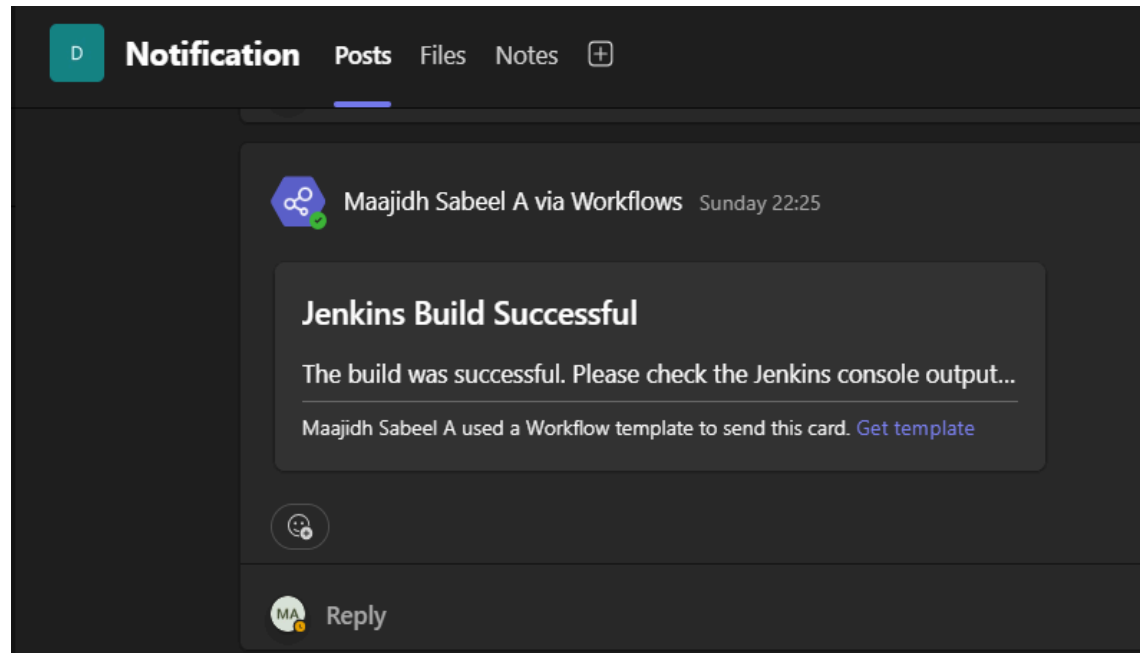


Deployment Process:

- **Fetching New Instances:** Identify the new instances that require deployment.
 - **Instance Identification:** Once the pipeline is triggered, it first fetches the list of all instances in the Auto Scaling group that are in the "InService" state. And fetches the instance ip using aws cli commands.
 - **Determining New Instances:** The pipeline reads a file that tracks instances that have already been handled. It compares the list of current instances against this file to determine which instances are new and need deployment.

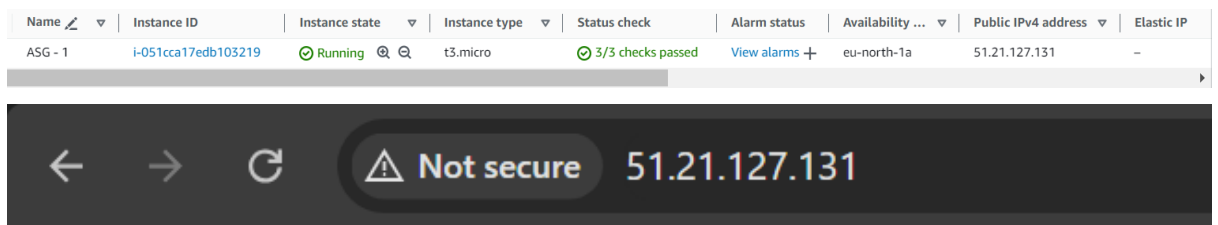
- **Deploying Code to New Instances:** Deploy the latest code from the GitHub repository to the identified new instances.
 - Clone Repository: The pipeline clones the latest code from the specified GitHub repository. This ensures that the deployment always uses the most up-to-date code.
 - SSH Connection Setup: The pipeline connects to each new instance ip via SSH. This requires:
 - SSH Key: The private key stored securely in Jenkins is used to authenticate and connect to the new EC2 instances.
 - SSH Options: Options like StrictHostKeyChecking=no are used to bypass host key verification, simplifying connections to fresh instances.
 - Deploy Commands Execution: After establishing an SSH connection, the pipeline executes a series of commands on the new instance to deploy the code:
 - Install Required Software: Ensure necessary software (e.g., Git, application dependencies) is installed.
 - Code Deployment: Clone the GitHub repository or copy files to the appropriate directories on the instance.
 - Service Restart: Restart any services (e.g., web servers, application servers) to apply the new code.
 - Error Handling and Retries: If an SSH connection fails or any command errors out, the pipeline retries the connection multiple times to ensure robustness. This is crucial for handling network instability or temporary instance unavailability.
- **Updating Handled Instances:** Keep track of all instances that have been processed to avoid redeploying to the same instances in future runs.
 - Handled Instances: After successfully deploying to an instance, the pipeline adds its IP address to the handled_instances.txt file. This file is then used in future pipeline runs to skip instances that have already been updated.
- **Notifications:** In the post action of the pipeline the build status of the pipeline will be notified.

- Success Notification: If the deployment is successful for all instances, a success message is sent to a configured Microsoft Teams channel using a webhook.
- Failure Notification: If the deployment fails for any instance, a failure message is sent, allowing for quick troubleshooting and response.



- Code Deployment:

The pipeline fetches the latest code from a GitHub repository and deploys it to the new instance. Now the new instance launched in Autoscaling group was deployed successfully.



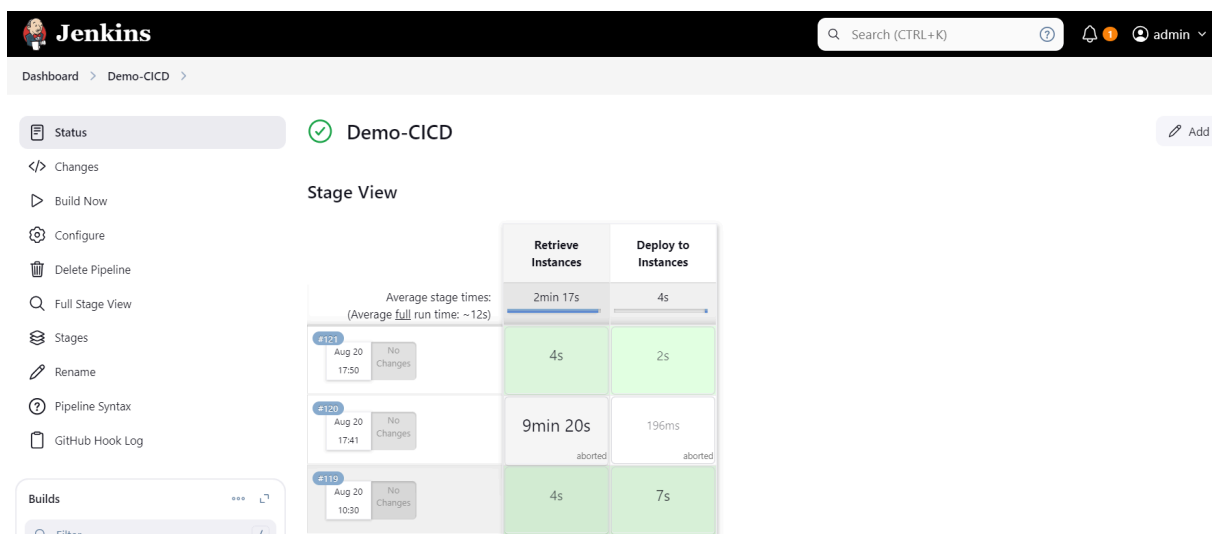
My Server IP address is: 172.31.26.38

Continuous Deployment Pipeline

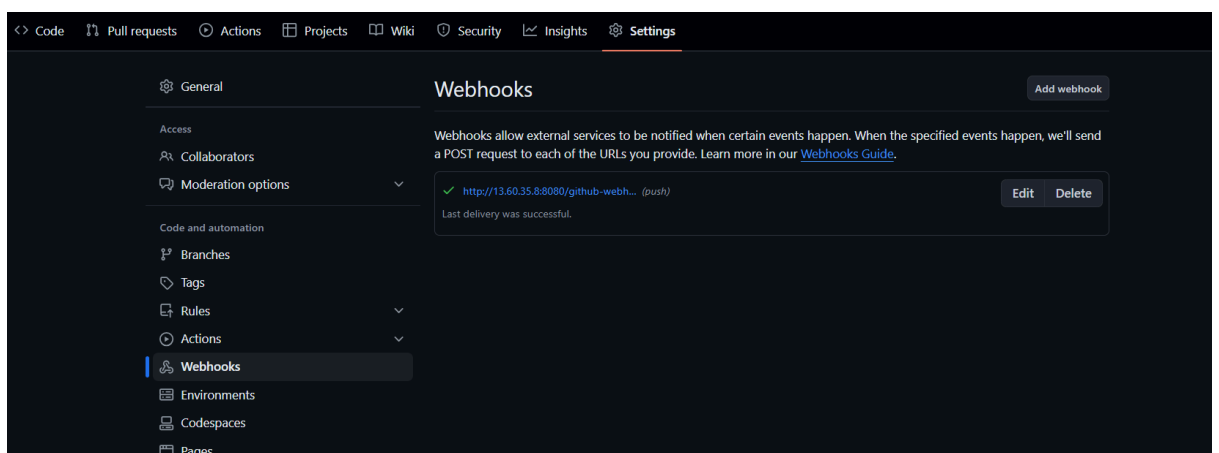
The continuous deployment pipeline ensures that all current servers managed under the Load Balancer are updated with the latest code changes.

Deployment process

- Create a new pipeline for continuous deployment of latest code from Github repo to all current instances under load balancer target group from Autoscaling Group.

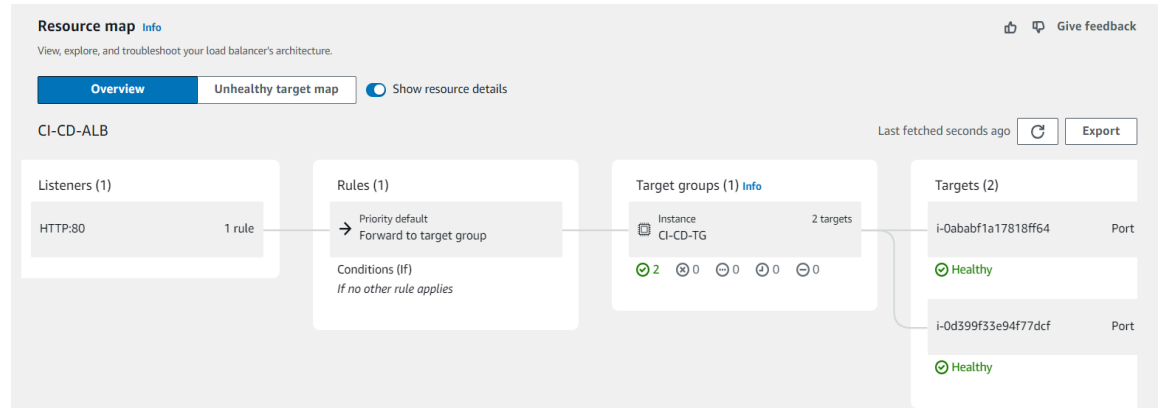


- **Trigger:** This pipeline will be automatically triggered using a Github Webhook when any changes or commit occurs to the Github repo.



- **Stages :**

- Retrieve Instance List: Fetch all instance IDs associated with the Load Balancer target group and get those instance ip using aws cli commands



- Deploy Application to All Instances: SSH into each instance and deploy the latest code.
 - Stores the list of IP addresses in the environment variable INSTANCE_IPS, separated by commas.
 - SSH Key: The private key stored securely in Jenkins is used to authenticate and connect to the new EC2 instances.
 - SSH Options: Options like StrictHostKeyChecking=no are used to bypass host key verification, simplifying connections to fresh instances.
- Deploy Commands Execution: After establishing an SSH connection, the pipeline executes a series of commands to deploy the latest code to each instance.

Conclusion

This project demonstrates a robust and scalable deployment strategy by integrating AWS Auto Scaling with Jenkins pipelines. The initial deployment pipeline ensures that new instances are automatically configured with the latest code upon launch, while the continuous deployment pipeline keeps all existing servers updated. Together, these pipelines provide an efficient and automated solution for maintaining and scaling applications, minimizing downtime, and ensuring consistency across all servers.