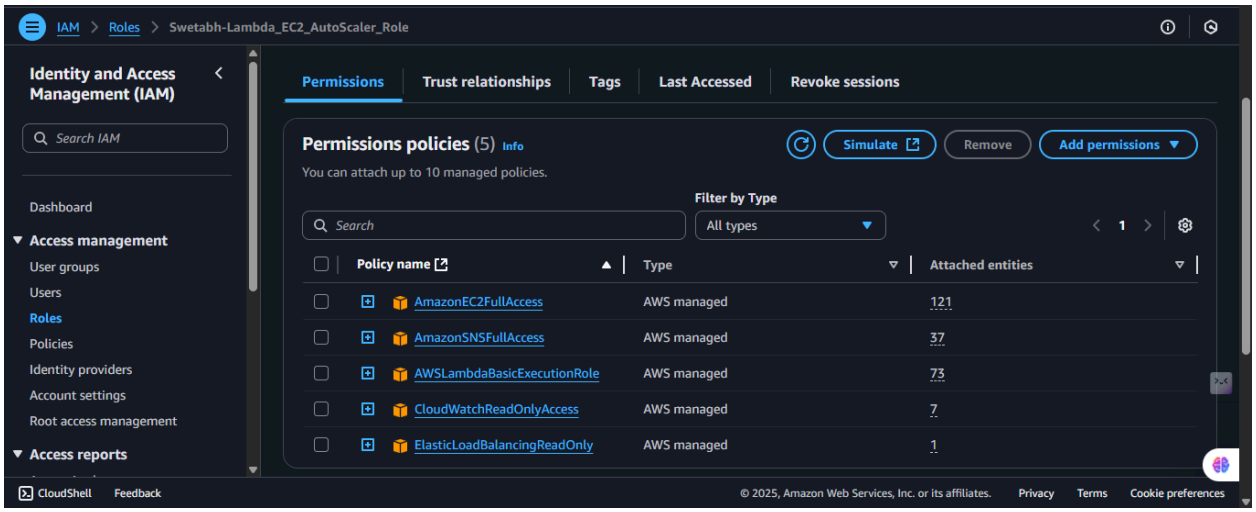
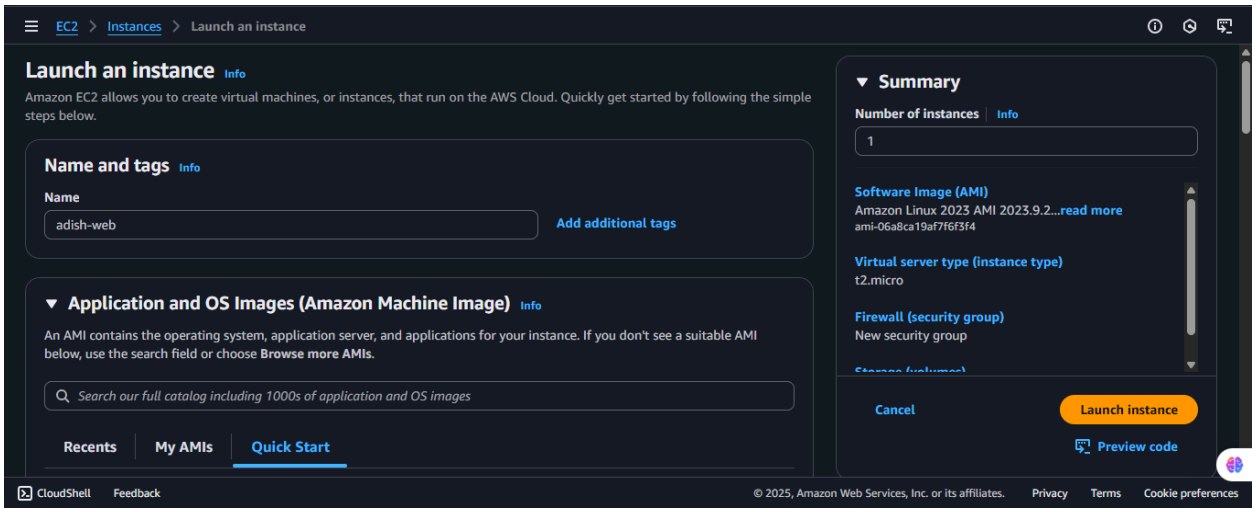
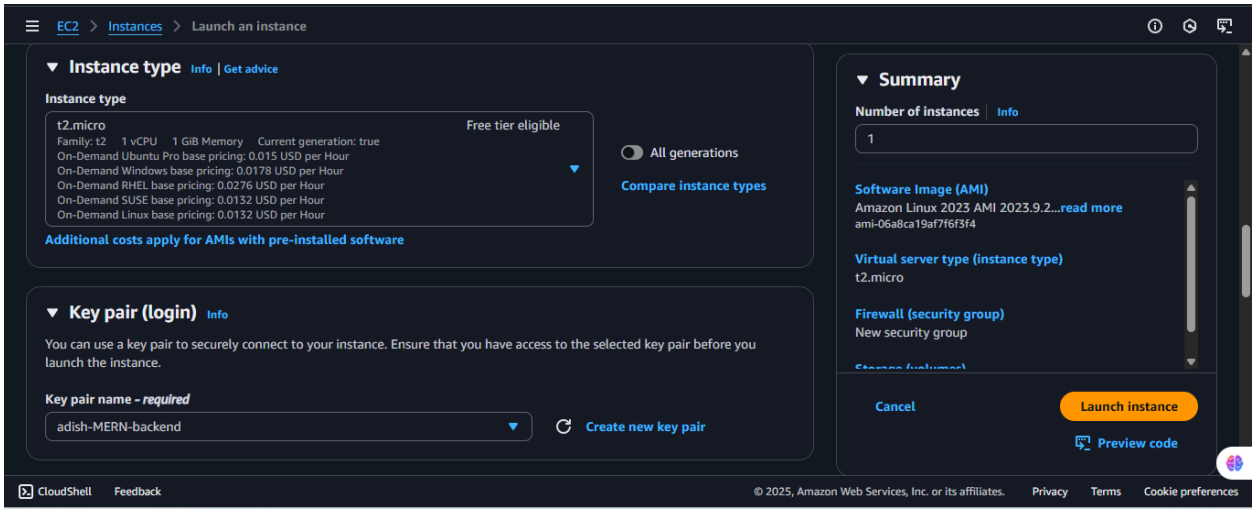
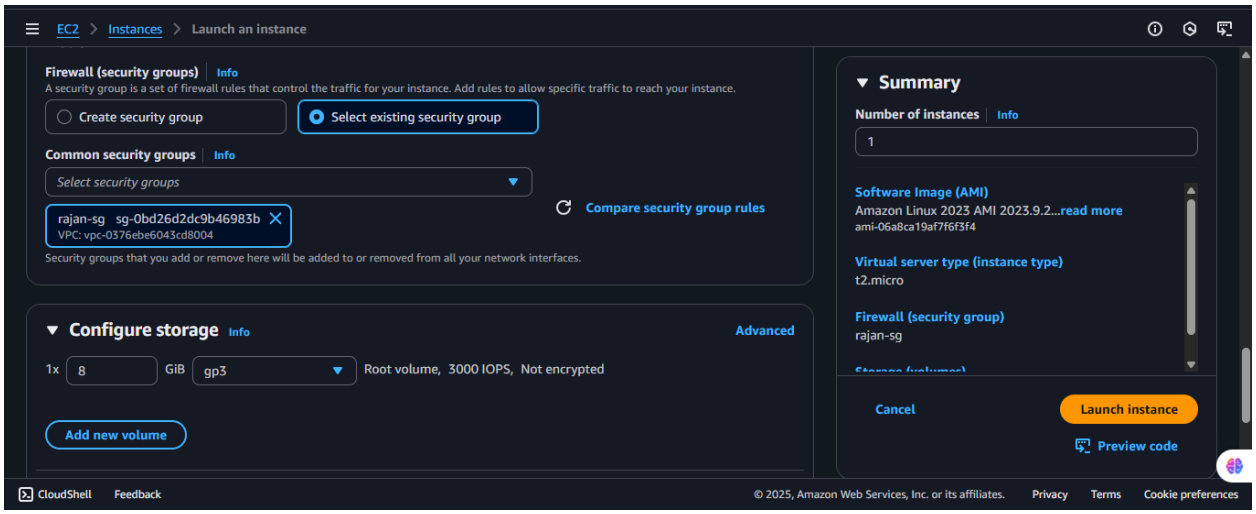
1:- Create IAM Role

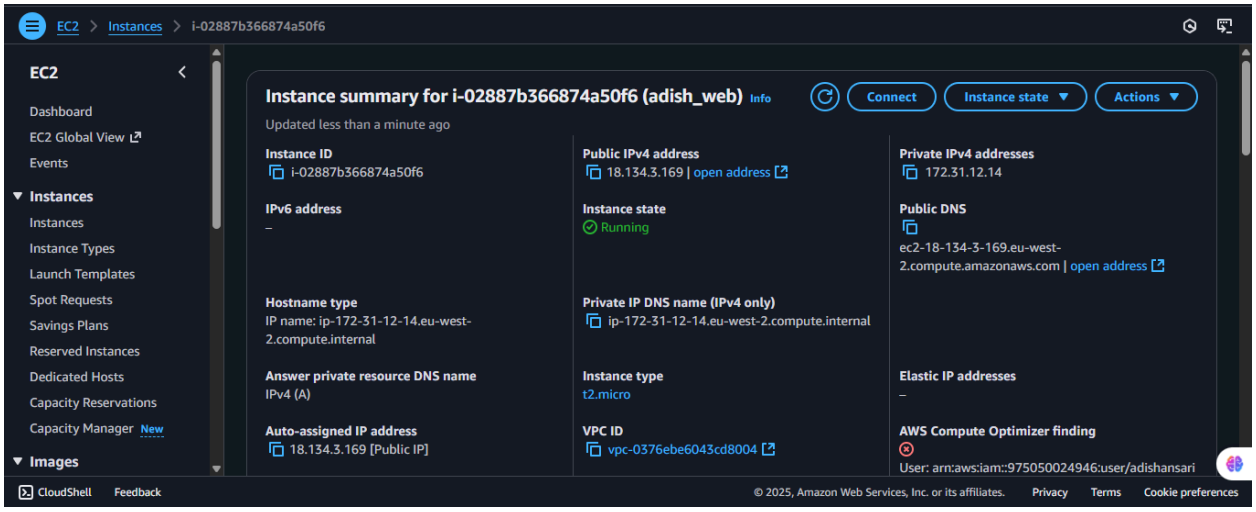


2:- Create EC2 Instance

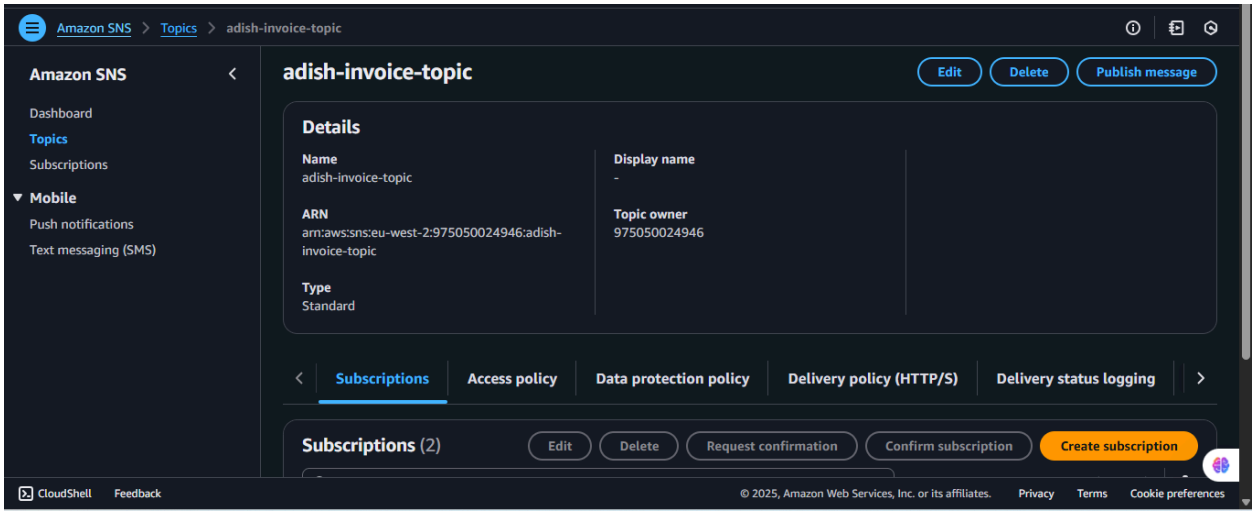




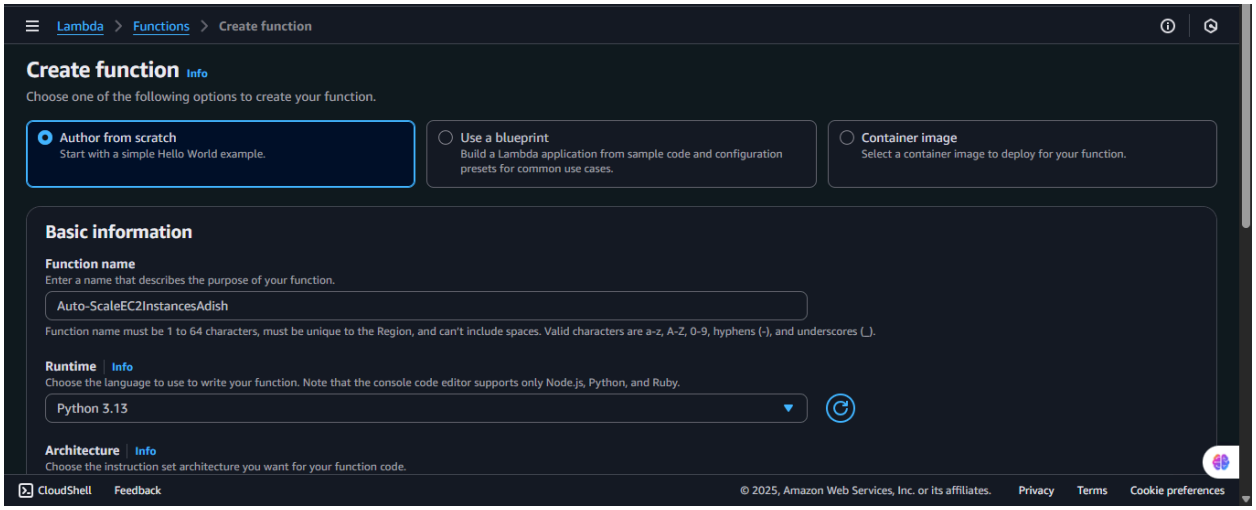


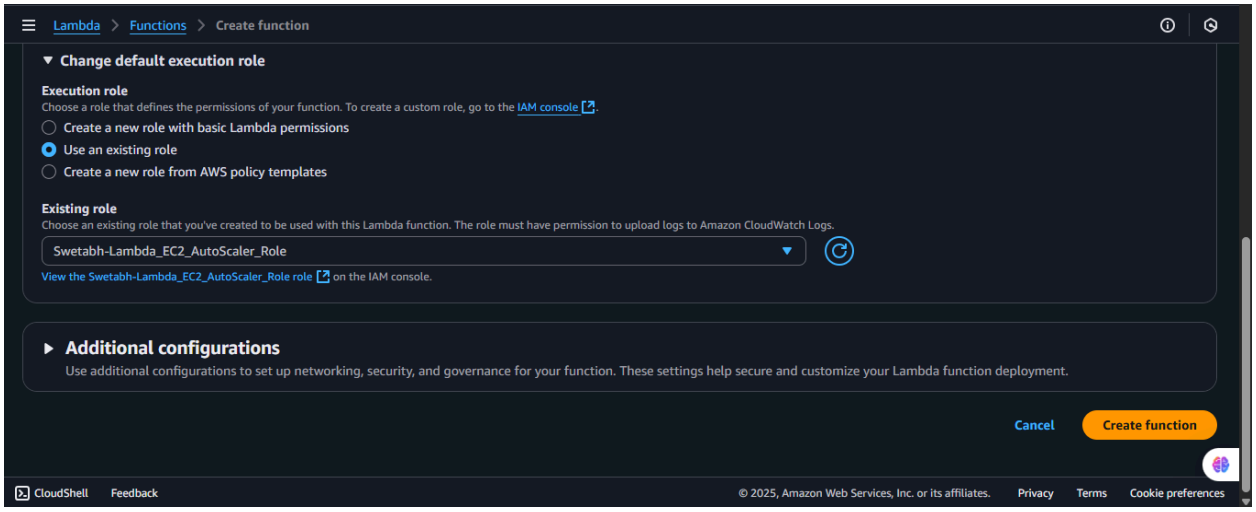


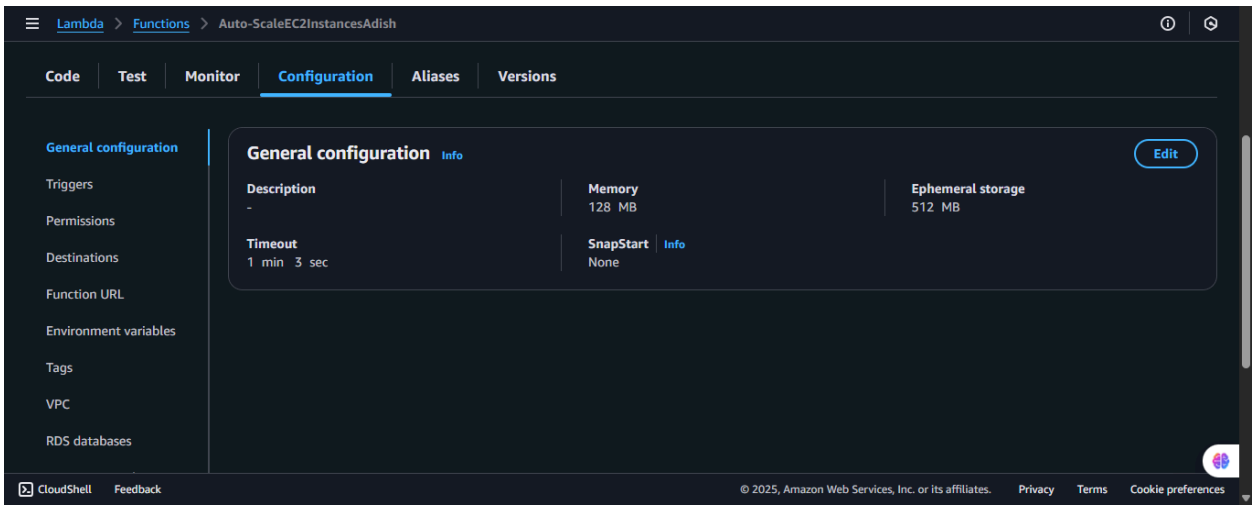
3:- Create SNS

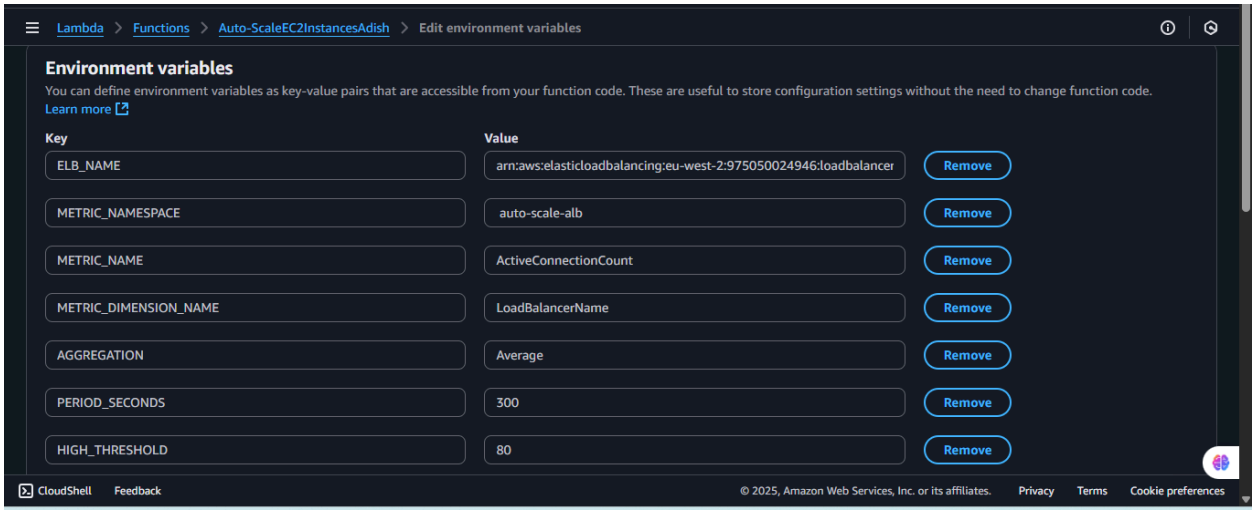


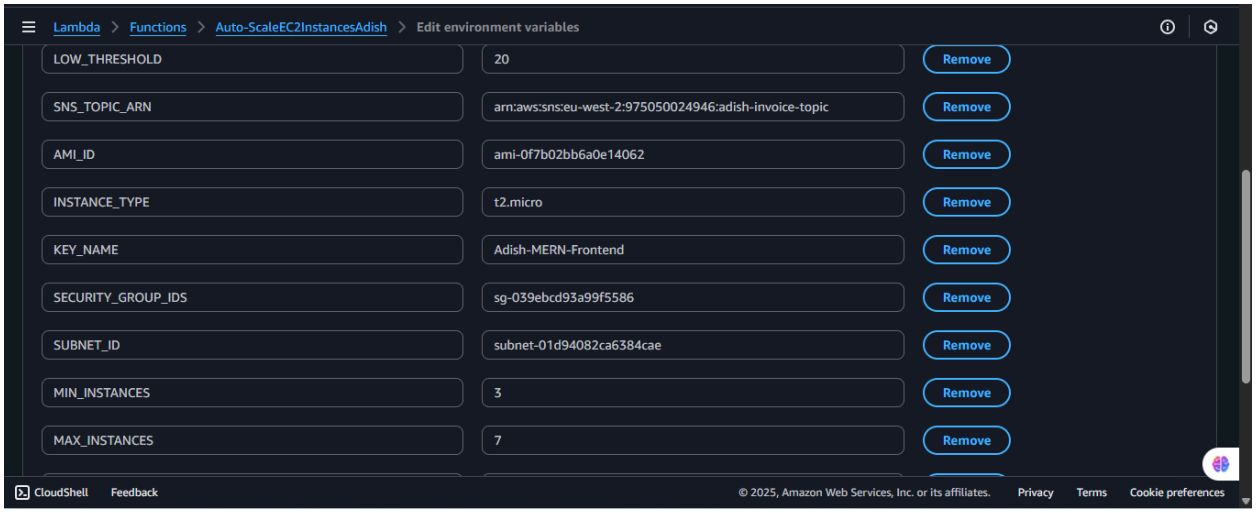
4 :- Create Lambda Function

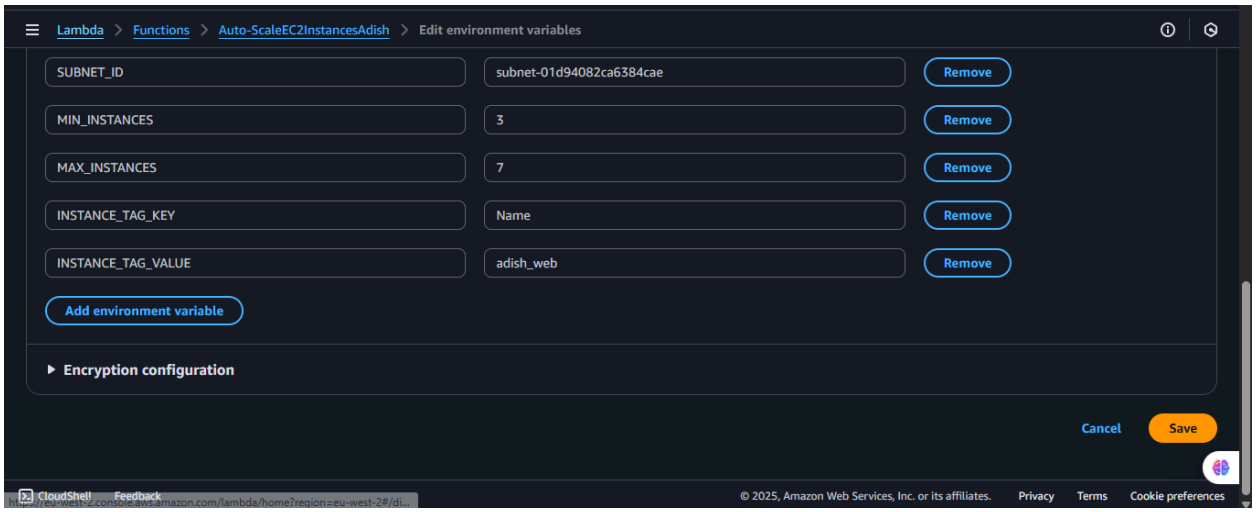

Lambda > Functions > Create function
Create function Info
Choose one of the following options to create your function.
Author from scratch
Start with a simple Hello World example.
Use a blueprint
Build a Lambda application from sample code and configuration presets for common use cases.
Container image
Select a container image to deploy for your function.
Basic information
Function name
Enter a name that describes the purpose of your function.
Auto-ScaleEC2InstancesAdish
Function name must be 1 to 64 characters, must be unique to the Region, and can't include spaces. Valid characters are a-z, A-Z, 0-9, hyphens (-), and underscores (_).
Runtime Info
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.
Python 3.13
Architecture Info
Choose the instruction set architecture you want for your function code.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Lambda > Functions > Create function
Change default execution role
Execution role
Choose a role that defines the permissions of your function. To create a custom role, go to the IAM console.
Create a new role with basic Lambda permissions
Use an existing role
Create a new role from AWS policy templates
Existing role
Choose an existing role that you've created to be used with this Lambda function. The role must have permission to upload logs to Amazon CloudWatch Logs.
Swetabh-Lambda_EC2_AutoScaler_Role
View the Swetabh-Lambda_EC2_AutoScaler_Role role on the IAM console.
Additional configurations
Use additional configurations to set up networking, security, and governance for your function. These settings help secure and customize your Lambda function deployment.
Cancel
Create function
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

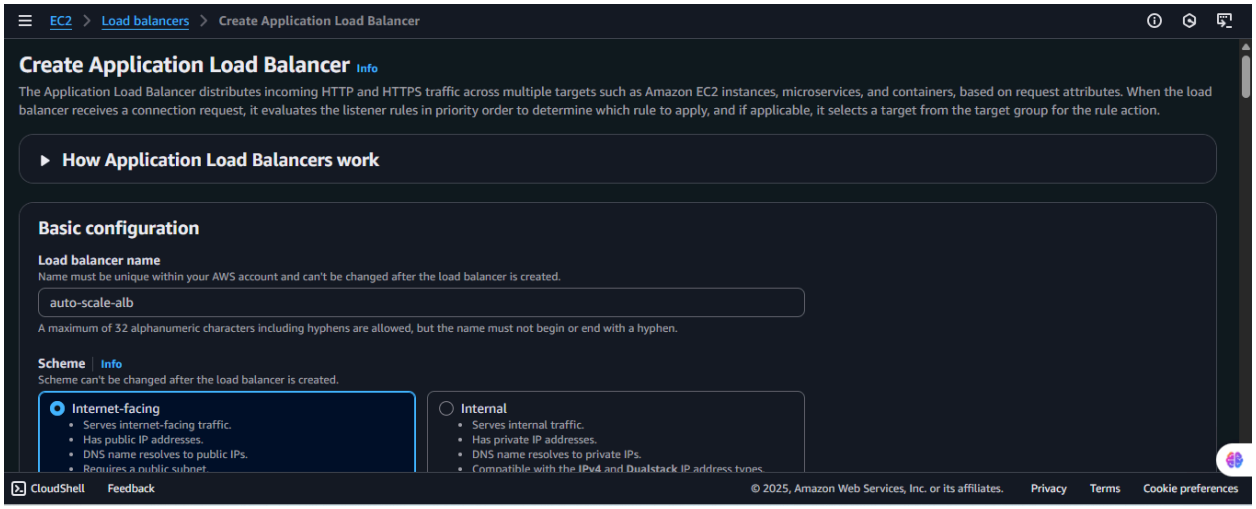

Lambda > Functions > Auto-ScaleEC2InstancesAdish
Code
Test
Monitor
Configuration
Aliases
Versions
General configuration
Triggers
Permissions
Destinations
Function URL
Environment variables
Tags
VPC
RDS databases
General configuration Info
Edit
Description
-
Memory
128 MB
Ephemeral storage
512 MB
Timeout
1 min 3 sec
SnapStart Info
None
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

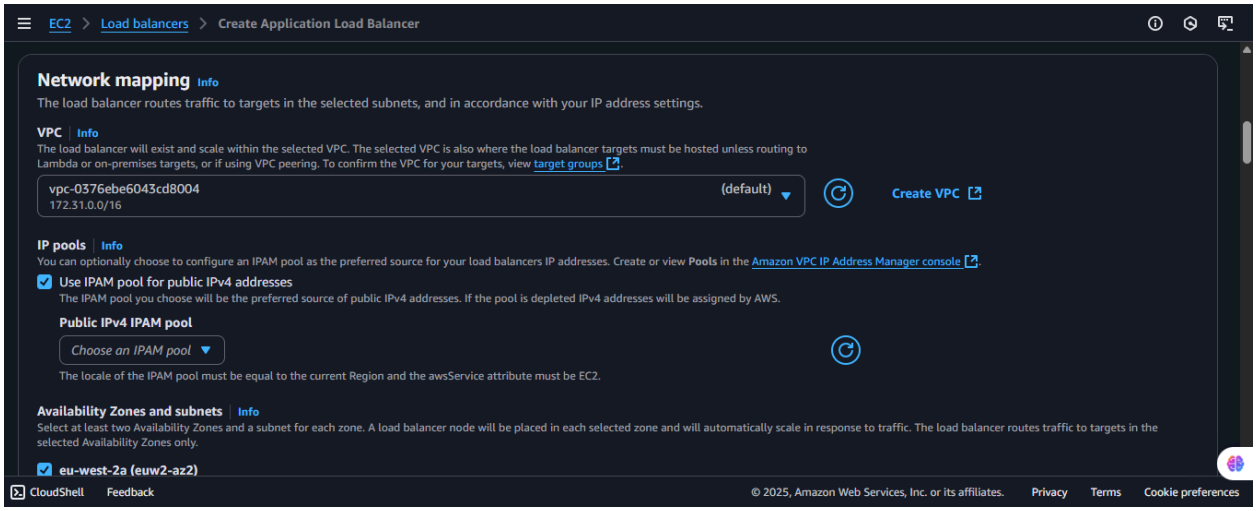

Lambda > Functions > Auto-ScaleEC2InstancesAdish > Edit environment variables
Environment variables
You can define environment variables as key-value pairs that are accessible from your function code. These are useful to store configuration settings without the need to change function code.
Learn more
Key
Value
ELB_NAME
arn:aws:elasticloadbalancing:eu-west-2:975050024946:loadbalancer
METRIC_NAMESPACE
auto-scale-alb
METRIC_NAME
ActiveConnectionCount
METRIC_DIMENSION_NAME
LoadBalancerName
AGGREGATION
Average
PERIOD_SECONDS
300
HIGH_THRESHOLD
80
Remove
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

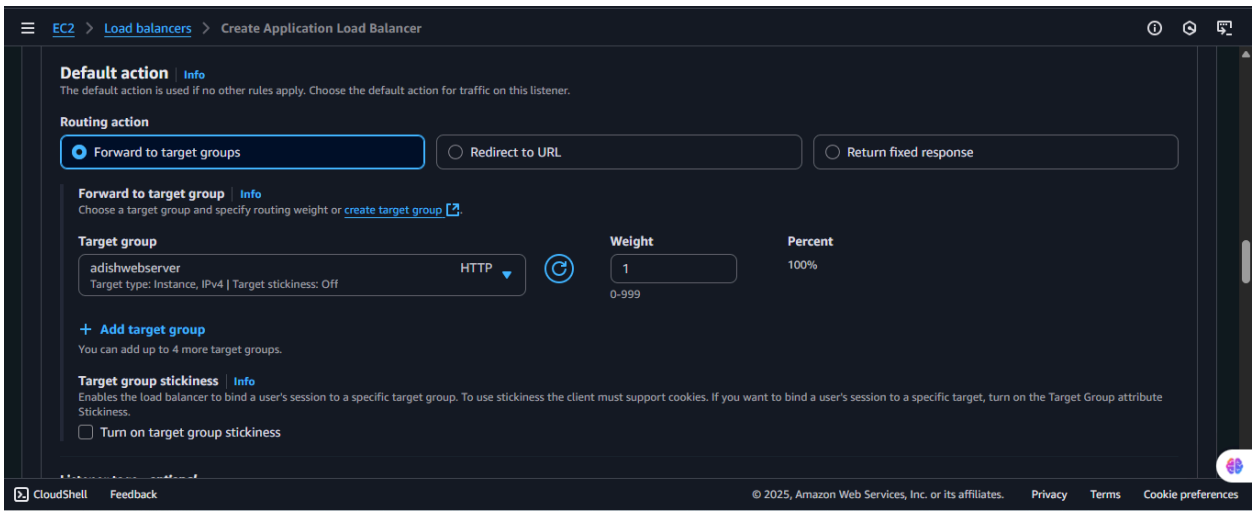

Lambda > Functions > Auto-ScaleEC2InstancesAdish > Edit environment variables
LOW_THRESHOLD
20
SNS_TOPIC_ARN
arn:aws:sns:eu-west-2:975050024946:adish-invoice-topic
AMI_ID
ami-0f7b02bb6a0e14062
INSTANCE_TYPE
t2.micro
KEY_NAME
Adish-MERN-Frontend
SECURITY_GROUP_IDS
sg-039ebcd93a99f5586
SUBNET_ID
subnet-01d94082ca6384cae
MIN_INSTANCES
3
MAX_INSTANCES
7
Remove
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


Lambda > Functions > Auto-ScaleEC2InstancesAdish > Edit environment variables
SUBNET_ID
subnet-01d94082ca6384cae
MIN_INSTANCES
3
MAX_INSTANCES
7
INSTANCE_TAG_KEY
Name
INSTANCE_TAG_VALUE
adish_web
Remove
Add environment variable
Encryption configuration
Cancel
Save
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

5:- Create ALB


EC2 > Load balancers > Create Application Load Balancer
Create Application Load Balancer Info
The Application Load Balancer distributes incoming HTTP and HTTPS traffic across multiple targets such as Amazon EC2 instances, microservices, and containers, based on request attributes. When the load balancer receives a connection request, it evaluates the listener rules in priority order to determine which rule to apply, and if applicable, it selects a target from the target group for the rule action.
How Application Load Balancers work
Basic configuration
Load balancer name
Name must be unique within your AWS account and can't be changed after the load balancer is created.
auto-scale-alb
A maximum of 32 alphanumeric characters including hyphens are allowed, but the name must not begin or end with a hyphen.
Scheme Info
Scheme can't be changed after the load balancer is created.
Internet-facing
Serves internet-facing traffic.
Has public IP addresses.
DNS name resolves to public IPs.
Internal
Serves internal traffic.
Has private IP addresses.
DNS name resolves to private IPs.
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


EC2 > Load balancers > Create Application Load Balancer
Network mapping Info
The load balancer routes traffic to targets in the selected subnets, and in accordance with your IP address settings.
VPC Info
The load balancer will exist and scale within the selected VPC. The selected VPC is also where the load balancer targets must be hosted unless routing to Lambda or on-premises targets, or if using VPC peering. To confirm the VPC for your targets, view target groups.
vpc-0376ebe6043cd8004
172.31.0.0/16
(default)
Create VPC
IP pools Info
You can optionally choose to configure an IPAM pool as the preferred source for your load balancers IP addresses. Create or view Pools in the Amazon VPC IP Address Manager console.
Use IPAM pool for public IPv4 addresses
The IPAM pool you choose will be the preferred source of public IPv4 addresses. If the pool is depleted IPv4 addresses will be assigned by AWS.
Public IPv4 IPAM pool
Choose an IPAM pool
The locale of the IPAM pool must be equal to the current Region and the awsService attribute must be EC2.
Availability Zones and subnets Info
Select at least two Availability Zones and a subnet for each zone. A load balancer node will be placed in each selected zone and will automatically scale in response to traffic. The load balancer routes traffic to targets in the selected Availability Zones only.
eu-west-2a (euw2-az2)
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences


EC2 > Load balancers > Create Application Load Balancer
Default action Info
The default action is used if no other rules apply. Choose the default action for traffic on this listener.
Routing action
Forward to target groups
Redirect to URL
Return fixed response
Forward to target group Info
Choose a target group and specify routing weight or create target group.
Target group
adishwebserver
HTTP
Target type: Instance, IPv4 | Target stickiness: Off
Weight
1
0-999
Percent
100%
Add target group
You can add up to 4 more target groups.
Target group stickiness Info
Enables the load balancer to bind a user's session to a specific target group. To use stickiness the client must support cookies. If you want to bind a user's session to a specific target, turn on the Target Group attribute Stickiness.
Turn on target group stickiness
CloudShell
Feedback
© 2025, Amazon Web Services, Inc. or its affiliates.
Privacy
Terms
Cookie preferences

```
import os
import boto3
import logging
from datetime import datetime, timedelta
from botocore.exceptions import ClientError

logger = logging.getLogger()
logger.setLevel(logging.INFO)

# clients
cw = boto3.client('cloudwatch')
ec2 = boto3.client('ec2')
sns = boto3.client('sns')

# Environment variables (set these in Lambda configuration)
ELB_NAME = os.environ.get('ELB_NAME')  # e.g.
"app/my-alb/1234567890abcdef"
METRIC_NAMESPACE = os.environ.get('METRIC_NAMESPACE',
'AWS/ApplicationELB')  # default for ALB
METRIC_NAME = os.environ.get('METRIC_NAME', 'RequestCount')  # or
'ActiveConnectionCount' for NLB, etc.
METRIC_DIMENSION_NAME = os.environ.get('METRIC_DIMENSION_NAME',
'LoadBalancer')  # for AWS/ApplicationELB
HIGH_THRESHOLD = float(os.environ.get('HIGH_THRESHOLD', '80'))  # percent
or absolute depending on metric
LOW_THRESHOLD = float(os.environ.get('LOW_THRESHOLD', '20'))
# Interpret thresholds relative to "per-instance" request capacity when
using RequestCount.
AGGREGATION = os.environ.get('AGGREGATION', 'Average')  # Average, Sum,
Maximum
PERIOD_SECONDS = int(os.environ.get('PERIOD_SECONDS', '300'))  # 5 minutes
SNS_TOPIC_ARN = os.environ.get('SNS_TOPIC_ARN')

# If you want Lambda to create instances directly:
AMI_ID = os.environ.get('AMI_ID')  # e.g. "ami-0123456789abcdef0"
INSTANCE_TYPE = os.environ.get('INSTANCE_TYPE', 't3.micro')
KEY_NAME = os.environ.get('KEY_NAME')  # optional
SECURITY_GROUP_IDS = os.environ.get('SECURITY_GROUP_IDS')  #
comma-separated ids
```

```
SUBNET_ID = os.environ.get('SUBNET_ID')  # subnet to launch into
(optional)
MIN_INSTANCES = int(os.environ.get('MIN_INSTANCES', '1'))  # do not go
below
MAX_INSTANCES = int(os.environ.get('MAX_INSTANCES', '5'))  # cap max
INSTANCE_TAG_KEY = os.environ.get('INSTANCE_TAG_KEY', 'AutoScaleManaged')
INSTANCE_TAG_VALUE = os.environ.get('INSTANCE_TAG_VALUE', 'true')

# Which instances to count/manage: filter by tag.
MANAGED_FILTERS = [{'Name': f'tag:{INSTANCE_TAG_KEY}', 'Values':
[INSTANCE_TAG_VALUE]}]

def publish_sns(subject: str, message: str):
    if not SNS_TOPIC_ARN:
        logger.warning("SNS_TOPIC_ARN not configured; skipping
notification.")
        return
    try:
        sns.publish(TopicArn=SNS_TOPIC_ARN, Subject=subject,
Message=message)
        logger.info("SNS published: %s", subject)
    except ClientError as e:
        logger.exception("Failed to publish SNS: %s", e)

def get_metric_value():
    """
    Fetch metric from CloudWatch for the last PERIOD_SECONDS window.
    Returns the requested statistic value (float) or None.
    """
    end = datetime.utcnow()
    start = end - timedelta(seconds=PERIOD_SECONDS)
    dims = [{'Name': METRIC_DIMENSION_NAME, 'Value': ELB_NAME}]
    try:
        resp = cw.get_metric_statistics(
            Namespace=METRIC_NAMESPACE,
            MetricName=METRIC_NAME,
            Dimensions=dims,
            StartTime=start,
            EndTime=end,
            Period=PERIOD_SECONDS,
```

```
            Statistics=[AGGREGATION]
        )
        datapoints = resp.get('Datapoints', [])
        if not datapoints:
            logger.info("No datapoints returned for metric query.")
            return None
        # pick the latest datapoint by timestamp
        latest = max(datapoints, key=lambda d: d['Timestamp'])
        value = latest.get(AGGREGATION)
        logger.info("Metric fetched: %s=%s (timestamp=%s)", METRIC_NAME,
value, latest.get('Timestamp'))
        return float(value)
    except ClientError:
        logger.exception("Error fetching metric")
        return None

def count_managed_instances():
    """Count running/stopped instances that match our tag filters (and are
EC2-managed by this function)."""
    try:
        resp = ec2.describe_instances(Filters=MANAGED_FILTERS + [{'Name':
'instance-state-name', 'Values':
['pending','running','stopping','stopped']}])
        instances = []
        for r in resp.get('Reservations', []):
            for i in r.get('Instances', []):
                instances.append(i)
        logger.info("Found %d managed instances", len(instances))
        return instances
    except ClientError:
        logger.exception("Failed to describe instances")
        return []

def start_new_instance():
    """Start a single new EC2 instance, tagged so it is managed."""
    if not AMI_ID:
        raise RuntimeError("AMI_ID not configured; cannot launch
instance.")
    sg_ids = [x.strip() for x in SECURITY_GROUP_IDS.split(',')] if
SECURITY_GROUP_IDS else None
```

```
    launch_args = {
        'ImageId': AMI_ID,
        'InstanceType': INSTANCE_TYPE,
        'MinCount': 1,
        'MaxCount': 1,
        'TagSpecifications': [{
            'ResourceType': 'instance',
            'Tags': [{'Key': INSTANCE_TAG_KEY, 'Value':
INSTANCE_TAG_VALUE}]
        }]
    }
    if sg_ids:
        launch_args['SecurityGroupIds'] = sg_ids
    if KEY_NAME:
        launch_args['KeyName'] = KEY_NAME
    if SUBNET_ID:
        launch_args['SubnetId'] = SUBNET_ID

    try:
        resp = ec2.run_instances(**launch_args)
        instance_id = resp['Instances'][0]['InstanceId']
        logger.info("Launched instance %s", instance_id)
        publish_sns("Scale Up: Launched EC2", f"Launched instance
{instance_id} due to high load.")
        return instance_id
    except ClientError:
        logger.exception("Failed to launch instance")
        return None

def terminate_one_instance(instances):
    """
    Terminate one instance from the provided list (prefer stopped or
oldest running).
    Instances is list of instance dicts from describe_instances.
    """
    # select candidate: prefer instances in 'stopped', then oldest
'running'
    stopped = [i for i in instances if i['State']['Name'] == 'stopped']
    if stopped:
        target = stopped[0]
```

```
    else:
        # pick oldest running (by LaunchTime)
        running = [i for i in instances if i['State']['Name'] ==
'running']
        if not running:
            logger.info("No running instances to terminate.")
            return None
        target = min(running, key=lambda i: i['LaunchTime'])

    inst_id = target['InstanceId']
    try:
        ec2.terminate_instances(InstanceIds=[inst_id])
        logger.info("Terminated instance %s", inst_id)
        publish_sns("Scale Down: Terminated EC2", f"Terminated instance
{inst_id} due to low load.")
        return inst_id
    except ClientError:
        logger.exception("Failed to terminate instance")
        return None

def lambda_handler(event, context):
    logger.info("Auto-scale Lambda started")
    metric_value = get_metric_value()
    if metric_value is None:
        logger.warning("Metric not available; exiting without action.")
        return {"status": "no_metric"}

    # You must decide what metric_value represents: percent or absolute.
    # This sample assumes metric is percent-like (0-100). If metric is
RequestCount, you may need to normalize to per-instance rate.
    try:
        managed = count_managed_instances()
        current_count = len([i for i in managed if i['State']['Name'] in
('pending', 'running', 'stopping', 'stopped')])
        logger.info("Current managed instance count: %d", current_count)
    except Exception as e:
        logger.exception("Failed to count managed instances: %s", e)
        current_count = 0

    # Scaling decisions
```

```
    # Scale up
    if metric_value >= HIGH_THRESHOLD and current_count < MAX_INSTANCES:
        logger.info("High threshold exceeded: %s >= %s", metric_value,
HIGH_THRESHOLD)
        launched = start_new_instance()
        return {"action": "scale_up", "launched": launched}

    # Scale down
    if metric_value <= LOW_THRESHOLD and current_count > MIN_INSTANCES:
        logger.info("Low threshold met: %s <= %s", metric_value,
LOW_THRESHOLD)
        terminated = terminate_one_instance(managed)
        return {"action": "scale_down", "terminated": terminated}

    logger.info("No scaling action required. metric=%s thresholds=(%s,%s)
count=%d", metric_value, LOW_THRESHOLD, HIGH_THRESHOLD, current_count)
    return {"action": "no_action", "metric": metric_value}
```