

# AWS Assignments

## ASSIGNMENT-1

### Assignment 1

In this assignment you have to learn about deployment strategies.

- Recreate deployment
- Rolling deployment
- Blue-green deployment
- A/B deployment
- Canary deployment

After this you need to implement these 2 below deployment strategies.

Must Do

- Recreate deployment
- Rolling deployment

Good to Do

- Blue Green deployment
- Canary deployment

Recommendation

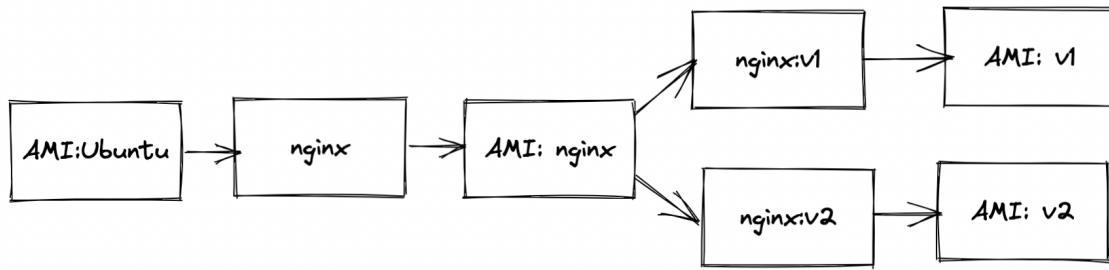
- Don't straight forward jump into creating the utility, first understand the concepts.

github-link --> <https://github.com/OT-TRAINING/DeploymentStrategies>

Hint :

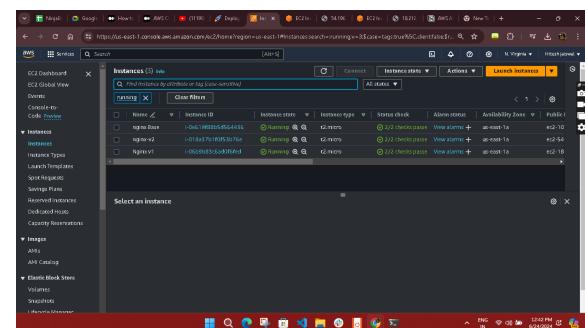
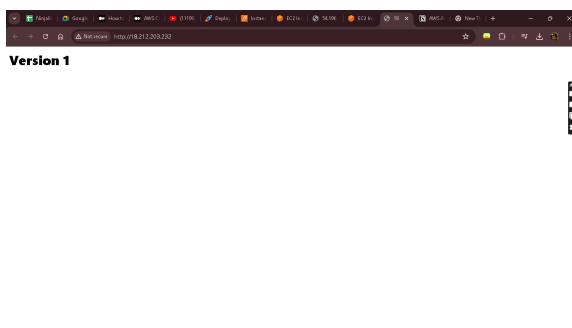
1. Use ASG
2. Use LB
3. Use EC2 service
4. Use AMI Snapshot

## PART-1 SERVER, IMAGE AND SNAPSHOT



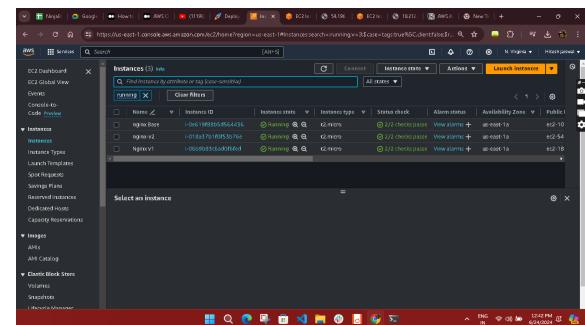
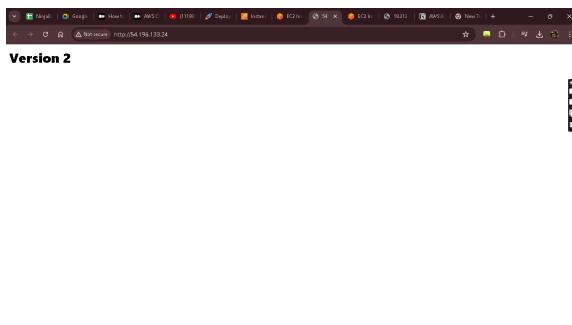
## Server Base

nginx-v1  
AMI nginx-v1  
AMI Snapshot



## Server Base

nginx-v2  
AMI nginx-v2  
AMI Snapshot



## PART -2 Application Load Balancer



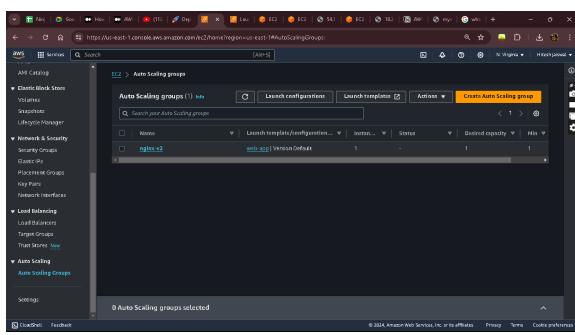
## Target Group & Load Balancer



Version 1



Version 2



## Auto-Scaling Group

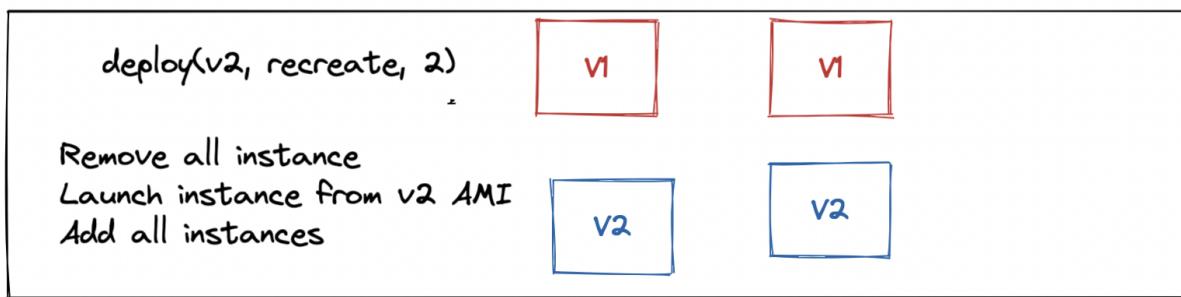
nginx v1 Removal - Deleted via Auto Scaling group  
nginx v2 Addition Template - Adding Via Auto Scaling

## PART-3

### Deployment Strategy

- 1. Recreate Strategy :- a basic deployment pattern which simply shuts down all the old pods and replaces them with new ones**  
. We don't use it much as the downtime is high in this.

Removed all Instance associated with v1 at one time



## Through Version 2 AMI Targets

## Health Checks for

2. Rolling Development Strategy :- A rolling deployment is a deployment strategy that slowly replaces previous versions of an application with new versions of an application by completely replacing the infrastructure on which the application is running.

Deregister nginx v2 AMI associated instance 1-by -1

Added nginx v2 and New-nginx-v2

## Now deregistering new-nginx-v2 and Deleting

## NOW THROUGH AWS CLI

1. Launch Instance nginx - We had launched one main ubuntu AMI then using that AMI we launch AMI nginx as base server, now we will launch nginx v1 and nginx v2 using base server

### Command :-

```
aws ec2 run-instances \
--image-id ami-0e001c9271cf7f3b9 \
--count 2 \
--instance-type t2.micro \
--key-name aws \
--security-group-ids sg-04caf7a20c859a1e3 \
--subnet-id subnet-052ce261d67d5cc06 \
--associate-public-ip-address
```

```
aws ec2 run-instances \
--image-id ami-0e001c9271cf7f3b9 \
--count 2 \
--instance-type t2.micro \
--key-name aws \
--security-group-ids sg-04caf7a20c859a1e3 \
--subnet-id subnet-052ce261d67d5cc06 \
--associate-public-ip-address
```

The screenshot shows the AWS EC2 Dashboard with the 'Instances' section selected. There are four instances listed:

- nginx v1: Instance ID i-059b10dfc25884db, Status Running, Type t2.micro, Status check 2/2 checks pass, Alarm status View alarms, Availability zone us-east-1b.
- nginx v2: Instance ID i-0bb43793fc18135c7, Status Running, Type t2.micro, Status check 2/2 checks pass, Alarm status View alarms, Availability zone us-east-1b.
- nginx: Instance ID i-0ba7ed9b051e01fa4, Status Running, Type t2.micro, Status check 2/2 checks pass, Alarm status View alarms, Availability zone us-east-1a.
- Main-Server-ubuntu: Instance ID i-0f0sea7b7793819d7, Status Running, Type t2.micro, Status check 2/2 checks pass, Alarm status View alarms, Availability zone us-east-1a.

## 2. Create Image & Snapshot and Templates of Instance

### Image & Snapshot

```
root@kern1:~# aws ec2 create-image --instance-id i-00437a07ff719b029 --name "Nginx Base" --no-reboot --block-device-mappings [
    {
        "DeviceName": "/dev/sda1",
        "DeleteOnTermination": true,
        "VolumeSize": 8,
        "VolumeType": "gp2"
    }
]
{
    "ImageId": "ami-05dfc0671141518a"
}
root1:~# kern1:~# aws ec2 create-image --instance-id i-012d322f72fd0bb6 --name "Nginx-V1" --no-reboot --block-device-mappings [
    {
        "DeviceName": "/dev/sda1",
        "DeleteOnTermination": true,
        "VolumeSize": 8,
        "VolumeType": "gp2"
    }
]
{
    "ImageId": "ami-061dd6cb7bde9icbd"
}
root1:~# kern1:~# aws ec2 create-image --instance-id i-0b42021b5ba3c25a --name "Nginx-V2" --no-reboot --block-device-mappings [
    {
        "DeviceName": "/dev/sda1",
        "DeleteOnTermination": true,
        "VolumeSize": 8,
        "VolumeType": "gp2"
    }
]
{
    "ImageId": "ami-061dd6cb7bde9icbd"
```

The screenshot shows the AWS EC2 Dashboard with the 'AMIs' section selected. Three AMIs are listed:

- Nginx-V2: AMI ID ami-05dfc0671141518a
- Nginx Base: AMI ID ami-05dfc0671141518a
- Nginx-V1: AMI ID ami-061dd6cb7bde9icbd

### Template Launched

```
root1:~# kern1:~# aws ec2 create-launch-template --launch-template-name nginx-V1-server --version-description "Initial version" --launch-template-data [
    {
        "ImageId": "ami-061dd6cb7bde9icbd",
        "InstanceId": "i-0bb43793fc18135c7",
        "InstanceType": "t2.micro",
        "KeyName": "test",
        "SecurityGroupIds": ["sg-0ca2a928c899a1e3"]
    }
]
{
    "LaunchTemplate": [
        {
            "LaunchTemplateId": "lt-0931a932ac6fc1e8",
            "LaunchTemplateName": "nginx-V1-server",
            "Version": "1"
        }
    ]
}
root1:~# kern1:~# aws ec2 create-launch-template --launch-template-name nginx-V2-server --version-description "Initial version" --launch-template-data [
    {
        "ImageId": "ami-061dd6cb7bde9icbd",
        "InstanceId": "i-0ba7ed9b051e01fa4",
        "InstanceType": "t2.micro",
        "KeyName": "test",
        "SecurityGroupIds": ["sg-0ca2a928c899a1e3"]
    }
]
{
    "LaunchTemplate": [
        {
            "LaunchTemplateId": "lt-0ff92bc316228011e9",
            "LaunchTemplateName": "nginx-V2-server",
            "Version": "1"
        }
    ]
}
root1:~# kern1:~#
```

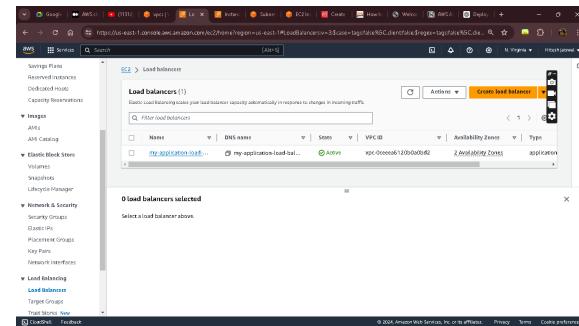
The screenshot shows the AWS EC2 Dashboard with the 'Launch Templates' section selected. Two launch templates are listed:

- nginx-V1-server: Launch Template ID lt-0931a932ac6fc1e8, Created Time 2024-06-25T08:40:04.000Z
- nginx-V2-server: Launch Template ID lt-0ff92bc316228011e9, Created Time 2024-06-25T08:44:58.000Z

## 3. Create Load Balancer, Target Group & Auto Scaling

## Load Balancer

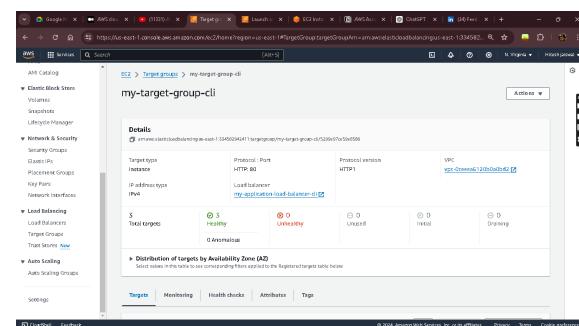
```
root@LinuxDESKTOP-IRPHM0D:~# aws elbv2 create-load-balancer --name my-application-load-balancer-cli --subnets subnet-05c2e201d67d0cc06 subnet-0f6d3110a0d02d18 --security-groups sg-0ecfa7ab0d99fa1 --scheme Internet-facing --type application --ip-address-type ipv4
{
    "LoadBalancers": [
        {
            "LoadBalancerArn": "arn:aws:elasticloadbalancing:us-east-1:334582942011:loadbalancer/app/my-application-load-balancer-cli/7f8ea11f5a090f",
            "DNSName": "my-application-load-balancer-cli-41381304.us-east-1.elb.amazonaws.com",
            "CanonicalHostedZoneId": "Z2HJ86-2T8H8-99-82-31080-00-00-00",
            "CreatedTime": "2024-06-27T08:59:02.310800-00:00",
            "Type": "application",
            "Scheme": "internet-facing",
            "State": "active",
            "IpAddressType": "ipv4",
            "SecurityGroups": [
                {
                    "GroupName": "ut-east-1a",
                    "SubnetId": "subnet-092c201d67d0cc06",
                    "LoadBalancerAddresses": []
                }
            ],
            "Subnets": [
                {
                    "SubnetId": "subnet-092c201d67d0cc06",
                    "LoadBalancerAddresses": []
                }
            ],
            "SecurityGroups": [
                "sg-0ecfa7ab0d99fa1"
            ],
            "IpAddressType": "ipv4"
        }
    ]
}
root@LinuxDESKTOP-IRPHM0D:~#
```



## Target Group

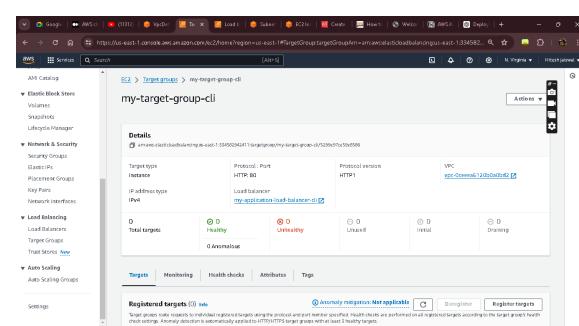
### Created TG

```
root@LinuxDESKTOP-IRPHM0D:~# aws elbv2 create-target-group --name my-target-group-cli --protocol HTTP --port 80 --vpc-id vpc-0ceaaad128d8a2a0 --unhealthy-threshold-count 2 --target-type instance
{
    "TargetGroups": [
        {
            "TargetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:334582942011:targetgroup/my-target-group-cli/5299e97ce0e8500",
            "Protocol": "HTTP",
            "Port": 80,
            "VpcId": "vpc-0ceaaad128d8a2a0",
            "HealthCheckProtocol": "HTTP",
            "HealthCheckPort": "traffic-port",
            "HealthCheckIntervalSeconds": 10,
            "HealthCheckTimeoutSeconds": 5,
            "HealthyThresholdCount": 5,
            "UnhealthyThresholdCount": 2,
            "HealthCheckGracePeriod": 10,
            "HealthCheckGracePeriodUnit": "seconds",
            "HealthCheckPath": "/",
            "HealthCheckProtocolVersion": "HTTP1",
            "IpAddressType": "ipv4"
        }
    ]
}
root@LinuxDESKTOP-IRPHM0D:~#
```



## Attach to Load Balancer

```
root@LinuxDESKTOP-IRPHM0D:~# aws elbv2 describe-load-balancers --names my-application-load-balancer-cli
{
    "LoadBalancers": [
        {
            "LoadBalancerArn": "arn:aws:elasticloadbalancing:us-east-1:334582942011:loadbalancer/app/my-application-load-balancer-cli/7f8ea11f5a090f",
            "DNSName": "my-application-load-balancer-cli-41381304.us-east-1.elb.amazonaws.com",
            "CanonicalHostedZoneId": "Z2HJ86-2T8H8-99-82-31080-00-00-00",
            "CreatedTime": "2024-06-27T08:59:02.310800-00:00",
            "Type": "application",
            "Scheme": "internet-facing",
            "State": "active",
            "IpAddressType": "ipv4",
            "AvailabilityZones": [
                {
                    "GroupName": "ut-east-1a",
                    "SubnetId": "subnet-092c201d67d0cc06",
                    "LoadBalancerAddresses": []
                }
            ],
            "Subnets": [
                {
                    "SubnetId": "subnet-092c201d67d0cc06",
                    "LoadBalancerAddresses": []
                }
            ],
            "SecurityGroups": [
                "sg-0ecfa7ab0d99fa1"
            ],
            "IpAddressType": "ipv4"
        }
    ]
}
root@LinuxDESKTOP-IRPHM0D:~#
```

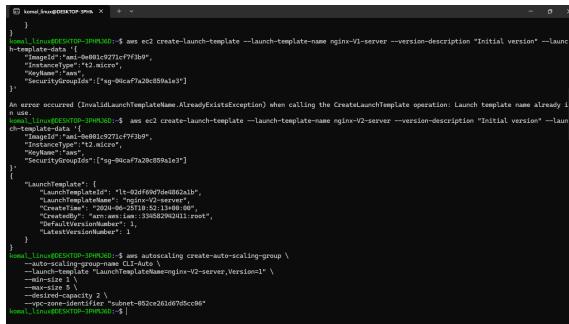


## Attach Listener

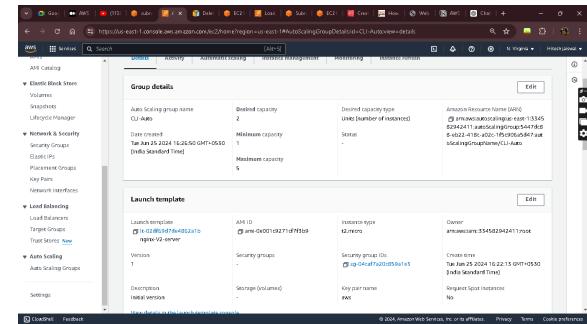


```
komal_lxmx@DESKTOP-3PNM360:~$ aws elbv2 create-target-group --load-balancer-arn arn:aws:elasticloadbalancing:us-east-1:334582942411:loadbalancer/app/my-application-load-balancer-cl/1 --protocol HTTP --port 80 --default-actions Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:us-east-1:334582942411:targetgroup/my-target-group-cl/5299e97ce59e8588
komal_lxmx@DESKTOP-3PNM360:~$ aws elbv2 create-listener --load-balancer-arn arn:aws:elasticloadbalancing:us-east-1:334582942411:loadbalancer/app/my-application-load-balancer-cl/7f5eea215f5a890f --protocol HTTP --port 80 --default-actions Type=forward,TargetGroupArn=arn:aws:elasticloadbalancing:us-east-1:334582942411:targetgroup/my-target-group-cl/7f5eea215f5a890f
{
    "Listeners": [
        {
            "ListenerArn": "arn:aws:elasticloadbalancing:us-east-1:334582942411:listener/app/my-application-load-balancer-cl/7f5eea215f5a890f"
        }
    ],
    "LoadBalancerArn": "arn:aws:elasticloadbalancing:us-east-1:334582942411:loadbalancer/app/my-application-load-balancer-cl/7f5eea215f5a890f"
}
588",
        "Port": 80,
        "Protocol": "HTTP",
        "DefaultActions": [
            {
                "Type": "Forward",
                "TargetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:334582942411:targetgroup/my-target-group-cl/5299e97ce59e8588",
                "ForwardConfig": {
                    "TargetGroups": [
                        {
                            "TargetGroupArn": "arn:aws:elasticloadbalancing:us-east-1:334582942411:targetgroup/my-target-group-cl/5299e97ce59e8588",
                            "Weight": 1
                        }
                    ],
                    "TargetGroupStickinessConfig": {
                        "Enabled": false
                    }
                }
            }
        ]
    }
}
589",
}
}
komal_lxmx@DESKTOP-3PNM360:~$ |
```

1. Recreate Strategy - using AMI of V1 we will launch template associate it with auto scaling group so when V1 will be deleted V2 will launch itself with new version. and it will be updated. As I have done through cli in above screenshot (Launched Template with Auto Scaling group ami is of V1)



```
komal_lxmx@DESKTOP-3PNM360:~$ aws ec2 create-launch-template --launch-template-name nginx-v1-server --version-description "Initial version" --launch-template-data '{ "ImageId": "ami-040f697d6e82a1b", "InstanceType": "t2.micro", "KeyName": "ans", "SecurityGroupIds": ["sg-0cfa7a20c858ale"] }'
An error occurred (InvalidLaunchTemplateName.AlreadyExistsException) when calling the CreateLaunchTemplate operation: Launch template name already in use.
komal_lxmx@DESKTOP-3PNM360:~$ aws ec2 create-launch-template --launch-template-name nginx-v2-server --version-description "Initial version" --launch-template-data '{ "ImageId": "ami-040f697d6e82a1b", "InstanceType": "t2.micro", "KeyName": "ans", "SecurityGroupIds": ["sg-0cfa7a20c858ale"] }'
{
    "LaunchTemplate": {
        "LaunchTemplateId": "lt-040f697d6e82a1b",
        "LaunchTemplateName": "nginx-v2-server",
        "Version": "1",
        "CreatedBy": "arn:aws:iam::334582942411:root",
        "DefaultVersionNumber": 1
    }
}
komal_lxmx@DESKTOP-3PNM360:~$ aws autoscaling create-auto-scaling-group \
--auto-scaling-group-name my-asg \
--launch-template "LaunchTemplateName=nginx-v2-server,Version=1" \
--max-size 5 \
--min-size 1 \
--desired-capacity 2 \
--vpc-security-group-identifier "subnet-052ce201d97dc90"
komal_lxmx@DESKTOP-3PNM360:~$ |
```



Group details

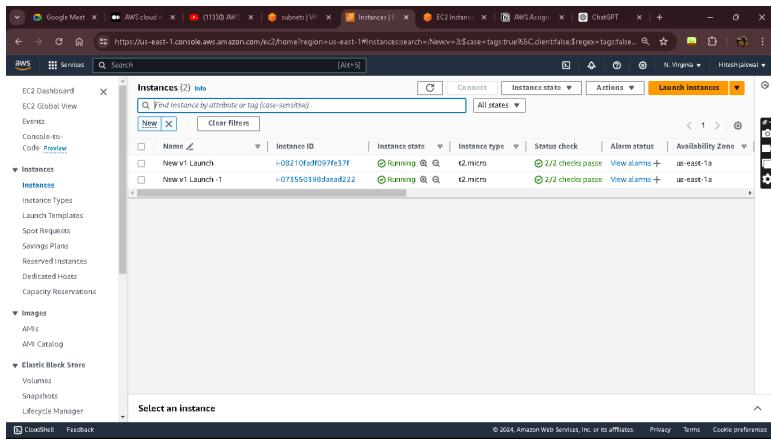
Auto Scaling Group Name	Desired capacity	Desired capacity step	Amazon Resource Name (ARN)
my-asg	2	1	arn:aws:autoscaling:us-east-1:334582942411:autoScalingGroup:my-asg:5299e97ce59e8588

Launch template

Launch template	AMI ID	Instance type	Owner
nginx-v2-server	ami-040f697d6e82a1b	t2.micro	amazonaws.com:334582942411:root

Target groups

Target group	Security group ID	Create time
my-target-group-cl	sg-0cfa7a20c858ale	The target group was created on 2024-10-22T13:00:00Z (1 day ago)



Command to terminate

```
aws ec2 terminate-instances --instance-ids $i-0492bd517bfb25fc8
```

We face downtime as every version deletes and it will time to make new version

## 2. Rolling Instance

In this we can roll back to previous version we can simply deregister using command

```
aws elbv2 deregister-targets \
--target-group-arn arn:aws:elasticloadbalancing:us-east-
1:1334582942411:targetgroup/my-target-group-cli/5299e97ce59e8580 \
--targets Id=$id of TG
```

we remove version line by line so we don't face downtime

## ASSIGNMENT-2

### Assignment 2

""Suppose your client has infrastructure running over the AWS cloud, but for the past few days, the client application has not been able to handle enough traffic and load, so he is thinking of setting up any reverse proxy for handling the load as middleware. He called you from the DevOps team

and explained the problem statement. You have explained very well about the plan and design that you are going to perform."""

You were told that nginx needs to be set up in between your client and application service, which can handle the request, but we need to setup one more thing, which is the basic HA (high availability) and DR (disaster recovery) of that middleware setup. Tasks would be done day-wise.

Day 1:

- Create instance and install nginx on that, after that create an AMI-1
- Create instance from the above AMI-1 and make V1
- Do some changes in same instance and again create AMI-2
- Create instance from the above AMI-2 and make V2
- Make both V1 and V2 AMI's
- Now you have 2 AMI's total

You have to continue the strategy with autoscaling group, attach asg on LB and follow the same strategy and make this highly available by testing load test on the nginx server so that it would autoscale at the time of load.

Attach the policy to asg and increase the stress according to policy mentioned then analyse the result. ( Try every policy )

- Avg CPU utilization
- Network bytes in/out
- ALB requests count per target.

Somehow client needs version upgrade in Nginx so please upgrade V1 to V2 that you have made

"""But the client has complained to us that our new version V2 is not compatible with the service, so please revert back. You need to revert back to the previous version. """

Day 2: """Client has told you about the change in it's Ngin

x as he needs this tool as hosting webpage too, so you need to plan accordingly. Also, he has specially mentioned that all tasks should be performed from their AWS environment, which means all O/P would be done through EC2 only. """

- You need to pull image content which would be display at the time of webpage hosting from VCS git and push your images on S3 bucket using aws-cli. (Do not use secrete and access keys)
- You need to create a frontend including images in nginx page which will be fetched directly from S3 bucket.

Day3:

"""Now client has again told us to test it continuously to see if ASG is working or not if my server got unhealthy."""

Enter in the server and make some changes in server in order to make server unhealthy. Now analyze the result to see how ASG can help you to maintain your instance desire state.

=====

(Remember the special points from client)

NOTE 1: You need to create the utility in such a way that it will make AMI of specific version and attach to the asg and perform the rolling deployment strategie. In case of revert back you should also have the function of revert back feature.

NOTE 2: But always remember first do it all the tasks manually.

=====

GOOD TO DO:

Do the whole automation with Blue Green Deployment and add CloudFront in front of Load Balancer to access the website in order to overcome latency.

Continue to the previous assignment 1 and use your both ngi

```
nx AMI's for displaying webpage and use your images from S3 bucket.
```

Day4:

```
"""Client called devOps and asked if it was possible to get the two different details with single LB DNS, as he has two webpages and wants to pass the traffic through a single ALB but with a different path, like app.opstree.com/path1 or app.opstree.com/path2. """
```

You have told him yes, it is possible with path-based routing in the ALB. Now do the following:

```
## Use existing Nginx and there should be 1 ec2 in each private subnet .
    - 1 bastion host in public subnet .
    - port 22 of bastion host should only accessible from your public ip.
        - nginx welcome page with path '/ninja1' on first ec2 should display `Image-1` .
        - nginx welcome page with path '/ninja2' on second ec2 should display `Image-2` .
    - port 22 of both the nginx servers should only be accessible from bastion host.
```

```
## Create target group for each nginx server. (2 servers)
## Create Application load balancer.
    - port 80 open of nginx server should only be accessible from your ALB.
        - port 80 open of ALB should only be accessible from your public IP.
            - ALB should only be accessible though your public IP none else.
                - create listener rule so that {YOUR-ALB-DNS-NAME}/ninja1 should display welcome page of first ec2.
                - create listener rule so that {YOUR-ALB-DNS-NAME}/ninja2 should display welcome page of second ec2.
```

```
## Push all the updated images of the webpage to S3 bucket defined folders through any EC2 instance and maintain repo on that server only.
```

Day5:

```
"""Client asked for S3 service and bucket to be deployed in the US-East-1 region and segregate the folders with env names with proper restrictions.""""
```

```
## You need to create a IAM user and a S3 Bucket on any of the region, Kindly follow as mentioned below :
```

- Create 2 folders prod and nonprod inside that bucket.
- Upload any different image files inside both folders.
- Create a role for the above specific task and it should only be access S3 full access
- A bucket should only be accessible from your both root, IAM user and nginx application.
- Restrict the access of IAM user to not access prod folder but able to access the nonprod folder.
- Set IAM and bucket policies in such a way that it accomplished the above points.

Day6:

```
"""Client facing issue with some policy in IAM. Earlier, he assigned one policy to S3, but if he assigned the same to CDN, then it showed an error. He called the devOps for this anonymous issue.""""
```

DevOps told him that it needs to be validated through a trust relationship in the IAM service

```
## suppose organization wants to get the page access quick, so you need to implement CDN and fetch the images through CDN instead S3, use same role in the CDN which you are using in EC2 without any modification of policy.
```

```
## Make your own IAM user and assign minimal permissions to yourself for this task.
```

## Instance with version and AMI, Template

The first screenshot shows the EC2 Instances page with several instances listed, including V1 and V2. The second screenshot shows the Launch Templates page with two entries: V2 employee (version 1) and V1 employee (version 1). The third screenshot shows the AMI Catalog page with two AMIs: AMI-1 and AMI-2.

This screenshot shows the AMI catalog page with two AMIs listed: AMI-1 and AMI-2. Both are owned by the user and have the same source and owner information.

## Load Balancer, Target Group

The left screenshot shows the Load Balancers page with one ALB named mylb. The right screenshot shows the Target Groups page for the mylb ALB, with one target group named mytg. The target group details show it is using the ALB as its IP address type and is healthy.

Auto Scaling Group For CPU Optimization, Network Inbound, Network Outbound , ALB Request Per Target with Monitoring and AWS CloudWatch Alarm

## For Version-1 Template & Version 2

The following screenshots illustrate the differences between the AWS Auto Scaling group details and monitoring pages for Version-1 and Version 2 templates.

**Version-1 Template (Left Column):**

- Auto Scaling Group Details:** Shows the configuration for ASG-1-RCPT and ASG-NI. Both groups have a Desired capacity of 1, Minimum capacity of 1, and Maximum capacity of 2. They use the same launch template (AMI ID: ami-0572e05d459a7f8) and VPC security group (sg-02a279317c2906).
- Monitoring Metrics:** Shows CloudWatch Metrics for CPU Utilization (Percent), Disk Read (Bytes), Disk Write Operations (Ops/Sec), Network In (Bytes), Network Out (Bytes), and Status Check Failed (Actions/Sec) over the last 1 hour.

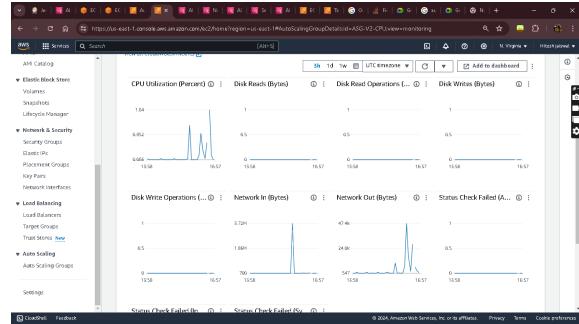
**Version 2 Template (Right Column):**

- Auto Scaling Group Details:** Shows the configuration for ASG-1 and ASG-NO. ASG-1 has a Desired capacity of 1, Minimum capacity of 1, and Maximum capacity of 2. ASG-NO has a Desired capacity of 1, Minimum capacity of 1, and Maximum capacity of 1. They use different launch templates (AMI IDs: ami-0572e05d459a7f8 and ami-0572e05d459a7f8) and different VPC security groups (sg-02a279317c2906 and sg-02a279317c2906).
- Monitoring Metrics:** Shows CloudWatch Metrics for CPU Utilization (Percent), Disk Read (Bytes), Disk Write Operations (Ops/Sec), Network In (Bytes), Network Out (Bytes), and Status Check Failed (Actions/Sec) over the last 1 hour.

ASG-V2-CPU

**Group details**

- Auto Scaling group name: ASG-V2-CPU
- Desired capacity: 1
- Maximum capacity: 1
- Status: Status last updated: Wed Jun 25 2024 16:52:49 GMT+05:30 (India Standard Time)
- Launch template: AMI ID: ami-0cb21a028a5f8d0, Version: V2-2024-06-25



## Alarm Cloud Watch

CloudWatch

Alarms (2)

- NetworkOutbound: Last state update (UTC): 2024-06-26 12:57:13, Condition: NetworkOut > 50 For 1 responses within 5 minutes.
- NetworkInbound: Last state update (UTC): 2024-06-26 12:55:19, Condition: NetworkIn > 50 For 1 responses within 5 minutes.

CloudWatch

Alarms (15)

- TargetGroup-ASG-1: Last state update (UTC): 2024-06-26 16:57:16, Condition: CPUUtilization > 50 For 3 datapoints within 3 minutes. Status: In alarm.
- TargetGroup-ASG-2: Last state update (UTC): 2024-06-26 16:57:16, Condition: CPUUtilization > 50 For 3 datapoints within 3 minutes. Status: In alarm.
- TargetGroup-ASG-3: Last state update (UTC): 2024-06-26 16:57:16, Condition: CPUUtilization > 70 For 5 datapoints within 5 minutes. Status: OK.
- NetworkOutbound: Last state update (UTC): 2024-06-26 12:57:13, Condition: NetworkOut > 50 For 1 responses within 5 minutes. Status: OK.
- NetworkInbound: Last state update (UTC): 2024-06-26 12:55:19, Condition: NetworkIn > 50 For 1 responses within 5 minutes. Status: OK.
- EC2: Last state update (UTC): 2024-06-26 12:52:32, Condition: RequestCountPerTarget > 50 For 1 datapoint within 5 minutes. Status: OK.

CloudWatch

Alarms (17)

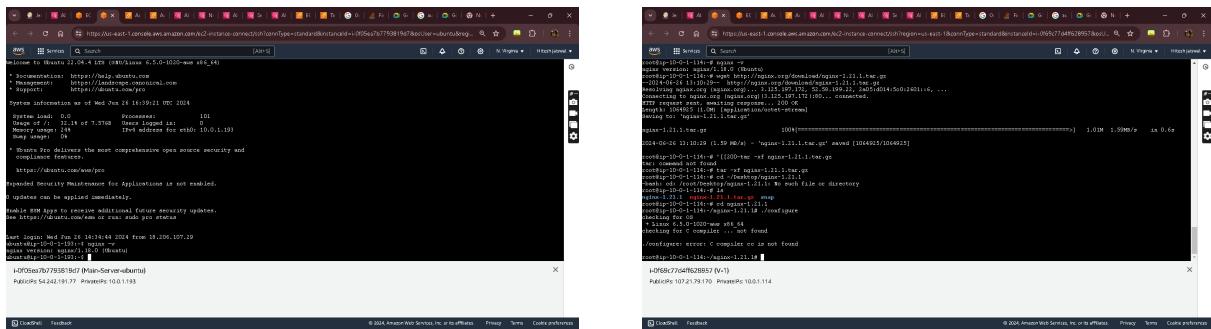
- TargetGroup-ASG-1: Last state update (UTC): 2024-06-26 16:58:45, Condition: CPUUtilization > 50 For 3 datapoints within 3 minutes. Status: In alarm.
- TargetGroup-ASG-2: Last state update (UTC): 2024-06-26 16:58:17, Condition: CPUUtilization > 50 For 3 datapoints within 3 minutes. Status: In alarm.
- TargetGroup-ASG-3: Last state update (UTC): 2024-06-26 16:57:16, Condition: CPUUtilization > 70 For 5 datapoints within 5 minutes. Status: OK.
- NetworkOutbound: Last state update (UTC): 2024-06-26 12:57:13, Condition: NetworkOut > 50 For 1 responses within 5 minutes. Status: OK.
- NetworkInbound: Last state update (UTC): 2024-06-26 12:55:19, Condition: NetworkIn > 50 For 1 responses within 5 minutes. Status: OK.
- EC2: Last state update (UTC): 2024-06-26 12:52:32, Condition: RequestCountPerTarget > 50 For 1 datapoint within 5 minutes. Status: OK.

CloudWatch

Alarms (12)

- TargetGroup-ASG-1: Last state update (UTC): 2024-06-26 11:12:26, Condition: CPUUtilization > 40 For 3 datapoints within 3 minutes. Status: In alarm.
- TargetGroup-ASG-2: Last state update (UTC): 2024-06-26 11:10:54, Condition: RequestCountPerTarget > 35 For 15 datapoints within 15 minutes. Status: In alarm.
- TargetGroup-ASG-3: Last state update (UTC): 2024-06-26 11:09:50, Condition: NetworkIn < 15 For 15 datapoints within 15 minutes. Status: OK.
- EC2: Last state update (UTC): 2024-06-26 11:06:42, Condition: RequestCountPerTarget > 50 For 3 datapoints within 5 minutes. Status: OK.

Revert Back to version 1 of Nginx 1.18.0 and Delete the 2nd Version of Nginx 1.21.1 we downloaded using wget Now we deleted v2 and just launch the instance through AMI V1



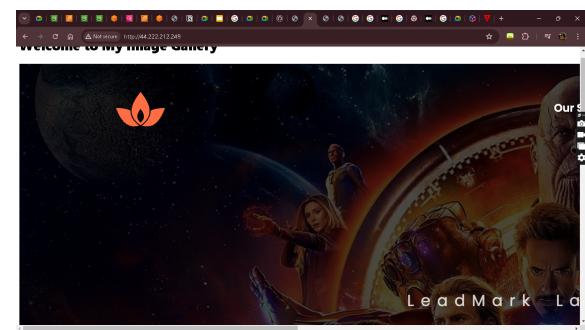
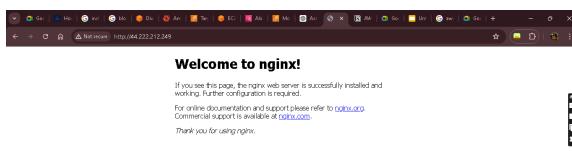
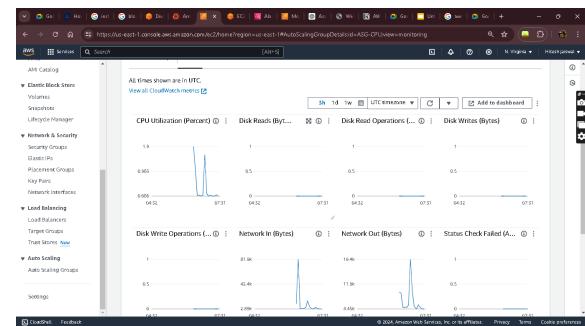
## Healthy

**Details**

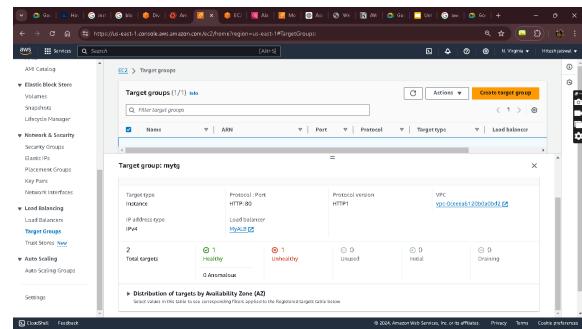
Target type	Protocol	Protocol version	VPC
Instance	HTTP/1.1	v2	vpn://vpcid/1200000000000000
IP address type	IP address		
Health	Unhealthy		

**Distribution of targets by Availability Zone (AZ)**

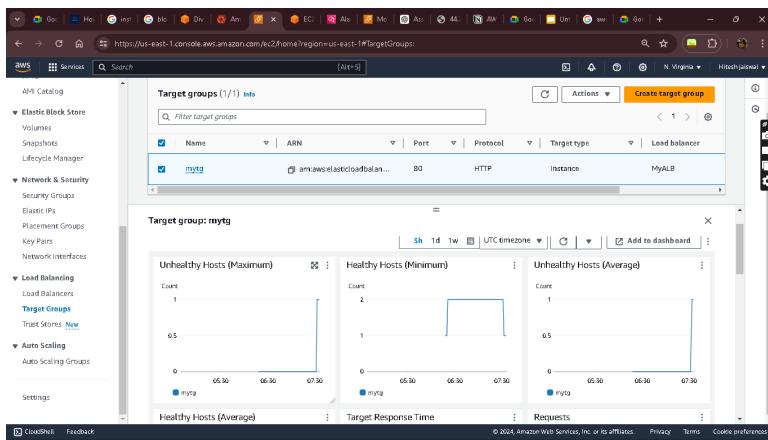
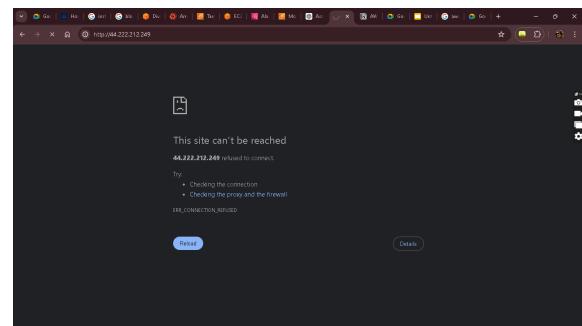
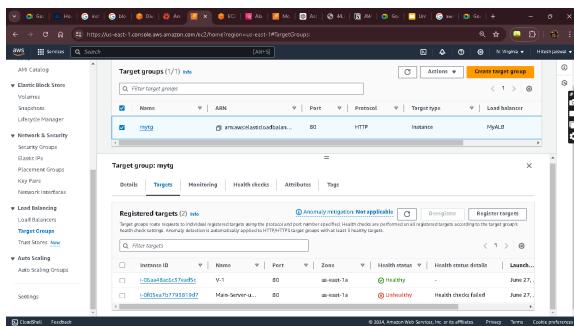
Two healthy targets in us-east-1a (10.0.1.114, 10.0.1.115) and one unhealthy target in us-east-1c (10.0.1.116).



To check the monitor I made server unhealthy by stopping the nginx



I stopped Nginx nginx on my main server It is showing Unhealthy to that Not the V1

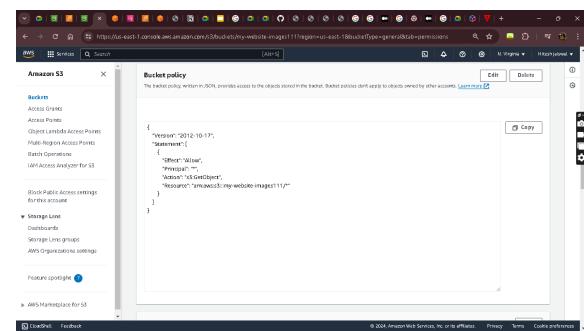
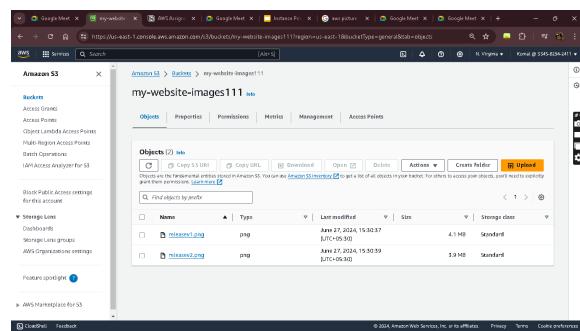


Make IAM user and give it s3 permission and do ssh through that so we don't need access and secret key we can check that while using curl website of that instance . Now S3 made

Images are Displayed using nginx the images are in S3 bucket

In Nginx html I have given image source of S3 bucket

I have also made customized bucket policy



```
{ "Version": "2012-10-17", "Statement": [ { "Effect": "Allow", "Principal": "*", "Action": "GetObject", "Resource": "arn:aws:s3:::my-website-images111/*" } ] }
```

