



HashiCorp

Terraform



Deploy a sample web application in AWS using Terraform

TASK

Steps required for building infrastructure and deploying our application according to the below Scenario/Use-case using Terraform

1. Create the key and security group which allows the port 80 .
2. Launch the EC2 instance.
3. In this EC2 instance use the key and security group which we have created in step 1.
4. Launch one Volume (EBS) and mount that volume into /var/www/html
5. Developer has uploaded the code into github repo which also contains images.
6. Copy the github repo code into /var/www/html
7. Create S3 bucket, and copy/deploy the images from github repo into the S3 bucket and change the permission to public readable.
8. Launch your webpage

Create an AWS account & credentials

First, we need to sign up for an AWS account. Although you can pick the “Free” account, because of the resources we will be using there will still be a charge. You will want to tear down the infrastructure once you’ve finished and I’ll show you how to do that in this blog.

Once you have your AWS account, sign in as your root user. The email that you used to sign up is your username, but make sure you click **login as root account**. Once signed in, select **IAM** (Identity Access and Management) from the **Services** menu.

Select **Users** from the nav bar on the left side of the screen and click **Add User**. Create a user and make sure you give it both programmatic and console access.

Add user



Set user details

You can add multiple users at once with the same access type and permissions. [Learn more](#)

User name*

[+ Add another user](#)

Select AWS access type

Select how these users will access AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Access type* ☒

Programmatic access

Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other development tools.

☒

AWS Management Console access

Enables a **password** that allows users to sign-in to the AWS Management Console.

Console password*

☒

Autogenerated password

☐

Custom password

Require password reset ☒

☒

User must create a new password at next sign-in.


Users automatically get the [IAMUserChangePassword](#) policy to allow them to change their own password.


Give the user Administrator Access either by creating a new role or by attaching the existing policy directly.


Add user

1 2 3 4 5

Set permissions

 Add user to group

 Copy permissions from existing user

 Attach existing policies directly

Create policy


Filter policies Search Showing 424 results

	Policy name	Type	Used as	Description
<input checked="" type="checkbox"/>	AdministratorAccess	Job function	None	Provides full access to AWS services and r...
<input type="checkbox"/>	AlexaForBusinessDe...	AWS managed	None	Provide device setup access to AlexaForB...
<input type="checkbox"/>	AlexaForBusinessFul...	AWS managed	None	Grants full access to AlexaForBusiness res...
<input type="checkbox"/>	AlexaForBusinessGa...	AWS managed	None	Provide gateway execution access to Alex...
<input type="checkbox"/>	AlexaForBusinessRe...	AWS managed	None	Provide read only access to AlexaForBusin...
<input type="checkbox"/>	AmazonAPIGateway...	AWS managed	None	Provides full access to create/edit/delete ...
<input type="checkbox"/>	AmazonAPIGatewayl...	AWS managed	None	Provides full access to invoke APIs in Ama...
<input type="checkbox"/>	AmazonAPIGateway...	AWS managed	None	Allows API Gateway to push logs to user's ...

You can keep the defaults and accept everything else in the setup wizard, but make sure you take careful note of the `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY` before you leave the Add user Success screen.



Add user

1 2 3 4 5

 **Success**
You successfully created the users shown below. You can view and download user security credentials. You can also email users instructions for signing in to the AWS Management Console. This is the last time these credentials will be available to download. However, you can create new credentials at any time.

Users with AWS Management Console access can sign-in at: <https://731430262381.signin.aws.amazon.com/console>

Download .csv

	User	Access key ID	Secret access key	Password	Email login instructions
	filmappuser	AKIAJRYJTJELES2VG7JA	***** Show	***** Show	Send email 

1.Create the key and security group which allows the port 80 .

- Create Key Pair:** In the EC2 dashboard, generate a new `.pem` key pair and save it.
- Move to Working Directory:** Place the downloaded `.pem` file in the directory where you're working with your Terraform files.

- *Create a security group which allows the port 80 (optional)*

2. Software Requirements

- **AWS CLIv2**

To download :-

<https://awscli.amazonaws.com/AWSCLIV2.msi>

- **Terraform**

To download:-

https://releases.hashicorp.com/terraform/1.5.6/terraform_1.5.6_windows_386.zip

- **VS Code**

To download:-

<https://code.visualstudio.com/download>

By adding these installation paths to your system's PATH environment variable, you'll be able to use the AWS CLI and Terraform commands without needing to specify their full paths each time.

3. Create S3 bucket, and copy/deploy the images from github repo into the S3 bucket and change the permission to public readable.

Step 1: Upload Images (if not done already)

1. Inside your S3 bucket, click the "Upload" button.
2. Select the images you want to upload from your local machine.
3. Follow the prompts to complete the upload.

Step 2: Change Object Permissions to Read-Only

1. Inside your S3 bucket, select the uploaded image objects.
2. Click on the "Actions" button, then select "Change permissions."
3. In the "Manage public permissions" dialog:
 - Choose "Grant public read access to this object(s)."
 - This will allow anyone to read the objects but not modify them.
 - Click "Save."

4. Code for building infrastructure

```
provider "aws" {
  region = "ap-south-1" # Set the AWS region
}

# *****
# Define an AWS security group
resource "aws_security_group" "allow_http_ssh" {
  name        = "allow_http"
  description = "Allow http inbound traffic"
  ingress {
    description = "http"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
  }
}
```

```

    cidr_blocks = ["0.0.0.0/0"]
}
ingress {
    description = "ssh"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
}
egress {
    from_port = 0
    to_port   = 0
    protocol  = "-1"
    cidr_blocks = ["0.0.0.0/0"]
}
tags = {
    Name = "allow_http_ssh"
}
}
# *****

# Define an AWS EC2 instance
resource "aws_instance" "web" {
    ami            = "ami-06f621d90fa29f6d0" # Specify the AMI ID
    instance_type = "t2.micro"
    key_name       = "27_firstkey" # Specify the key pair
    security_groups = ["launch-wizard-1"] # Associate security groups

    connection {
        type      = "ssh"
        user       = "ec2-user"
        private_key = file("27_firstkey.pem") # Provide private key path
        host       = aws_instance.web.public_ip
    }

    provisioner "remote-exec" {
        inline = [
            "sudo yum install httpd php git -y",      # Install necessary packages
            "sudo systemctl restart httpd",           # Restart the web server
            "sudo systemctl enable httpd",             # Enable the web server on boot
        ]
    }

    tags = {

```

```

    Name = "lws1"
  }
}

# Define an AWS EBS volume
resource "aws_ebs_volume" "esb1" {
  availability_zone = aws_instance.web.availability_zone
  size              = 1
  tags = {
    Name = "lwebs"
  }
}

# Attach the EBS volume to the EC2 instance
resource "aws_volume_attachment" "esb_att" {
  device_name = "/dev/sdh"
  volume_id   = aws_ebs_volume.esb1.id
  instance_id = aws_instance.web.id
  force_detach = true
}

# Define an output for the public IP of the instance
output "myos_ip" {
  value = aws_instance.web.public_ip
}

# Define a null resource for local execution
resource "null_resource" "nulllocal2" {
  provisioner "local-exec" {
    command = "echo ${aws_instance.web.public_ip} > publicip.txt" # Store public
IP in a file
  }
}

# Define another null resource for remote execution
resource "null_resource" "nullremote3" {
  depends_on = [
    aws_volume_attachment.esb_att,
  ]

  connection {
    type      = "ssh"
    user      = "ec2-user"
    private_key = file("27_firstkey.pem") # Provide private key path
    host      = aws_instance.web.public_ip
  }
}

```

```

}

provisioner "remote-exec" {
  inline = [
    "sudo mkfs.ext4 /dev/xvdx",          # Format the attached EBS volume
    "sudo mount /dev/xvdx /var/www/html", # Mount the volume to
/var/www/html
    "sudo rm -rf /var/www/html/*",        # Remove existing content
    "sudo git clone https://github.com/Rizwantech5/tf-web-project.git
/var/www/html/" # Clone a GitHub repository
  ]
}
}

# Define one more null resource for local execution
resource "null_resource" "nulllocal1" {
  depends_on = [
    null_resource.nullremote3,
  ]

  provisioner "local-exec" {
    command = "start chrome ${aws_instance.web.public_ip}" # Open the instance's
public IP in Chrome
  }
}
}

```

For complete/detailed code refer to this link:-

<https://github.com/Rizwantech5/tf-web-project.git>

5.How to run the code !!

Go to your current directory --> where your xyz.tf extension file is saved in your system

Open the file which contains the terraform code in VS code, click on "terminal". Once the terminal opens, run the following commands:

1. terraform init
2. terraform validate
3. terraform plan
4. terraform apply
5. terraform destroy

To configure your AWS account, we will run the "aws configure" command