Expert Cloud Consulting

Enhance Optimise & Scale

ASCP GPUonCLOUD Pvt Ltd

“Expert Cloud Consulting” -

SOP | Introduction to Infrastructure as Code

03.January.2025

version 1.0

—

Contributed by Tejal Kale
Approved by Akshay Shinde
Expert Cloud Consulting
Office #811, Gera Imperium Rise,
Hinjewadi Phase-II Rd, Pune, India – 411057

“Expert Cloud Consulting”
Introduction to Infrastructure as Code

1.0 Contents

2.0 General Information:

2.1 Document Purpose

This document provides an introduction to **Infrastructure as Code (IaC)**, focusing on automation and management of cloud resources using **Terraform** and **CloudFormation**. It includes hands-on assignments to provision a multi-tier architecture on AWS with Terraform and automate S3-Lambda integrations with CloudFormation. The purpose is to equip users with practical skills in IaC for scalable and repeatable cloud infrastructure deployments.

2.2 Document Revisions

Date	Version	Contributor(s)	Approver(s)	Section(s)	Change(s)
03/Jan/2025	1.0	Tejal Kale	Akshay Shinde	All Sections	New Document Created

3.0 Document Overview:

Infrastructure as Code (IaC) is a contemporary method for automating the provisioning and management of cloud infrastructure through code. Tools like Terraform and CloudFormation enable users to define and deploy cloud resources, including virtual networks, compute instances, and storage, in a consistent and efficient way. This document outlines the steps to leverage Terraform and CloudFormation for setting up a multi-tier architecture and automating S3-Lambda integrations on AWS.

4.0 Steps / Procedure for Terraform Script

4.1 : Main Terraform Configuration (main.tf):

This **`main.tf`** file provides a modular Terraform configuration to deploy AWS infrastructure, including a custom VPC, public and private subnets, EC2 instances, and an Application Load Balancer (ALB).

main.tf > module "alb"

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

data "aws_availability_zones" "available" {
  state = "available"
}

module "vpc" {
  source = "./modules/vpc"

  project_name         = var.project_name
  vpc_cidr             = var.vpc_cidr
  public_subnet_cidrs  = var.public_subnet_cidrs
  private_subnet_cidrs = var.private_subnet_cidrs
  availability_zones   = data.aws_availability_zones.available.names
}

module "ec2" {
  source = "./modules/ec2"

  project_name         = var.project_name
  vpc_id               = module.vpc.vpc_id
  private_subnet_ids   = module.vpc.private_subnet_ids
  instance_count       = var.instance_count
  instance_type        = var.instance_type
}
```

```hcl
module "alb" {
  source = "./modules/alb"

  project_name      = var.project_name
  vpc_id            = module.vpc.vpc_id
  public_subnet_ids = module.vpc.public_subnet_ids
  instance_ids      = module.ec2.instance_ids
}
```

4.2: Variables (variables.tf):

variables.tf > variable "instance_type"

```hcl
variable "aws_region" {
  description = "AWS region where resources will be deployed"
  type        = string
  default     = "us-east-1"
}

variable "project_name" {
  description = "Name of the project"
  type        = string
  default     = "multi-tier"
}

variable "vpc_cidr" {
  description = "CIDR block for VPC"
  type        = string
  default     = "10.0.0.0/16"
}

variable "public_subnet_cidrs" {
  description = "CIDR blocks for public subnets"
  type        = list(string)
  default     = ["10.0.1.0/24", "10.0.2.0/24"]
}
```

variables.tf > variable "instance_type"

```hcl
variable "private_subnet_cidrs" {
  description = "CIDR blocks for private subnets"
  type        = list(string)
  default     = ["10.0.3.0/24", "10.0.4.0/24"]
}

variable "instance_count" {
  description = "Number of EC2 instances"
  type        = number
  default     = 2
}

variable "instance_type" {
  description = "Instance type for EC2 instances"
  type        = string
  default     = "t2.micro"

}
```

4.3: Outputs.tf

output.tf > ...

```
output "vpc_id" {
  description = "ID of the VPC"
  value       = module.vpc.vpc_id
}

output "alb_dns_name" {
  description = "DNS name of the load balancer"
  value       = module.alb.alb_dns_name
}

output "instance_ids" {
  description = "IDs of EC2 instances"
  value       = module.ec2.instance_ids
}

```

4.4: terraform.tfvars

This configuration sets specific values for AWS region, project name, VPC CIDR, subnet CIDRs, EC2 instance count, and type to customize and deploy a scalable AWS infrastructure.

terraform.tfvars > ...

```
aws_region         = "us-east-1"
project_name       = "my-project"
vpc_cidr           = "10.0.0.0/16"
public_subnet_cidrs = ["10.0.1.0/24", "10.0.2.0/24"]
private_subnet_cidrs = ["10.0.3.0/24", "10.0.4.0/24"]
instance_count      = 2
instance_type       = "t2.micro"

```

4.5: Modules/Vpc

4.5.1: main.tf

This Terraform configuration provisions a VPC with public and private subnets, an internet gateway, and a NAT gateway to enable internet access.

modules > vpc > main.tf > ...

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

  tags = {
    Name = "${var.project_name}-vpc"
  }
}

resource "aws_internet_gateway" "main" {
  vpc_id = aws_vpc.main.id

  tags = {
    Name = "${var.project_name}-igw"
  }
}

resource "aws_subnet" "public" {
  count             = length(var.public_subnet_cidrs)
  vpc_id            = aws_vpc.main.id
  cidr_block        = var.public_subnet_cidrs[count.index]
  availability_zone = var.availability_zones[count.index % length(var.availability_zones)]

  map_public_ip_on_launch = true

  tags = {
    Name = "${var.project_name}-public-subnet-${count.index + 1}"
  }
```

modules > vpc > main.tf > ...

```hcl
resource "aws_subnet" "public" {
}

resource "aws_subnet" "private" {
  count             = length(var.private_subnet_cidrs)
  vpc_id            = aws_vpc.main.id
  cidr_block        = var.private_subnet_cidrs[count.index]
  availability_zone = var.availability_zones[count.index % length(var.availability_zones)]

  tags = {
    Name = "${var.project_name}-private-subnet-${count.index + 1}"
  }
}
resource "aws_eip" "nat" {
  domain = "vpc"
}

resource "aws_nat_gateway" "main" {
  allocation_id = aws_eip.nat.id
  subnet_id     = aws_subnet.public[0].id

  tags = {
    Name = "${var.project_name}-nat"
  }

  depends_on = [aws_internet_gateway.main]
}
```

modules > vpc > main.tf > resource "aws_route_table" "private"

```
57  resource "aws_route_table" "public" {
58    vpc_id = aws_vpc.main.id
59
60    route {
61      cidr_block = "0.0.0.0/0"
62      gateway_id = aws_internet_gateway.main.id
63    }
64
65    tags = {
66      Name = "${var.project_name}-public-rt"
67    }
68  }
69  resource "aws_route_table" "private" {
70    vpc_id = aws_vpc.main.id
71
72    route {
73      cidr_block     = "0.0.0.0/0"
74      nat_gateway_id = aws_nat_gateway.main.id
75    }
76
77    tags = {
78      Name = "${var.project_name}-private-rt"
79    }
80  }
81  resource "aws_route_table_association" "public" {
82    count          = length(var.public_subnet_cidrs)
83    subnet_id      = aws_subnet.public[count.index].id
84    route_table_id = aws_route_table.public.id
85  }
```

4.5.2: variables.tf

modules > vpc > variables.tf > ...

```hcl
variable "project_name" {
  description = "Name of the project"
  type        = string
}

variable "vpc_cidr" {
  description = "CIDR block for VPC"
  type        = string
}

variable "public_subnet_cidrs" {
  description = "CIDR blocks for public subnets"
  type        = list(string)
}

variable "private_subnet_cidrs" {
  description = "CIDR blocks for private subnets"
  type        = list(string)
}

variable "availability_zones" {
  description = "Availability zones"
  type        = list(string)
}
```

4.5.3: outputs.tf

modules > vpc > output.tf > ...

```hcl
output "vpc_id" {
  description = "ID of the VPC"
  value       = aws_vpc.main.id
}

output "public_subnet_ids" {
  description = "IDs of public subnets"
  value       = aws_subnet.public[*].id
}

output "private_subnet_ids" {
  description = "IDs of private subnets"
  value       = aws_subnet.private[*].id
}
```

4.6: Modules/ec2

4.6.1: main.tf

This Terraform configuration deploys Amazon Linux 2023 EC2 instances in dynamically assigned private subnets, associating a security group to control inbound and outbound traffic.

modules > ec2 > main.tf > ...

```hcl
data "aws_ami" "amazon_linux_2023" {
  most_recent = true
  owners      = ["amazon"]

  filter {
    name   = "name"
    values = ["al2023-ami-2023.*-x86_64"]
  }

  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
}

resource "aws_security_group" "ec2" {
  name        = "${var.project_name}-ec2-sg"
  description = "Security group for EC2 instances"
  vpc_id      = var.vpc_id

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

modules > ec2 > main.tf > ...

```hcl
resource "aws_security_group" "ec2" {
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags = {
    Name = "${var.project_name}-ec2-sg"
  }
}
resource "aws_instance" "app" {
  count                  = var.instance_count
  ami                    = data.aws_ami.amazon_linux_2023.id
  instance_type          = var.instance_type
  subnet_id              = var.private_subnet_ids[count.index]
  vpc_security_group_ids = [aws_security_group.ec2.id]

  user_data = <<-EOF
              #!/bin/bash
              dnf update -y
              dnf install -y httpd
              systemctl start httpd
              systemctl enable httpd
              echo "<h1>Hello from EC2 instance $(hostname -f)</h1>" > /var/www/html/index.html
              EOF
```

```
54
55      tags = {
56        Name = "${var.project_name}-app-${count.index + 1}"
57      }
58    }
```

4.6.2: variables.tf

modules > ec2 > variables.tf > ...

```
1   variable "project_name" {
2     description = "Name of the project"
3     type        = string
4   }
5
6   variable "vpc_id" {
7     description = "ID of the VPC"
8     type        = string
9   }
10
11  variable "private_subnet_ids" {
12    description = "IDs of private subnets"
13    type        = list(string)
14  }
15
16  variable "instance_count" {
17    description = "Number of EC2 instances"
18    type        = number
19  }
20
21  variable "instance_type" {
22    description = "Instance type for EC2"
23  type        = string
24  }
```

4.6.3: outputs.tf

modules > ec2 > output.tf > ...

```
1   output "instance_ids" {
2     description = "IDs of EC2 instances"
3     value       = aws_instance.app[*].id
4   }
```

4.7: Modules/alb

4.7.1: main.tf

This Terraform configuration sets up an Application Load Balancer (ALB) along with its security group, target group, listener, and dynamically attached EC2 instances. It ensures efficient distribution of incoming HTTP traffic across instances in public subnets, incorporating health checks to maintain reliability and support scalability.

modules > alb > main.tf > resource "aws_lb" "main" > tags

```hcl
resource "aws_security_group" "alb" {
  name        = "${var.project_name}-alb-sg"
  description = "Security group for ALB"
  vpc_id      = var.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"]
  }

  tags = {
    Name = "${var.project_name}-alb-sg"
  }
}
resource "aws_lb" "main" {
  name               = "${var.project_name}-alb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.alb.id]
  subnets            = var.public_subnet_ids
```

modules > alb > main.tf > resource "aws_lb" "main" > tags

```hcl
resource "aws_lb" "main" {
  tags = {
    Name = "${var.project_name}-alb"
  }
}

resource "aws_lb_target_group" "main" {
  name     = "${var.project_name}-tg"
  port     = 80
  protocol = "HTTP"
  vpc_id   = var.vpc_id

  health_check {
    enabled             = true
    healthy_threshold   = 2
    interval            = 30
    matcher             = "200"
    path                = "/"
    port                = "traffic-port"
    timeout             = 5
    unhealthy_threshold = 2
  }
}

resource "aws_lb_listener" "main" {
  load_balancer_arn = aws_lb.main.arn
  port              = 80
  protocol          = "HTTP"
```

```
resource "aws_lb_listener" "main" {
  load_balancer_arn = aws_lb.main.arn
  port              = 80
  protocol          = "HTTP"

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

resource "aws_lb_target_group_attachment" "main" {
  count            = length(var.instance_ids)
  target_group_arn = aws_lb_target_group.main.arn
  target_id        = var.instance_ids[count.index]
  port             = 80
}
```

4.7.2: variables.tf

modules > alb > variables.tf > ...

```
variable "project_name" {
  description = "Name of the project"
  type        = string
}

variable "vpc_id" {
  description = "ID of the VPC"
  type        = string
}

variable "public_subnet_ids" {
  description = "IDs of public subnets"
  type        = list(string)
}

variable "instance_ids" {
  description = "IDs of EC2 instances"
  type        = list(string)
}
```

4.7.3: outputs.tf

modules > alb > output.tf > ...

```
output "alb_dns_name" {
  description = "DNS name of the load balancer"
  value       = aws_lb.main.dns_name
}

output "target_group_arn" {
  description = "ARN of target group"
  value       = aws_lb_target_group.main.arn
}
```

4.8: Run Commands

4.8.1: Initialize:

terraform init

4.8.2: Plan:

terraform plan

4.8.3: Apply:

terraform apply

4.9: Outputs:

Figure 1: EC2 Instance 1 Created in Private Subnet 1

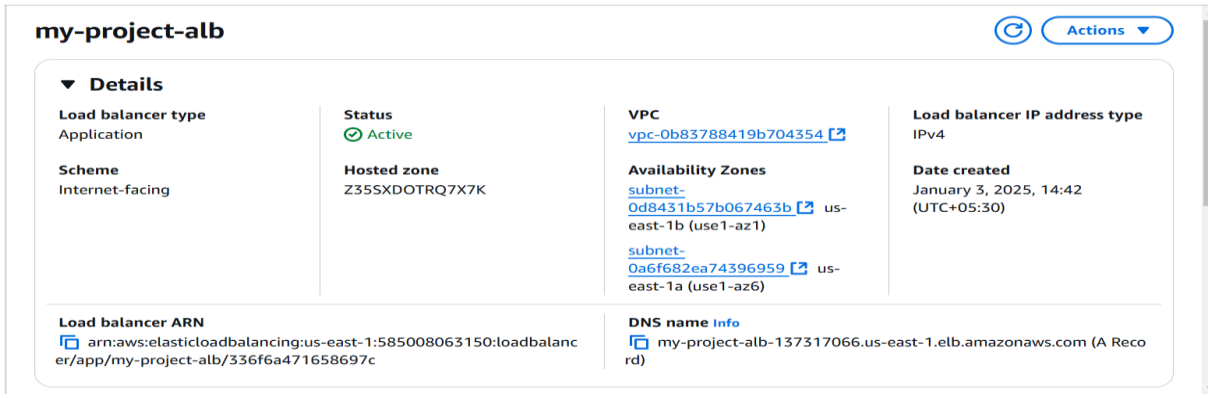
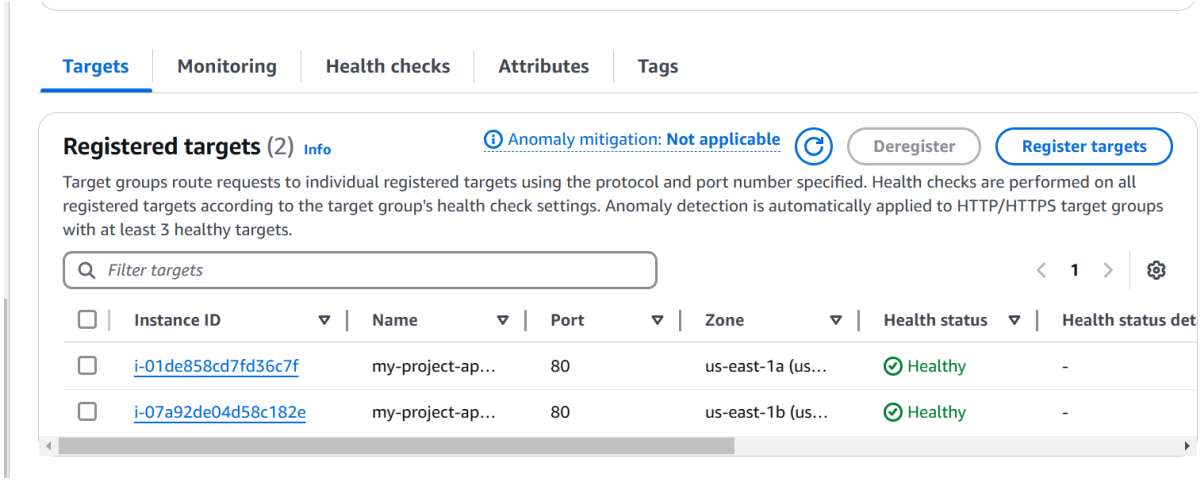


Figure 2: ALB Configured Across different Availability Zones



5.0 **CloudFormation Implementation - S3-Lambda Integration**

5.1: Cloudformation Template:

This CloudFormation template creates an S3 bucket with versioning enabled and a Lambda function configured to trigger on S3 object creation events. An IAM role with the necessary permissions is provisioned to enable Lambda execution. The Lambda function logs details of newly added objects to CloudWatch Logs. Outputs include the S3 bucket name and the Lambda function ARN.

Implementation Steps:

Welcome ! bucket.yml 2 ×

! bucket.yml > CloudFormation Linter > {} Parameters > {} BucketName

Visualize with Infrastructure Composer

```yaml
AWSTemplateFormatVersion: '2010-09-09'
Description: 'Template to create S3 bucket with versioning and Lambda email notifications'

Parameters:
  EmailAddress:
    Type: String
    Default: tejalbkale@gmail.com
    Description: Email address to receive notifications
  BucketName:
    Type: String
    Description: Name of the S3 bucket to create
```

Welcome ! bucket.yml 2 ×

! bucket.yml > CloudFormation Linter > {} Resources > {} LambdaExecutionRole > abc Type

```yaml
Resources:
  # SNS Topic (Independent resource)
  NotificationTopic:
    Type: 'AWS::SNS::Topic'
    Properties:
      TopicName:
        Fn::Sub: ${AWS::StackName}-notifications
      Subscription:
        - Protocol: email
          Endpoint:
            Ref: EmailAddress

  # Lambda Execution Role (Depends on SNS Topic)
  LambdaExecutionRole:
    Type: 'AWS::IAM::Role'
    DependsOn: NotificationTopic
    Properties:
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: lambda.amazonaws.com
```

```
36                  Action: 'sts:AssumeRole'
37            ManagedPolicyArns:
38              - 'arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole'
39            Policies:
40              - PolicyName: S3AndSNSAccess
41                PolicyDocument:
42                  Version: '2012-10-17'
43                  Statement:
44                    - Effect: Allow
45                      Action:
46                        - 's3:GetObject'
47                      Resource: '*'
48                    - Effect: Allow
49                      Action:
50                        - 'sns:Publish'
51                      Resource:
52                        Ref: NotificationTopic
53
```

Welcome | ! bucket.yml 2 ×

! bucket.yml > CloudFormation Linter > {} Resources > {} LambdaExecutionRole > abc Type

```
13  Resources:
54    # Lambda Function (Depends on Role)
55    ProcessingFunction:
56      Type: 'AWS::Lambda::Function'
57      DependsOn: LambdaExecutionRole
58      Properties:
59        Handler: 'index.handler'
60        Role:
61          Fn::GetAtt:
62            - LambdaExecutionRole
63            - Arn
64        Code:
65          ZipFile: |
66            const AWS = require('aws-sdk');
67            const sns = new AWS.SNS();
68
69            exports.handler = async (event, context) => {
70                try {
71                    const bucket = event.Records[0].s3.bucket.name;
72                    const key = event.Records[0].s3.object.key;
73                    const size = event.Records[0].s3.object.size;
74
75                    const message = `
```

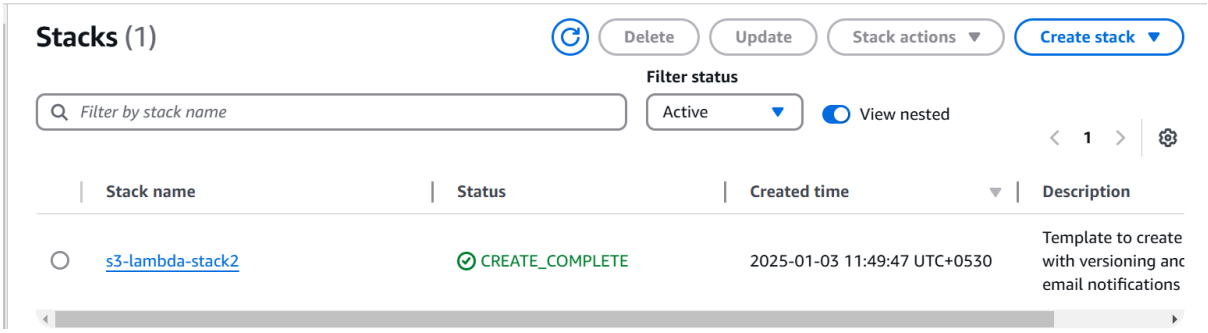
```
                    New file uploaded to S3:
                    Bucket: ${bucket}
                    File: ${key}
                    Size: ${size} bytes
                `;

                await sns.publish({
                    TopicArn: process.env.SNS_TOPIC_ARN,
                    Subject: 'New S3 Upload Notification',
                    Message: message
                }).promise();

                return {
                    statusCode: 200,
                    body: 'Notification sent successfully'
                };
            } catch (error) {
                console.error('Error:', error);
                throw error;
            }
        }
      Runtime: 'nodejs18.x'
      Timeout: 30
      MemorySize: 128
      Environment:
        Variables:
          SNS_TOPIC_ARN:
            Ref: NotificationTopic

  # Bucket Permission (Depends on Lambda)
  BucketPermission:
    Type: 'AWS::Lambda::Permission'
    DependsOn: ProcessingFunction
    Properties:
      Action: 'lambda:InvokeFunction'
      FunctionName:
```

```
      Ref: ProcessingFunction
    Principal: 's3.amazonaws.com'
    SourceArn:
      Fn::Sub: arn:aws:s3:::${BucketName}

# S3 Bucket
S3Bucket:
  Type: 'AWS::S3::Bucket'
  DependsOn: BucketPermission
  Properties:
    VersioningConfiguration:
      Status: Enabled
    BucketName:
      Ref: BucketName
    NotificationConfiguration:
      LambdaConfigurations:
        - Event: 's3:ObjectCreated:*'
          Function:
            Fn::GetAtt:
              - ProcessingFunction
                - Arn

Outputs:
  BucketName:
    Description: 'Name of the created S3 bucket'
    Value:
      Ref: S3Bucket
  SNSTopicARN:
    Description: 'ARN of the SNS Topic'
    Value:
      Ref: NotificationTopic
```

5.2 Create the CloudFormation stack:

```
PS C:\Users\tejal\cloudformation> aws cloudformation create-stack  --stack-name s3-lambda-stack2 --t
emplate-body file://bucket.yml --parameters ParameterKey=BucketName,ParameterValue=s3-object-yxxl  P
arameterKey=EmailAddress,ParameterValue=tejalbkale@gmail.com  --capabilities CAPABILITY_IAM
{
    "StackId": "arn:aws:cloudformation:us-east-1:585008063150:stack/s3-lambda-stack2/9bd11b50-c994-1
1ef-9d7c-12f34c2b2f1d"
}

PS C:\Users\tejal\cloudformation>
```

5.3 Verify the deployment:



5.4 S3 events trigger Lambda execution: