INTERNATIONAL
UNIVERSITY OF
APPLIED SCIENCES

COURSE OF STUDY: SOFTWARE DEVELOPMENT

COURSE TITLE: CLOUD PROGRAMMING

MATRICULATION NO: 92107689

TUTOR: GEORGI DIMCHEV

AUGUST 4, 2024

1. Introduction

In today's digital world, the necessity of having a solid and resistant web application infrastructure is incontestable. While designing our cloud architecture for our firm’s site, we have to concentrate on ensuring it caters for a global audience and is easily scalable. The design will be geared towards offering end users a seamless experience that will also be able to handle increased traffic as well as workload cost-effectively.

1.1. Project Goals

i. **High Availability:** This means that the website must be kept running at all times, even if one or more components fail. It requires AWS services to ensure resilience and fault tolerance (Amazon Web Services [AWS], 2024a).

ii. **Global Performance:** In order to minimize latency while providing consistent performance worldwide, this project should strategically place resources across several locations depending on the network traffic pattern such as AWS Content Delivery Network (CDN) (AWS, 2024b).

iii. **Scalability:** Backend infrastructure should scale up on demand by accommodating varying levels of traffic hence maintaining optimum performance even during peak periods of usage (AWS, 2024a).

1.2. Proposed AWS Services and Architecture

To meet these objectives, the following AWS services will be integrated into the cloud architecture:

• **Amazon EC2 and Auto Scaling Groups:** To host services, including any dynamic content or application logic. Auto Scaling Groups will ensure that the number of EC2 instances scales up or down based on traffic demands, providing both scalability and cost-efficiency (AWS, 2024a).

• **Virtual Private Cloud (VPC):** Virtual Private Cloud (VPC) is a cloud computing service that allows you to create a private, isolated network within a cloud environment. It provides control over network configuration, security, and routing, similar to a traditional data center, but with the flexibility and scalability of cloud resources (AWS, 2024b).

• **Internet Gateway:** An Internet Gateway is a horizontally scaled, redundant, and highly available component that allows resources in a VPC to connect to the internet and vice versa. It supports both inbound and outbound traffic and serves as the entry and exit point for data traveling to and from the internet (AWS, 2024c).

• **Load Balancer:** A Load Balancer is a device or service that manages the distribution of network or application traffic across multiple servers or instances to ensure that no single server becomes overwhelmed. It acts as an intermediary between clients and servers, directing traffic based on predefined rules or algorithms (AWS, 2024d).

• **Security Groups:** Security Groups are used to control the inbound and outbound traffic to your AWS resources, providing a flexible and stateful firewall solution (AWS, 2024e).

• **Subnets:** Subnets allow you to partition a VPC into smaller, manageable segments, enabling you to control and isolate network traffic and resources effectively (AWS, 2024f).

• **Terraform:** For Infrastructure as Code (IaC), these tools will be utilized to define and deploy the architecture in a repeatable and manageable manner. They will help in versioning, automating, and maintaining the infrastructure (HashiCorp, 2024).

• **Docker-compose:** The web app is built and packaged into Docker containers from Dockerfiles (Docker, 2024).

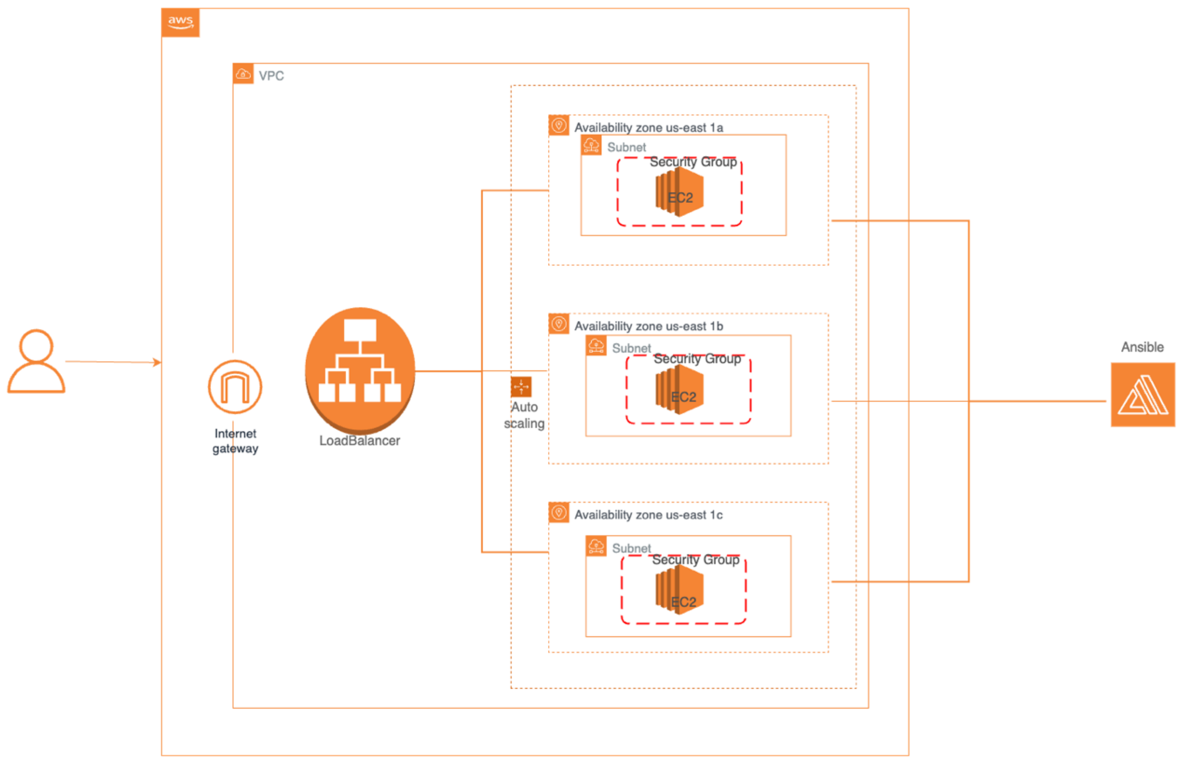
• **Ansible:** Ansible is an automation platform that uses human-readable YAML (Yet Another Markup Language) files, known as playbooks, to define configurations and orchestrate the deployment and management of systems. Three virtual servers used in this project will be configured and the Docker container will be deployed to our virtual server through Ansible (Red Hat, 2024).

2. Implementation Plan:

The implementation is categorized into three phase namely:

2.1. **Design Phase:**

2.1.1. Archectural design



(Chris, 2024).

2.2. **IaC Configuration:**

2.2.1. **Access Key:**

When it comes to AWS, an Access Key is like a password combination used for allowing access to AWS services and resources. It consists of two-bit information:

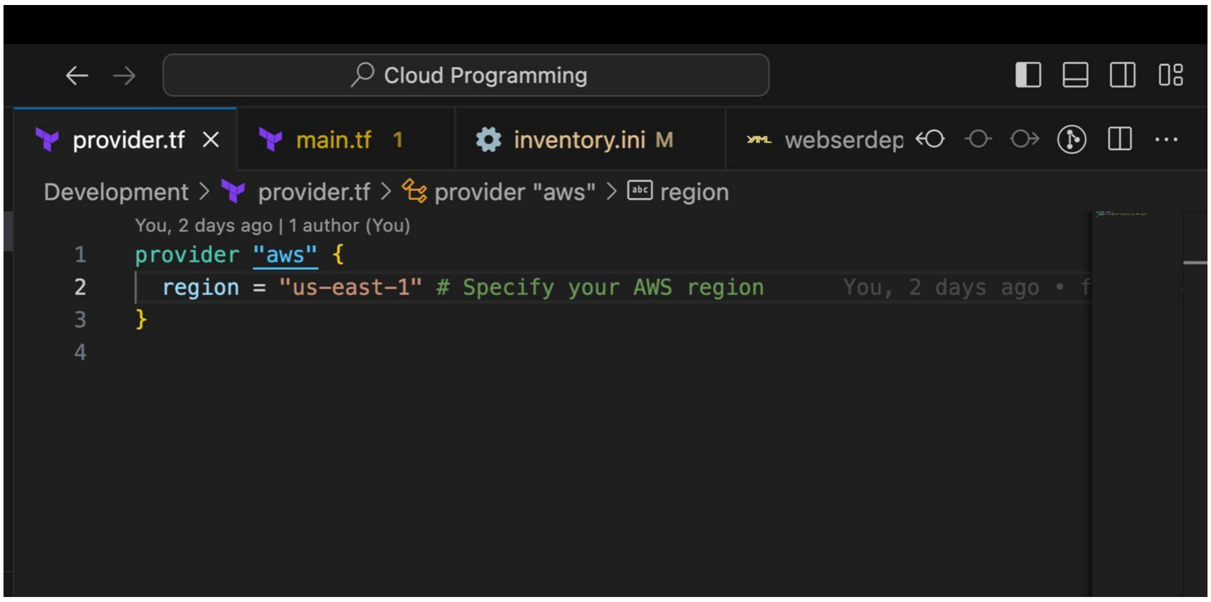
- Access Key ID: A unique identifier of the access key.
- Secret Access Key: Secret key used to sign requests sent to AWS services.

These two keys help authenticate who someone is and what they can do with their data by both individuals and applications. The main uses of these keys include API calls, CLI commands, and SDKs for managing the lifecycle of AWS resources through code. To avoid unauthorized entrance into your AWS environment, these keys are kept private.

This project assumes that one already have AWS account. To get the Access Key ID and Secret Access Key, login to AWS console and navigate to the security credential and create one. then navigate to the area where the access key will be used through the Command Line Interface(CLI) and use the command “aws configure” this will prompt to enter the access key id and secret key id you created. This action will successfully grant access to use the resources in aws. It is worth mentioning that there are also other ways to this like using Environment variables.

2.2.2. Choosing Provider:

Terraform being a universal Infrastructure as Code(IaC) can be use for AWS, Google platform, Oracle and other virtual server providers. Here, specifically define the provider the project will be build on. In this project, the chosen choice is AWS. AWS is chosen because it the most widely use, occupies about 80 percent of the market share with huge community of users. (Synergy Research Group, 2024)


Cloud Programming
provider.tf
main.tf 1
inventory.ini M
webserdep
Development > provider.tf > provider "aws" > region
You, 2 days ago | 1 author (You)

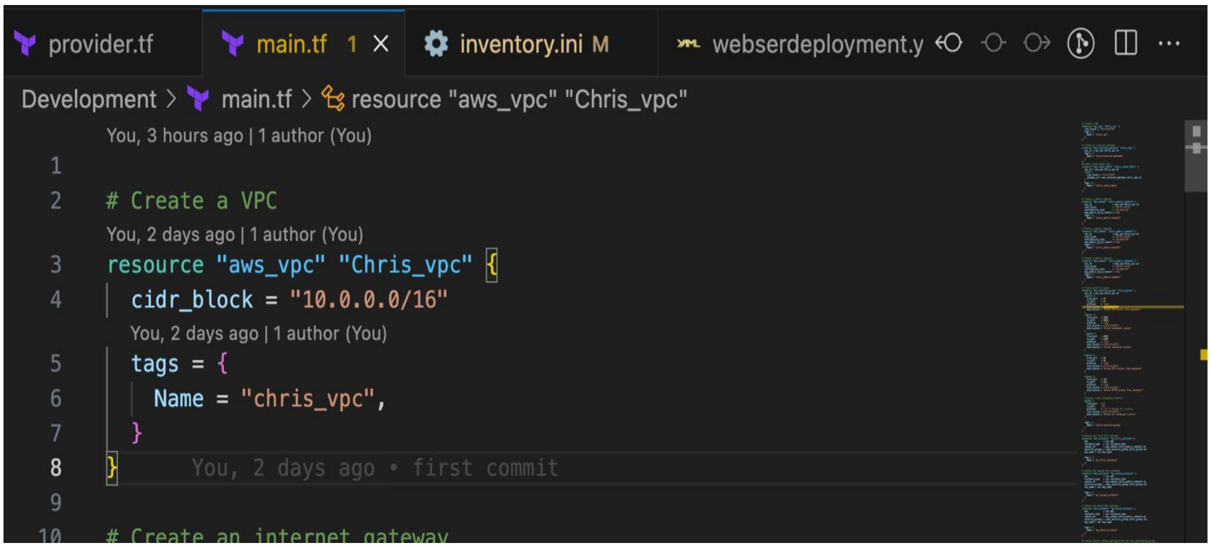
```
provider "aws" {
  region = "us-east-1" # Specify your AWS region
}
```

(Etiti27.2024a)

the above image states that the provider used and the region. In this case, its AWS and us-east-1 respectively.

2.2.3. Creation of Virtual Private Cloud:

Inorder to segregate and add layer of security to the architecture, Virtual private Cloud is create.



provider.tf | main.tf 1 | inventory.ini M | webserdeployment.y

Development > main.tf > resource "aws_vpc" "Chris_vpc"

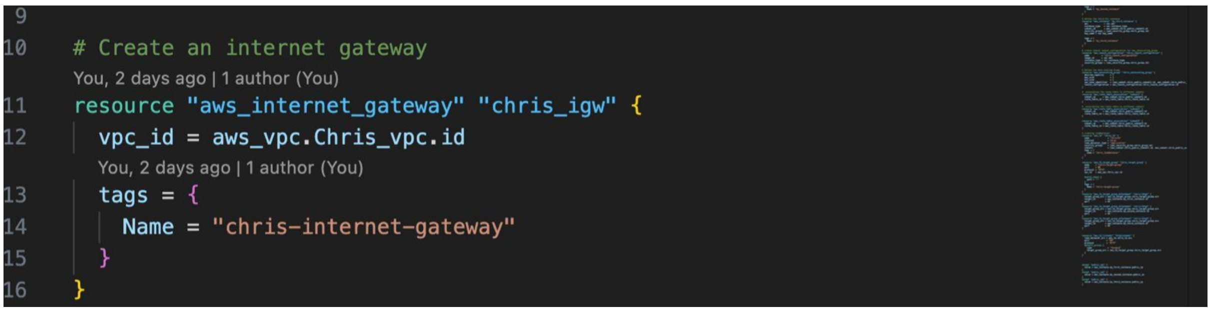
```
You, 3 hours ago | 1 author (You)

# Create a VPC
You, 2 days ago | 1 author (You)
resource "aws_vpc" "Chris_vpc" {
  cidr_block = "10.0.0.0/16"
  You, 2 days ago | 1 author (You)
  tags = {
    Name = "chris_vpc",
  }
}       You, 2 days ago • first commit

# Create an internet gateway
```

(Etiti27.2024b).

here, Cidr_block which stands for Classless Inter-Domain Routing block was assigned. This gives the range of Internet Protocol (IP) within the network

2.2.4. Internet Gateway:

Internet gateway allows internet access to the VPC network



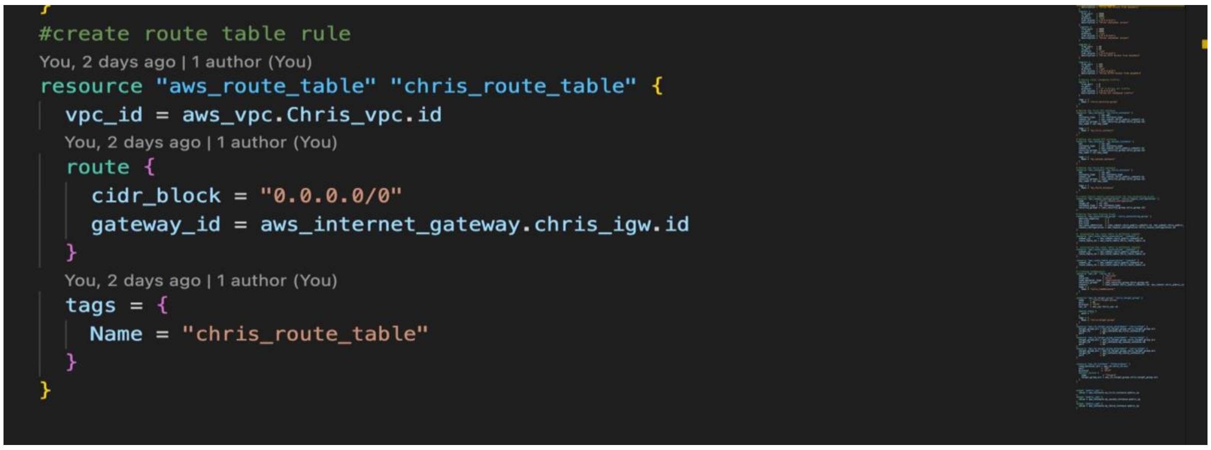
```

# Create an internet gateway
You, 2 days ago | 1 author (You)
resource "aws_internet_gateway" "chris_igw" {
  vpc_id = aws_vpc.Chris_vpc.id
  You, 2 days ago | 1 author (You)
  tags = {
    Name = "chris-internet-gateway"
  }
}
```

(Etiti27. 2024c).

here, The Virtual Private Cloud where internet should be allowed to flow is defined

2.2.5. Route Table:

Routing table is a set of rules used by network devices to determine the best path for sending packets to the destination.



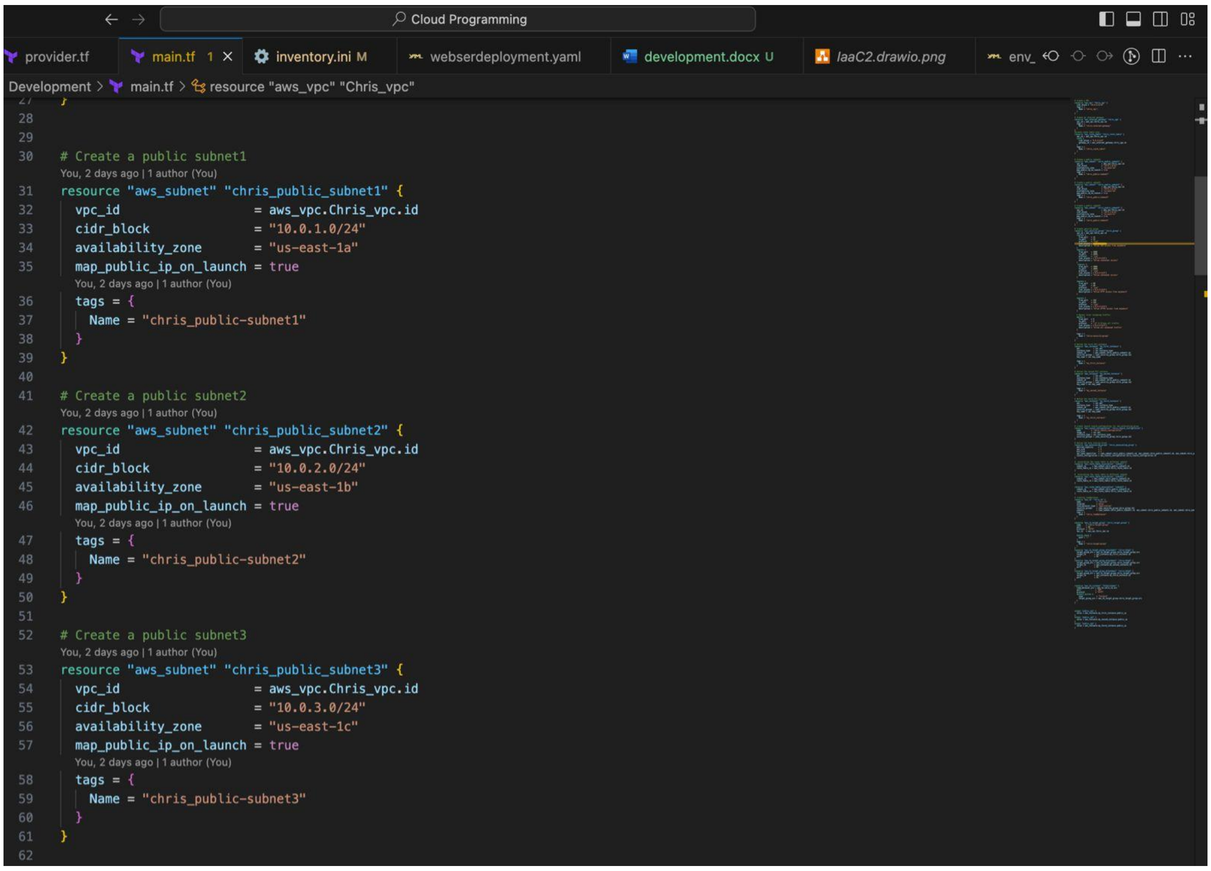
```
#create route table rule
You, 2 days ago | 1 author (You)
resource "aws_route_table" "chris_route_table" {
  vpc_id = aws_vpc.Chris_vpc.id
  You, 2 days ago | 1 author (You)
  route {
    cidr_block = "0.0.0.0/0"
    gateway_id = aws_internet_gateway.chris_igw.id
  }
  You, 2 days ago | 1 author (You)
  tags = {
    Name = "chris_route_table"
  }
}
```

(Etiti27. 2024d).

Here, Virtual Private Cloud is specified, the specific internet gateway to follow and the ip address range. here traffics are allowed from everywhere

2.2.6. Subnets:

This divides and segregate resources within the Virtual private network. In simple understanding, we build a room within our house, here VPC is the house and subnet is the rooms. From our archectural designed, we have three subnets



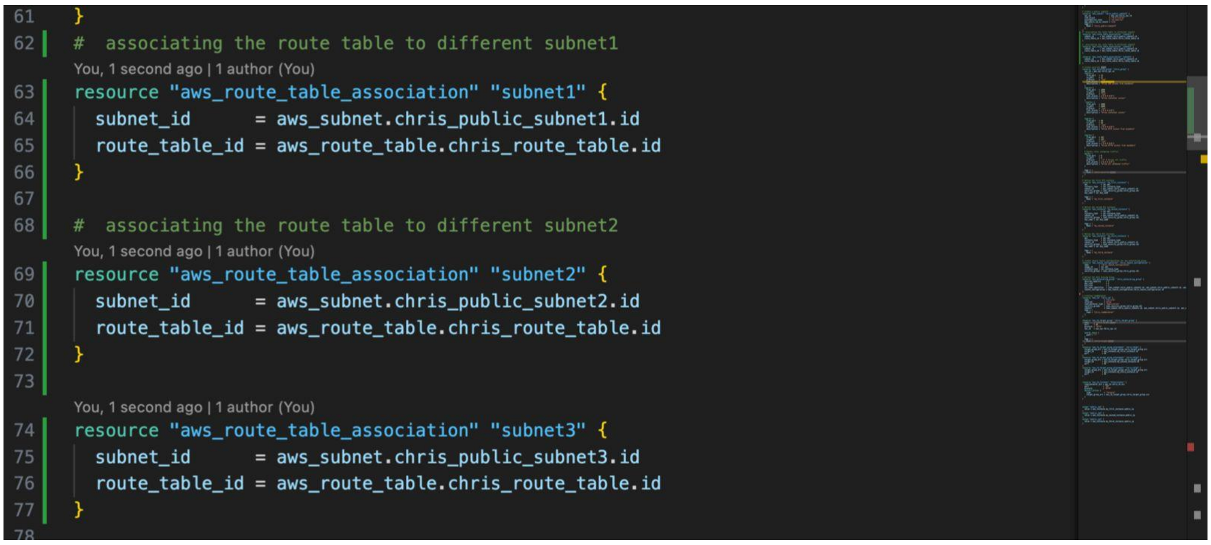
```
# Create a public subnet1
You, 2 days ago | 1 author (You)
resource "aws_subnet" "chris_public_subnet1" {
  vpc_id                  = aws_vpc.Chris_vpc.id
  cidr_block              = "10.0.1.0/24"
  availability_zone       = "us-east-1a"
  map_public_ip_on_launch = true
  You, 2 days ago | 1 author (You)
  tags = {
    Name = "chris_public-subnet1"
  }
}

# Create a public subnet2
You, 2 days ago | 1 author (You)
resource "aws_subnet" "chris_public_subnet2" {
  vpc_id                  = aws_vpc.Chris_vpc.id
  cidr_block              = "10.0.2.0/24"
  availability_zone       = "us-east-1b"
  map_public_ip_on_launch = true
  You, 2 days ago | 1 author (You)
  tags = {
    Name = "chris_public-subnet2"
  }
}

# Create a public subnet3
You, 2 days ago | 1 author (You)
resource "aws_subnet" "chris_public_subnet3" {
  vpc_id                  = aws_vpc.Chris_vpc.id
  cidr_block              = "10.0.3.0/24"
  availability_zone       = "us-east-1c"
  map_public_ip_on_launch = true
  You, 2 days ago | 1 author (You)
  tags = {
    Name = "chris_public-subnet3"
  }
}
```

(Etiti27. 2024e).

The VPC where the subnets will be is defined, the internet protocol range is also defined. Based on the project requirement which states that it should be available in multiple zone, therefore subnet1 is assigned the to us-east-1a, subnet2 to us-east-1b and subnet3 to us-east-1c. In addition, the type of subnet is specified and in this scenario we specified all three subnets to be public subnet.

2.2.7. Associate the subnets to the Route Table:

The path at which packet is sent to the subnets are defined.

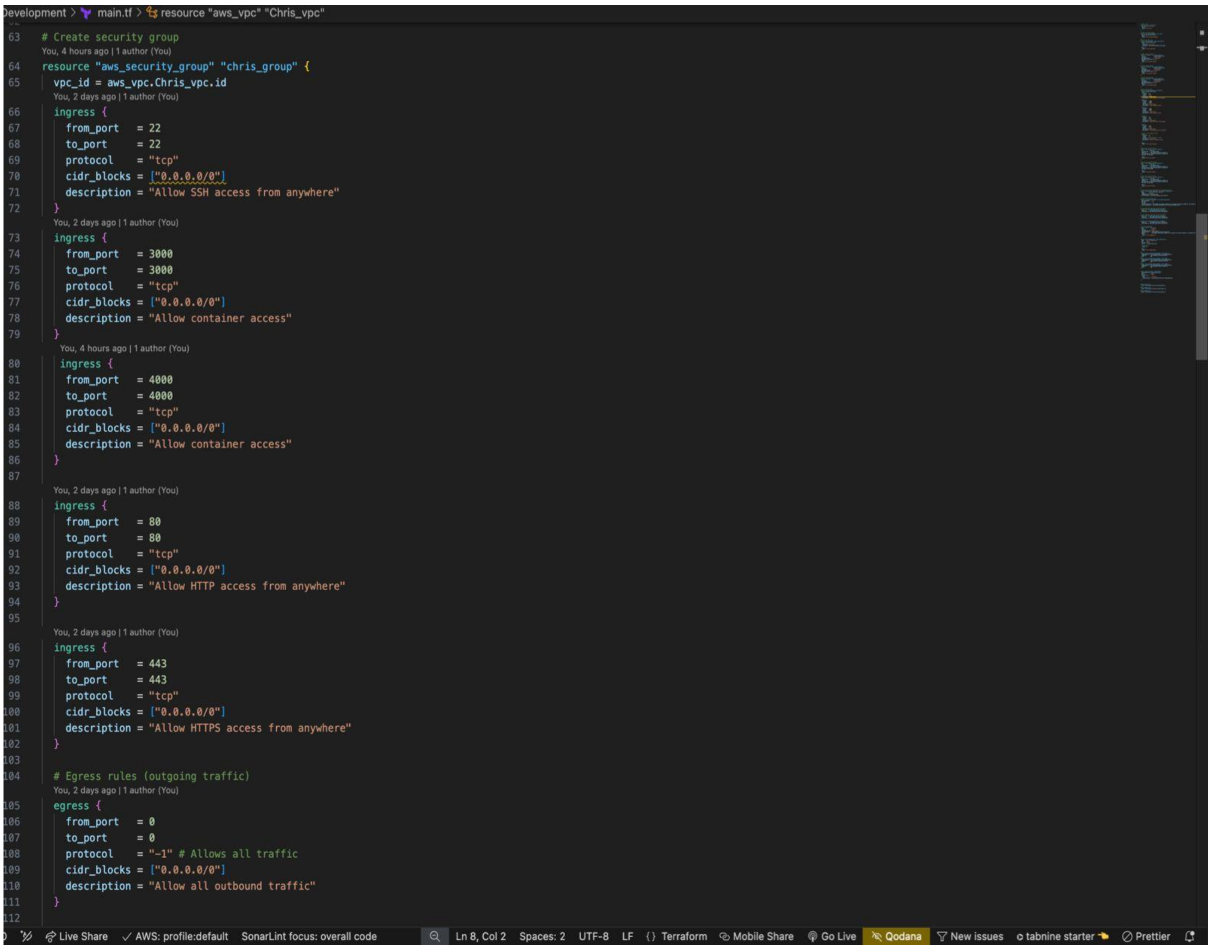


```
}
#  associating the route table to different subnet1
You, 1 second ago | 1 author (You)
resource "aws_route_table_association" "subnet1" {
  subnet_id      = aws_subnet.chris_public_subnet1.id
  route_table_id = aws_route_table.chris_route_table.id
}

#  associating the route table to different subnet2
You, 1 second ago | 1 author (You)
resource "aws_route_table_association" "subnet2" {
  subnet_id      = aws_subnet.chris_public_subnet2.id
  route_table_id = aws_route_table.chris_route_table.id
}

You, 1 second ago | 1 author (You)
resource "aws_route_table_association" "subnet3" {
  subnet_id      = aws_subnet.chris_public_subnet3.id
  route_table_id = aws_route_table.chris_route_table.id
}
```

(Etiti27. 2024f).

2.2.8. Security Group:

Security group as the name implies is a security rules and protocol acting in the subnets that define what must and must not access the resources they are attached to.



```
# Create security group
resource "aws_security_group" "chris_group" {
  vpc_id = aws_vpc.Chris_vpc.id
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = "Allow SSH access from anywhere"
  }
  ingress {
    from_port   = 3000
    to_port     = 3000
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = "Allow container access"
  }
  ingress {
    from_port   = 4000
    to_port     = 4000
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = "Allow container access"
  }

  ingress {
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = "Allow HTTP access from anywhere"
  }

  ingress {
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
    description = "Allow HTTPS access from anywhere"
  }

  # Egress rules (outgoing traffic)
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1" # Allows all traffic
    cidr_blocks = ["0.0.0.0/0"]
    description = "Allow all outbound traffic"
  }
```

(Etiti27. 2024g).

Here, the actual VPC this security group is meant for is defined, the ingress rule specifies the Internet Protocol (IP) range that can access the resources that this security group is protecting and from what port numbers. In this project, the ingress port number specified are port 22 which is for SSH, SSH is necessary for this project because our artifact is being deployed through ansible and ansible connect to the servers through SSH, Port 3000 and 4000 is also defined because we are using React for frontend and Node JS for backend and they run on port 3000 and 4000 respectively

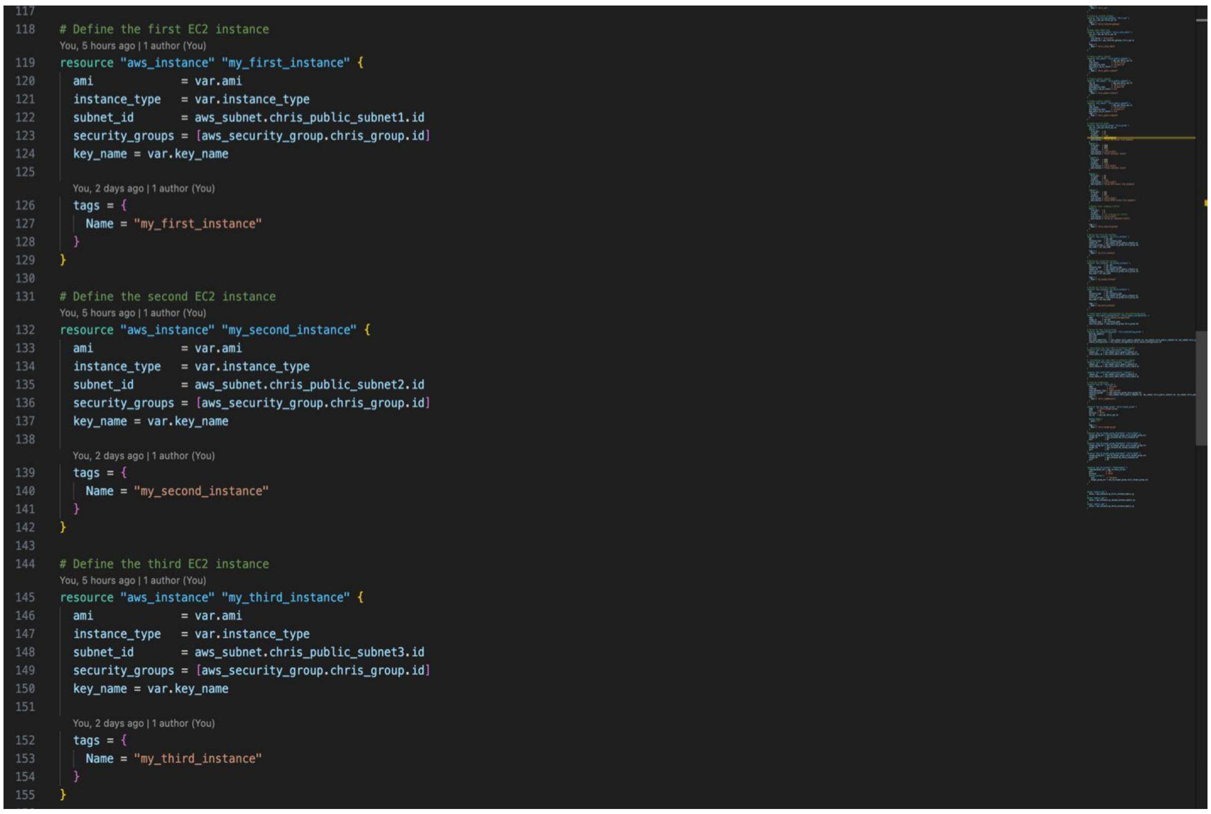
In the egress rule, we allowed all ports

2.2.9. Elastic Cloud Computing (EC2):

There is the virtual server where we deploy the created artifacts

- **ami:** defines the type of Operating System (OS) that the server will be using
- **instance type:** allows the use to specifies and choose the amount of storage and capacity the server will take
- **subnet id:** In our case, it is use to specify among the three subnets created which one the EC2 should be placed on

- **Security groups:** The security ground which gives the rule and protocol at which the EC2 can be accessed is given.
- **key name: T**his is an added layer of security which is optional but recommended which is use to login to the server. moslyt in .pem format

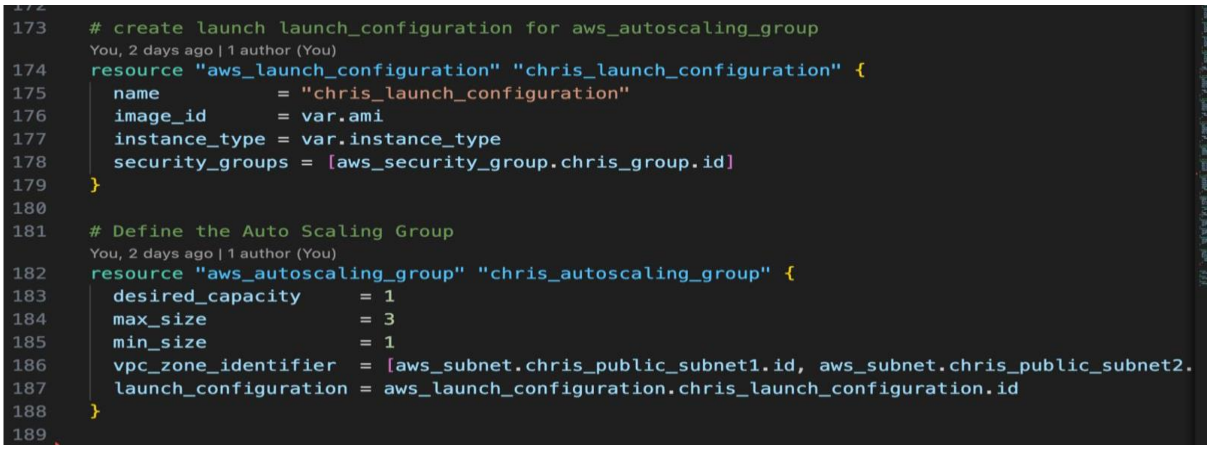


```
# Define the first EC2 instance
You, 5 hours ago | 1 author (You)
resource "aws_instance" "my_first_instance" {
  ami             = var.ami
  instance_type   = var.instance_type
  subnet_id       = aws_subnet.chris_public_subnet1.id
  security_groups = [aws_security_group.chris_group.id]
  key_name = var.key_name

  You, 2 days ago | 1 author (You)
  tags = {
    Name = "my_first_instance"
  }
}

# Define the second EC2 instance
You, 5 hours ago | 1 author (You)
resource "aws_instance" "my_second_instance" {
  ami             = var.ami
  instance_type   = var.instance_type
  subnet_id       = aws_subnet.chris_public_subnet2.id
  security_groups = [aws_security_group.chris_group.id]
  key_name = var.key_name

  You, 2 days ago | 1 author (You)
  tags = {
    Name = "my_second_instance"
  }
}

# Define the third EC2 instance
You, 5 hours ago | 1 author (You)
resource "aws_instance" "my_third_instance" {
  ami             = var.ami
  instance_type   = var.instance_type
  subnet_id       = aws_subnet.chris_public_subnet3.id
  security_groups = [aws_security_group.chris_group.id]
  key_name = var.key_name

  You, 2 days ago | 1 author (You)
  tags = {
    Name = "my_third_instance"
  }
}
```

(Etiti27. 2024h).

2.3.0. Autoscaling Group:

Based on the project specification, the server should be able to scale and recreate itself if the load beco mes too much.



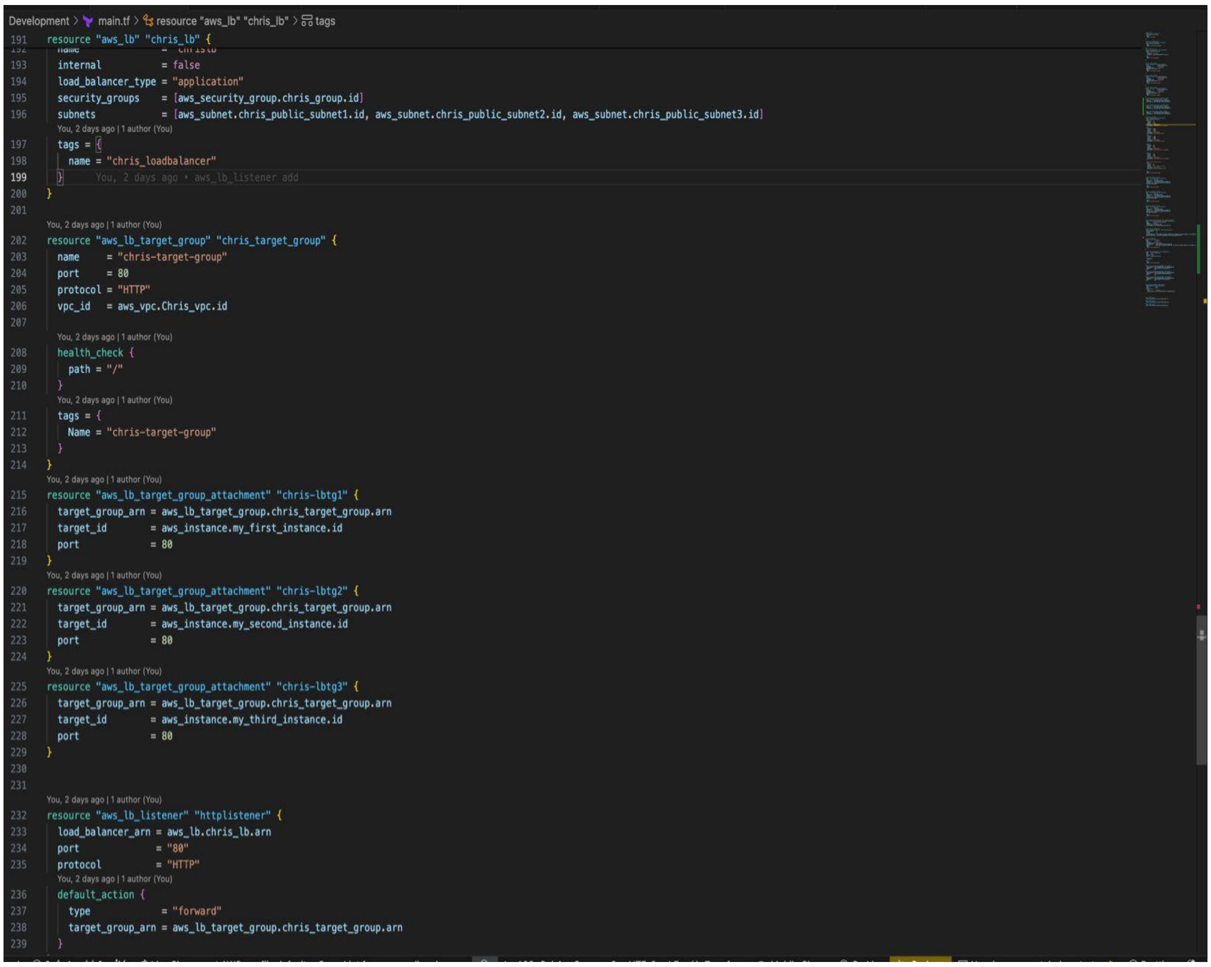
```
# create launch launch_configuration for aws_autoscaling_group
You, 2 days ago | 1 author (You)
resource "aws_launch_configuration" "chris_launch_configuration" {
  name          = "chris_launch_configuration"
  image_id      = var.ami
  instance_type = var.instance_type
  security_groups = [aws_security_group.chris_group.id]
}

# Define the Auto Scaling Group
You, 2 days ago | 1 author (You)
resource "aws_autoscaling_group" "chris_autoscaling_group" {
  desired_capacity     = 1
  max_size             = 3
  min_size             = 1
  vpc_zone_identifier  = [aws_subnet.chris_public_subnet1.id, aws_subnet.chris_public_subnet2.
  launch_configuration = aws_launch_configuration.chris_launch_configuration.id
}
```

(Etiti27. 2024i).

The server that will be monitoring the server on their load capacity is created and configured first. In the autoscaling group, the desired capacity of instances the servers will be at launch is defined, in our case, we want just one instances to be created. the maximum number of instance that can be replicated is specified, In our case, the maxium is three and the minimun is taken also. In the vpc_zone_identifier, the subnets that will join in this autoscaling is added. and finally, the server that will monitor this process is added.

2.3.1. Loadbalancer:

This distributes traffics across all the instances, this is done to make sure that all instances can get traffic and not one instances getting all the traffics. Loadbalancer also perform health check



Development > main.tf > resource "aws_lb" "chris_lb" > tags

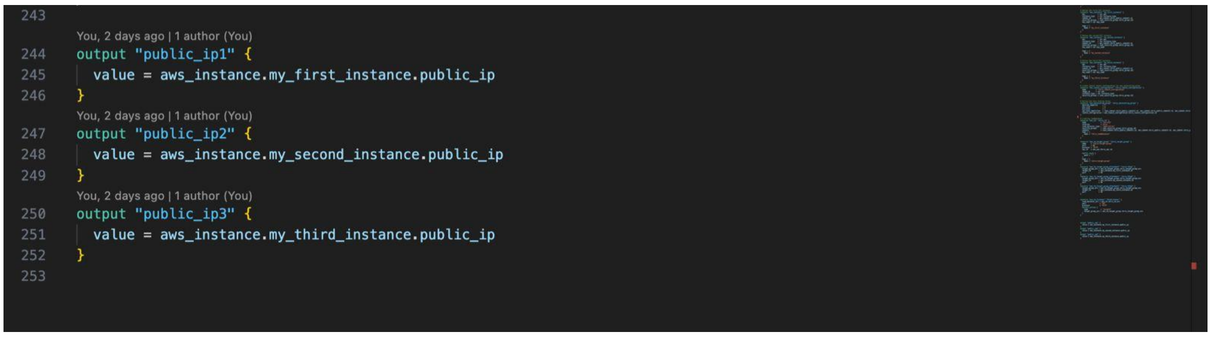
```
resource "aws_lb" "chris_lb" {
  name               = "chris_lb"
  internal           = false
  load_balancer_type = "application"
  security_groups    = [aws_security_group.chris_group.id]
  subnets            = [aws_subnet.chris_public_subnet1.id, aws_subnet.chris_public_subnet2.id, aws_subnet.chris_public_subnet3.id]

  tags = {
    name = "chris_loadbalancer"
  }       You, 2 days ago • aws_lb_listener add
}

resource "aws_lb_target_group" "chris_target_group" {
  name     = "chris-target-group"
  port     = 80
  protocol = "HTTP"
  vpc_id   = aws_vpc.Chris_vpc.id

  health_check {
    path = "/"
  }
  tags = {
    Name = "chris-target-group"
  }
}
resource "aws_lb_target_group_attachment" "chris-lbtg1" {
  target_group_arn = aws_lb_target_group.chris_target_group.arn
  target_id        = aws_instance.my_first_instance.id
  port             = 80
}
resource "aws_lb_target_group_attachment" "chris-lbtg2" {
  target_group_arn = aws_lb_target_group.chris_target_group.arn
  target_id        = aws_instance.my_second_instance.id
  port             = 80
}
resource "aws_lb_target_group_attachment" "chris-lbtg3" {
  target_group_arn = aws_lb_target_group.chris_target_group.arn
  target_id        = aws_instance.my_third_instance.id
  port             = 80
}


resource "aws_lb_listener" "httplistener" {
  load_balancer_arn = aws_lb.chris_lb.arn
  port              = "80"
  protocol          = "HTTP"
  default_action {
    type             = "forward"
    target_group_arn = aws_lb_target_group.chris_target_group.arn
  }
```

(Etiti27. 2024j).

Here, we pass a flag " Internal" This flag determines whether the Load Balancer is internal or internet-facing. Setting it to false means that the Load Balancer is **internet-facing**, meaning it will be accessible from the internet. The type of loadbalancer is defined and in my case, It is an application. the security groups and the subnets are defined. After this, the targets that will receive traffic from the Load Balancer is specified through loadbalance target group. In this project, we targets http traffics that passes through our VPC and we perform a health check on home directory. After this, the servers that traffics will be distributed to is attached to the loadbalancer and listen to all forward traffic requests on port 80.

2.3.2. Output:

Since this project is provisioned to use ansible to deploy our artifact, it needs the public address of all the instances. Inorder to achiever that without login to aws console, I decided to use output to display the public ip addresses



```
You, 2 days ago | 1 author (You)
output "public_ip1" {
  value = aws_instance.my_first_instance.public_ip
}
You, 2 days ago | 1 author (You)
output "public_ip2" {
  value = aws_instance.my_second_instance.public_ip
}
You, 2 days ago | 1 author (You)
output "public_ip3" {
  value = aws_instance.my_third_instance.public_ip
}
```

(Etiti27. 2024k).

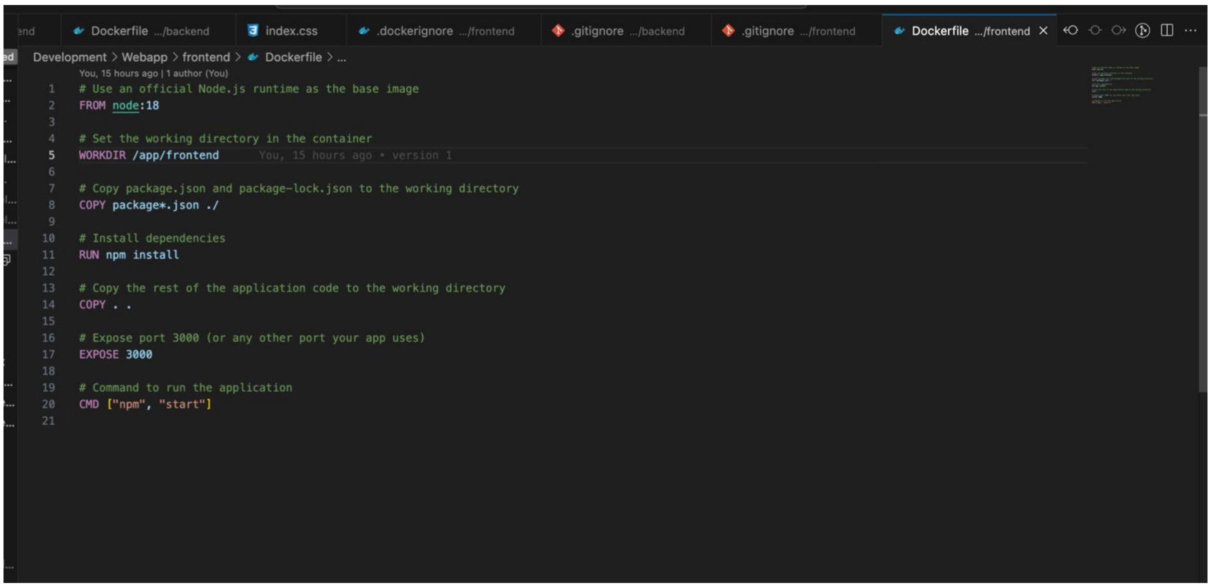
3. Deployment Phase:

3.1. Running Terraform:

- **terraform init:** After the setup phase, the terraform is run to create the resources defined therein. The first action is to initialize the terraform and prepare the working directory for other terraform commands to work.
- **terraform plan:** This command is run to give output information on the what will be created.
- **terraform validate:** This comand runs through the entire setup and check the validity of the setup.
- **terraform apply:** This command will apply the terraform setup and create the architecture

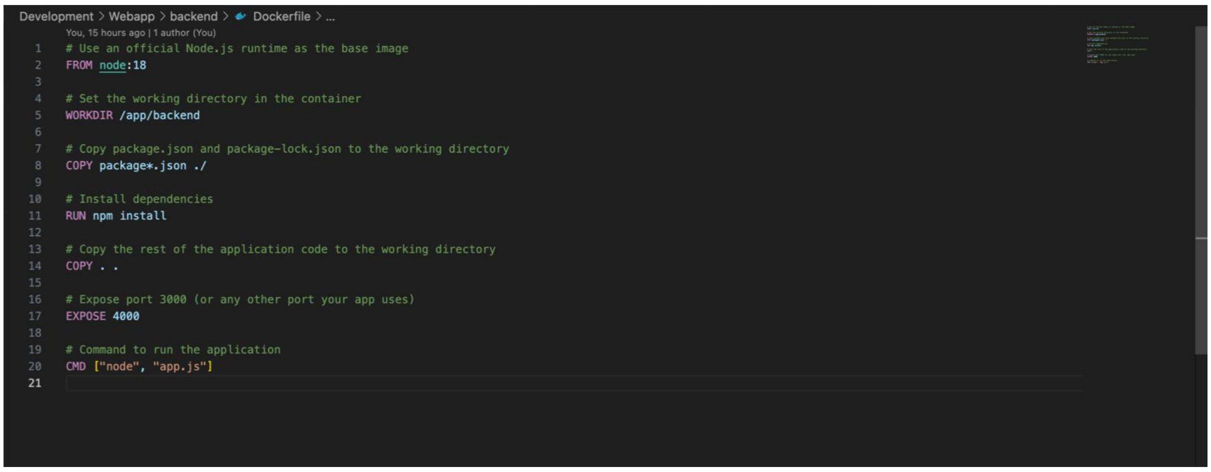
3.2. Declaration of Dockerfile:

Dockerfile is a text file that contains a set of instructions used to build a Docker image


Dockerfile .../backend
index.css
.dockerignore .../frontend
.gitignore .../backend
.gitignore .../frontend
Dockerfile .../frontend
Development > Webapp > frontend > Dockerfile > ...
You, 15 hours ago | 1 author (You)

```
# Use an official Node.js runtime as the base image
FROM node:18

# Set the working directory in the container
WORKDIR /app/frontend      You, 15 hours ago • version 1

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose port 3000 (or any other port your app uses)
EXPOSE 3000

# Command to run the application
CMD ["npm", "start"]
```

(Etiti27. 2024l).

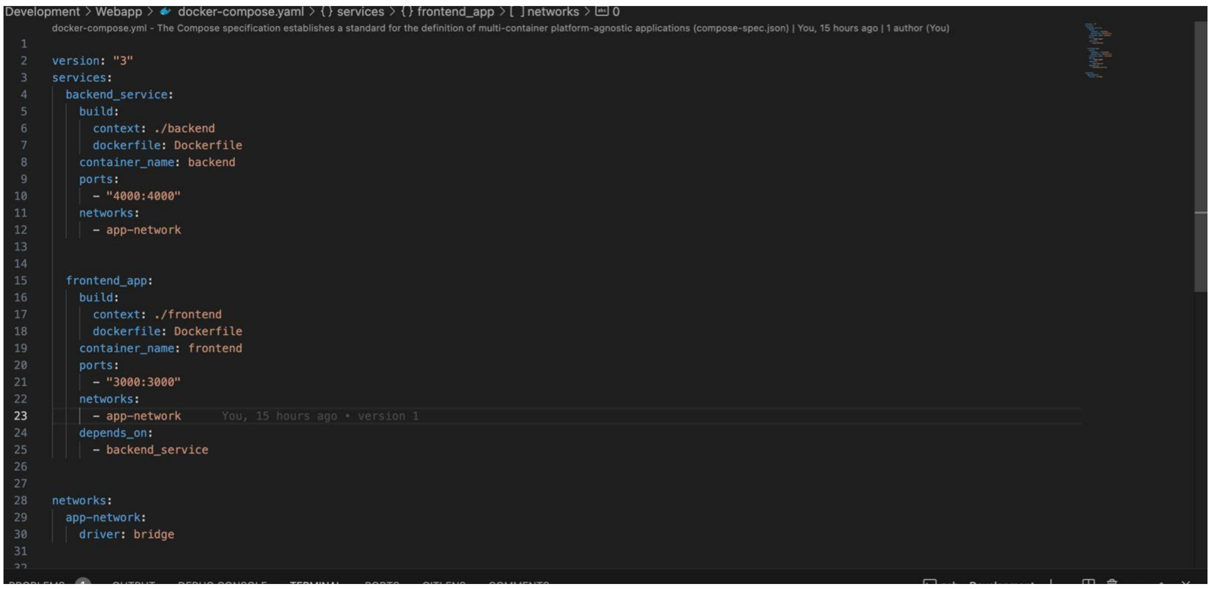

Development > Webapp > backend > Dockerfile > ...
You, 15 hours ago | 1 author (You)

```
# Use an official Node.js runtime as the base image
FROM node:18

# Set the working directory in the container
WORKDIR /app/backend

# Copy package.json and package-lock.json to the working directory
COPY package*.json ./

# Install dependencies
RUN npm install

# Copy the rest of the application code to the working directory
COPY . .

# Expose port 3000 (or any other port your app uses)
EXPOSE 4000

# Command to run the application
CMD ["node", "app.js"]
```

(Etiti27. 2024m).

then, docker-compose will be use to deploy both of the frontend and backend at a go.

below is docker compose file



Development > Webapp > docker-compose.yaml > {} services > {} frontend_app > [ ] networks > 0

docker-compose.yml - The Compose specification establishes a standard for the definition of multi-container platform-agnostic applications (compose-spec.json) | You, 15 hours ago | 1 author (You)

```yaml

version: "3"
services:
  backend_service:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: backend
    ports:
      - "4000:4000"
    networks:
      - app-network


  frontend_app:
    build:
      context: ./frontend
      dockerfile: Dockerfile
    container_name: frontend
    ports:
      - "3000:3000"
    networks:
      - app-network       You, 15 hours ago • version 1
    depends_on:
      - backend_service


networks:
  app-network:
    driver: bridge

```

(Etiti27. 2024n).

3.3. Ansible:

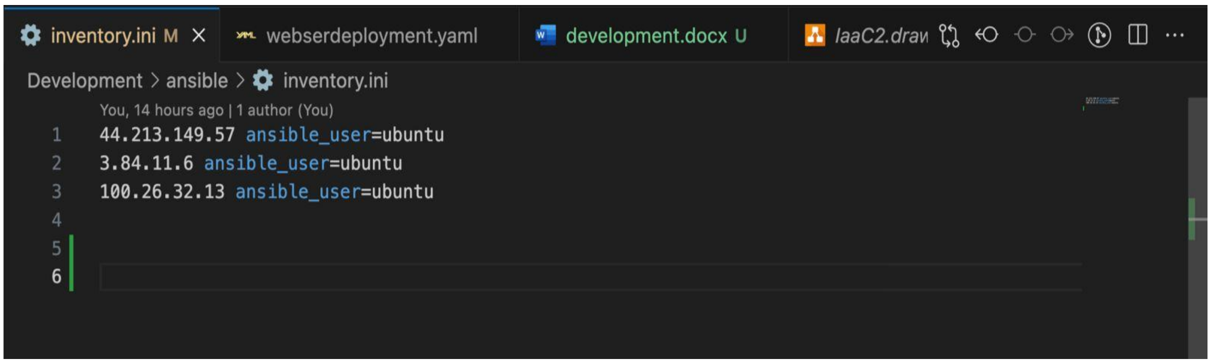
Since there are multiple serves, ansible is use to configure all of them by running a single command.

- **Setup:**

  As stated previously, ansible communicates to the serves through SSH, as such port 22 must be enabled in the server. The servers must also allow authorize the local server where the ansible playbook is being run on to access them and connect to them. This is done by running the following command on the servers `ssh-keygen` this will create and generate activation key. then copy the public rsa key and paste it in the authorize key session in the virtual server.

- **Create hosts:**

  Hosts (IP) of the servers are stored in inventory.ini file.



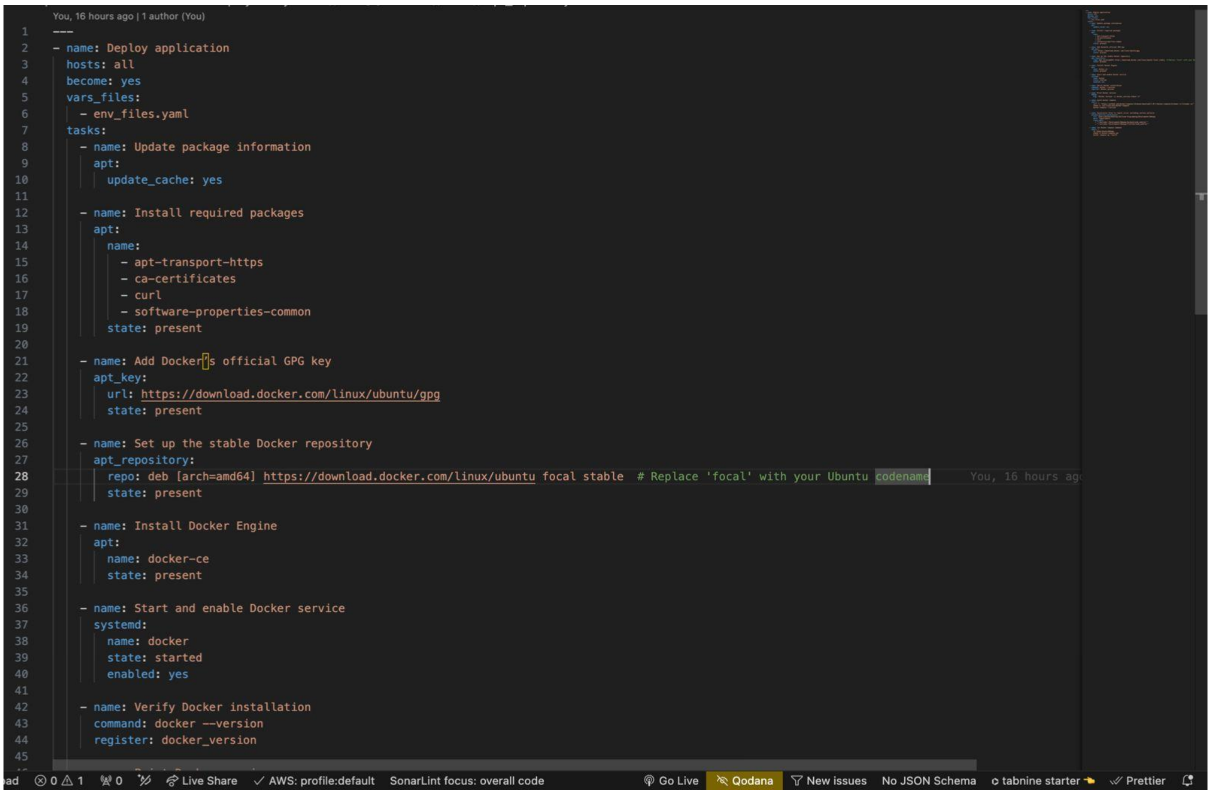
Development > ansible > inventory.ini

You, 14 hours ago | 1 author (You)

```ini
44.213.149.57 ansible_user=ubuntu
3.84.11.6 ansible_user=ubuntu
100.26.32.13 ansible_user=ubuntu


```

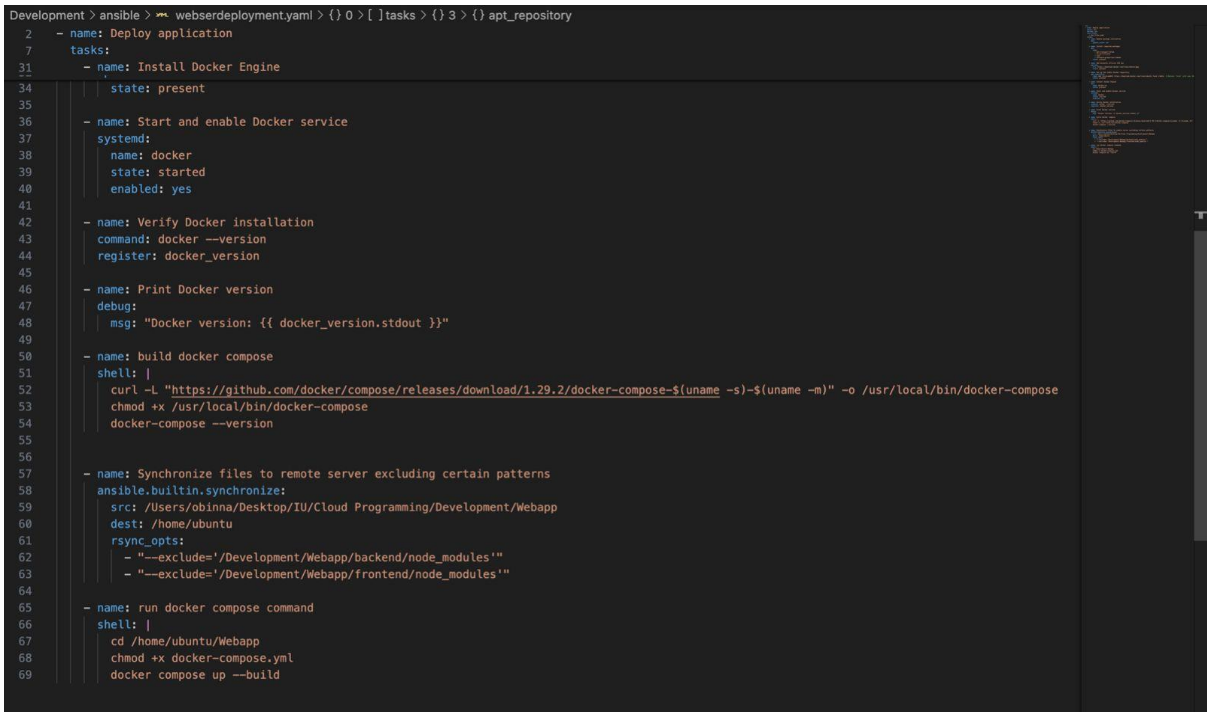
(Etiti27. 2024o).

- **Create Ansible playbook:**
  An Ansible playbook is a YAML file that defines a set of tasks to be executed on remote machines. Playbooks are the core of Ansible's automation capabilities and allow you to configure systems, deploy applications, and orchestrate tasks.



```yaml
---
- name: Deploy application
  hosts: all
  become: yes
  vars_files:
    - env_files.yaml
  tasks:
    - name: Update package information
      apt:
        update_cache: yes

    - name: Install required packages
      apt:
        name:
          - apt-transport-https
          - ca-certificates
          - curl
          - software-properties-common
        state: present

    - name: Add Docker's official GPG key
      apt_key:
        url: https://download.docker.com/linux/ubuntu/gpg
        state: present

    - name: Set up the stable Docker repository
      apt_repository:
        repo: deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable  # Replace 'focal' with your Ubuntu codename
        state: present

    - name: Install Docker Engine
      apt:
        name: docker-ce
        state: present

    - name: Start and enable Docker service
      systemd:
        name: docker
        state: started
        enabled: yes

    - name: Verify Docker installation
      command: docker --version
      register: docker_version
```

You, 16 hours ago
ad ⊗ 0 ⚠ 1 0 Live Share ✓ AWS: profile:default SonarLint focus: overall code Go Live Qodana New issues No JSON Schema tabnine starter Prettier


Development > ansible > webserdeployment.yaml > {} 0 > [ ] tasks > {} 3 > {} apt_repository

```yaml
- name: Deploy application
  tasks:
    - name: Install Docker Engine
        state: present

    - name: Start and enable Docker service
      systemd:
        name: docker
        state: started
        enabled: yes

    - name: Verify Docker installation
      command: docker --version
      register: docker_version

    - name: Print Docker version
      debug:
        msg: "Docker version: {{ docker_version.stdout }}"

    - name: build docker compose
      shell: |
        curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
        chmod +x /usr/local/bin/docker-compose
        docker-compose --version


    - name: Synchronize files to remote server excluding certain patterns
      ansible.builtin.synchronize:
        src: /Users/obinna/Desktop/IU/Cloud Programming/Development/Webapp
        dest: /home/ubuntu
        rsync_opts:
          - "--exclude='/Development/Webapp/backend/node_modules'"
          - "--exclude='/Development/Webapp/frontend/node_modules'"

    - name: run docker compose command
      shell: |
        cd /home/ubuntu/Webapp
        chmod +x docker-compose.yml
        docker compose up --build
```

(Etiti27. 2024p).

In the hosts flag, “all” was specified which means this commands will be applied in all the IP address in the inventory.ini file. Since, we are deploying docker images, the server must be setup for docker to run on them. this is done by installing docker and docker dependencies. The folder in which our application is copied to the virtual servers along with their Dockerfile and docker-compose file. `docker-compose up` command is use to build and start our docker image.

To run the playbook, `ansible-playbook -i inventory.ini playbook.yaml`

4. Conclusion

The successful deployment of a highly available and low-latency cloud architecture for a company's website involves a strategic combination of AWS services and modern cloud technologies. This project achieved the key objectives of high availability, global latency optimization, and backend auto-scaling, all while ensuring the infrastructure is manageable and replicable through Infrastructure as Code (IaC).

Modern cloud architectures depend on high availability; therefore, if there are any faults or support activities, it will still be possible to access the site. For instance, with Amazon Web Services (AWS) Elastic Load Balancer(ELB) together with Auto Scaling Groups(ASG), this architecture will work well in handling unpredictable changes in traffic volume and system status. Moreover, inbound traffic spread over many EC2 instances by ELB mitigates against the risk of service failure due to single points of failure. Finally, ASG can increase or decrease the number of instances on demand basis to ensure that consistent performance and availability is maintained within the back-end infrastructure.

Creating an infrastructure that is reproducible and manageable is important and can be done with Infrastructure as Code (IaC) tools like Terraform. IaC allows for the whole cloud architecture to be defined using codes which can be kept in version control and used repeatedly in multiple environments. This way, the deployment process is made simple, mistakes are limited and infrastructural changes managed across time. In order to ensure consistency and enable automated deployments, resources such as VPCs, subnets, security groups, and load balancers are coded.

4.1. Future Plan

The chosen architecture not only meets the current requirements but is also designed with future scalability in mind. The modular nature of AWS services and the IaC approach allows for easy updates and expansions. As the company's needs evolve, additional resources can be integrated into the infrastructure with minimal disruption, ensuring that the architecture remains adaptable and capable of supporting growth.

4.2. Framework use.

React js: https://legacy.reactjs.org/docs/getting-started.html.

Node js (Express): https://legacy.reactjs.org/docs/getting-started.html.

Ansible:https://docs.ansible.com/ansible/latest/getting_started/get_started_playbook.html?extIdCarryOver=true&sc_cid=701f2000001OH7YAAW.

docker: https://docs.docker.com/guides/workshop/02_our_app/.

Dockercompose: https://docs.docker.com/compose/gettingstarted/**.**

References

Amazon Web Services. (2024a). Amazon EC2. Retrieved from https://aws.amazon.com/ec2/

Amazon Web Services. (2024b). Amazon VPC. Retrieved from https://aws.amazon.com/vpc/

Amazon Web Services. (2024c). Internet Gateway. Retrieved from https://aws.amazon.com/vpc/features/internet-gateway/

Amazon Web Services. (2024d). Elastic Load Balancing. Retrieved from https://aws.amazon.com/elasticloadbalancing/

Amazon Web Services. (2024e). Security Groups. Retrieved from https://aws.amazon.com/ec2/security-groups/

Amazon Web Services. (2024f). Subnets. Retrieved from https://aws.amazon.com/vpc/subnets/

Chris. (2024). Iu architecture. Retrieved from https://app.diagrams.net/?src=about#G19_-AIxKP0JJH2qSwD8a4Jj6I8UmGg09q#%7B%22pageId%22%3A%220pvrcRmcpLezZoWpGFq1%22%7D

Docker. (2024). Docker-compose. Retrieved from https://docs.docker.com/compose/

Etiti27. (2024a). AWS provider [Screenshot]. Visual Studio Code.

Etiti27. (2024b). AWS Virtual Private Cloud [Screenshot]. Visual Studio Code.

Etiti7. (2024c). AWS Internet gateway [Screenshot]. Visual Studio Code.

Etiti27. (2024d). AWS route table [Screenshot]. Visual Studio Code.

Etiti27. (2024e). AWS subnets [Screenshot]. Visual Studio Code.

Etiti27. (2024f). AWS subnet association [Screenshot]. Visual Studio Code.

Etiti27. (2024g). AWS Security group [Screenshot]. Visual Studio Code.

Etiti27. (2024h). AWS EC2 [Screenshot]. Visual Studio Code.

Etiti27. (2024i). AWS Autoscaling group [Screenshot]. Visual Studio Code.

Etiti27. (2024j). AWS Loadbalancer [Screenshot]. Visual Studio Code.

Etiti27. (2024k). AWS Output [Screenshot]. Visual Studio Code.

Etiti27. (2024l). AWS Dockerfile frontend [Screenshot]. Visual Studio Code.

Etiti27. (2024m). AWS Dockerfile backend [Screenshot]. Visual Studio Code.

Etiti27. (2024n). AWS Docker compose yaml [Screenshot]. Visual Studio Code.

Etiti27. (2024o). AWS Ansible inventory file [Screenshot]. Visual Studio Code.

Etiti27. (2024p). AWS Ansible playbook yaml [Screenshot]. Visual Studio Code.

HashiCorp. (2024). Terraform. Retrieved from https://www.terraform.io/

Red Hat. (2024). Ansible. Retrieved from https://www.ansible.com/

Synergy Research Group. (2024). *Cloud infrastructure services market share*. Retrieved from https://www.srgresearch.com/