Assesement Documentation

Task-1

Commands using AWS & GCP - create, assign roles & permissions , policies and MFA Config – IAM

aws iam create-role --role-name MyRole --assume-role-policy-document file://"C:\trustpolicy.json" --> This command helps you create a role using trustpolicy . The trust policy defines which entities (e.g., AWS services, users) can assume the role.

Trustpolicy.json:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sts:AssumeRole"
            ],
            "Principal": {
                "Service": [
                    "ec2.amazonaws.com"
                ]
            }
        }
    ]
}
```

aws iam attach-role-policy --role-name MyRole --policy-arn arn:aws:iam::aws:policy/AmazonS3FullAccess --> Attach policy to role by using aws existing policies.

aws iam put-role-policy --role-name MyRole --policy-name MyPolicy --policy-document file://policy.json --> create and attach a custom inline policy.

Policy.json:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::assesmenttasks3bucket",
                "arn:aws:s3:::assesmenttasks3bucket/*"
            ]
        }
    ]
}
```

create user:

aws iam create-user --user-name <username> --> To create a user in IAM.

Attach policy to user:

aws iam attach-user-policy --user-name <username> --policy-arn arn:aws:iam::aws:policy/AmazonS3ReadOnlyAccess --> To attach a policy to user here we are attaching existing policy which is s3 readonly access.

aws iam put-user-policy --user-name <username> --policy-name MyPolicy --policy-document file://policy.json --> To attach a custom policy to the user .

aws iam get-role --role-name MyRoleName --> Check the role's trust policy and confirm it was created correctly.

aws iam list-attached-role-policies --role-name MyRoleName --> To Verify which AWS-managed or customer-managed policies are associated with the role.

aws iam list-role-policies --role-name MyRoleName --> To Identify any custom, inline policies directly created within the role (as opposed to attached managed policies).

aws iam delete-role --role-name <RoleName> --> To delete IAM role .

aws iam list-users --> To list IAM users .

aws iam get-user --user-name <UserName> --> To get details of a user .

aws iam delete-user --user-name <UserName> --> To delete a user .

aws iam list-mfa-devices --user-name <UserName> --> lists the MFA devices assigned to a specific user.

aws iam enable-mfa-device --user-name <UserName> --serial-number <SerialNumber> --authentication-code-1 <Code1> --authentication-code-2 <Code2> --> MFA device for a user, you need to assign the MFA device and provide its serial number and authentication codes.

aws iam deactivate-mfa-device --user-name <UserName> --serial-number <SerialNumber> --> To deactivate an MFA device for a user.

aws iam delete-virtual-mfa-device --serial-number <SerialNumber> --> To remove an MFA device permanently.

aws iam list-policies --scope AWS --> To list aws managed policies.

aws iam list-policies --scope Local --> To list customer managed policies.

aws iam detach-role-policy --role-name <RoleName> --policy-arn <PolicyArn> --> To detach a policy from a role.
aws iam put-role-policy --role-name <RoleName> --policy-name <PolicyName> --policy-document file://policy.json --> To create an inline policy.

aws iam delete-role-policy --role-name <RoleName> --policy-name <PolicyName> --> To delete an inline policy.

Gcloud commnads:

gcloud iam service-accounts create SERVICE_ACCOUNT_NAME \ --description="DESCRIPTION" \ --display-name="DISPLAY_NAME" --> create service account

gcloud projects add-iam-policy-binding PROJECT_ID \ --member="MEMBER_TYPE:EMAIL" \ --role="ROLE_NAME" --> Add a Member (User or Service Account)

Grant a Role to a Member:

```
gcloud projects add-iam-policy-binding PROJECT_ID \
   --member="MEMBER_TYPE:EMAIL" \
   --role="ROLE_NAME"\
```

Revoke a Role from a Member:
gcloud projects remove-iam-policy-binding PROJECT_ID \ --member="MEMBER_TYPE:EMAIL" \ --role="ROLE_NAME"

Create a Custom Role:

gcloud iam roles create ROLE_ID

--project=PROJECT_ID

```
}

--title="TITLE"

--description="DESCRIPTION"

--permissions="PERMISSION_1,PERMISSION_2"

--stage="GA"
```

Update a Custom Role:

```
gcloud iam roles update ROLE_ID \ --project=PROJECT_ID \ --
permissions="UPDATED_PERMISSIONS"
```

View IAM Policy:

```
gcloud projects get-iam-policy PROJECT_ID
```

Policy.json:

```
{
  "bindings": [
    {
      "role": "ROLE_NAME",
      "members": [
        "MEMBER_TYPE:EMAIL"
      ]
    }
  ]
}
```

Add a Condition to a Role:

```
gcloud projects add-iam-policy-binding PROJECT_ID
--member="MEMBER_TYPE:EMAIL"
--role="ROLE_NAME"
--condition="title=CONDITION_TITLE,expression=CONDITION_EXPRESSION"
```

Enable MFA:

```
gcloud resource-manager org-policies enable-enforce \
   constraints/iam.allowedPolicyMemberDomains \
   --organization=ORGANIZATION_ID
```

Task-2

S3 bucket

aws s3api create-bucket --bucket <bucket name> --region <region name> --> To create s3 bucket.

aws s3 cp "C:\filename" s3://Bucket name/ --> To upload file to bucket.

aws s3 ls s3://Bucket name/ --> To list objects in s3 bucket.

IAM attaching policy to s3

aws iam create-policy --policy-name S3Policy --policy-document file://"C:\s3policy.json" --> To create a policy named s3 policy .

s3policy.json:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "s3:ListBucket",
                "s3:GetObject",
                "s3:PutObject"
            ],
            "Resource": [
                "arn:aws:s3:::assesmenttasks3bucket",
                "arn:aws:s3:::assesmenttasks3bucket/*"
            ]
        }
    ]
}
```

aws iam attach-user-policy --user-name r-user --policy-arn arn:aws:iam::354918405309: policy/S3Policy --> To attach policy to user.

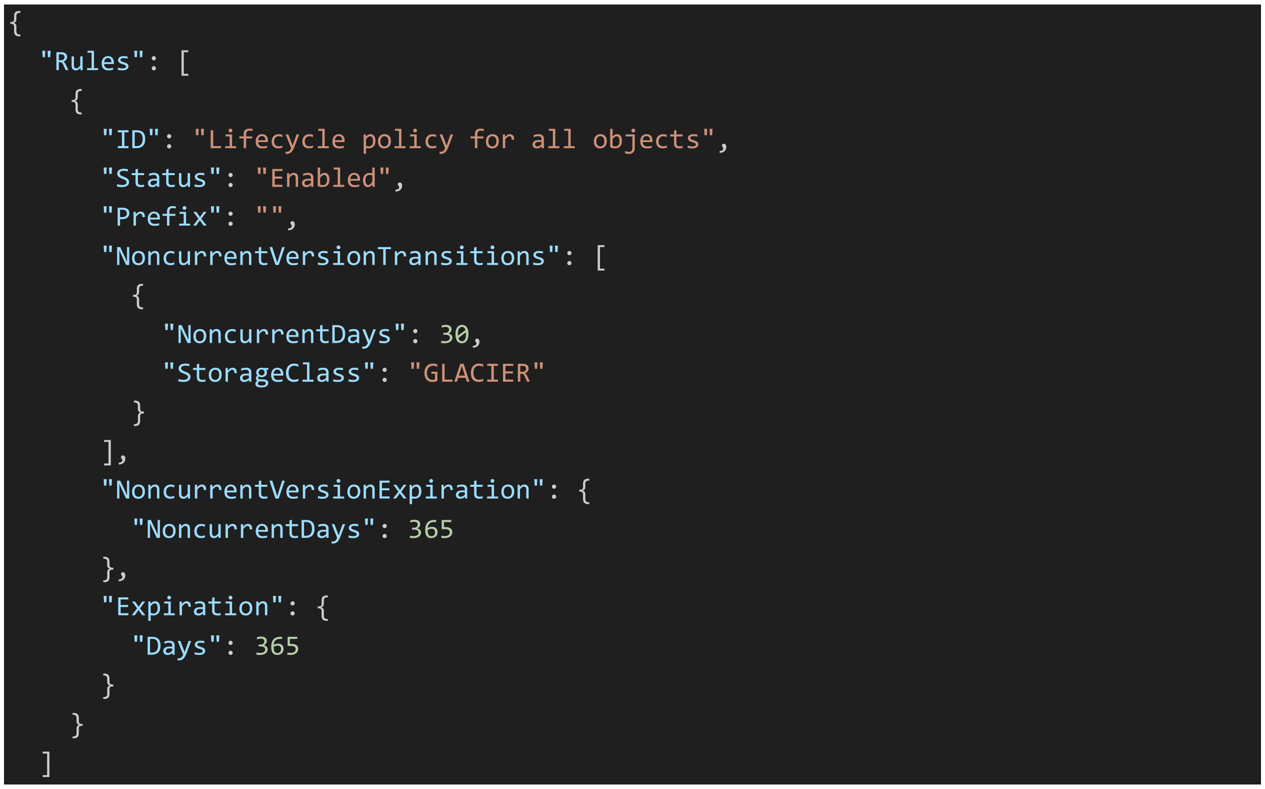
Create group
aws iam create-group --group-name s3group --> To create a group.

Attach policy to group
aws iam attach-group-policy --group-name s3group --policy-arn arn:aws:iam::354918405309: policy/S3Policy --> To attach a policy to group.

Add user to group
aws iam add-user-to-group --user-name r-user --group-name s3group --> To attach user to group.

Bucket versioning on s3
aws s3api put-bucket-versioning --bucket <bucket name> --versioning-configuration Status=Enabled --> To enable versioning on s3 bucket.

Lifecycle configuration on s3
aws s3api put-bucket-lifecycle-configuration --bucket assesmenttasks3bucket --lifecycle-configuration file://"C:\lifecycle.json" --> To put lifecycle configuration on s3 bucket.
Lifecycle.json:



```
{
  "Rules": [
    {
      "ID": "Lifecycle policy for all objects",
      "Status": "Enabled",
      "Prefix": "",
      "NoncurrentVersionTransitions": [
        {
          "NoncurrentDays": 30,
          "StorageClass": "GLACIER"
        }
      ],
      "NoncurrentVersionExpiration": {
        "NoncurrentDays": 365
      },
      "Expiration": {
        "Days": 365
      }
    }
  ]
}
```

In the above json file you can select storage class and non-concurent days so that after that it is moved to different storage class.
As mentioned above it will be moved to Glacier after 30 days, this helps in saving cost and after 365 days it will be deleted.

Enable object-Lock on s3(Retention):

aws s3api put-object-lock-configuration --bucket assesmenttasks3bucket --object-lock-configuration "ObjectLockEnabled=Enabled,Rule={DefaultRetention={Mode=GOVERNANCE,Days=30}}" --> To enable retention Allows bucket owners or users with special permissions to delete or overwrite objects, but they must explicitly bypass the governance mode using a `BypassGovernanceRetention` permission.

aws s3api put-object-lock-configuration --bucket assesmenttasks3bucket --object-lock-configuration "ObjectLockEnabled=Enabled,Rule={DefaultRetention={Mode=COMPLIANCE,Days=30}}" --> To enable retention, Objects are completely immutable. No user, not even the root account, can delete or overwrite the objects during the retention period.

Create a Bucket

Using gsutil

gsutil mb -l -c gs://

: The region where the bucket is created (e.g., us-central1). : The storage class (e.g., STANDARD, NEARLINE, COLDLINE).

Using gcloud

gcloud storage buckets create --location=

--storage-class=

Upload Files

Using gsutil

gsutil cp gs:///

Using gcloud

gcloud storage cp gs:///

Using gsutil

To assign a role (e.g., roles/storage.objectViewer) to a user: gsutil iam ch user::roles/storage.objectViewer gs://

To assign a role to a group: gsutil iam ch group::roles/storage.objectViewer gs://

Using gcloud

Assign a role to a user: gcloud storage buckets add-iam-policy-binding

--member=user:

--role=roles/storage.objectViewer

Assign a role to a group: gcloud storage buckets add-iam-policy-binding

--member=group:

--role=roles/storage.objectViewer

Enable Versioning

Using gsutil

gsutil versioning set on gs://

Using gcloud

gcloud storage buckets update --versioning

Set Lifecycle Policies

Using gsutil

Create a lifecycle configuration JSON file, e.g., lifecycle.json: { "rule": [{ "action": {"type": "Delete"}, "condition": {"age": 30} }] } Then apply it to the bucket:

gsutil lifecycle set lifecycle.json gs://

Using gcloud

gcloud storage buckets update

--lifecycle-file=lifecycle.json

Set Retention Policy

Using gsutil

To set a 30-day retention policy: gsutil retention set 30d gs://

Using gcloud

gcloud storage buckets update --retention-period=30d

Task -3

To Create 2 VMS, Setup , Elastic/Application Load Balancer , Deploy helloworld Application, Running lb dns - should show the helloworld, Configure Auto Scaling & whenever server crashes/down , it should auto create the server.

Below is **main.tf** file to create all the required resources mentioned above.

Main.tf

```
# VPC Resource
resource "aws_vpc" "Hello_vpc" {
  cidr_block           = var.vpc_cidr
  enable_dns_support   = true
  enable_dns_hostnames = true

  tags = {
    Name = "Hello_vpc"
  }
}

# Subnets
resource "aws_subnet" "subnets" {
  count                   = length(var.subnet_cidrs)
  vpc_id                  = aws_vpc.Hello_vpc.id
  cidr_block              = var.subnet_cidrs[count.index]
  availability_zone       = var.availability_zones[count.index]
  map_public_ip_on_launch = true

  tags = {
    Name = "Subneth${count.index + 1}"
  }
}

# Internet Gateway
resource "aws_internet_gateway" "Hello_igw" {
  vpc_id = aws_vpc.Hello_vpc.id

  tags = {
    Name = "Hello-igw"
  }
}

# Route Table
resource "aws_route_table" "main_route_table" {
  vpc_id = aws_vpc.Hello_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.Hello_igw.id
  }

  tags = {
    Name = "HelloRouteTable"
  }
}

# Route Table Associations
resource "aws_route_table_association" "subnet_assoc" {
  count          = length(aws_subnet.subnets)
  subnet_id      = aws_subnet.subnets[count.index].id
  route_table_id = aws_route_table.main_route_table.id
}

# Security Group for Instances
resource "aws_security_group" "hello_sg" {
  vpc_id = aws_vpc.Hello_vpc.id

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "WebSecurityGroup"
  }
```

```
}

# Instance Resource
resource "aws_instance" "app" {
  ami           = var.ami_id
  instance_type = var.instance_type
  count         = 1
  key_name      = var.key_name
  subnet_id     = aws_subnet.subnets[0].id
  vpc_security_group_ids = [aws_security_group.hello_sg.id]

  user_data = <<-EOF
             #!/bin/bash
             sudo yum update -y
             sudo yum install -y httpd
             echo "<h1>Hello, World!</h1>" > /var/www/html/index.html
             sudo systemctl start httpd
             sudo systemctl enable httpd
             EOF

  tags = {
    Name = "HelloWorldApp-${count.index}"
  }

  lifecycle {
    create_before_destroy = true
  }
}

# Security Group for Load Balancer
resource "aws_security_group" "lb_sg" {
  name_prefix = "lb-sg"
  vpc_id      = aws_vpc.Hello_vpc.id

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# Load Balancer
resource "aws_lb" "app_lb" {
  name               = "app-lb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.lb_sg.id]
  subnets            = aws_subnet.subnets[*].id

  tags = {
    Name = "AppLoadBalancer"
  }

  lifecycle {
    create_before_destroy = true
  }
}

# Target Group
resource "aws_lb_target_group" "app_tg" {
  name     = "app-tg"
  port     = 80
  protocol = "HTTP"
  vpc_id   = aws_vpc.Hello_vpc.id

  health_check {
    path                = "/"
    interval            = 30
    timeout             = 5
    healthy_threshold   = 2
    unhealthy_threshold = 2
    matcher             = "200"
  }
}

# Load Balancer Listener
resource "aws_lb_listener" "http" {
  load_balancer_arn = aws_lb.app_lb.arn
  port                 = "80"
  protocol             = "HTTP"

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.app_tg.arn
  }
}

# Launch Template
resource "aws_launch_template" "app_lt" {
  name          = "app-lt"
  image_id      = var.ami_id
  instance_type = var.instance_type
  key_name      = var.key_name
  vpc_security_group_ids = [aws_security_group.lb_sg.id]

  user_data = base64encode(<<-EOF
    #!/bin/bash
    sudo yum update -y
    sudo yum install -y httpd
    echo "<h1>Hello, World!</h1>" > /var/www/html/index.html
    sudo systemctl start httpd
    sudo systemctl enable httpd
    EOF
  )

  tag_specifications {
    resource_type = "instance"

    tags = {
      Name = "HelloWorldApp"
    }
  }
}

# Auto Scaling Group
resource "aws_autoscaling_group" "app_asg" {
  launch_template {
    id      = aws_launch_template.app_lt.id
    version = "$Latest"
  }
  min_size             = var.min_capacity
  max_size             = var.max_capacity
  desired_capacity     = var.desired_capacity
  vpc_zone_identifier = aws_subnet.subnets[*].id

  target_group_arns = [aws_lb_target_group.app_tg.arn]

  tag {
    key                 = "Name"
    value               = "HelloWorldApp"
    propagate_at_launch = true
  }

  lifecycle {
    create_before_destroy = true
  }
}
```

Here in this main.tf terraform script Firstly we create required resources for ec2 instances to run on.

Here we create vpc named Hello_vpc

- **`cidr_block`:** Specifies the IP range for the VPC.
- **`enable_dns_support`:** Allows DNS resolution within the VPC.
- **`enable_dns_hostnames`:** Assigns DNS hostnames to instances in the VPC.
- **Tags:** Metadata for easier identification.

```
resource "aws_vpc" "Hello_vpc" {
  cidr_block           = var.vpc_cidr
  enable_dns_support   = true
  enable_dns_hostnames = true

  tags = {
    Name = "Hello_vpc"
  }
}
```

Here we create subnets in vpc

- **`count`:** Creates multiple subnets, one for each CIDR block in `var.subnet_cidrs`.
- **`availability_zone`:** Ensures subnets are spread across different zones for fault tolerance.
- **`map_public_ip_on_launch`:** Assigns public IPs to instances launched in these subnets.

```
resource "aws_subnet" "subnets" {
  count                   = length(var.subnet_cidrs)
  vpc_id                  = aws_vpc.Hello_vpc.id
  cidr_block              = var.subnet_cidrs[count.index]
  availability_zone       = var.availability_zones[count.index]
  map_public_ip_on_launch = true

  tags = {
    Name = "Subneth${count.index + 1}"
  }
}
```

Then we create internet gateway and attach to vpc and create a route

`vpc_id`: Associates the gateway with the VPC.

Then we create routing table it manages routing rules for the VPC.

`route`: Directs all traffic (`0.0.0.0/0`) through the internet gateway

```
# Internet Gateway
resource "aws_internet_gateway" "Hello_igw" {
  vpc_id = aws_vpc.Hello_vpc.id

  tags = {
    Name = "Hello-igw"
  }
}

# Route Table
resource "aws_route_table" "main_route_table" {
  vpc_id = aws_vpc.Hello_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.Hello_igw.id
  }

  tags = {
    Name = "HelloRouteTable"
  }
}
```

Here routetable association Links each subnet to the route table, ensuring traffic flows correctly.

Then we create a security group it Controls inbound and outbound traffic for EC2 instances.

Inbound Rules: Allows HTTP (port 80) and SSH (port 22) access from any IP.
Outbound Rules: Allows all outbound traffic.

```
resource "aws_route_table_association" "subnet_assoc" {
  count          = length(aws_subnet.subnets)
  subnet_id      = aws_subnet.subnets[count.index].id
  route_table_id = aws_route_table.main_route_table.id
}

# Security Group for Instances
resource "aws_security_group" "hello_sg" {
  vpc_id = aws_vpc.Hello_vpc.id

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "WebSecurityGroup"
  }
}
```

Then we launch an EC2 instance with a web server.Also we create them in above vpc and subnets .

`ami`: Specifies the AMI for the instance.

User Data: Configures the instance to install and start Apache, serving a "Hello, World!" web page.

```
resource "aws_instance" "app" {
  ami           = var.ami_id
  instance_type = var.instance_type
  count         = 1
  key_name      = var.key_name
  subnet_id     = aws_subnet.subnets[0].id
  vpc_security_group_ids = [aws_security_group.hello_sg.id]

  user_data = <<-EOF
            #!/bin/bash
            sudo yum update -y
            sudo yum install -y httpd
            echo "<h1>Hello, World!</h1>" > /var/www/html/index.html
```

```
              sudo systemctl start httpd
              sudo systemctl enable httpd
              EOF

  tags = {
    Name = "HelloWorldApp-${count.index}"
  }

  lifecycle {
    create_before_destroy = true
  }
}
```

We then create a load balancer security group in same vpc as above

Inbound Rules: Allows HTTP traffic on port 80 from any IP.

Outbound Rules: Allows all outbound traffic.

After above step we create Load balancer Distributes incoming HTTP traffic across instances.

- **`security_groups`:** Secures the load balancer using the previously defined security group.
- **`subnets`:** Spreads the load balancer across multiple subnets.

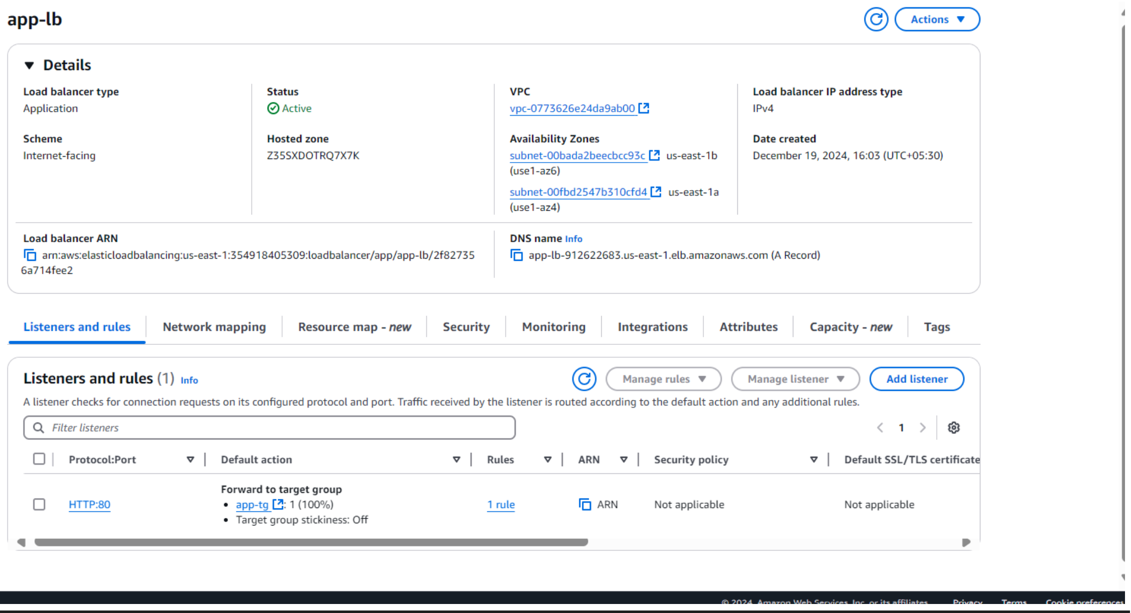
Target group defines a group of instances for the load balancer to forward traffic to.
Health Check: Verifies the health of targets via HTTP requests.

We then create a load balancer listner it Configures the load balancer to listen for HTTP traffic on port 80 and forward it to the target group.

```
resource "aws_security_group" "lb_sg" {
  name_prefix = "lb-sg"
  vpc_id      = aws_vpc.Hello_vpc.id

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# Load Balancer
resource "aws_lb" "app_lb" {
  name               = "app-lb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.lb_sg.id]
  subnets            = aws_subnet.subnets[*].id

  tags = {
    Name = "AppLoadBalancer"
  }

  lifecycle {
    create_before_destroy = true
  }
}

# Target Group
resource "aws_lb_target_group" "app_tg" {
  name     = "app-tg"
  port     = 80
  protocol = "HTTP"
  vpc_id   = aws_vpc.Hello_vpc.id

  health_check {
    path                = "/"
    interval            = 30
    timeout             = 5
    healthy_threshold   = 2
    unhealthy_threshold = 2
    matcher             = "200"
  }
}

# Load Balancer Listener
resource "aws_lb_listener" "http" {
  load_balancer_arn = aws_lb.app_lb.arn
  port              = "80"
  protocol          = "HTTP"

  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.app_tg.arn
  }
}
```

We create a launch tempalte it defines reusable configuration for launching EC2 instances.

- **User Data:** Same as the individual instance configuration



app-lb

Details

Load balancer type Application	**Status** Active	**VPC** vpc-0773626e24da9ab00	**Load balancer IP address type** IPv4
Scheme Internet-facing	**Hosted zone** Z35SXDOTRQ7X7K	**Availability Zones** subnet-00bada2beecbcc93c us-east-1b (use1-az6) subnet-00fbd2547b310cfd4 us-east-1a (use1-az4)	**Date created** December 19, 2024, 16:03 (UTC+05:30)

Load balancer ARN arn:aws:elasticloadbalancing:us-east-1:354918405309:loadbalancer/app/app-lb/2f82735 6a714fee2

DNS name Info app-lb-912622683.us-east-1.elb.amazonaws.com (A Record)

Listeners and rules

Listeners and rules (1) Info

A listener checks for connection requests on its configured protocol and port. Traffic received by the listener is routed according to the default action and any additional rules.

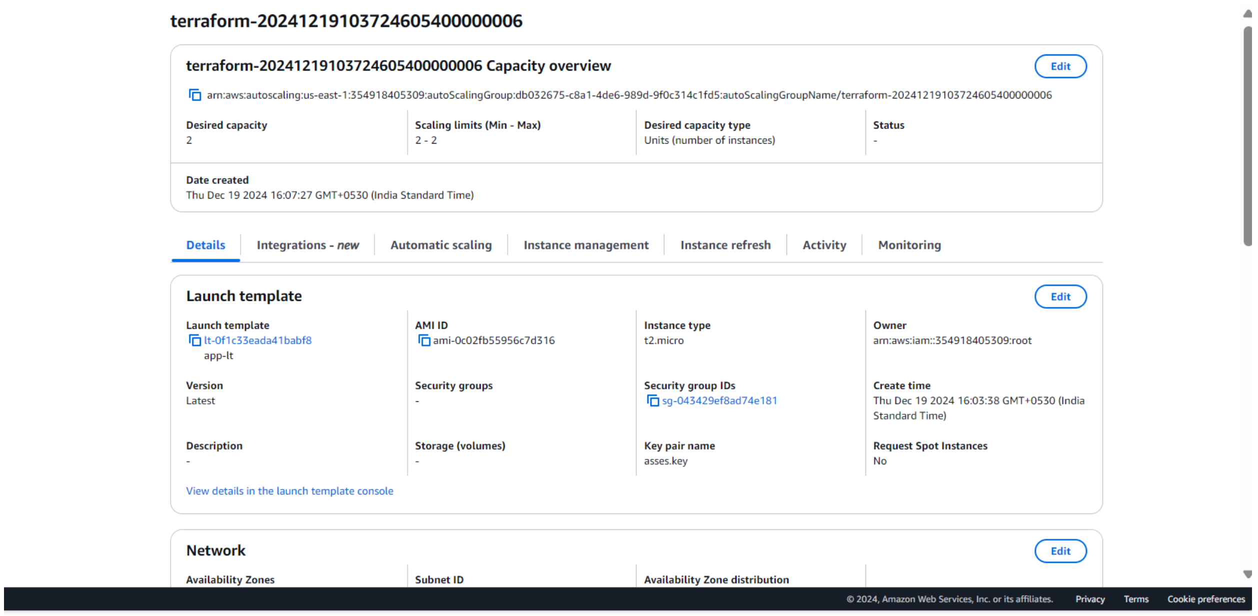
Filter listeners

Protocol:Port	Default action	Rules	ARN	Security policy	Default SSL/TLS certificate
HTTP:80	Forward to target group • app-tg: 1 (100%) • Target group stickiness: Off	1 rule	ARN	Not applicable	Not applicable

© 2024, Amazon Web Services, Inc. or its affiliates. Privacy Terms Cookie preferences

```
resource "aws_launch_template" "app_lt" {
  name          = "app-lt"
  image_id      = var.ami_id
  instance_type = var.instance_type
  key_name      = var.key_name
  vpc_security_group_ids = [aws_security_group.lb_sg.id]

  user_data = base64encode(<<-EOF
    #!/bin/bash
    sudo yum update -y
    sudo yum install -y httpd
    echo "<h1>Hello, World!</h1>" > /var/www/html/index.html
    sudo systemctl start httpd
    sudo systemctl enable httpd
    EOF
  )

  tag_specifications {
    resource_type = "instance"

    tags = {
      Name = "HelloWorldApp"
    }
  }
}
```

- We then create an autoscaling group to Automatically scale the number of instances based on demand.we mention **min,max and desired capacity** to autoscale the instances if any instance goes down.
- **`launch_template`:** Uses the defined launch template.
- **`target_group_arns`:** Associates the Auto Scaling group with the load balancer's target group.



terraform-20241219103724605400000006

terraform-20241219103724605400000006 Capacity overview

arn:aws:autoscaling:us-east-1:354918405309:autoScalingGroup:db032675-c8a1-4de6-989d-9f0c314c1fd5:autoScalingGroupName/terraform-20241219103724605400000006

Desired capacity	Scaling limits (Min - Max)	Desired capacity type	Status
2	2 - 2	Units (number of instances)	-

Date created
Thu Dec 19 2024 16:07:27 GMT+0530 (India Standard Time)

Launch template

Launch template lt-0f1c33eada41babf8 app-lt	**AMI ID** ami-0c02fb55956c7d316	**Instance type** t2.micro	**Owner** arn:aws:iam::354918405309:root
Version Latest	**Security groups** -	**Security group IDs** sg-043429ef8ad74e181	**Create time** Thu Dec 19 2024 16:03:38 GMT+0530 (India Standard Time)
Description -	**Storage (volumes)** -	**Key pair name** asses.key	**Request Spot Instances** No

View details in the launch template console

Network

Availability Zones | Subnet ID | Availability Zone distribution

© 2024, Amazon Web Services, Inc. or its affiliates.

```
resource "aws_autoscaling_group" "app_asg" {
  launch_template {
    id      = aws_launch_template.app_lt.id
    version = "$Latest"
  }

  min_size            = var.min_capacity
  max_size            = var.max_capacity
  desired_capacity    = var.desired_capacity
  vpc_zone_identifier = aws_subnet.subnets[*].id

  target_group_arns = [aws_lb_target_group.app_tg.arn]

  tag {
    key                 = "Name"
    value               = "HelloWorldApp"
    propagate_at_launch = true
  }

  lifecycle {
    create_before_destroy = true
  }
```

Variable.tf

Below are the reasons we use variable.tf

This Variables allow you to use the same code for different environments (e.g., production, staging) by passing different values.

Changes can be made in one place (e.g., updating the region or instance type).

Easier to handle configurations across large infrastructures.

Keeps configuration values separate from resource definitions, making the code cleaner and more readable.

```
variable "region" {
  type = string
}

variable "vpc_cidr" {
  type = string
}

variable "subnet_cidrs" {
  type = list(string)
}

variable "availability_zones" {
  type = list(string)
}

variable "ami_id" {
  type = string
}

variable "instance_type" {
  type = string
}

variable "key_name" {
  type = string
}

variable "min_capacity" {
  type = number
}

variable "max_capacity" {
  type = number
}

variable "desired_capacity" {
  type = number
}

provider "aws" {
  region = var.region
}
```

By combining variables and a provider, your Terraform script becomes highly flexible and configurable, making it easier to deploy and manage infrastructure across multiple environments. Let me know if you need any further clarifications!

Terraformtf.vars

By using a `terraform.tfvars` file, you can separate the **infrastructure code** (which defines the resources) from the **configuration data** (which provides the actual values for those resources). This promotes reusability and flexibility, as you can change variables without modifying the core infrastructure code.

It is useful for managing **different environments** (e.g., development, staging, production) by providing environment-specific variable values in separate `.tfvars` files.

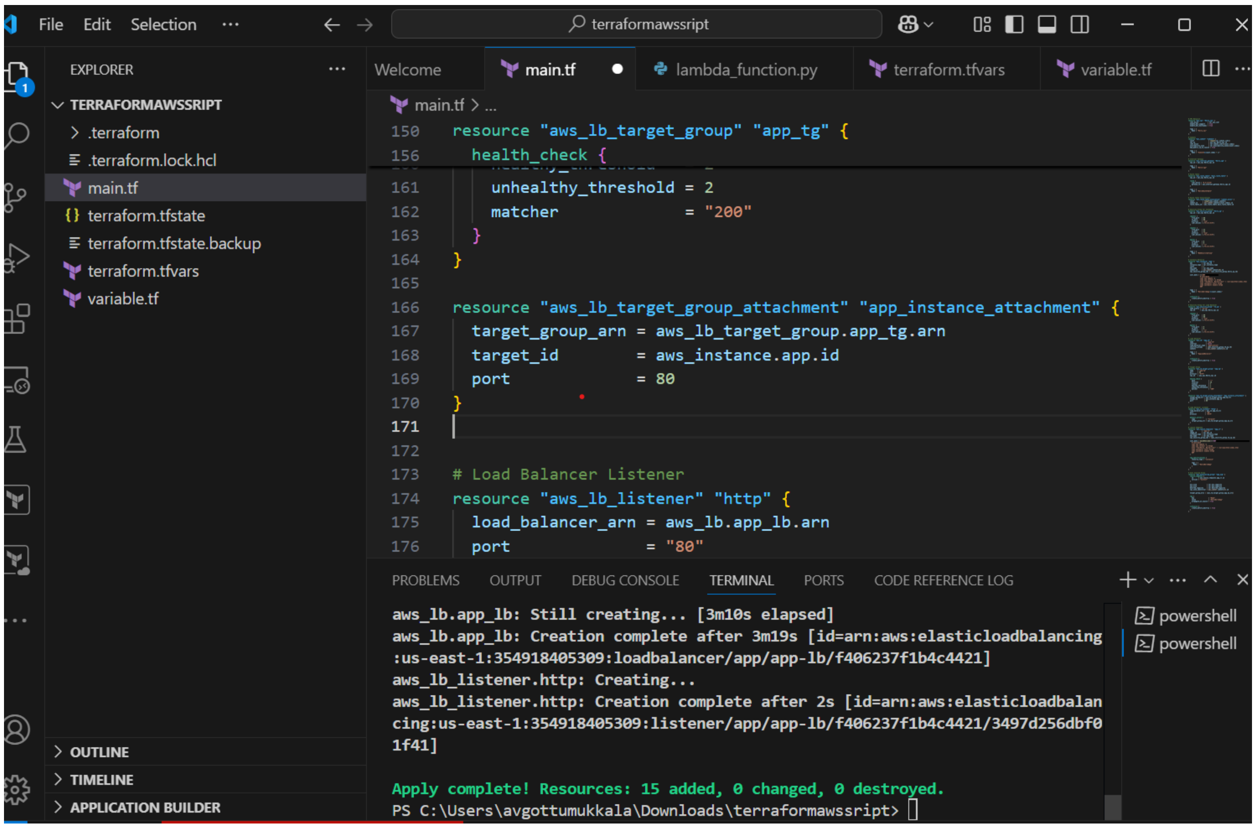
```
region = "us-east-1"
vpc_cidr = "10.0.0.0/16"
subnet_cidrs = ["10.0.1.0/24", "10.0.2.0/24"]
availability_zones = ["us-east-1a", "us-east-1b"]
ami_id = "ami-0c02fb55956c7d316"
instance_type = "t2.micro"
key_name = "asses.key"
```

```
min_capacity = 1
max_capacity = 3
desired_capacity = 2
```

We create above terraform files main.tf,variable.tf and terraformtf.vars and then we login to aws using aws configure we give aws acess key and secret access key once we are logged in we run terraform init command It initializes your working directory and prepares Terraform to run correctly.

Then we run terraform plan command it is used to preview the changes Terraform will make to your infrastructure. It doesn't make any actual changes; rather, it shows what actions Terraform would take if you run terraform apply. Once this is successful we run below command .

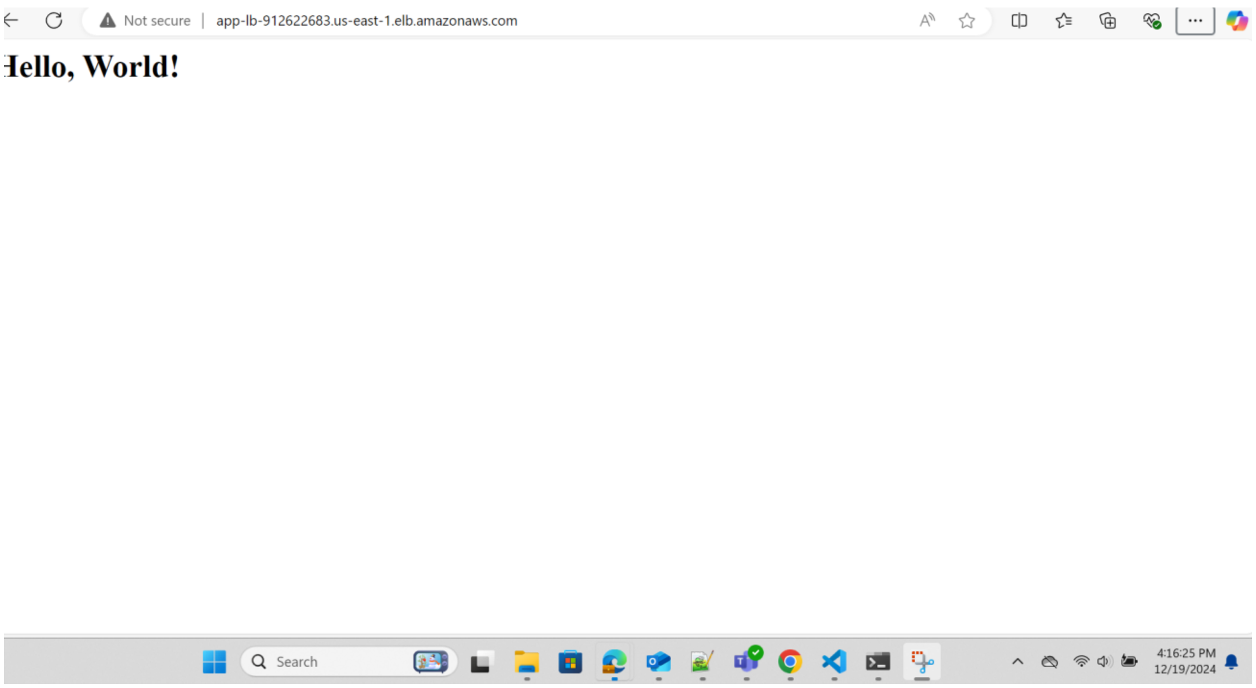
We run terraform apply this creates the infrastructure in aws as mentioned in terraform script and all the resources are created.



```
resource "aws_lb_target_group" "app_tg" {
  health_check {
    unhealthy_threshold = 2
    matcher             = "200"
  }
}

resource "aws_lb_target_group_attachment" "app_instance_attachment" {
  target_group_arn = aws_lb_target_group.app_tg.arn
  target_id        = aws_instance.app.id
  port             = 80
}


# Load Balancer Listener
resource "aws_lb_listener" "http" {
  load_balancer_arn = aws_lb.app_lb.arn
  port              = "80"
```

```
aws_lb.app_lb: Still creating... [3m10s elapsed]
aws_lb.app_lb: Creation complete after 3m19s [id=arn:aws:elasticloadbalancing
:us-east-1:354918405309:loadbalancer/app/app-lb/f406237f1b4c4421]
aws_lb_listener.http: Creating...
aws_lb_listener.http: Creation complete after 2s [id=arn:aws:elasticloadbalan
cing:us-east-1:354918405309:listener/app/app-lb/f406237f1b4c4421/3497d256dbf0
1f41]

Apply complete! Resources: 15 added, 0 changed, 0 destroyed.
PS C:\Users\avgottumukkala\Downloads\terraformawssript>
```

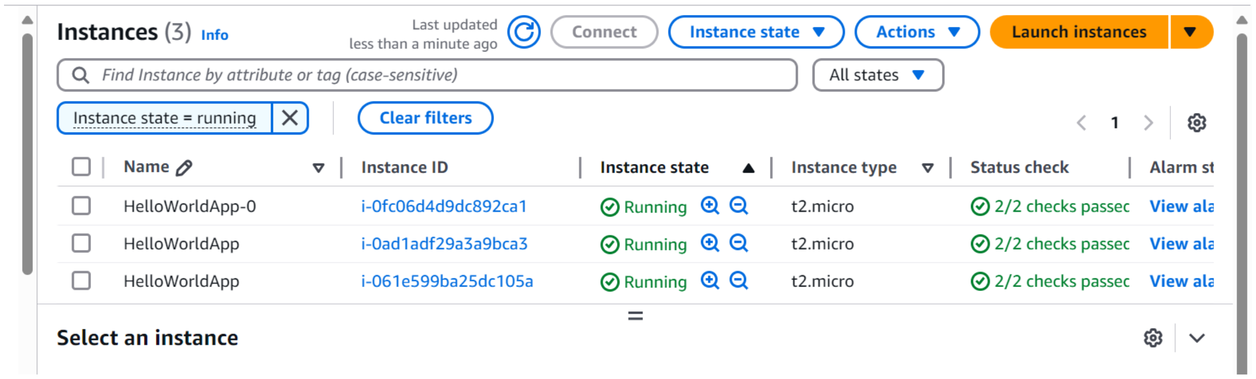
Below screenshot shows the helloworld is running using lb-dns.


Not secure | app-lb-912622683.us-east-1.elb.amazonaws.com

Hello, World!

4:16:25 PM
12/19/2024

This shows that the ec2 instances are running


Instances (3) Info

Last updated less than a minute ago

Find Instance by attribute or tag (case-sensitive)

Instance state = running

Name	Instance ID	Instance state	Instance type	Status check	Alarm st
HelloWorldApp-0	i-0fc06d4d9dc892ca1	Running	t2.micro	2/2 checks passec	View ala
HelloWorldApp	i-0ad1adf29a3a9bca3	Running	t2.micro	2/2 checks passec	View ala
HelloWorldApp	i-061e599ba25dc105a	Running	t2.micro	2/2 checks passec	View ala

Select an instance

For this task you Need aws cli and terrform installed in your local host .

Task –4

Write the python script and configure in AWS Lambda - Start/stop the Vms , Upload the file from server to S3 Bucket, Download the file from S3 bucket to Server.

Before we do this we have some prerequisites they are:

1. Ensure the EC2 instance is accessible and has the necessary permissions for file transfer if needed. ()

2. Create an S3 bucket for uploading and downloading files.

3. Use the AWS Management Console or AWS CLI to create the Lambda function.

4. Lambda should have an IAM role with following permissions

EC2 permissions: To start and stop EC2 instances "ec2:StartInstances","ec2:StopInstances","ec2:DescribeInstances" or ec2 fullaccess

Systems Manager (SSM): To execute commands on EC2 instances using Systems Manager (SSM)

"ssm:SendCommand","ssm:ListCommandInvocations", "ssm:GetCommandInvocation", "ssm:DescribeInstanceInformation"

S3access: To upload and download files from S3.

"s3:PutObject", "s3:GetObject", "s3:ListBucket",s3:fullaccess

5. Permissions for EC2 Instances

S3 Permissions:To upload and download files.(same as above)

SSM Permissions (Optional if using SSM to execute commands):To allow Systems Manager access(same as above).

6. Permissions for S3 Bucket :

Ensure the S3 bucket has appropriate permissions. The IAM Role for Lambda or EC2 should be explicitly allowed access. Optionally, you can use an S3 bucket policy to restrict access to specific roles or action.

Example:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::account-id:role/your-lambda-or-ec2-role"
      },
      "Action":
    "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": [
        "arn:aws:s3:::your-bucket-name",
        "arn:aws:s3:::your-bucket-name/*"
      ]
    }
  ]
}
```

Then go to lambda function created if you havent created follow below steps

Create the Lambda Function

1. **Go to the Lambda Console**:
   a. Navigate to the AWS Lambda Management Console.
2. **Create a New Function**:
   a. Click **Create function**.
   b. Choose **Author from scratch**.
   c. Provide a function name (e.g., `StartStopVMs`).
3. **Runtime**:
   a. Select the runtime for your function (e.g., Python 3.9).
4. **Execution Role**:
   a. Under **Execution role**, select:
      i. **Use an existing role**.
      ii. Choose the role created in **Step 1** from the dropdown.
5. **Create the Function**:
   a. Click **Create function**

Then get into lambda function deploy the python script and press on deploy you can directly deploy it there or you can create a zip file in local machine and deploy it .

Below is the python code

```
import boto3
import os
import time

# AWS Clients
ec2 = boto3.client('ec2')
s3 = boto3.client('s3')
ssm = boto3.client('ssm')

def lambda_handler(event, context):
    action = event.get("action", "").lower()  # Define the action via event
trigger

    if action == "start_ec2":
        return start_ec2_instances(event)
    elif action == "stop_ec2":
        return stop_ec2_instances(event)
    elif action == "upload_file":
        return upload_file_to_s3(event)
    elif action == "download_file":
        return download_file_from_s3(event)
    else:
        return {"message": "Invalid action. Use start_ec2, stop_ec2, upload_file,
or download_file."}

# 1. Start EC2 Instances
def start_ec2_instances(event):
    instance_ids = event.get("instance_ids", [])
    if not instance_ids:
        return {"message": "Instance IDs not provided."}

    response = ec2.start_instances(InstanceIds=instance_ids)
    return {"message": f"Starting instances: {instance_ids}", "response":
response}

# 2. Stop EC2 Instances
def stop_ec2_instances(event):
    instance_ids = event.get("instance_ids", [])
    if not instance_ids:
        return {"message": "Instance IDs not provided."}

    response = ec2.stop_instances(InstanceIds=instance_ids)
```

```python
    return {"message": f"Stopping instances: {instance_ids}", "response":
response}

# 3. Upload a File to S3
def upload_file_to_s3(event):
    instance_id = event.get("instance_id")
    local_file_path = event.get("local_file_path")
    s3_bucket = event.get("s3_bucket")
    s3_key = event.get("s3_key", os.path.basename(local_file_path))

    if not all([instance_id, local_file_path, s3_bucket]):
        return {"message": "Missing parameters for file upload."}

    try:
        # Access the working directory of the EC2 instance
        working_directory = get_current_working_directory(instance_id)
        if not working_directory:
            return {"message": "Could not get the working directory from the EC2
instance."}

        # Ensure the file exists in the working directory (or modify for another
directory)
        local_file_path = os.path.join(working_directory, local_file_path)

        # Upload file to S3 using SSM
        command = f"aws s3 cp {local_file_path} s3://{s3_bucket}/{s3_key}"
        run_command_on_instance(instance_id, command)

        return {"message": f"File uploaded to S3 bucket {s3_bucket} with key
{s3_key}"}

    except Exception as e:
        return {"error": str(e)}

# 4. Download a File from S3 to EC2 instance
def download_file_from_s3(event):
    instance_id = event.get("instance_id")
    s3_bucket = event.get("s3_bucket")
    s3_key = event.get("s3_key")
    remote_file_path = event.get("remote_file_path")

    if not all([instance_id, s3_bucket, s3_key, remote_file_path]):
        return {"message": "Missing parameters for file download."}

    try:
        # Download file from S3 to EC2 instance using SSM
        command = f"aws s3 cp s3://{s3_bucket}/{s3_key} {remote_file_path}"
        run_command_on_instance(instance_id, command)

        return {"message": f"File {s3_key} downloaded from S3 bucket {s3_bucket}
to {remote_file_path}"}
    except Exception as e:
        return {"error": str(e)}

# Helper Function: Get Current Working Directory from EC2 instance
def get_current_working_directory(instance_id):
    try:
        # Run "pwd" command on the EC2 instance to get the current working
directory
        command = "pwd"
        response = ssm.send_command(
            InstanceIds=[instance_id],
            DocumentName="AWS-RunShellScript",
            Parameters={"commands": [command]}
        )

        command_id = response['Command']['CommandId']
        max_retries = 5
        retry_delay = 2  # seconds

        # Retry until the command completes
        for attempt in range(max_retries):
            time.sleep(retry_delay)
            invocation_response = ssm.get_command_invocation(
                CommandId=command_id,
                InstanceId=instance_id,
            )

            if invocation_response['Status'] == "Success":
                # Return the current working directory
                return invocation_response['StandardOutputContent'].strip()
            elif invocation_response['Status'] == "Failed":
                raise Exception(
                    f"Command failed:
{invocation_response['StandardErrorContent']}"
```

```
                )

        # If all retries are exhausted, raise an exception
        raise Exception("Failed to retrieve working directory within retry
limit.")

    except Exception as e:
        print(f"Error getting working directory: {str(e)}")
        return None

# Helper Function: Run a Command on EC2 Instance Using SSM
def run_command_on_instance(instance_id, command):
    response = ssm.send_command(
        InstanceIds=[instance_id],
        DocumentName="AWS-RunShellScript",
        Parameters={"commands": [command]}
    )

    command_id = response['Command']['CommandId']
    max_retries = 5
    retry_delay = 2  # seconds

    for attempt in range(max_retries):
        time.sleep(retry_delay)
        invocation_response = ssm.get_command_invocation(
        CommandId=command_id,
        InstanceId=instance_id,
      )

      if invocation_response['Status'] == "Success":
        return invocation_response['StandardOutputContent']
      elif invocation_response['Status'] == "Failed":
        raise Exception(f"Command failed: {invocation_response['StandardErrorContent']}")

   raise Exception("Command execution did not complete within retry limit.")
```

Components of the Script

1. Lambda Function Entry Point (lambda_handler)

- Determines what action to perform based on the action parameter in the event.
- Valid actions:
  - "start_ec2": Start EC2 instances.
  - "stop_ec2": Stop EC2 instances.
  - "upload_file": Upload a file from an EC2 instance to an S3 bucket.
  - "download_file": Download a file from an S3 bucket to an EC2 instance.

2. EC2 Management

- **Start EC2 Instances (start_ec2_instances)**:
  - Starts the EC2 instances specified in the instance_ids list.
  - Uses the ec2.start_instances() function.
- **Stop EC2 Instances (stop_ec2_instances)**:
  - Stops the EC2 instances specified in the instance_ids list.
  - Uses the ec2.stop_instances() function.

3. File Management

- **Upload File to S3 (upload_file_to_s3)**:
  - Uploads a file from an EC2 instance to an S3 bucket.
  - Steps:
    - Get the current working directory of the EC2 instance using the get_current_working_directory function.
    - Build the file path and upload it to S3 using the aws s3 cp command via SSM.

- **Download File from S3 (download_file_from_s3)**:
    - Downloads a file from an S3 bucket to an EC2 instance.
    - Uses the aws s3 cp command via SSM.

4. Helper Functions

- **Get Current Working Directory (get_current_working_directory)**:
    - Runs the pwd command on an EC2 instance using SSM to retrieve the working directory.
    - Keeps checking the command status until it completes or fails.
- **Run Command on EC2 (run_command_on_instance)**:
    - Runs any shell command on an EC2 instance using SSM (e.g., upload/download files).
    - Waits for the command to complete, retries if necessary, and handles success or failure.

How It Works

1. **Lambda is Triggered**:
    a. An event is passed to the Lambda function with details like action, instance IDs, S3 bucket name, file paths, etc.
2. **Action Execution**:
    a. Based on the `action`, the script calls the appropriate function.
3. **File Operations**:
    a. For upload/download, the script interacts with the EC2 instance via **AWS Systems Manager (SSM)** to run commands like `aws s3 cp`.

Example Event Payloads

Start EC2 Instances:

```
{
  "action": "start_ec2",
  "instance_ids": ["i-0abcd1234efgh5678"] ## instance id of ec2 instance
}
```

Stop EC2 Instances:

```
{
  "action": "stop_ec2",
  "instance_ids": ["i-0abcd1234efgh5678"] # instance id of ec2 instance
}
```

Upload File to S3:

```
{
  "action": "upload_file",
  "instance_id": "i-0abcd1234efgh5678", # instance id of ec2 instance
  "local_file_path": "example.txt", # path of file in vm
  "s3_bucket": "my-s3-bucket", # bucket name
  "s3_key": "example.txt" # path of s3 bucket
}
```

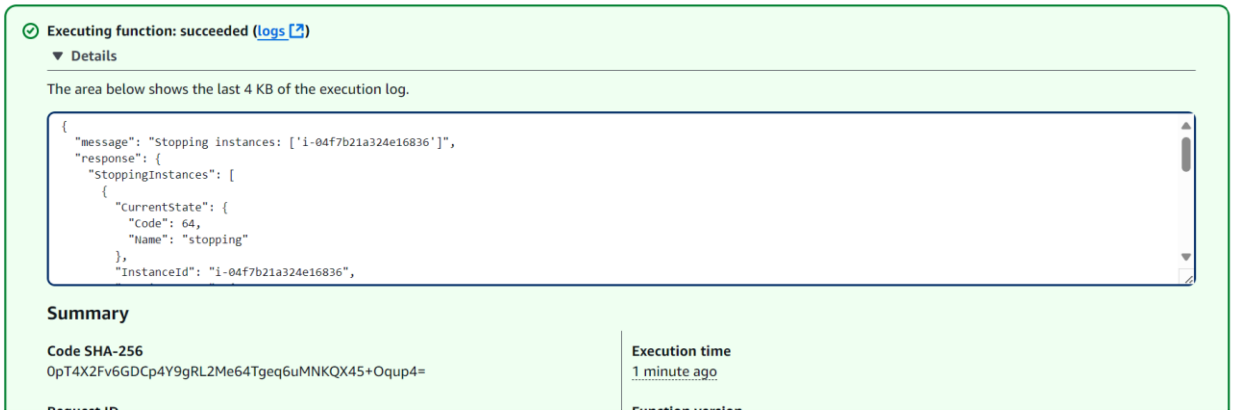
Download File from S3:

```
{
  "action": "download_file",
  "instance_id": "i-0abcd1234efgh5678", # instance id of ec2 instance
  "s3_bucket": "my-s3-bucket", #bucket name
  "s3_key": "example.txt", # path of s3 bucket
  "remote_file_path": "/home/ec2-user/example.txt" path where you want file to
be downloaded from s3 in vm
}
```

When you run any of above events and test it your script will work accordingly .

- This script Automates EC2 instance management and file handling with minimal user input.
- Utilizes AWS-managed services (like SSM) for secure and seamless operations

Below are screen shots which shows the working of script:



Executing function: succeeded (logs)

Details

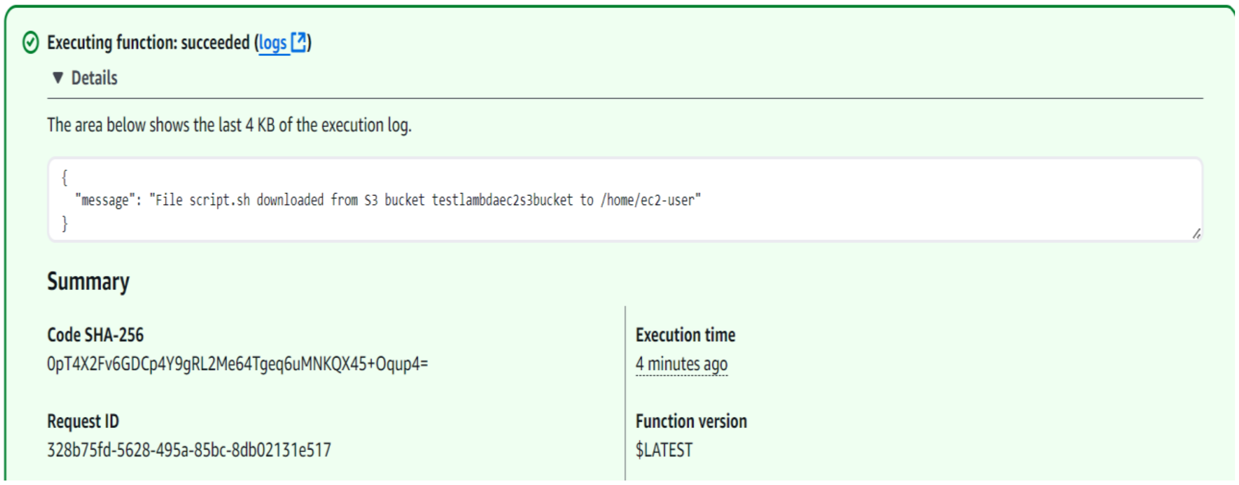
The area below shows the last 4 KB of the execution log.

```
{
  "message": "Stopping instances: ['i-04f7b21a324e16836']",
  "response": {
    "StoppingInstances": [
      {
        "CurrentState": {
          "Code": 64,
          "Name": "stopping"
        },
        "InstanceId": "i-04f7b21a324e16836",
```

Summary

Code SHA-256
0pT4X2Fv6GDCp4Y9gRL2Me64Tgeq6uMNKQX45+Oqup4=

Execution time
1 minute ago



Executing function: succeeded (logs)

Details

The area below shows the last 4 KB of the execution log.

```
{
  "message": "File script.sh downloaded from S3 bucket testlambdaec2s3bucket to /home/ec2-user"
}
```

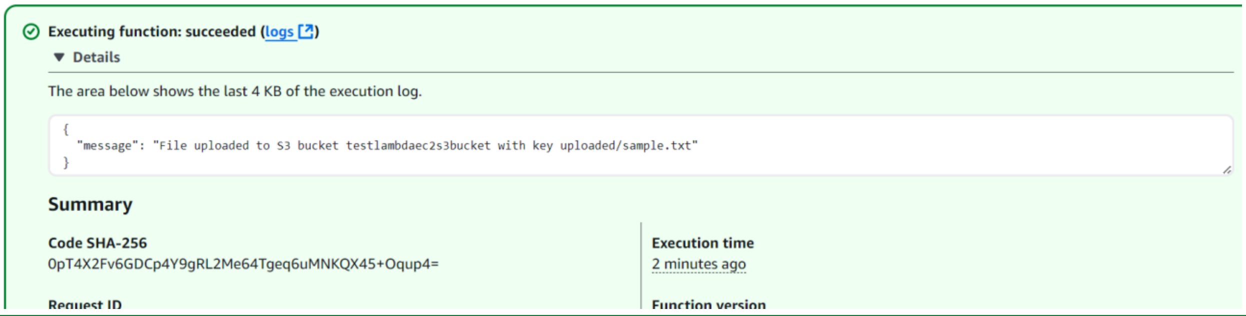
Summary

Code SHA-256
0pT4X2Fv6GDCp4Y9gRL2Me64Tgeq6uMNKQX45+Oqup4=

Execution time
4 minutes ago

Request ID
328b75fd-5628-495a-85bc-8db02131e517

Function version
$LATEST



Executing function: succeeded (logs)

Details

The area below shows the last 4 KB of the execution log.

```
{
  "message": "File uploaded to S3 bucket testlambdaec2s3bucket with key uploaded/sample.txt"
}
```

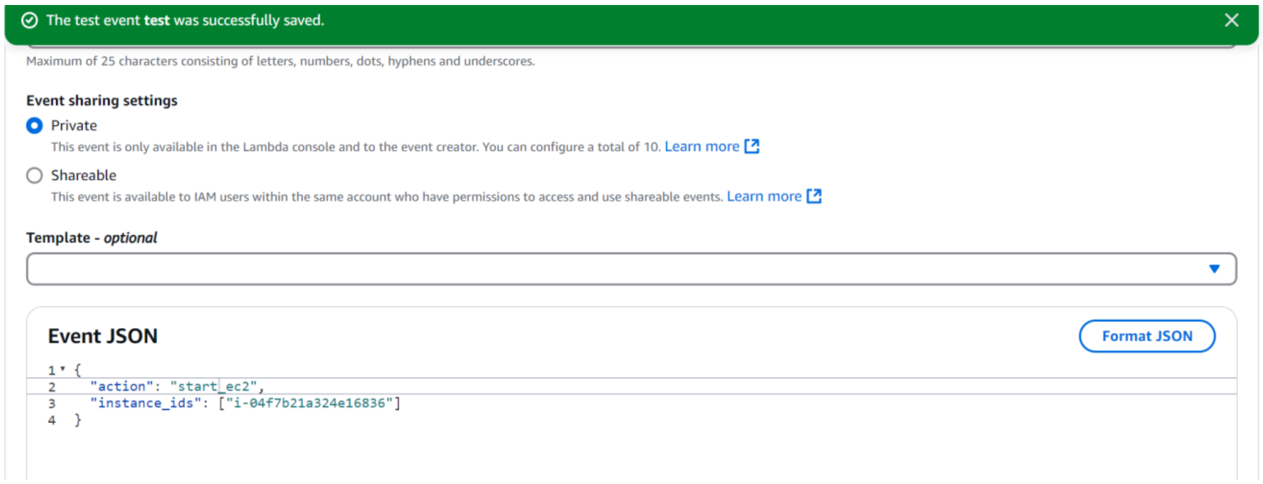
Summary

Code SHA-256
0pT4X2Fv6GDCp4Y9gRL2Me64Tgeq6uMNKQX45+Oqup4=

Execution time
2 minutes ago

Request ID

Function version



The test event **test** was successfully saved.

Maximum of 25 characters consisting of letters, numbers, dots, hyphens and underscores.

Event sharing settings

[✓ Private]
This event is only available in the Lambda console and to the event creator. You can configure a total of 10. Learn more

Shareable
This event is available to IAM users within the same account who have permissions to access and use shareable events. Learn more

Template - *optional*

Event JSON

```
{
  "action": "start_ec2",
  "instance_ids": ["i-04f7b21a324e16836"]
}
```

This is the process to write a python script and configure in AWS Lambda - Start/stop the Vms , Upload the file from server to S3 Bucket, Download the file from S3 bucket to Server.