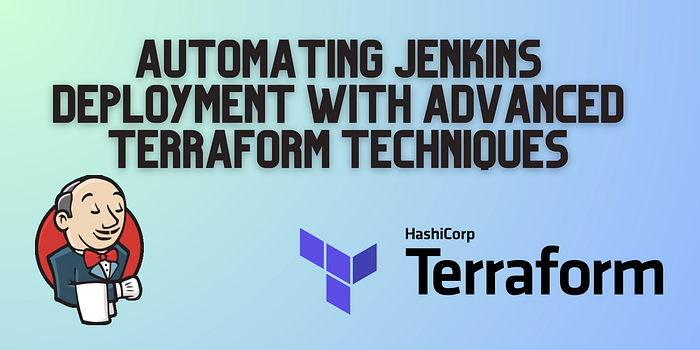
**Automating Jenkins Deployment with Advanced Terraform Techniques**



***TASKS***

**FOUNDATIONAL**

* *Your team would like to start using Jenkins as their CI/CD tool to create pipelines for DevOps projects. They need you to create the Jenkins server using Terraform so that it can be used in other environments and so that changes to the environment are better tracked.*
* *For the Foundational project you are allowed to have all your code in a single* ***main.tf*** *file (known as a monolith) with hardcoded data.*
* *Include the link to your repo in your documentation.*
* *Deploy 1 EC2 Instances in your Default VPC.  
  Bootstrap the EC2 instance with a script that will install and start Jenkins.*
* *Create and assign a Security Group to the Jenkins EC2 that allows traffic on port 22 from your ip and allows traffic from port 8080.*
* *Create a S3 bucket for your Jenkins Artifacts that is not open to the public.*
* *Verify that you can reach your Jenkins install via port 8080 in your browser. Be sure to include a screenshot of the Jenkins login screen in your documentation.*
* *Push your code to GitHub and include the link in your write up.*

**ADVANCED**

* *Add a variables.tf file and make sure nothing is hardcoded in your* ***main.tf***
* *Create separate file for your* ***providers.tf*** *if you have not already.*

**COMPLEX**

* *Create an IAM Role that allows S3 read/write access for the Jenkins Server and assign that role to your Jenkins Server EC2 instance.*
* *You can confirm this by sshing into your instance and without using your credentials test some S3 AWS CLI commands.*

**FOUNDATIONAL**

For the Foundational project you are allowed to have all your code in a single **main.tf** file (known as a monolith) with hardcoded data.

Include the link to your repo in your documentation.

We will first create a main.tf file, and then modify it once we have our EC2 running to build the project from there.

Our assigned task allows us to have all of our code in a single **‘main.tf’** file (known as a monolith) with hardcoded data.

With more complex files, it is common to split the code into multiple files and use modules for better organization and reusability. However we will keep everything in one file for simplicity.

The term **monolith** describes a single, large file that contains all the code, as opposed to a modular approach where it is separated into smaller more manageable pieces

The **“hardcoded data”** aspect of our instructions means we can directly include values in our **‘main.tf’** file instead of using variables or other means of parameterization. In more advanced setups, we would use variables or other techniques to make the configuration more dynamic and reusable.

* While it’s generally good practice to use variables for values that might change, we will stick with the monolith

**Install Terraform**

First we must install Terraform, if we do not already have it installed. You can follow the directions on the [Terraform documentation](https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli) page depending on your OS.

For this project we will be using AWS Cloud9 with an Amazon Linux OS, which comes pre installed with Terraform.

**Create a new directory**

We will create a new directory for our project and navigate to it in the terminal. You can do so with the following code:

mkdir week20-terraform  
cd week20-terraform



**Create the main.tf file**

We will create a file named **‘main.tf’** in the project directory, and use this file to build our code.



touch main.tf

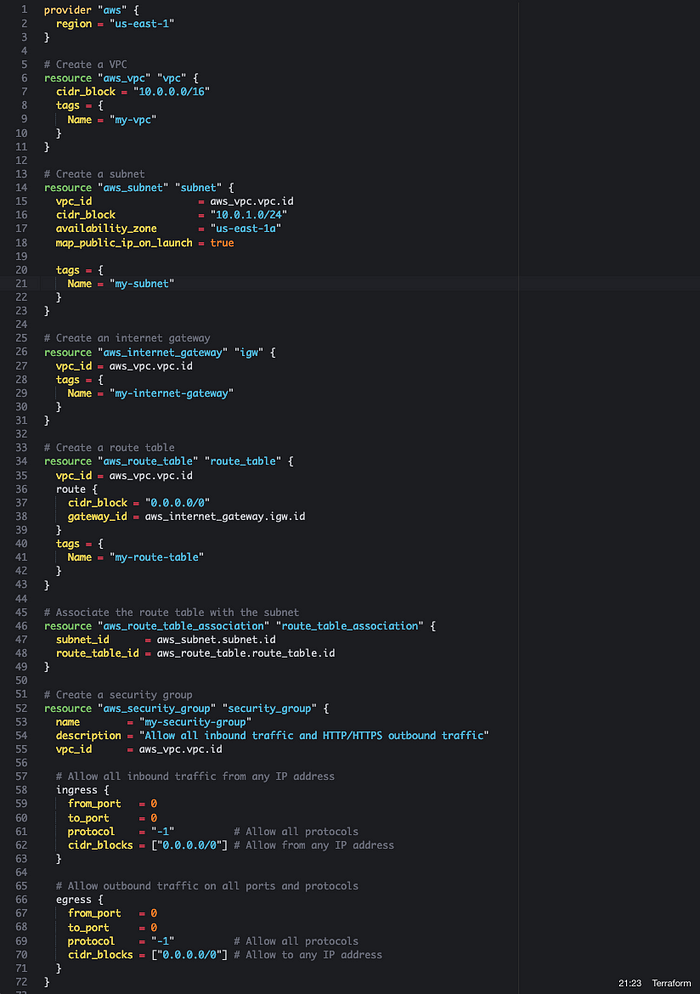
For this project we will need to establish a VPC, a subnet, an internet gateway, a route table, a security group, and an EC2 instance

We will need to also consider the correct order in which we will need to build these resources, by examining the dependencies of each resource.

* We will build in this order: *AWS provider, region and availability zone, VPC, subnet, internet gateway, route table, route table association, security group, and EC2 instance.*

We are going to create a **‘main.tf’** file, and break it down with explanations below:

provider "aws" {  
 region = "us-east-1"  
}  
  
# Create a VPC  
resource "aws\_vpc" "vpc" {  
 cidr\_block = "10.0.0.0/16"  
 tags = {  
 Name = "my-vpc"  
 }  
}  
  
# Create a subnet  
resource "aws\_subnet" "subnet" {  
 vpc\_id = aws\_vpc.vpc.id  
 cidr\_block = "10.0.1.0/24"  
 availability\_zone = "us-east-1a"  
 map\_public\_ip\_on\_launch = true  
  
 tags = {  
 Name = "my-subnet"  
 }  
}  
  
# Create an internet gateway  
resource "aws\_internet\_gateway" "igw" {  
 vpc\_id = aws\_vpc.vpc.id  
 tags = {  
 Name = "my-internet-gateway"  
 }  
}  
  
# Create a route table  
resource "aws\_route\_table" "route\_table" {  
 vpc\_id = aws\_vpc.vpc.id  
 route {  
 cidr\_block = "0.0.0.0/0"  
 gateway\_id = aws\_internet\_gateway.igw.id  
 }  
 tags = {  
 Name = "my-route-table"  
 }  
}  
  
# Associate the route table with the subnet  
resource "aws\_route\_table\_association" "route\_table\_association" {  
 subnet\_id = aws\_subnet.subnet.id  
 route\_table\_id = aws\_route\_table.route\_table.id  
}  
  
# Create a security group  
resource "aws\_security\_group" "security\_group" {  
 name = "my-security-group"  
 description = "Allow all inbound traffic and HTTP/HTTPS outbound traffic"  
 vpc\_id = aws\_vpc.vpc.id  
  
 # Allow all inbound traffic from any IP address  
 ingress {  
 from\_port = 0  
 to\_port = 0  
 protocol = "-1" # Allow all protocols  
 cidr\_blocks = ["0.0.0.0/0"] # Allow from any IP address  
 }  
  
 # Allow outbound traffic on all ports and protocols  
 egress {  
 from\_port = 0  
 to\_port = 0  
 protocol = "-1" # Allow all protocols  
 cidr\_blocks = ["0.0.0.0/0"] # Allow to any IP address  
 }  
}  
  
# Create an EC2 instance  
resource "aws\_instance" "instance" {  
 ami = "ami-00c39f71452c08778" # Amazon Linux 2023 AMI  
 instance\_type = "t2.micro"  
 subnet\_id = aws\_subnet.subnet.id  
 vpc\_security\_group\_ids = [aws\_security\_group.security\_group.id]  
 tags = {  
 Name = "my-instance"  
 }  
 user\_data = <<-EOF  
 #!/bin/bash  
 sudo yum -y update  
 sudo yum -y install httpd  
 sudo systemctl enable httpd  
 sudo systemctl start httpd  
 EOF  
}  
  
# Output the pulic IP address of the EC2 instance  
output "instance\_public\_ip" {  
 value = aws\_instance.instance.public\_ip  
 description = "The public IP address of the EC2 instance"  
}



main.tf

Text

Description automatically generated

main.tf (cont.)

**Configure the AWS provider:**

# Configure the AWS provider  
provider "aws" {  
 region = "us-east-1"  
}

* *The provider block in* ***main.tf*** *is used to configure the Terraform provider (AWS in our case). It sets the foundation for working with AWS resources. The provider manages the resources and API calls to the cloud provider*

**Create a VPC:**

# Create a VPC  
resource "aws\_vpc" "vpc" {  
 cidr\_block = "10.0.0.0/16"  
 tags = {  
 Name = "my-vpc"  
 }  
}

* *The VPC is the foundational networking component in AWS, and must be created before any subnets or other network resources. No dependencies exist for the VPC, but many resources to follow will depend on it.*

**Create a Subnet**:

# Create a subnet  
resource "aws\_subnet" "subnet" {  
 vpc\_id = aws\_vpc.vpc.id  
 cidr\_block = "10.0.1.0/24"  
 availability\_zone = "us-east-1a"  
 map\_public\_ip\_on\_launch = true  
  
 tags = {  
 Name = "my-subnet"  
 }  
}

* *The subnet depends on the VPC, as it needs to be associated with the VPD during creation. Setting the ‘vpc\_id’ attribute links the subnet to the VPC*

**Create an Internet Gateway:**

# Create an internet gateway  
resource "aws\_internet\_gateway" "igw" {  
 vpc\_id = aws\_vpc.vpc.id  
 tags = {  
 Name = "my-internet-gateway"  
 }  
}

* *This block creates an internet gateway and associates it with your VPC*

**Create a route table:**

# Create a route table  
resource "aws\_route\_table" "route\_table" {  
 vpc\_id = aws\_vpc.vpc.id  
 route {  
 cidr\_block = "0.0.0.0/0"  
 gateway\_id = aws\_internet\_gateway.igw.id  
 }  
 tags = {  
 Name = "my-route-table"  
 }  
}

* *This block creates a route table in your VPC, with a route that directs all traffic to the internet gateway we created*

**Associate the route table with the subnet:**

# Associate the route table with the subnet  
resource "aws\_route\_table\_association" "route\_table\_association" {  
 subnet\_id = aws\_subnet.subnet.id  
 route\_table\_id = aws\_route\_table.route\_table.id  
}

* *This block associates the route table with our subnet, enabling traffic from the subnet to be routed through the internet gateway*

**Create a security group:**

# Create a security group  
resource "aws\_security\_group" "security\_group" {  
 name = "my-security-group"  
 description = "Allow all inbound traffic and HTTP/HTTPS outbound traffic"  
 vpc\_id = aws\_vpc.vpc.id  
  
 # Allow all inbound traffic from any IP address  
 ingress {  
 from\_port = 0  
 to\_port = 0  
 protocol = "-1" # Allow all protocols  
 cidr\_blocks = ["0.0.0.0/0"] # Allow from any IP address  
 }  
  
 # Allow outbound traffic on all ports and protocols  
 egress {  
 from\_port = 0  
 to\_port = 0  
 protocol = "-1" # Allow all protocols  
 cidr\_blocks = ["0.0.0.0/0"] # Allow to any IP address  
 }  
}

* *This block creates a security group in the VPC, allowing all inbound traffic from any IP address*

**Create an EC2 instance:**

# Create an EC2 instance  
resource "aws\_instance" "instance" {  
 ami = "ami-00c39f71452c08778" # Amazon Linux 2023 AMI  
 instance\_type = "t2.micro"  
 subnet\_id = aws\_subnet.subnet.id  
 vpc\_security\_group\_ids = [aws\_security\_group.security\_group.id]  
 tags = {  
 Name = "my-instance"  
 }  
 user\_data = <<-EOF  
 #!/bin/bash  
 sudo yum -y update  
 sudo yum -y install httpd  
 sudo systemctl enable httpd  
 sudo systemctl start httpd  
 EOF  
}

* *This creates an EC2 instance using the Amazon Linux 2 AMI and the ‘t2 micro’ instance type. It associates the instance with the subnet and security group we created earlier.*
* *The ‘user\_data’ script will install apache*

Terraform output:

# Output the pulic IP address of the EC2 instance  
output "instance\_public\_ip" {  
 value = aws\_instance.instance.public\_ip  
 description = "The public IP address of the EC2 instance"  
}

* *This section will output our public IP of the instance so we dont need to go find it in the AWS console.*

***The code above, ‘main.tf,’ sets up a basic infrastructure with a VPC, subnet, internet gateway, route table, security group, and EC2 instance.***

**Initialize Terraform**

Fist make sure you are in the correct directory. Then we will run terraform fmt to make sure our file is formatted properly

terraform fmt

We can then run the following command to initialize terraform:

terraform init

This initializes the Terraform working directory, which is necessary before running any other terraform commands. It does several things:

* It will download the necessary provider plugins (i.e. the AWS provider)
* Sets up the backend for storing the Terraform state

Text

Description automatically generated

terraform init

Next we will validate to check if our code is valid, with **‘terraform validate’**:

terraform validate

If our code is valid, we will see a message telling us that our configuration is valid:

Text

Description automatically generated

terraform validate

Next we will review our execution plan with ‘terraform plan’ to review any changes that will be made to our infrastructure:

terraform plan

We are given the following output, which is a detailed overview of the resources that Terraform will create:

Text

Description automatically generated

terraform plan

Text

Description automatically generated

terraform plan 2

Text

Description automatically generated

terraform plan 3

If we are happy with how this looks, we will then apply the changes with **‘terraform apply’:**

Text

Description automatically generated

terraform apply

Text

Description automatically generated

terraform apply 2

Text

Description automatically generated

terraform apply 3

Text

Description automatically generated

terraform apply 4

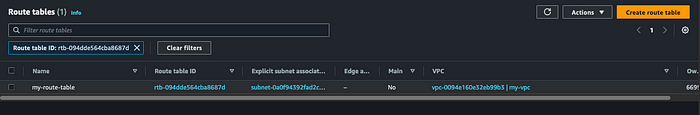
**Verify the resources**

After the ‘terraform apply’ command has successfully completed, we can verify the created resources in the AWS management console. We should see the VPC, subnet, internet gateway, route table security group, and EC2 instance that we defined in our Terraform code.

A screenshot of a computer

Description automatically generated with medium confidence

VPC



Route Table

A screenshot of a computer

Description automatically generated

Internet gateway

Graphical user interface, application

Description automatically generated

Subnet

A screenshot of a computer

Description automatically generated

Security Group

We will take our public IP and make sure it works:

Shape, rectangle

Description automatically generated

excellent

Before we move on to modifying this code for the next part of our project, we will **‘terraform destroy’** and break down this instance:

Text

Description automatically generated

terraform destroy

*You can find my code for the tasks up to this point in the project at the following link at my GitHub:* [*https://github.com/bsalvaggio/terraform/tree/main/week20-terraform/week20-basic*](https://github.com/bsalvaggio/terraform/tree/main/week20-terraform/week20-basic)

**ADVANCED**

**Modify our base main.tf code to install and start Jenkins**

We will now make several changes to our script now to allow for the installation of Jenkins on our EC2. The updated code is below with explanations of our modifications which allow us to proceed with our Jenkins install:

# Configure the AWS provider  
provider "aws" {  
 region = "us-east-1"  
}  
  
# Create a VPC  
resource "aws\_vpc" "vpc" {  
 cidr\_block = "10.0.0.0/16"  
 tags = {  
 Name = "my-vpc"  
 }  
}  
  
# Create a subnet  
resource "aws\_subnet" "subnet" {  
 vpc\_id = aws\_vpc.vpc.id  
 cidr\_block = "10.0.1.0/24"  
 availability\_zone = "us-east-1a"  
 map\_public\_ip\_on\_launch = true  
  
 tags = {  
 Name = "my-subnet"  
 }  
}  
  
# Create an internet gateway  
resource "aws\_internet\_gateway" "igw" {  
 vpc\_id = aws\_vpc.vpc.id  
 tags = {  
 Name = "my-internet-gateway"  
 }  
}  
  
# Create a route table  
resource "aws\_route\_table" "route\_table" {  
 vpc\_id = aws\_vpc.vpc.id  
 route {  
 cidr\_block = "0.0.0.0/0"  
 gateway\_id = aws\_internet\_gateway.igw.id  
 }  
 tags = {  
 Name = "my-route-table"  
 }  
}  
  
# Associate the route table with the subnet  
resource "aws\_route\_table\_association" "route\_table\_association" {  
 subnet\_id = aws\_subnet.subnet.id  
 route\_table\_id = aws\_route\_table.route\_table.id  
}  
  
# Create a security group for the Jenkins EC2 instance  
resource "aws\_security\_group" "jenkins\_security\_group" {  
 name = "jenkins-security-group"  
 description = "Security group for Jenkins EC2 instance"  
 vpc\_id = aws\_vpc.vpc.id  
  
 # Allow traffic on port 22 (SSH) from any IP address  
 ingress {  
 from\_port = 22  
 to\_port = 22  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
 # Allow traffic on port 8080 (Jenkins) from any IP address  
 ingress {  
 from\_port = 8080  
 to\_port = 8080  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
 ingress {  
 description = "http"  
 from\_port = 80  
 to\_port = 80  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
 # Allow outbound traffic on all ports and protocols  
 egress {  
 from\_port = 0  
 to\_port = 0  
 protocol = "-1" # Allow all protocols  
 cidr\_blocks = ["0.0.0.0/0"] # Allow to any IP address  
 }  
}  
  
# Create an EC2 instance  
resource "aws\_instance" "instance" {  
 ami = "ami-04581fbf744a7d11f" # Amazon Linux 2 AMI  
 instance\_type = "t2.micro"  
 subnet\_id = aws\_subnet.subnet.id  
 vpc\_security\_group\_ids = [aws\_security\_group.jenkins\_security\_group.id] # Use the new jenkins security group  
 tags = {  
 Name = "my-instance"  
 }  
  
 user\_data = file("jenkinsuserdata.sh")  
}  
  
# Create an S3 bucket for Jenkins artifacts  
resource "aws\_s3\_bucket" "jenkins\_artifacts" {  
 bucket = "billsjenkinsbucket"  
 tags = {  
 Name = "Bills Jenkins Artifacts Bucket"  
 }  
}  
resource "aws\_s3\_bucket\_acl" "jenkins\_artifacts\_acl" {  
 bucket = aws\_s3\_bucket.jenkins\_artifacts.id  
 acl = "private"  
}  
  
# Output the public IP address of the EC2 instance  
output "instance\_public\_ip" {  
 value = aws\_instance.instance.public\_ip  
 description = "The public IP address of the EC2 instance"  
}

Text

Description automatically generated

new main.tf file

**Security Group changes:**

We added an additional ingress rule to allow inbound traffic on ports 22 (SSH), 80 (HTTP), and 8080 (alternative to port 80 used for Jenkins) from any IP address and allows outbound traffic on all ports and protocols:

# Create a security group for the Jenkins EC2 instance  
resource "aws\_security\_group" "jenkins\_security\_group" {  
 name = "jenkins-security-group"  
 description = "Security group for Jenkins EC2 instance"  
 vpc\_id = aws\_vpc.vpc.id  
  
 # Allow traffic on port 22 (SSH) from any IP address  
 ingress {  
 from\_port = 22  
 to\_port = 22  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
 # Allow traffic on port 8080 (Jenkins) from any IP address  
 ingress {  
 from\_port = 8080  
 to\_port = 8080  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
 ingress {  
 description = "http"  
 from\_port = 80  
 to\_port = 80  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
 # Allow outbound traffic on all ports and protocols  
 egress {  
 from\_port = 0  
 to\_port = 0  
 protocol = "-1" # Allow all protocols  
 cidr\_blocks = ["0.0.0.0/0"] # Allow to any IP address  
 }  
}

**User Data:**

Our original user data script installed the Apache HTTP server. To install Jenkins in this part, we moved the user data script to a separate file, called **‘jenkinsuserdata.sh’**, and updated it to install the Jenkins server:

#!/bin/bash  
  
# Update the package index  
sudo yum update -y  
  
# Download the Jenkins repository configuration file and place it in the appropriate directory  
sudo wget -O /etc/yum.repos.d/jenkins.repo https://pkg.jenkins.io/redhat-stable/jenkins.repo  
  
# Import the Jenkins repository GPG key to ensure package integrity  
sudo rpm --import https://pkg.jenkins.io/redhat-stable/jenkins.io.key  
  
# Upgrade all installed packages to their latest versions  
sudo yum upgrade  
  
# Install Java OpenJDK 11  
sudo amazon-linux-extras install java-openjdk11 -y  
  
# Install Jenkins from the repository  
sudo yum install jenkins -y  
  
# Enable the Jenkins service to start automatically at boot  
sudo systemctl enable jenkins  
  
# Start the Jenkins service  
sudo systemctl start jenkins

Text

Description automatically generated

jenkinsuserdata.sh

We added code for a new S3 bucket as well for our Jenkins artifacts, and configured it with a private access control list to ensure that only authorized users can access it.

We create this S3 bucket to provide a centralized and easily accessible storage location for our artifacts. This makes it easier to collaborate with others, it allows us to keep multiple build outputs, have better security, and make our artifacts safely stored and available when we need them

# Create an S3 bucket for Jenkins artifacts  
resource "aws\_s3\_bucket" "jenkins\_artifacts" {  
 bucket = "billsjenkinsbucket"  
 tags = {  
 Name = "Bills Jenkins Artifacts Bucket"  
 }  
}  
resource "aws\_s3\_bucket\_acl" "jenkins\_artifacts\_acl" {  
 bucket = aws\_s3\_bucket.jenkins\_artifacts.id  
 acl = "private"  
}

* We set our bucket to “private,” which means by default only the IAM user who created it has access.

Once we have our new **main.tf** file ready to install jenkins, we can run **‘terraform plan’** and **‘terraform apply’** to apply the new terraform settings:

Text

Description automatically generated

terraform apply

Text

Description automatically generated

terraform apply

Text

Description automatically generated

terraform apply

Text

Description automatically generated

terraform apply

We are given the public IP as our output

Text

Description automatically generated

public ip output

We can then take the public IP and visit it at port 8080 in our web browser to find the Jenkins login screen:

Graphical user interface, text, application, email

Description automatically generated

Congratulations, we have successfully set up a Jenkins server using Terraform on AWS.

*You can find my code for this section of the project at the following link at my GitHub, including the updated* ***main.tf*** *and* ***jenkinsuserdata.sh*** *files:*

[*https://github.com/bsalvaggio/terraform/tree/main/week20-terraform/week20-jenkins*](https://github.com/bsalvaggio/terraform/tree/main/week20-terraform/week20-jenkins)

**ADVANCED**

In the next section of our project we will introduce variables to make our configuration more flexible. Variables are used to define customizable input values that can be passed to your Terraform configuration. This makes it easier to adapt the code for different environments or requirements without changing the underlying infrastructure code.

We will add a **variables.tf** file and make sure nothing is hardcoded in our **main.tf** file, as well as a **providers.tf** file.

**variables.tf:**

variable "aws\_region" {  
 description = "AWS region to deploy the infrastructure"  
 default = "us-east-1"  
}  
  
variable "vpc\_cidr\_block" {  
 description = "CIDR block for the VPC"  
 default = "10.0.0.0/16"  
}  
  
variable "subnet\_cidr\_block" {  
 description = "CIDR block for the subnet"  
 default = "10.0.1.0/24"  
}  
  
variable "auto\_ipv4" {  
 type = bool  
 description = "enable auto-assign ipv4"  
 default = true  
}  
  
variable "ami\_id" {  
 description = "AMI ID for the EC2 instance"  
 default = "ami-0fa1de1d60de6a97e" # Amazon Linux 2 AMI   
}  
  
variable "instance\_type" {  
 description = "Instance type for the EC2"  
 default = "t2.micro"  
}  
  
variable "bucket\_name" {  
 description = "S3 bucket name for Jenkins artifacts"  
 default = "billsjenkins-artifacts-bucketwk20"  
}  
  
variable "all\_traffic" {  
 description = "CIDR block for allowing all traffic"  
 default = "0.0.0.0/0"  
}  
  
variable "app\_name" {  
 description = "Name of the application"  
 default = "Jenkins"  
}

Text

Description automatically generated

variables.tf

* This variables.tf file contains our variable declarations with their names, types, and default values and descriptions

**providers.tf:**

provider "aws" {  
 region = var.aws\_region  
}

Graphical user interface

Description automatically generated with low confidence

providers.tf

**main.tf:**

# Create a VPC  
resource "aws\_vpc" "vpc" {  
 cidr\_block = var.vpc\_cidr\_block  
 tags = {  
 Terraform = "true"  
 Project = "Jenkins"  
 Name = "Jenkins VPC"  
 }  
}  
  
# Create a subnet within the VPC  
resource "aws\_subnet" "subnet" {  
 cidr\_block = var.subnet\_cidr\_block  
 vpc\_id = aws\_vpc.vpc.id  
 map\_public\_ip\_on\_launch = var.auto\_ipv4  
 tags = {  
 Name = "Jenkins Subnet"  
 }  
}  
  
# Create an Internet Gateway and attach it to the VPC  
resource "aws\_internet\_gateway" "igw" {  
 vpc\_id = aws\_vpc.vpc.id  
  
 tags = {  
 Name = "Jenkins Internet Gateway"  
 }  
}  
  
# Modify the default route table  
resource "aws\_default\_route\_table" "public\_route\_table" {  
 default\_route\_table\_id = aws\_vpc.vpc.default\_route\_table\_id  
  
 route {  
 cidr\_block = var.all\_traffic  
 gateway\_id = aws\_internet\_gateway.igw.id  
 }  
 tags = {  
 Name = "${var.app\_name}\_public\_rt"  
 Terraform = "true"  
 }  
}  
  
# Create a security group for the Jenkins EC2 instance  
resource "aws\_security\_group" "jenkins\_security\_group" {  
 name = "jenkins-security-group"  
 description = "Security group for Jenkins EC2 instance"  
 vpc\_id = aws\_vpc.vpc.id  
  
 # Allow traffic on port 22 (SSH) from any IP address  
 ingress {  
 from\_port = 22  
 to\_port = 22  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
 # Allow traffic on port 8080 (Jenkins) from any IP address  
 ingress {  
 from\_port = 8080  
 to\_port = 8080  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
 # Allow traffic on port 80 (HTTP) from any IP address  
 ingress {  
 from\_port = 80  
 to\_port = 80  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
 # # Allow inbound traffic on all ports and protocols  
 # ingress {  
 # from\_port = 0  
 # to\_port = 0  
 # protocol = "-1"  
 # cidr\_blocks = ["0.0.0.0/0"]  
 # }  
  
 # Allow outbound traffic on all ports and protocols  
 egress {  
 from\_port = 0  
 to\_port = 0  
 protocol = "-1"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
}  
  
# Create an EC2 instance  
resource "aws\_instance" "instance" {  
 ami = var.ami\_id  
 instance\_type = var.instance\_type  
 subnet\_id = aws\_subnet.subnet.id  
 vpc\_security\_group\_ids = [aws\_security\_group.jenkins\_security\_group.id]  
  
 tags = {  
 Terraform = "true"  
 Project = "Jenkins"  
 Name = "my-instance"  
 }  
  
 user\_data = file("jenkinsuserdata.sh")  
}  
  
# Create an S3 bucket for Jenkins artifacts  
resource "aws\_s3\_bucket" "billsjenkins\_artifacts" {  
 bucket = var.bucket\_name  
 tags = {  
 Name = "${var.bucket\_name}\_bucket"  
 }  
}  
  
# Output the public IP of the instance  
output "instance\_public\_ip" {  
 value = aws\_instance.instance.public\_ip  
 description = "The public IP address of the EC2 instance"  
}

**Troubleshooting Intermission**

**NOTE:** I have left the below security group rule hidden, but in my main.tf file, to show that I used it while troubleshooting:

# # Allow inbound traffic on all ports and protocols  
 # ingress {  
 # from\_port = 0  
 # to\_port = 0  
 # protocol = "-1"  
 # cidr\_blocks = ["0.0.0.0/0"]  
 # }

For a while, I was having difficulty getting my Jenkins page to load. I could not figure out why it wasn’t working and spent hours going through each variable and every part of my main.tf, variables, and userdata file . One of the things i tried was to open all ports to all incoming traffic to see if it was a port issue.

You will also notice some echo commands in my jenkinsuserdata.sh file. This was another attempt to troubleshoot where my problem was happening.

I later figured out that Jenkins was not successfully installing because it was not properly downloading the key from the URL. I learned this by connecting to the EC2 and watching and working in the terminal to check on the install progress and attempt to run commands in the instance.I then replaced the original URL in this line with a new URL for the Jenkins public key, and observed the EC2 properly download and import the Jenkins public key, which allowed the installation of Jenkins to proceed without any further issues.



Text

Description automatically generated

output from terraform apply

Graphical user interface, text, application, email

Description automatically generated

*You can find my code for the tasks up to this point in the project at the following link at my GitHub:* <https://github.com/bsalvaggio/terraform/tree/main/week20-terraform/week-20-advanced>

**Complex Tasks:**

**Create an IAM Role that allows S3 read/write access for the Jenkins Server and assign that role to your Jenkins Server EC2 instance. 2.**

**You can confirm this by sshing into your instance and without using your credentials test some S3 AWS CLI commands.**

**Modify our main.tf file:**

# Create a VPC  
resource "aws\_vpc" "vpc" {  
 cidr\_block = var.vpc\_cidr\_block  
 tags = {  
 Terraform = "true"  
 Project = "Jenkins"  
 Name = "Jenkins VPC"  
 }  
}  
  
# Create a subnet within the VPC  
resource "aws\_subnet" "subnet" {  
 cidr\_block = var.subnet\_cidr\_block  
 vpc\_id = aws\_vpc.vpc.id  
 map\_public\_ip\_on\_launch = var.auto\_ipv4  
 tags = {  
 Name = "Jenkins Subnet"  
 }  
}  
  
# Create an Internet Gateway and attach it to the VPC  
resource "aws\_internet\_gateway" "igw" {  
 vpc\_id = aws\_vpc.vpc.id  
  
 tags = {  
 Name = "Jenkins Internet Gateway"  
 }  
}  
  
# Modify the default route table  
resource "aws\_default\_route\_table" "public\_route\_table" {  
 default\_route\_table\_id = aws\_vpc.vpc.default\_route\_table\_id  
  
 route {  
 cidr\_block = var.all\_traffic  
 gateway\_id = aws\_internet\_gateway.igw.id  
 }  
 tags = {  
 Name = "${var.app\_name}\_public\_rt"  
 Terraform = "true"  
 }  
}  
  
# Create a security group for the Jenkins EC2 instance  
resource "aws\_security\_group" "jenkins\_security\_group" {  
 name = "jenkins-security-group"  
 description = "Security group for Jenkins EC2 instance"  
 vpc\_id = aws\_vpc.vpc.id  
  
 # Allow traffic on port 22 (SSH) from any IP address  
 ingress {  
 from\_port = 22  
 to\_port = 22  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
 # Allow traffic on port 8080 (Jenkins) from any IP address  
 ingress {  
 from\_port = 8080  
 to\_port = 8080  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
 # Allow traffic on port 80 (HTTP) from any IP address  
 ingress {  
 from\_port = 80  
 to\_port = 80  
 protocol = "tcp"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
  
 # Allow outbound traffic on all ports and protocols  
 egress {  
 from\_port = 0  
 to\_port = 0  
 protocol = "-1"  
 cidr\_blocks = ["0.0.0.0/0"]  
 }  
}  
  
# Create an IAM policy for S3 read/write access  
resource "aws\_iam\_policy" "s3\_policy" {  
 name = "S3ReadWriteAccess"  
 description = "Policy for Jenkins server to access S3"  
  
 policy = jsonencode({  
 Version = "2012-10-17"  
 Statement = [  
 {  
 Action = [  
 "s3:ListBucket",  
 "s3:GetObject",  
 "s3:PutObject",  
 "s3:DeleteObject"  
 ]  
 Effect = "Allow"  
 Resource = ["arn:aws:s3:::${var.bucket\_name}", "arn:aws:s3:::${var.bucket\_name}/\*"]  
 }  
 ]  
 })  
}  
  
# Create an IAM role for the Jenkins EC2 instance  
resource "aws\_iam\_role" "jenkins\_instance\_role" {  
 name = "JenkinsInstanceRole"  
  
 assume\_role\_policy = jsonencode({  
 Version = "2012-10-17"  
 Statement = [  
 {  
 Action = "sts:AssumeRole"  
 Effect = "Allow"  
 Principal = {  
 Service = "ec2.amazonaws.com"  
 }  
 }  
 ]  
 })  
}  
  
# Attach the S3 policy to the Jenkins EC2 instance role  
resource "aws\_iam\_role\_policy\_attachment" "jenkins\_s3\_policy\_attachment" {  
 policy\_arn = aws\_iam\_policy.s3\_policy.arn  
 role = aws\_iam\_role.jenkins\_instance\_role.name  
}  
  
# Create an IAM instance profile for the Jenkins EC2 instance  
resource "aws\_iam\_instance\_profile" "jenkins\_instance\_profile" {  
 name = "JenkinsInstanceProfile"  
 role = aws\_iam\_role.jenkins\_instance\_role.name  
}  
  
# Create an EC2 instance  
resource "aws\_instance" "instance" {  
 ami = var.ami\_id  
 instance\_type = var.instance\_type  
 subnet\_id = aws\_subnet.subnet.id  
 vpc\_security\_group\_ids = [aws\_security\_group.jenkins\_security\_group.id]  
 iam\_instance\_profile = aws\_iam\_instance\_profile.jenkins\_instance\_profile.name  
  
 tags = {  
 Terraform = "true"  
 Project = "Jenkins"  
 Name = "my-instance"  
 }  
  
 user\_data = file("jenkinsuserdata.sh")  
}  
  
# Create an S3 bucket for Jenkins artifacts  
resource "aws\_s3\_bucket" "billsjenkins\_artifacts" {  
 bucket = var.bucket\_name  
 tags = {  
 Name = "${var.bucket\_name}\_bucket"  
 }  
}  
  
# Output the public IP of the instance  
output "instance\_public\_ip" {  
 value = aws\_instance.instance.public\_ip  
 description = "The public IP address of the EC2 instance"  
}

We have updated our **main.tf** to include an **IAM policy**, an **IAM role**, the attachment of the policy to the role, and an **IAM instance profile** that we will need for the Jenkins server

**IAM policy:**

resource "aws\_iam\_policy" "s3\_policy" {  
 name = "jenkins-s3-policy"  
 description = "Allows Jenkins server to read and write to S3 buckets"  
 policy = jsonencode({  
 Version = "2012-10-17"  
 Statement = [  
 {  
 Action = [  
 "s3:GetObject",  
 "s3:PutObject",  
 "s3:ListBucket"  
 ]  
 Effect = "Allow"  
 Resource = "\*"  
 }  
 ]  
 })  
}

* This IAM policy grants read and write access to S3. We will attach this to the IAM role and then associate it with the Jenkins EC2 instance

**IAM role:**

* We will create an IAM role using the ‘**aws\_iam\_role’** resource in Terraform configuration. We set the ‘assume role policy’ to allow EC2 instances to assume the role.

resource "aws\_iam\_role" "jenkins\_role" {  
 name = "jenkins-role"  
 assume\_role\_policy = jsonencode({  
 Version = "2012-10-17"  
 Statement = [  
 {  
 Action = "sts:AssumeRole"  
 Effect = "Allow"  
 Principal = {  
 Service = "ec2.amazonaws.com"  
 }  
 }  
 ]  
 })  
}

Attach the policy to the role:

resource "aws\_iam\_role\_policy\_attachment" "jenkins\_role\_policy\_attachment" {  
 policy\_arn = aws\_iam\_policy.s3\_policy.arn  
 role = aws\_iam\_role.jenkins\_role.name  
}

**Create an IAM instance** profile and associate it with the IAM role:

resource "aws\_iam\_instance\_profile" "jenkins\_instance\_profile" {  
 name = "jenkins-instance-profile"  
 role = aws\_iam\_role.jenkins\_role.name  
}

Associate the IAM instance profile with the Jenkins EC2 instance:

resource "aws\_instance" "instance" {  
 ...  
 iam\_instance\_profile = aws\_iam\_instance\_profile.jenkins\_instance\_profile.name  
 ...  
}

With these changes, we have created the necessary IAM resources and also granted the Jenkins server with permissions to interact with the S3 bucket

We will then run our usual ‘**terraform init**,’ ‘**terraform plan**,’ and “**terraform apply**’ :

Text

Description automatically generated

Once successful we will be presented with the public IP as shown in the picture above.

**Testing**

We will connect to our instance via Amazon Instance Connect. This will allow us to have temporary access so that we can verify the IAM role and permissions using the AWS CLI. EC2 Instance Connect lets us connect to our without having to manage SSH keys

Text

Description automatically generated

We will use typical commands to test our read and write privileges such as:

listing the buckets in our account

aws s3 ls

Creating a test file on our EC2 instance:

echo "This is a test file" > test.txt

and then uploading that file to our S3 bucket:

aws s3 cp test.txt s3://<your-bucket-name>/test.txt

Downloading the test file from our S3 bucket to a different file name:

aws s3 cp s3://<your-bucket-name>/test.txt downloaded\_test.txt

and deleting the test file from our S3 bucket:

aws s3 rm s3://<your-bucket-name>/test.txt

As you can see we were able to perform these commands without any issues, which confirms that our EC2 instance has the necessary IAM crole and permissions to access our bucket:

Text

Description automatically generated

**Conclusion:**

In this week’s project, we showcased the power and flexibility of Terraform as an infrastructure as Code tool.

Some of the tasks we accomplished:

* Set up an AWS environment using Terraform, including creating a VPC, subnets, security groups, and an EC2 instance.
* Automated the installation and configuration of Jenkins on the EC2 instance using a user data script
* Used Terraform variables to make the infrastructure code more flexible and reusable
* Set up an S3 bucket to store Jenkins artifacts, ensuring that we have a centralized and durable storage for these files
* Overcame Jenkins install installation issues by trial and error and eventually pinpointing our issue by connecting to our EC2 and manually installing Jenkins line by line until we figured out that it was the key URL. By replacing this with a new URL, we solved our problem. Through troubleshooting however, I tried many different configurations and changed many parts of all our files, which led to a much better understanding of all aspects of this project.
* Created an IAM role and policy for the Jenkins EC2
* Attached the IAM policy to the role
* Tested our setup by connecting to the instance and verifying access to S3 without using credentials.

This project has many real world implications, as Infrastructure as Code tools such as Terraform can provide businesses with many benefits and improve overall efficiency.

* Terraform allows businesses to quickly deploy infrastructure for new projects, or scale existing ones. This can reduce time-to-market for new products and services
* By codifying infrastructure in a version-controlled repository, businesses can maintain consistent configurations across multiple environments
* Businesses can optimize resources and automate the provisioning and of resources. This can lead to significant cost savings by avoiding over-provisioning as well as minimizing human error.
* Tools like Terraform can enable teams to work together more effectively, as they can review and collaborate on infrastructure code. This allows for better communication, reduces the risk of misconfigurations, and ensures that everyone has a clear understanding of the infrastructure.
* As was demonstrated in the complex portion, IaC allows businesses to define and enforce security policies and compliance requirements in the code itself. This ensures that every deployment adheres to the necessary standards, which reduces the risk of security breaches.

*You can find my code for the complex portion of the project at the following link at my GitHub:* <https://github.com/bsalvaggio/terraform/tree/main/week20-terraform/week20-complex>

**Thank you for reading!**