IADT PROJECT REPORT

GROUP 1

Single click Geo-location based Latest News Display

Group Members

Vishal Gupta

Vikrant Vaze

Miloni Kapadia

Rohit Kapadi

Omer Yousafzai

Introduction

In this project we have created a website that tracks the location of clients accessing the website by capturing and analyzing the IP address. Based on the location of the client the website will display the latest news and activities in the region. The whole project is designed and implemented on Amazon Web Services Cloud Infrastructure.

The main reason for using AWS for deploying our website was the high availability and vast varieties of other resources. The whole project is deployed up and running in a matter of minutes which would take at least 45 days just to procure a server.

The tools like terraform and ansible adds another level of automation which have made our lives a lot easier. Just by using a single line ad hoc command inside our local machine we can do all the steps starting from provisioning the resources till running the website and making it available to the public.

Tools Used

We are using three most common and powerful tools in our project. All these tools are playing their role starting from resource provisioning to Infrastructure management and last but not the least programming version control and code deployment.

Terraform: Terraform is a powerful provisioning tool which has built-in modules designed specifically for AWS infrastructure. We are using it in our project to provision the services and enable the communication between the services.

Ansible: Ansible is another powerful tool used for provisioning and configuration management. In our project we are using Ansible to install a webserver and push the REST API configuration files from our Master node to all the public and Private EC2 instances. We have integrated the Ansible playbook inside our terraform script which on the run time creates dynamic inventory which is used by ansible to play its part.

GitHub: GitHub is a service that is used for programming and version control. It keeps track of all the changes made and helps users change the code simultaneously and keep different versions for debugging. In our project we use GitHub to develop our code and then securely pull it on EC2 instances.

Services used from Amazon Web Services

Virtual Private Cloud: It is a service provided by AWS for its users to isolate their deployed infrastructure from the rest of the world. It provides security in the form of security groups, Public and Private subnets etc.

Internet Gateway: This is a fundamental part of the whole infrastructure that provides access from outside of the world to the publicly available infrastructure deployed

Public and Private Subnets: Subnetting in a network architecture adds a level of security and makes the network scalable and manageable. In our infrastructure we are using two types of subnets.

> **Public Subnet:** This subnet includes two EC2 instances as shown in Network Diagram. These instances in subnet can be publicly accessed as per defined rules.
>
> **Private Subnet:** This subnet includes two API instances which are not publicly accessible. They can only be accessed internally from public subnets.

Public and Private Route Tables: Routing tables are used in Network to direct traffic from a source to a destination as per defined rules. We have defined two Route tables, one for routing traffic from outside the VPC cloud and another for internal routing among the infrastructure.

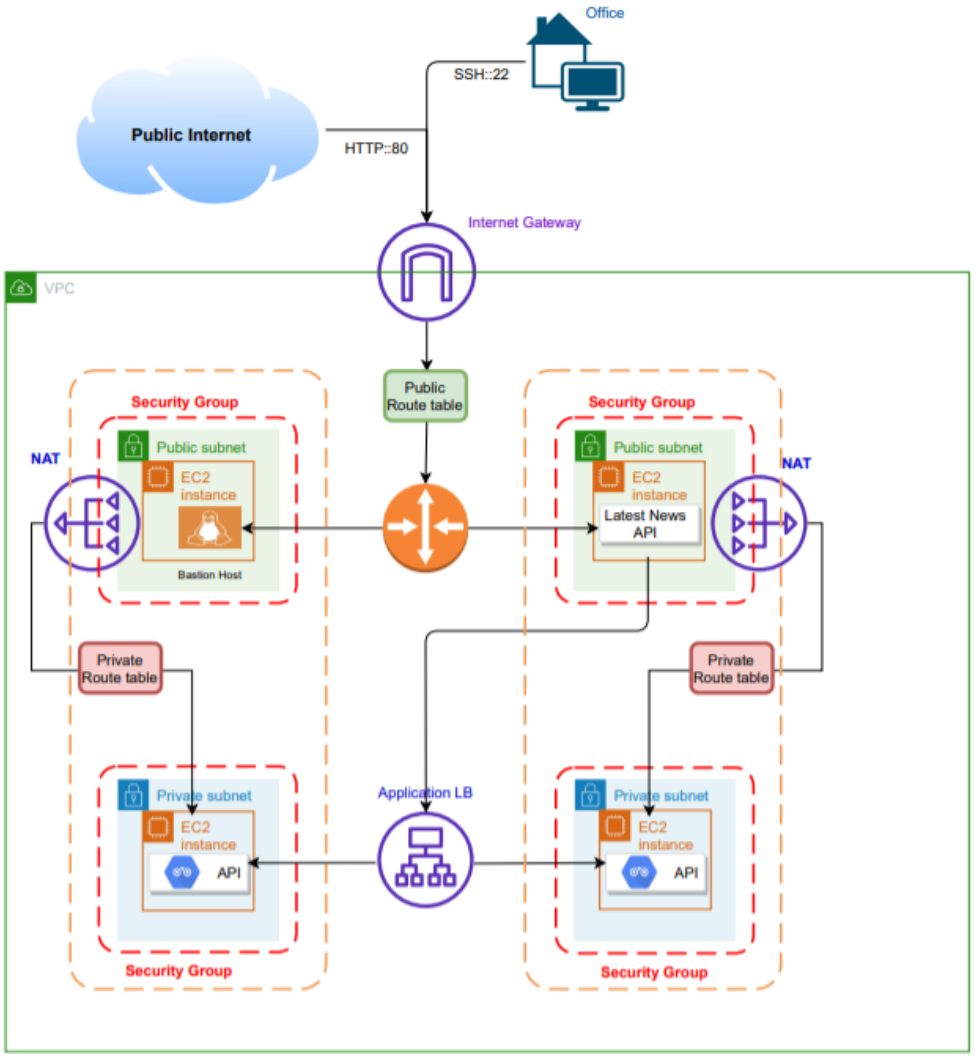
EC2 instances: Through AWS EC2, users get Elastic computing resources along with memory, which can be provisioned and destroyed in a matter of seconds. We have provisioned four EC2 instances. One as a bastion host and three as web servers. We have installed Apache web server, Java, and Maven on these instances to run our website. All the website handling and computing is done through these instances.

Application Load Balancers: Amazon Elastic Load Balancer serves as a single point of contact. All the requests are directed to the load balancer which then forwards those requests to different target groups as per defined rules and conditions.

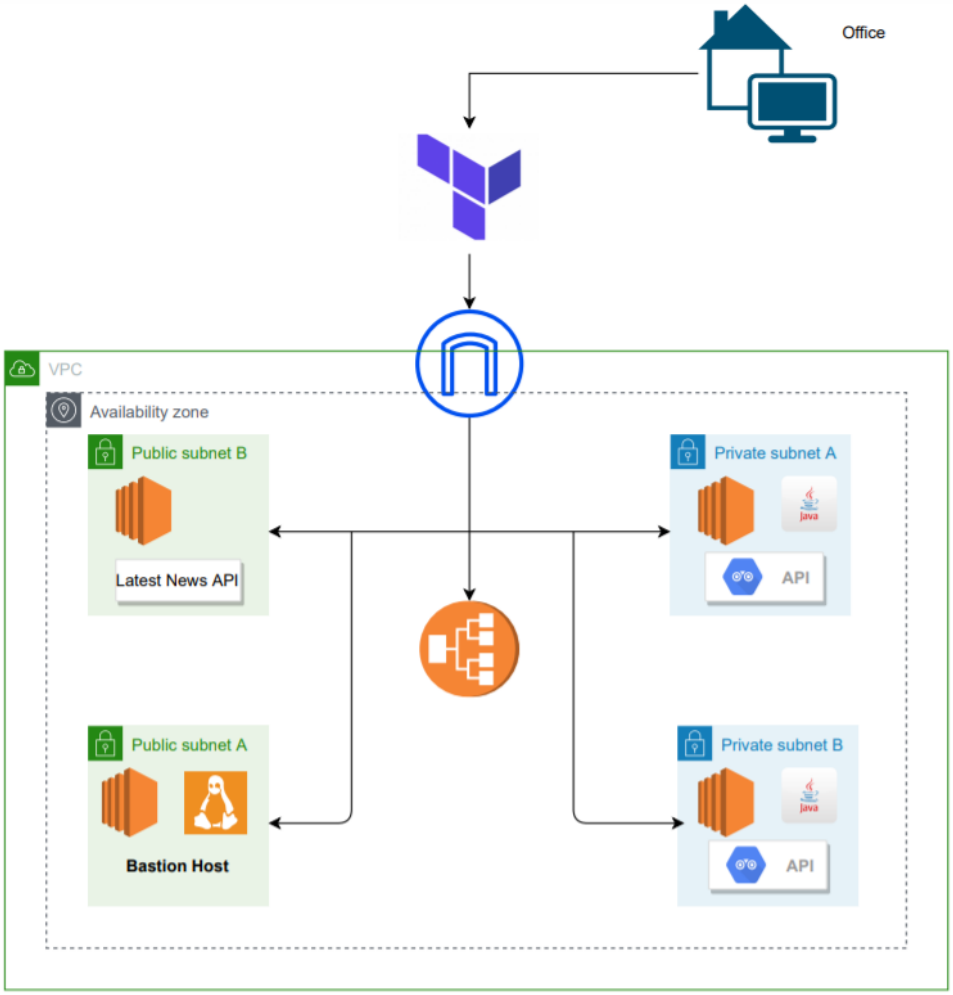
NAT Gateways: We have used two elastic IP addresses for our two instances that are located inside the private subnet at the backend. The NAT gateways translate the IP addresses back and forth based on the requests made.

Bastion Host: Bastion host is used as a jump host. It is the only way we can connect to the private EC2 instances. The ansible playbook tasks for the private instances are done through Bastion host. Bastion host adds an extra layer of security to our infrastructure.

Network Diagram



Terraform Automation Flow diagram



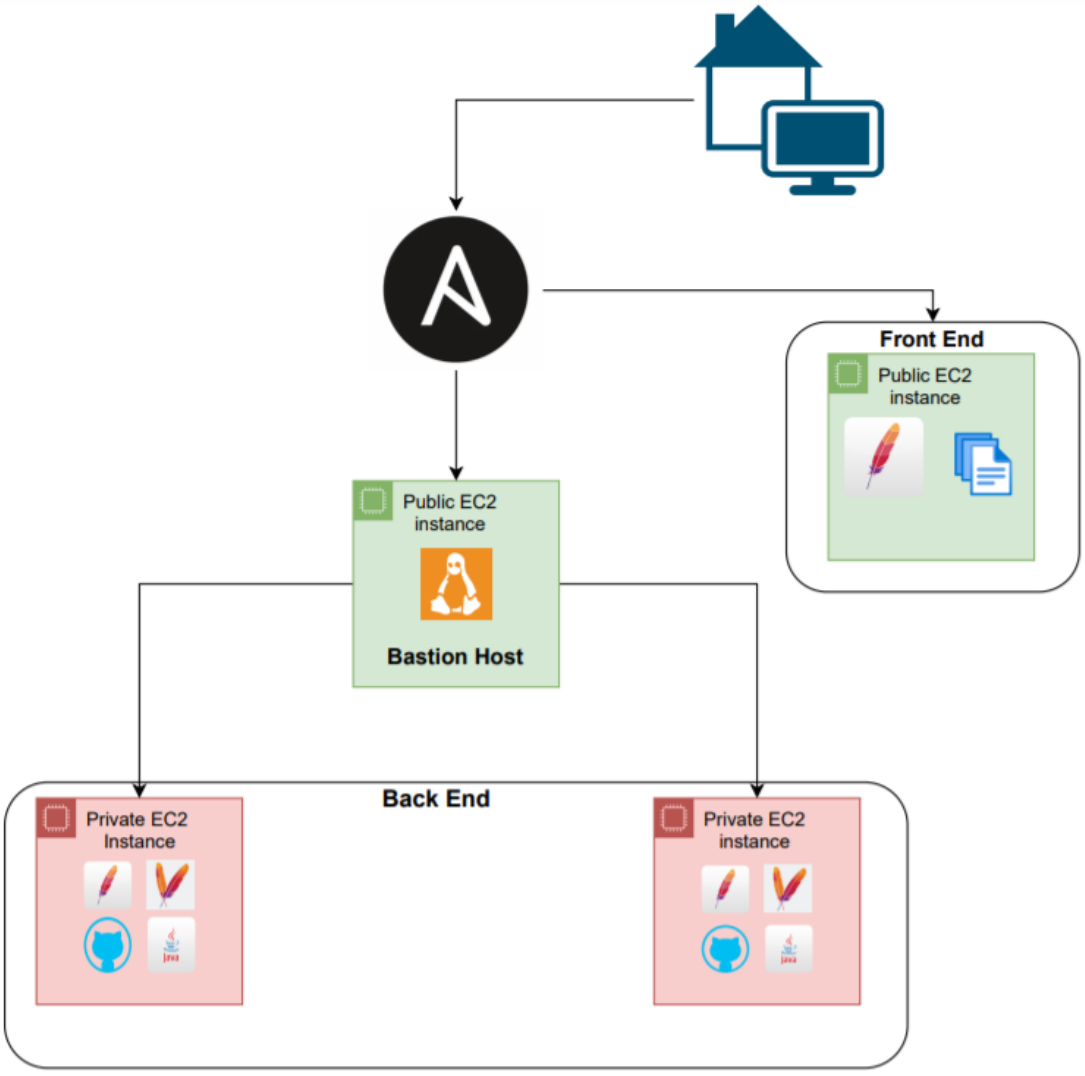
Resource Provisioning

Terraform is a resource provisioning tool that integrates well with cloud services like AWS, Google cloud and Microsoft Azure. We have created a terraform script on our local machine through which we are provisioning resources. We have also integrated Ansible through the Terraform provisioner inside our script.

We are provisioning Vpc, Internet gateway, Availability zone, Subnets, EC2 instances, Route tables and Application Load balancer through terraform. Within a matter of minutes we were able to deploy our infrastructure.

Terraform is a highly modular scripted tool that brings Automation at a very micro-level inside an architecture which is why it is highly utilized and accepted everywhere. We have also created a terraform variable file which stores all the variables that we want to use inside our script. The variables are called inside the main script.

Ansible managed Network Diagram



Resource Management

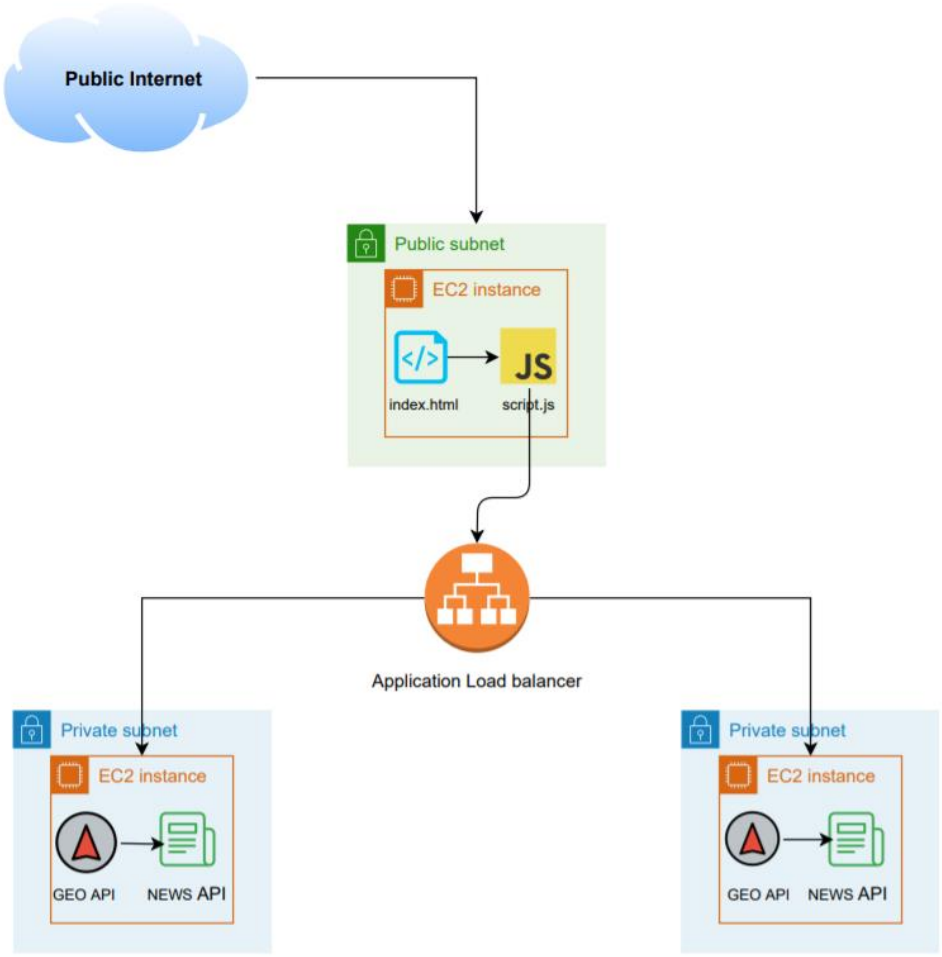
Since the resources have been successfully provisioned now it is time to make use of it. We have used Ansible which is a powerful and highly integratable tool used for resource management. It installs all the relevant tools and softwares on the remote hosts. Ansible has a builtin module for Git which we have used to clone the repository that is available on GitHub.

We have created two playbooks one for front end instances and another for backend instances. Both the playbooks are integrated inside the terraform script and hence are taken care of by terraform.

The Bastion host is added for the purpose to act as a jump host to execute the playbook on backend devices which cannot be accessed otherwise as they are inside a private subnet. In our architecture bastion host can only be accessed from a single IP address that is affiliated to our office.

Ansible playbooks can be utilized in the future to add more layers to the website. It can also be used to create and access databases and other related stuff. Ansible has a builtin module for Amazon Web Services which can be used to provision and manage infrastructure deployed on its cloud.

User request Flow diagram



User Request

Here as soon as a user hits the request on the website, it will be directed to the html page of the public instance. That html page will make a call to JavaScript which transfers the request to application load balancer. Now the application load balancer will distribute the load and take the response from the instance from which it got a health check first. And then the response will be displayed on the website page. In our architecture, application load balancer is facing towards private instances lying under private subnets. These instances are already having spring boot framework to make a GEO api call first and then NEWS api call. So, firstly it will extract the user's country details on the basis of GEO api and then by using that information it will extract the latest news for that country and display that to the webpage. Here, we have used public instances for the html page and javascript which transfer the request to ALB, while all the information which has been extracted as per apis, we have put those scripts under private instances to achieve the concept of security.

Project Budget Estimation

(Not considering free trial)

Sr. No.	Service Name	Service used	Cost/Month	Used	Total(USD)
1.	Instance	T2. micro	9.5	4	32
2.	Load balancer	Application	22.45	1	22.45
3.	VPC	Nat gateway	32.89	2	65.78
4.	Elastic IP	Elastic Ip	3.65	2	7.3
5.	Bastion host	Bastion Host	8.76	1	8.76
				Grand Total	142.29 USD

IAM Roles

An IAM role is a specific permissioned IAM identity that you can create in your account. An IAM role, like an IAM user, is an AWS identity with permission policies that govern what the identity can and cannot do in AWS.
The following are examples of roles which can be assigned:

- In the same AWS account as the role, an IAM user
- An AWS account with a different IAM user than the role
- Amazon Elastic Compute Cloud, for example, is a web service provided by AWS (Amazon EC2)
- An external user who has been authenticated by a custom-built identity broker or an external identity provider (IdP) service that is compatible with SAML 2.0 or OpenID Connect.

IAM Roles used in the project:
Admin Access: Full admin access over all of the services used in the amazon web service. So that the admin can edit, remove and add any services into the infrastructure.
Policy Name: AdministratorAccess

EC2 instance Access: Provides full access to EC2 instances including EC2 auto scaling and Load balancers in the infrastructure.

Policy Name: AmazonEC2FullAccess.

Future Scope

We can have following concepts in the near future in the terms of making website and architecture more feasible and more efficient:

- We can have a specific domain name for our website with route53 which can be implemented by attaching elastic IP or adding an ALB or also by using a service which provides DDNS
- We can have requestor counts using specific API in our spring boot implementation to have the records of user counts
- We can implement the concept of auto scaling by adding rules in the terms of memory utilization and load distribution on EC2 instances
- Rather than using javascript for the API implementation we can use lambda function service or cloudformation service where we can host the website and have the actual implementation with the service

Terraform Code:

- finalTerraformScript.tf

```
# Declaring the Cloud Service To be used and declaring a default region
provider "aws" {
  region = var.region
}

# Creation of Virtual Private Cloud
resource "aws_vpc" "group1_vpc" {
  cidr_block           = var.vpc_cidr   # 10.10.0.0/20
  enable_dns_hostnames = true

  tags = {
    Name = "group1_vpc-vpc"
  }
}

# Creating an Internet Gateway to provide internet connectivity to the VPC
resource "aws_internet_gateway" "group1_igw" {
  vpc_id = aws_vpc.group1_vpc.id

  tags = {
    Name = "group1_igw"
  }
}

# Creating 2 public and 2 Private Subnets
resource "aws_subnet" "public_a" {
  vpc_id                  = aws_vpc.group1_vpc.id
  cidr_block              = cidrsubnet(var.vpc_cidr, var.subnet_cidr_newbits, 0) # 10.10.0.0/20 would become 10.10.0.0/24
  availability_zone       = data.aws_availability_zones.available.names[0]
  map_public_ip_on_launch = true

  tags = {
    Name = "PUB_SUB_a"
  }
}
```

```
# Creating 2 public and 2 Private Subnets
resource "aws_subnet" "public_a" {
  vpc_id                  = aws_vpc.group1_vpc.id
  cidr_block              = cidrsubnet(var.vpc_cidr, var.subnet_cidr_newbits, 0) # 10.10.0.0/20 would become 10.10.0.0/24
  availability_zone       = data.aws_availability_zones.available.names[0]
  map_public_ip_on_launch = true

  tags = {
    Name = "PUB_SUB_a"
  }
}

resource "aws_subnet" "public_b" {
  vpc_id                  = aws_vpc.group1_vpc.id
  cidr_block              = cidrsubnet(var.vpc_cidr, var.subnet_cidr_newbits, 1)  # 10.10.0.0/20 would become 10.10.1.0/24
  availability_zone       = data.aws_availability_zones.available.names[1]
  map_public_ip_on_launch = true

  tags = {
    Name = "PUB_SUB_b"
  }
}

resource "aws_subnet" "private_a" {
  vpc_id                  = aws_vpc.group1_vpc.id
  cidr_block              = cidrsubnet(var.vpc_cidr, var.subnet_cidr_newbits, 2) # 10.10.0.0/20 would become 10.10.2.0/24
  availability_zone       = data.aws_availability_zones.available.names[0]
  map_public_ip_on_launch = true

  tags = {
    Name = "PRI_SUB_-a"
  }
}

resource "aws_subnet" "private_b" {
  vpc_id                  = aws_vpc.group1_vpc.id
  cidr_block              = cidrsubnet(var.vpc_cidr, var.subnet_cidr_newbits, 3) # 10.10.0.0/20 would become 10.10.3.0/24
  availability_zone       = data.aws_availability_zones.available.names[1]
  map_public_ip_on_launch = true

  tags = {
    Name = "PRI SUB -b"
```

```
# Creating an Elastic IPs for NAT Gateways
resource "aws_eip" "NGW_A" {
  vpc = true
}

resource "aws_eip" "NGW_B" {
  vpc = true
}

# Creating the NAT Gateways
resource "aws_nat_gateway" "ngw_a" {
  allocation_id = aws_eip.NGW_A.id
  subnet_id     = aws_subnet.public_a.id

  tags = {
    Name = "nat-gw-a"
  }
}

resource "aws_nat_gateway" "ngw_b" {
  allocation_id = aws_eip.NGW_B.id
  subnet_id     = aws_subnet.public_b.id

  tags = {
    Name = "nat-gw-b"
  }
}



# Creation of Route Tables
resource "aws_route_table" "public" {
  vpc_id = aws_vpc.group1_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.group1_igw.id
  }

  tags = {
    Name = "public-rt"
  }
}

resource "aws_route_table" "private_a" {
  vpc_id = aws_vpc.group1_vpc.id

  route {
    cidr_block     = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.ngw_a.id
  }

  tags = {
    Name = "private-rt-a"
  }
}

resource "aws_route_table" "private_b" {
  vpc_id = aws_vpc.group1_vpc.id

  route {
    cidr_block     = "0.0.0.0/0"
    nat_gateway_id = aws_nat_gateway.ngw_b.id
  }

  tags = {
    Name = "private-rt-b"
  }
}
```

```
# Route Table Subnet Assoiation
resource "aws_route_table_association" "public_a" {
  subnet_id      = aws_subnet.public_a.id
  route_table_id = aws_route_table.public.id
}

resource "aws_route_table_association" "public_b" {
  subnet_id      = aws_subnet.public_b.id
  route_table_id = aws_route_table.public.id
}

resource "aws_route_table_association" "private_a" {
  subnet_id      = aws_subnet.private_a.id
  route_table_id = aws_route_table.private_a.id
}

resource "aws_route_table_association" "private_b" {
  subnet_id      = aws_subnet.private_b.id
  route_table_id = aws_route_table.private_b.id
}

# Creating security_groups
resource "aws_security_group" "public" {
  name        = "public sg"
  description = "SG for Public Instances"
  vpc_id      = aws_vpc.group1_vpc.id
```

```
# Creating security_groups
resource "aws_security_group" "public" {
  name        = "public sg"
  description = "SG for Public Instances"
  vpc_id      = aws_vpc.group1_vpc.id


  ingress {
    description = "For SSH traffic"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["${var.ssh_location}/32"]
  }

  ingress {
    description = "For HTTP Traffic"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["${var.ssh_location}/32"]
  }

  ingress {
    description = "For Apache"
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["${var.ssh_location}/32"]
  }

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

  tags = {
    Name = "public-sg"
  }
}
```

```
resource "aws_security_group" "private" {
  name        = "private sg"
  description = "SG for Private instances"
  vpc_id      = aws_vpc.group1_vpc.id

  ingress {
    description = "Allow SSH"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = [aws_vpc.group1_vpc.cidr_block]
  }

  ingress {
    description = "For NewsAPI"
    from_port   = 8090
    to_port     = 8090
    protocol    = "tcp"
    cidr_blocks = [aws_vpc.group1_vpc.cidr_block]
  }

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

  tags = {
    Name = "private-sg"
  }
}



resource "aws_security_group" "alb" {
  name        = "alb sg"
  description = "SG For ALB"
  vpc_id      = aws_vpc.group1_vpc.id

  ingress {
    description = "For Apache"
    from_port   = 8090
    to_port     = 8090
    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 = "alb-sg"
  }
}
```

```
# Specification for EC2 Instances
resource "aws_instance" "bastion" {
  ami                    = var.ec2_image_ids[var.region]
  instance_type          = var.ec2_instance_type
  key_name               = var.private_key_name
  subnet_id              = aws_subnet.public_a.id
  vpc_security_group_ids = [aws_security_group.public.id]

  tags = {
    Name = "Bastion-Host"
  }
}

resource "aws_instance" "latest_news_api_a" {
  ami                         = var.ec2_image_ids[var.region]
  instance_type               = var.ec2_instance_type
  key_name                    = var.private_key_name
  subnet_id                   = aws_subnet.private_a.id
  vpc_security_group_ids      = [aws_security_group.private.id]
  associate_public_ip_address = false

  tags = {
    Name = "News-API-AZ-A"
  }

  depends_on = [aws_instance.bastion]
```

```
  # Initialize Server
  provisioner "remote-exec" {
    inline = ["echo 'Server Initialization ...'"]

    connection {
      type        = "ssh"
      agent       = false
      host        = self.private_ip
      user        = "ec2-user"
      private_key = file(var.private_key_file_path)

      bastion_host        = aws_instance.bastion.public_ip
      bastion_private_key = file(var.private_key_file_path)
    }
  }

  provisioner "local-exec" {
    command = ansible-playbook -i '${self.private_ip},' --ssh-common-args ' -o ProxyCommand="ssh -A  -q -W %h:%p ec2-user@${aws_instance.bastion.

  }
}

resource "aws_instance" "latest_news_api_b" {
  ami                         = var.ec2_image_ids[var.region]
  instance_type               = var.ec2_instance_type
  key_name                    = var.private_key_name
  subnet_id                   = aws_subnet.private_b.id
  vpc_security_group_ids      = [aws_security_group.private.id]
  associate_public_ip_address = false

  tags = {
    Name = "News-API-AZ-B"
  }

  depends_on = [aws_instance.bastion]

  # Initialize Server
  provisioner "remote-exec" {
    inline = ["echo 'Server Initialization ...'"]

    connection {
```

```
  # Initialize Server
  provisioner "remote-exec" {
    inline = ["echo 'Server Initialization ...'"]

    connection {
      type        = "ssh"
      agent       = false
      host        = self.private_ip
      user        = "ec2-user"
      private_key = file(var.private_key_file_path)

      bastion_host        = aws_instance.bastion.public_ip
      bastion_private_key = file(var.private_key_file_path)
    }
  }

  provisioner "local-exec" {
    command = ansible-playbook -i '${self.private_ip},' --ssh-common-args ' -o ProxyCommand="ssh -A -q -W %h:%p  ec2-user@${aws_instance.bastion.

  }
}

resource "aws_instance" "latest_news_website" {
  ami                    = var.ec2_image_ids[var.region]
  instance_type          = var.ec2_instance_type
  key_name               = var.private_key_name
  subnet_id              = aws_subnet.public_b.id
  vpc_security_group_ids = [aws_security_group.public.id]

  # To Ensure Dependency on ALB
  depends_on = [aws_lb.latest_news_api]

  tags = {
    Name = "News-Website"
  }

  # Initializing Server
  provisioner "remote-exec" {
    inline = ["echo 'Server Initialization ...'"]
```

```
  # Initializing Server
  provisioner "remote-exec" {
    inline = ["echo 'Server Initialization ...'"]

    connection {
      type        = "ssh"
      agent       = false
      host        = self.public_ip
      user        = "ec2-user"
      private_key = file(var.private_key_file_path)
    }
  }

  provisioner "local-exec" {
    command = ansible-playbook -i '${self.public_ip},' -u ec2-user --private-key ${var.private_key_file_path} --extra-vars "host=${aws_lb.latest_news_api.dns_name}"
  }
}

# Creating ALB
resource "aws_lb" "latest_news_api" {
  name               = "latest-news-api-lb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.alb.id]
  subnets            = [aws_subnet.public_a.id, aws_subnet.public_b.id]

  tags = {
    Name = "news-api-alb"
  }
}

# Creation of Target Group
resource "aws_lb_target_group" "latest_news_api" {
```

```
# Creation of Target Group
resource "aws_lb_target_group" "latest_news_api" {
  name     = "latest-news-api-lb-tg"
  port     = 8090
  protocol = "HTTP"
  vpc_id   = aws_vpc.group1_vpc.id

  health_check {
    interval            = 10
    path                = "/actuator/health"
    port                = 8090
    protocol            = "HTTP"
    timeout             = 5
    healthy_threshold   = 3
    unhealthy_threshold = 2
  }

  target_type = "instance"


  tags = {
    Name = "news-api-target-group"
  }
}

# adding listener to ALB
resource "aws_lb_listener" "latest_news_api" {
  load_balancer_arn = aws_lb.latest_news_api.arn
  port              = "8090"
  protocol          = "HTTP"

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

```
# adding listener to ALB
resource "aws_lb_listener" "latest_news_api" {
  load_balancer_arn = aws_lb.latest_news_api.arn
  port              = "8090"
  protocol          = "HTTP"

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

# ALB-TG Association
resource "aws_lb_target_group_attachment" "target_a" {
  target_group_arn = aws_lb_target_group.latest_news_api.arn
  target_id        = aws_instance.latest_news_api_a.id
  port             = 8090
}

resource "aws_lb_target_group_attachment" "target_b" {
  target_group_arn = aws_lb_target_group.latest_news_api.arn
  target_id        = aws_instance.latest_news_api_b.id
  port             = 8090
}
```

- Output.tf

```
output "news_website_address" {
  value       = "http://${aws_instance.latest_news_website.public_dns}"
  description = "Website URL"
}
```

- Variable.tf

```
# Specifyying Region for Infrastructue setup
variable "region" {
  type        = string
  default     = "us-east-1"
  description = "Default region"
}

# List of Availability Zones
data "aws_availability_zones" "available" {
  state = "available"
}

# Defining Cidr block for VPC
variable "vpc_cidr" {
  default     = "10.10.0.0/20"
  description = "VPC Cidr"
}

# For Configuring Subnet Ciders
variable "subnet_cidr_newbits" {
  type        = string
  default     = 4
  description = "Subnet Cider Configuration"
}

# User-Region Specific Private Key
variable "private_key_name" {
  type        = string
  description = "User-Region Specific Private Key"
}

# Path to AWS Private key
variable "private_key_file_path" {
  type        = string
  description = "Path to AWS Private key"
}
```

```
# Allowed IP for SSH into bastion host
variable "ssh_location" {
  type        = string
  description = "Allowed IP for SSH into bastion host"
}

# Properties for ec2  Instance
variable "ec2_instance_type" {
  type        = string
  default     = "t2.micro"
  description = "Properties for ec2  Instance"
}

# Ami Selection
variable "ec2_image_ids" {
  type        = map
  description = "Ami Selection"

  default = {
    us-east-1      = "ami-0c2b8ca1dad447f8a"
    # us-east-2      = "ami-0e01ce4ee18447327"
  }
}
```

Ansible Scripts:

- Backend.yml

```yaml
---
- hosts: all
  become: yes
  become_user: root

  tasks:
    # Update all packages
    - name: Updating All packaes on the system
      yum:
        name: "*"
        state: latest
    #--------------------
    # Installing Java packages
    - name: Istalling Java package
      shell: yum install java-1.8.0-openjdk -y
    # ------------------------

    # Installing Maven packages
    - name: Installing Maven
      yum:
        name: maven
        state: present
    #--------------------------

    # Installing Git and Cloning the project Repository
    - name: Installing Git Package
      yum:
        name: git
        state: present

    - name: Creating a new new directory to clone the project repository On EC2
      file:
        path: "/home/ec2-user/news-api"
        state: directory
      become: yes
      become_user: ec2-user

    - name: Creating a new new directory to clone the project repository On local machine
      git:
        repo: https://github.com/Vikrant279/Group1-IADT
        dest: "/home/ec2-user/news-api"
      become: yes
      become_user: ec2-user
    #-------------------------------------------------
```

```
    #-------------------------------------------------
    # Installing new-api files
    - name: Executing Maven
      shell: mvn clean install
      register: mvn_result
      args:
        chdir: news-api/
      become: yes
      become_user: ec2-user

    - name: Maven Output
      debug:
        var: mvn_result

    - name: Copy API files to server
      copy:
        src: files/services/news-api.service
        dest: /etc/systemd/system
        owner: root
        group: root
      notify:
        - Reloading the services
        - Starting the API service
    #-------------------------------------------------
  handlers:
    - name: Reloading the services
      systemd:
        daemon_reload: yes

    - name: Starting the API service
      systemd:
        name: news-api
        state: started
```

Frontend.yml

```yaml
---
- hosts: all
  become: yes
  become_user: root

  tasks:
    # Update All packages
    - name: Updating all the packages on the system
      yum:
        name: "*"
        state: latest
    #--------------------
    # Installing Apache packages
    - name: Installing httpd package
      yum:
        name: httpd

    - name: Starting the httpd Service
      service:
        name: httpd
        state: started
        enabled: yes

    - name: Enabling the httpd Service
      systemd:
        name: httpd
        enabled: yes
        masked: no
    #-----------------------------
    # Installing news_website files
    - name: Coping the website files on public EC2 Instance
      copy:
        src: "{{ item }}"
        dest: /var/www/html/
      with_fileglob:
        - ./files/website/*

    - name: Copying the java script file to required location
      template:
        src: ./files/templates/scripts.js.j2
        dest: /var/www/html/scripts.js
```

```yaml
    - name: Reloading the httpd service
      systemd:
        name: httpd
        state: reloaded
    #-----------------------------
```

References:

https://unix.stackexchange.com/questions/183951/what-do-the-h-and-p-do-in-this-command

https://registry.terraform.io/modules/cloudposse/ec2-bastion-server/aws/latest

https://www.terraform.io/docs/language/resources/provisioners/local-exec.html

https://www.udemy.com/join/login-popup/?next=/course/aws-certified-solutions-architect-associate-saa-c02/learn/lecture/26098020#overview

- Stephane Maarek AWS Solutions Architect Associate Certification

https://newsapi.org/

- NEWS api source

https://www.udemy.com/join/login-popup/?next=/course/aws-certified-developer-associate-dva-c01/learn/lecture/19771482#overview

- Stephane Maarek AWS Developer Associate Certification
-

https://parmarnaitik0909.medium.com/deploying-hosting-a-django-website-on-cpanel-with-git-version-control-6e8dce70a316

https://medium.com/@morenojr90/fully-understanding-web-tech-development-6e866442513c