Tutorial – AWS Infrastructure and Bookstack Set Up Using **Terraform** and **Ansible**

Tamim Hemat
A01278451

In this tutorial I will walk you through the process of setting up an AWS VPC, EC2 Instance, and RDS MySQL Database along with a static HTML page (instead of a Bookstack instance) through the EC2 Instance – all this using Terraform and Ansible.

First, I am going to list the components of the AWS Infrastructure:

- VPC
- Subnets
  - 1 public subnet for the EC2 Instance
  - 2 private subnets for our RDS database
- Internet Gateway
- Route table
- Security groups
  - 1 for the EC2 Instance, it will allow ssh and http traffic from anywhere
  - 1 for the RDS database, it will allow MySQL traffic from the VPC
- SSH key pair
- EC2 Instance
  - Ubuntu 22.04
- Subnet group
  - It will include the 2 private subnets and will be used for the database
- RDS database
  - MySQL

Our infrastructure will be created by a single "**main.tf**" file. The file will also include a block that uses a Terraform Cloud Workspace. After explaining the infrastructure I'll also explain how to set up Terraform Cloud.

Below I will show and explain all the code blocks for creating the AWS infrastructure:

```
terraform {
  cloud {
    organization = "tamim_hemat"

    workspaces {
      name = "4640-Terraform"
    }
  }

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }

  required_version = ">= 1.2.0"
}
```

The **terraform** block is used to define variables that will be used by Terraform to execute the code within the file.

The **cloud** block specifies information related to Terraform Cloud. In this block, we set the organization and workspace that we want to use. This block only works if you have created a Terraform Cloud account and have logged in with the CLI.

The **required_providers** block specifies the cloud provider(s) to be used for our infrastructure. In this example, we are using **aws**, with the source being "hashicorp/aws" and the version being "~> 4.16". The source specifies where to download the provider from, and the version specifies the version constraint for the provider. In this case, the version constraint is set to any version greater than or equal to 4.16, but less than version 5.

The **required_version** variable specifies the minimum version of Terraform required to run this code. In this case, the minimum required version is 1.2.0.

```
provider "aws" {
  region = "us-west-2"
}
```

The **provider** block specifies the cloud provider to be used for our infrastructure. We are using **aws**.

The **region** variable specifies the AWS region in which to create the resources. We will be using "us-west-2" – the Oregon Region.

```
variable "base_cidr_block" {
  description = "default cidr block for vpc"
  default     = "10.0.0.0/16"
}
```

Variable blocks are used to define variables that can be used throughout the Terraform code. Here we are defining a variable called "**base_cidr_block**" that will be used to set the CIDR block for a VPC (Virtual Private Cloud).
The **description** parameter is used to describe the purpose of the variable.
The **default** parameter is used to specify the default value for the variable. We are going to use "10.0.0.0/16" as our default value. If no value is provided for this variable during runtime, Terraform will use the default value specified here.

```
variable "subnet1_cidr_block" {
  description = "default cidr block for subnet1"
  default     = "10.0.1.0/24"
}

variable "subnet2_cidr_block" {
  description = "default cidr block for subnet2"
  default     = "10.0.2.0/24"
}

variable "subnet3_cidr_block" {
  description = "default cidr block for subnet3"
  default     = "10.0.3.0/24"
}
```

Here we are defining three **variables** that will be used to store the CIDR blocks for the three subnets that we'll create in the VPC.
Each variable block has a name (e.g. "subnet1_cidr_block"), a description (e.g. "default cidr block for subnet1"), and a default value (e.g. "10.0.1.0/24").
The format is the same for all three with only the values slightly differing.

```
resource "aws_vpc" "main" {
  cidr_block       = var.base_cidr_block
  instance_tenancy = "default"

  tags = {
    Name = "acit-4640-vpc"
  }
}
```

Resource blocks are used to declare and configure infrastructure resources.
In this block, we declare an AWS VPC **resource** with the resource type **aws_vpc**. The resource is identified by the name **main**, which is used to reference the resource elsewhere in the code.
The **cidr_block** attribute specifies the IP range for the VPC. In this example, we set it to the value of the **base_cidr_block variable**, which we already defined.
When using values from variables the format is: **<var>.<variable name>**
The **instance_tenancy** specifies the tenancy of instances launched in the VPC.

The **tags** attribute specifies metadata for the resource in the form of key-value pairs. Here we set the Name tag to "acit-4640-vpc".

```
resource "aws_subnet" "sub1" {
  vpc_id                  = aws_vpc.main.id
  cidr_block              = var.subnet1_cidr_block
  availability_zone       = "us-west-2a"
  map_public_ip_on_launch = true

  tags = {
    Name = "acit-4640-pub-sub"
  }
}
```

The **aws_subnet** resource creates a subnet. It creates it in the VPC specified by **aws_vpc.main.id** which queries the **id** attribute of our previously created **aws_vpc** with a name **main**.
Querying attributes from already created resources is used a lot in the code so here is the format for querying: **<resource_type>.<resource_name>.<attribute>**
This subnet is created with the name "sub1". The **cidr_block** variable is set to the variable for the first subnet that we already defined. The **availability_zone** specifies the AWS availability zone in which to create the subnet. This subnet is in "us-west-2a". The **map_public_ip_on_launch** attribute is set to **true**, which means that instances launched in this subnet will be automatically assigned a public IP address.

```
resource "aws_subnet" "sub2" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = var.subnet2_cidr_block
  availability_zone = "us-west-2a"

  tags = {
    Name = "acit-4640-rds-sub1"
  }
}

resource "aws_subnet" "sub3" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = var.subnet3_cidr_block
  availability_zone = "us-west-2b"

  tags = {
    Name = "acit-4640-rds-sub2"
  }
}
```

Above we define the other 2 subnets. The format is the same as the first one but here we don't set the **map_public_ip_on_launch** because we want them to be private.
We also need to change the names, **cidr_block** variables user respectively.
One last note is that we want one of the private subnets to be in the same AZ as our public subnet, but the other one needs to be in a different AZ. To achieve this we just specify "**us-west-2a**" for the first private subnet and "**us-west-2b**" for the second.

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

  tags = {
    Name = "acit-4640-igw"
  }
}
```

This **aws_internet_gateway** resource block creates an internet gateway for the VPC. This allows instances in the VPC to connect to the internet and vice versa.
The **vpc_id** argument specifies the ID of our VPC to which the internet gateway should be attached.

```
resource "aws_route_table" "rt" {
  vpc_id = aws_vpc.main.id

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

  tags = {
    Name = "acit-4640-rt"
  }
}
```

The **aws_route_table** resource creates a route table.
The **vpc_id** again references our VPC.
The **route** block specifies a routing rule for the route table. We are creating a default route to allow all traffic to flow through the Internet Gateway that we just created and we reference it the same way as our VPC.

```
resource "aws_route_table_association" "rta" {
  subnet_id      = aws_subnet.sub1.id
  route_table_id = aws_route_table.rt.id
}
```

This resource **associates a route table with a subnet**. We reference our route table and our public subnet (**sub1**). This is important to do, because if we don't do this then later our EC2 instance won't have access to the Internet even if in our public subnet.

```
resource "aws_security_group" "sg-ec2" {
  name        = "acit-4640-sg-ec2"
  description = "Allow SSH and HTTP traffic from anywhere"
  vpc_id      = aws_vpc.main.id

  ingress {
    description = "SSH from anywhere"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

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

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

This resource creates a security group for our EC2 instance. The **name** parameter is the name for the SG. The **description** parameter is optional but useful to give a short explanation of the SG. Again we need to reference our VPC.

The **ingress** block specifies an ingress security rule for the SG.

The parameters **from_port to_port** define the range of the rule (i.e. do we want to allow multiple ports, or just a single one). For out first ingress rule we allow SSH on port 22.

The **protocol** parameter sets the allowed protocol for the traffic.

The **cidr_blocks** parameter defines where traffic can come from. In our case it can come from anywhere (i.e. **0.0.0.0/0**).

The second ingress security rule is almost identical to the first, we just the from_port and to_port to 80 so we allow **HTTP** traffic.

The **egress** security rule defines outbound traffic allowed for the SG. It has the same parameters as the ingress rules. We want to allow all outbound traffic on all ports. To do so we set the from_port and to_port to 0 to allow all ports. We set the protocol to "**-1**" which means all protocols allowed. We also set the cidr_blocks to **0.0.0.0/0** to allow all destinations.

```
resource "aws_security_group" "sg-rds" {
  name        = "acit-4640-sg-rds"
  description = "Allow MySQL traffic from within the VPC"
  vpc_id      = aws_vpc.main.id

  ingress {
    description = "MySQL from within the VPC"
    from_port   = 3306
    to_port     = 3306
    protocol    = "tcp"
    cidr_blocks = [var.base_cidr_block]
  }

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

This code block also creates a security group, but for our MySQL database.
The format and parameters are the same as the previous security group. The difference is here we have **only 1 ingress** rule, which should allow traffic on port 3306 but only coming from within our VPC (which is referenced by our **base_cidr_block** variable).
The **egress** rule is the same as the EC2 instance SG (allows all traffic on all ports).

```
resource "aws_key_pair" "key" {
  key_name   = "acit-4640-key"
  public_key = <YOUR PUBLIC KEY HERE>
}
```

The above resource creates an AWS key pair. It only needs 2 parameters.
The **key_name** parameter specifies the name of the key.
The **public_key** parameter will contain the public key from the key pair.

For this to work we need to create a key pair in our Linux development environment.
The command to create an ED25519 key pair is:
ssh-keygen -t ed25519 -f acit-4640-key

This command uses the **ssh-keygen** utility to create a key pair.
The **-t** flag specifies the type of key we want to create
The **-f** flag specifies the name we want to use for out **.pub** and **.pem** files – so this command will create 2 files: "**acit-4640-key.pub**" and "**acit-4640-key.pem**".

After we create our key pair we can copy our public key and paste it into the **public_key** parameter of our **aws_key_pair** resource.
Then when we want to initialize SSH connections we are going to use the private key from the pair (i.e. the **.pem** file).

```
resource "aws_instance" "ec2" {
  ami                         = "ami-0735c191cf914754d"
  instance_type               = "t2.micro"
  key_name                    = aws_key_pair.key.key_name
  subnet_id                   = aws_subnet.sub1.id
  associate_public_ip_address = true
  vpc_security_group_ids      = [aws_security_group.sg-ec2.id]

  tags = {
    Name = "acit-4640-ec2"
  }
}
```

This block creates an EC2 instance. There are several parameters needed.
The **ami** parameter specifies the ID of the AMI image we want to use.

Note: I tried using a **data** block to get the latest Ubuntu 22.04 version, but even after many attempts with different code I was getting AMIs that were either invalid or expired, so I just copy pasted the latest Ubuntu 22.04 version from the AWS EC2 console.

The **instance_type** specifies, well, the type of instance we want. In this tutorial we will use "**t2.micro**" so we can use the free tier EC2 instance.
The **key_name** parameter specifies the key we want to use. Here we reference the key we created earlier.
The **subnet_id** parameter specifies the subnet we want our EC2 instance to use. Here we reference our public subnet (**sub1**).
The **associate_public_ip_address** parameter defines if our EC2 instance gets a public IP. Here we set it to **true**. Even though our public subnet automatically assigns a public IP, I noticed it is good to also include this parameter, because without it sometimes there are conflicts during creation of the EC2 instance.
The **vpc_security_group_ids** parameter specifies either the ID of a single SG or a list of IDs which will be applied to the EC2 instance. In our case we just reference the ID of the SG we created for our EC2 instance (**sg-ec2**).

```
resource "aws_db_subnet_group" "rds-subnet-group" {
  name       = "acit-4640-rds-subnet-group"
  subnet_ids = [aws_subnet.sub2.id, aws_subnet.sub3.id]

  tags = {
    Name = "acit-4640-rds-subnet-group"
```

```
  }
}
```

This block creates a DB subnet group that will be used by our RDS database.
The **name** parameter specifies the name of the subnet group.
The **subnet_ids** parameter specifies a list of subnet IDs to be used by the subnet group. We want out subnet group to use our 2 private subnets so we reference their IDs in our list (i.e. **sub2** and **sub3**).

```
resource "aws_db_instance" "rds-db" {
  identifier             = "acit-4640-rds"
  engine                 = "mysql"
  engine_version         = "8.0.28"
  instance_class         = "db.t3.micro"
  allocated_storage      = 20
  storage_type           = "gp2"
  multi_az               = false
  username               = "admin"
  password               = "password"
  vpc_security_group_ids = [aws_security_group.sg-rds.id]
  db_subnet_group_name   = aws_db_subnet_group.rds-subnet-group.name
  publicly_accessible    = false
  skip_final_snapshot    = true
  apply_immediately      = true
  availability_zone      = "us-west-2a"

  tags = {
    Name = "acit-4640-rds"
  }
}
```

This block creates our RDS MySQL database. I will list the parameters and explanation:

- **identifier**: The database identifier
- **engine**: The engine to be used. We want to use MySQL
- **engine_version**: This is optional. I set it to the default MySQL version provided for free tier MySQL RDS databases.
- **instance_class**: The class for our database instance. Because we want to use the free tier we should set this to either “**db.t2.micro**” or “**db.t3.micro**”.
- **allocated_storage**: The storage for the database in GiB.
- **storage_type**: The type of the storage that will be used. **gp2** is the default for free tier.
- **multi_az**: A boolean value that defines if our database instance will be available across multiple availability zones. We want to set this to **false**.
- **username**: The database master username. This username is used for initial login.
- **password**: The database master password. This password is used with the above master username for initial login.

Note: Because this is just a tutorial, I have left the username and password in plain text for the parameters. In any other case, it is highly advised to either have a terraform variable file or environmental variables that will be used for these 2 parameters.

- **vpc_security_group_ids**: A list of security group IDs. All SGs specified will be applied to the DB instance. Here we only reference the SG we created for the db.
- **db_subnet_group_name**: The name of the subnet group that will be used for the database instance. Here we reference the subnet group we created.
- **publicly_accessible**: A boolean value that defines if our database will be accessible by anyone. Because we don't want this, we set the value to **false**.
- **skip_final_snapshot**: A boolean value that indicates if we want to skip creating a final snapshot of the database. Here we set this to **true** so that the database is created faster and also so we ensure free tier.
- **apply_immediately**: A boolean value that specifies whether any database modifications are applied immediately, or during the next maintenance window. The default value is **false**. Here we set this to **true**.
- **availability_zone**: The availability zone where we want our DB instance to be placed. Here we want it to be the same as the AZ where our public subnet is placed (i.e. **us-west-2a**).

```
output "ec2_public_ip" {
  value       = aws_instance.ec2.public_ip
  description = "Public IP address of the EC2 instance"
}

output "rds_endpoint" {
  value       = aws_db_instance.rds-db.endpoint
  description = "Endpoint of the RDS instance. Follows the -h flag in the mysql
command when connecting."
}

output "rds_username" {
  value       = aws_db_instance.rds-db.username
  description = "Username of the RDS instance. Follows the -u flag in the mysql
command when connecting."
  sensitive   = true
}

output "rds_password" {
  value       = aws_db_instance.rds-db.password
  description = "Password of the RDS instance. Follows the -p flag in the mysql
command when connecting."
  sensitive   = true
```

```
}
```

The code blocks above are all **output** blocks. Each block has it's own name (e.g. **ec2_public_ip**, **rds_endpoint**).
Each block can contain a single **value** parameter. This parameter references some property or attribute of a resource.
There also is the optional **description** parameter, which can be used to include a short description of the output value and what it is used for.
For values like usernames and password (for example RDS username and password) the output blocks require to have the **sensitive** parameter set to **true**, otherwise the **terraform plan/apply** commands will give you an error.
These output blocks store the output values in the Terraform state file. These output values are displayed on the command line when **terraform apply** is ran and also they can be shown by using the **terraform output** command.

And that is all for our infrastructure code using Terraform!!!

Now I will explain what various **terraform** commands do and how to use them.

- **terraform init** – Prepares your working directory for other terraform commands
- **terraform validate** – Checks if the syntax of the configuration file is valid
- **terraform fmt** – Reformats the configuration (makes it look nice :p)
- **terraform plan** – Shows the changes required by the configuration (i.e. what will be created, added, deleted)
- **terraform apply** – Creates or updates the infrastructure. Only will apply changes if the configuration yields a different state than the current state. If portions of the configuration exist in the state and don't change, they will just be skipped.
- **terraform destroy** – Destroys all previously created infrastructure
- **terraform login** – Obtains an API token for a remote host
- **terraform output** – Shows output values from root module
- **terraform output <output_block_name>** - Shows a specific output value using the name of the output block
- **terraform providers** – Shows required providers for the current configuration
- **terraform show** – Shows the current state or a saved plan
- **terraform version** – Shows the current version of Terraform

Now I will explain how to use Terraform Cloud for managing Terraform state
I will list the steps to create a workspace, project, adding access keys as environment variables and logging in from the CLI.

Steps required:

- Create a free account in Terraform Cloud (https://app.terraform.io/)

- When creating an account you will be prompted to add an organization. You can just use your own name as the organization name
- Navigate to the "**Projects and workspaces**" tab
- Click on "**New**" → "**Workspace**"
- Choose "**CLI-driven workflow**" (because we're working with the Terraform CLI)
- Type in a name for your workspace and optionally a description
- Click "**Create workspace**"

Now you have a Terraform Cloud account and a Workspace. Next, to create a porject:

- Navigate to the "**Projects and workspaces**" tab
- Click on "**New**" → "**Project**"
- Type in a name from your new project and click "**Create**"

This is how you create a project. Now when creating workspaces you can select specific projects to use for the workspace.

Now to add access keys as environment variables the steps are:

- Create access keys in AWS and store the access key and secret access key.
- In Terraform Cloud navigate to the "**Settings**" tab and then to "**Variable sets**"
- Click the "**Create variable set**" button
- Type in a name for the variable set and optionally a description too
- Below select to either apply this variable set to all workspaces or choose a specific workspace for this variable
- Below that click on "**Add variable**"
- Choose the "**Environment variable**" option for the variable
- Inside the "**Key**" box type in "**AWS_ACCESS_KEY_ID**" and in "**Value**" paste the access key you stored earlier
- On the far right check the "**Sensitive**" box to ensure the value is protected
- Optionally add a description for that variable
- Click "**Save variable**"
- Click on "**Add variable**" again
- Choose the "**Environment variable**" option for the variable
- Inside the "**Key**" box type in "**AWS_SECRET_ACCESS_KEY**" and in "**Value**" paste the secret access key you stored earlier
- On the far right check the "**Sensitive**" box to ensure the value is protected
- Optionally add a description for that variable
- Click "**Save variable**"

Now you have successfully added a variable set that contains your AWS access keys

Last thing to do is login to Terraform Cloud from the CLI. Here are the steps:

- On your command line type in the command **terraform login**
- Type in **yes** and press Enter

- A browser page should open automatically. If it doesn't there will be a link provided on the command line that you can navigate to.
- Login to your Terraform Cloud account
- On the browser page type in a token name or leave the default name
- Click **Create API token** to generate the authentication token
- Save a copy of the token in a secure location
- After generating the token the CLI will prompt you to for the token
- Paste the token that you generated and press Enter

That is all. Now your token is stored locally in your home directory, and you can utilize Terraform Cloud in your configurations.

Output of **terraform show** for my infrastructure:

```
# aws_db_instance.rds-db:
resource "aws_db_instance" "rds-db" {
    address                               =  "acit-4640-rds.c5dqoalqa93e.us-west-2.rds.amazonaws.com"
    allocated_storage                     = 20
    apply_immediately                     = true
    arn                                   = "arn:aws:rds:us-west-2:741561355720:db:acit-4640-rds"
    auto_minor_version_upgrade            = true
    availability_zone                     = "us-west-2a"
    backup_retention_period               = 0
    backup_window                         = "09:34-10:04"
    ca_cert_identifier                    = "rds-ca-2019"
    copy_tags_to_snapshot                 = false
    customer_owned_ip_enabled             = false
    db_subnet_group_name                  = "acit-4640-rds-subnet-group"
    delete_automated_backups              = true
    deletion_protection                   = false
    enabled_cloudwatch_logs_exports       = []
    endpoint                              =  "acit-4640-rds.c5dqoalqa93e.us-west-2.rds.amazonaws.com:3306"
    engine                                = "mysql"
    engine_version                        = "8.0.28"
    engine_version_actual                 = "8.0.28"
    hosted_zone_id                        = "Z1PVIF0B656C1W"
    iam_database_authentication_enabled   = false
    id                                    = "acit-4640-rds"
    identifier                            = "acit-4640-rds"
    instance_class                        = "db.t3.micro"
```

```
    iops                                 = 0
    license_model                        = "general-public-license"
    listener_endpoint                    = []
    maintenance_window                   = "thu:07:57-thu:08:27"
    max_allocated_storage                = 0
    monitoring_interval                  = 0
    multi_az                             = false
    network_type                         = "IPV4"
    option_group_name                    = "default:mysql-8-0"
    parameter_group_name                 = "default.mysql8.0"
    password                             = (sensitive value)
    performance_insights_enabled         = false
    performance_insights_retention_period = 0
    port                                 = 3306
    publicly_accessible                  = false
    replicas                             = []
    resource_id                          = "db-6AOK2AFOVM3S5ZZXZRCZMWPMUE"
    security_group_names                 = []
    skip_final_snapshot                  = true
    status                               = "available"
    storage_encrypted                    = false
    storage_throughput                   = 0
    storage_type                         = "gp2"
    tags                                 = {
        "Name" = "acit-4640-rds"
    }
    tags_all                             = {
        "Name" = "acit-4640-rds"
    }
    username                             = "admin"
    vpc_security_group_ids               = [
        "sg-024b3b8e8cc65a5a4",
    ]
}

# aws_db_subnet_group.rds-subnet-group:
resource "aws_db_subnet_group" "rds-subnet-group" {
    arn                = "arn:aws:rds:us-west-2:741561355720:subgrp:acit-4640-rds-subnet-
group"
    description        = "Managed by Terraform"
    id                 = "acit-4640-rds-subnet-group"
    name               = "acit-4640-rds-subnet-group"
    subnet_ids         = [
```

```
        "subnet-05df56b32701e516e",
        "subnet-0a268617ebc1ba107",
    ]
    supported_network_types = [
        "IPV4",
    ]
    tags                   = {
        "Name" = "acit-4640-rds-subnet-group"
    }
    tags_all               = {
        "Name" = "acit-4640-rds-subnet-group"
    }
}

# aws_instance.ec2:
resource "aws_instance" "ec2" {
    ami                                  = "ami-0735c191cf914754d"
    arn                                  = "arn:aws:ec2:us-west-2:741561355720:instance/i-
047cbb6f541af70c1"
    associate_public_ip_address          = true
    availability_zone                    = "us-west-2a"
    cpu_core_count                       = 1
    cpu_threads_per_core                 = 1
    disable_api_stop                     = false
    disable_api_termination              = false
    ebs_optimized                        = false
    get_password_data                    = false
    hibernation                          = false
    id                                   = "i-047cbb6f541af70c1"
    instance_initiated_shutdown_behavior = "stop"
    instance_state                       = "running"
    instance_type                        = "t2.micro"
    ipv6_address_count                   = 0
    ipv6_addresses                       = []
    key_name                             = "acit-4640-key"
    monitoring                           = false
    placement_partition_number           = 0
    primary_network_interface_id         = "eni-0230eee5603d4e77c"
    private_dns                          = "ip-10-0-1-66.us-west-2.compute.internal"
    private_ip                           = "10.0.1.66"
    public_ip                            = "54.191.3.7"
    secondary_private_ips                = []
    security_groups                      = []
```

```
source_dest_check                    = true
subnet_id                            = "subnet-091cc506052c6fceb"
tags                                 = {
    "Name" = "acit-4640-ec2"
}
tags_all                             = {
    "Name" = "acit-4640-ec2"
}
tenancy                              = "default"
user_data_replace_on_change          = false
vpc_security_group_ids               = [
    "sg-0fc88382017e608a3",
]

capacity_reservation_specification {
    capacity_reservation_preference = "open"
}

credit_specification {
    cpu_credits = "standard"
}

enclave_options {
    enabled = false
}

maintenance_options {
    auto_recovery = "default"
}

metadata_options {
    http_endpoint               = "enabled"
    http_put_response_hop_limit = 1
    http_tokens                 = "optional"
    instance_metadata_tags      = "disabled"
}

private_dns_name_options {
    enable_resource_name_dns_a_record    = false
    enable_resource_name_dns_aaaa_record = false
    hostname_type                        = "ip-name"
}
```

```
    root_block_device {
        delete_on_termination = true
        device_name           = "/dev/sda1"
        encrypted             = false
        iops                  = 100
        tags                  = {}
        throughput            = 0
        volume_id             = "vol-04544f63ec708e8a9"
        volume_size           = 8
        volume_type           = "gp2"
    }
}

# aws_internet_gateway.igw:
resource "aws_internet_gateway" "igw" {
    arn                 =     "arn:aws:ec2:us-west-2:741561355720:internet-gateway/igw-
0fa9e6611285d9e1a"
    id       = "igw-0fa9e6611285d9e1a"
    owner_id = "741561355720"
    tags     = {
        "Name" = "acit-4640-igw"
    }
    tags_all = {
        "Name" = "acit-4640-igw"
    }
    vpc_id   = "vpc-09975faed02365018"
}

# aws_key_pair.key:
resource "aws_key_pair" "key" {
    arn         = "arn:aws:ec2:us-west-2:741561355720:key-pair/acit-4640-key"
    fingerprint = "2jtsZE3T3JqLKcKmYqJ0yCPSf/LKSROB1/To65S5Tn8="
    id          = "acit-4640-key"
    key_name    = "acit-4640-key"
    key_pair_id = "key-07b36f16fb1b131be"
    key_type    = "ed25519"
    public_key                                          =                   "ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAILvf+1MwjXpuYAj0C2aUEQ/QpRGJ8wk7bK8QtEdPpy
9A tamim@Tamim-Vivobook-Flip14"
    tags        = {}
    tags_all    = {}
}
```

```
# aws_route_table.rt:
resource "aws_route_table" "rt" {
    arn                              =   "arn:aws:ec2:us-west-2:741561355720:route-table/rtb-
0f995c7997f8f5d29"
    id               = "rtb-0f995c7997f8f5d29"
    owner_id         = "741561355720"
    propagating_vgws = []
    route            = [
        {
            carrier_gateway_id         = ""
            cidr_block                 = "0.0.0.0/0"
            core_network_arn           = ""
            destination_prefix_list_id = ""
            egress_only_gateway_id     = ""
            gateway_id                 = "igw-0fa9e6611285d9e1a"
            instance_id                = ""
            ipv6_cidr_block            = ""
            local_gateway_id           = ""
            nat_gateway_id             = ""
            network_interface_id       = ""
            transit_gateway_id         = ""
            vpc_endpoint_id            = ""
            vpc_peering_connection_id  = ""
        },
    ]
    tags             = {
        "Name" = "acit-4640-rt"
    }
    tags_all         = {
        "Name" = "acit-4640-rt"
    }
    vpc_id           = "vpc-09975faed02365018"
}

# aws_route_table_association.rta:
resource "aws_route_table_association" "rta" {
    id             = "rtbassoc-097cce30f7218f1bf"
    route_table_id = "rtb-0f995c7997f8f5d29"
    subnet_id      = "subnet-091cc506052c6fceb"
}

# aws_security_group.sg-ec2:
resource "aws_security_group" "sg-ec2" {
```

```
  arn                         =  "arn:aws:ec2:us-west-2:741561355720:security-group/sg-
0fc88382017e608a3"
  description           = "Allow SSH and HTTP traffic from anywhere"
  egress                = [
    {
      cidr_blocks      = [
        "0.0.0.0/0",
      ]
      description      = ""
      from_port        = 0
      ipv6_cidr_blocks = []
      prefix_list_ids  = []
      protocol         = "-1"
      security_groups  = []
      self             = false
      to_port          = 0
    },
  ]
  id                    = "sg-0fc88382017e608a3"
  ingress               = [
    {
      cidr_blocks      = [
        "0.0.0.0/0",
      ]
      description      = "HTTP from anywhere"
      from_port        = 80
      ipv6_cidr_blocks = []
      prefix_list_ids  = []
      protocol         = "tcp"
      security_groups  = []
      self             = false
      to_port          = 80
    },
    {
      cidr_blocks      = [
        "0.0.0.0/0",
      ]
      description      = "SSH from anywhere"
      from_port        = 22
      ipv6_cidr_blocks = []
      prefix_list_ids  = []
      protocol         = "tcp"
      security_groups  = []
```

```
            self             = false
            to_port          = 22
        },
    ]
    name                   = "acit-4640-sg-ec2"
    owner_id               = "741561355720"
    revoke_rules_on_delete = false
    tags                   = {}
    tags_all               = {}
    vpc_id                 = "vpc-09975faed02365018"
}

# aws_security_group.sg-rds:
resource "aws_security_group" "sg-rds" {
    arn                    =  "arn:aws:ec2:us-west-2:741561355720:security-group/sg-
024b3b8e8cc65a5a4"
    description            = "Allow MySQL traffic from within the VPC"
    egress                 = [
        {
            cidr_blocks      = [
                "0.0.0.0/0",
            ]
            description      = ""
            from_port        = 0
            ipv6_cidr_blocks = []
            prefix_list_ids  = []
            protocol         = "-1"
            security_groups  = []
            self             = false
            to_port          = 0
        },
    ]
    id                     = "sg-024b3b8e8cc65a5a4"
    ingress                = [
        {
            cidr_blocks      = [
                "10.0.0.0/16",
            ]
            description      = "MySQL from within the VPC"
            from_port        = 3306
            ipv6_cidr_blocks = []
            prefix_list_ids  = []
            protocol         = "tcp"
```

```
        security_groups  = []
        self             = false
        to_port          = 3306
      },
  ]
  name                 = "acit-4640-sg-rds"
  owner_id             = "741561355720"
  revoke_rules_on_delete = false
  tags                 = {}
  tags_all             = {}
  vpc_id               = "vpc-09975faed02365018"
}

# aws_subnet.sub1:
resource "aws_subnet" "sub1" {
  arn                                            = "arn:aws:ec2:us-west-2:741561355720:subnet/subnet-
091cc506052c6fceb"
  assign_ipv6_address_on_creation                = false
  availability_zone                              = "us-west-2a"
  availability_zone_id                           = "usw2-az1"
  cidr_block                                     = "10.0.1.0/24"
  enable_dns64                                   = false
  enable_resource_name_dns_a_record_on_launch    = false
  enable_resource_name_dns_aaaa_record_on_launch = false
  id                                             = "subnet-091cc506052c6fceb"
  ipv6_native                                    = false
  map_customer_owned_ip_on_launch                = false
  map_public_ip_on_launch                        = true
  owner_id                                       = "741561355720"
  private_dns_hostname_type_on_launch            = "ip-name"
  tags                                           = {
    "Name" = "acit-4640-pub-sub"
  }
  tags_all                                       = {
    "Name" = "acit-4640-pub-sub"
  }
  vpc_id                                         = "vpc-09975faed02365018"
}

# aws_subnet.sub2:
resource "aws_subnet" "sub2" {
  arn                                            = "arn:aws:ec2:us-west-2:741561355720:subnet/subnet-
05df56b32701e516e"
```

```
    assign_ipv6_address_on_creation                = false
    availability_zone                              = "us-west-2a"
    availability_zone_id                           = "usw2-az1"
    cidr_block                                     = "10.0.2.0/24"
    enable_dns64                                   = false
    enable_resource_name_dns_a_record_on_launch    = false
    enable_resource_name_dns_aaaa_record_on_launch = false
    id                                             = "subnet-05df56b32701e516e"
    ipv6_native                                    = false
    map_customer_owned_ip_on_launch                = false
    map_public_ip_on_launch                        = false
    owner_id                                       = "741561355720"
    private_dns_hostname_type_on_launch            = "ip-name"
    tags                                           = {
        "Name" = "acit-4640-rds-sub1"
    }
    tags_all                                       = {
        "Name" = "acit-4640-rds-sub1"
    }
    vpc_id                                         = "vpc-09975faed02365018"
}

# aws_subnet.sub3:
resource "aws_subnet" "sub3" {
    arn                                            = "arn:aws:ec2:us-west-2:741561355720:subnet/subnet-
0a268617ebc1ba107"
    assign_ipv6_address_on_creation                = false
    availability_zone                              = "us-west-2b"
    availability_zone_id                           = "usw2-az2"
    cidr_block                                     = "10.0.3.0/24"
    enable_dns64                                   = false
    enable_resource_name_dns_a_record_on_launch    = false
    enable_resource_name_dns_aaaa_record_on_launch = false
    id                                             = "subnet-0a268617ebc1ba107"
    ipv6_native                                    = false
    map_customer_owned_ip_on_launch                = false
    map_public_ip_on_launch                        = false
    owner_id                                       = "741561355720"
    private_dns_hostname_type_on_launch            = "ip-name"
    tags                                           = {
        "Name" = "acit-4640-rds-sub2"
    }
    tags_all                                       = {
```

```
        "Name" = "acit-4640-rds-sub2"
    }
    vpc_id                                  = "vpc-09975faed02365018"
}

# aws_vpc.main:
resource "aws_vpc" "main" {
    arn                                     = "arn:aws:ec2:us-west-2:741561355720:vpc/vpc-
09975faed02365018"
    assign_generated_ipv6_cidr_block     = false
    cidr_block                          = "10.0.0.0/16"
    default_network_acl_id               = "acl-02740e68947e8ce06"
    default_route_table_id              = "rtb-0cd7092abfc9f2ec2"
    default_security_group_id            = "sg-094457f687ec51aa2"
    dhcp_options_id                     = "dopt-01164ccd1cf085931"
    enable_classiclink                  = false
    enable_classiclink_dns_support       = false
    enable_dns_hostnames                 = false
    enable_dns_support                   = true
    enable_network_address_usage_metrics = false
    id                                  = "vpc-09975faed02365018"
    instance_tenancy                    = "default"
    ipv6_netmask_length                  = 0
    main_route_table_id                 = "rtb-0cd7092abfc9f2ec2"
    owner_id                            = "741561355720"
    tags                                = {
        "Name" = "acit-4640-vpc"
    }
    tags_all                            = {
        "Name" = "acit-4640-vpc"
    }
}


Outputs:

ec2_public_ip = "54.191.3.7"
rds_endpoint = "acit-4640-rds.c5dqoalqa93e.us-west-2.rds.amazonaws.com:3306"
rds_password = (sensitive value)
rds_username = (sensitive value)
```

Now I will explain how to provision the EC2 instance using **Ansible**

First here is a look of what my Project tree looks like:

```
PROJECT
├── Ansible
│       ├── ansible.cfg
│       ├── default.conf
│       ├── hello.html
│       ├── inventory.ini
│       └── playbook.yml
├── Terraform
│       └── main.tf
├── acit-4640-key.pem
└── acit-4640-key.pub
```

We will focus on the files inside the **Ansible** directory

- **ansible.cfg** is our configuration for Ansible
- **inventory.ini** is the default inventory that we'll use
- **playbook.yml** is the playbook that contains our tasks
- **hello.html** is our static HTML file
- **default.conf** is the Nginx configuration that we'll use

First, I will explain the **ansible.cfg** file:

```
[defaults]
host_key_checking = False
remote_user = ubuntu
private_key_file = /home/tamim/ACIT4640_Assignment3/acit-4640-key.pem
inventory = inventory.ini
```

The contents of the file contain:

- **[defaults]** specifies a section called "**defaults**" that will be applied to any playbook used
- **host_key_checking** - this setting, when set to **False**, disables strict host key checking. This means that Ansible will not prompt you to confirm the authenticity of a host key before connecting to a new host.
- **remote_user** – the default username to be used for remote connections
- **private_key_file** – the absolute path to the ssh private key that will be used for SSH connections

- **inventory** - this specifies the inventory file that Ansible should use. Here we set it to the our inventory file "**inventory.ini**"

This **ansible.cfg** file sets some default options for Ansible to use when connecting to hosts.

Next, I will explain the **inventory.ini** file:

```
[ec2_instance]
54.191.3.7 ansible_user=ubuntu ansible_python_interpreter=/usr/bin/python3
```

- **[ec2_instance]** – a group named **ec2_instance**
- 54.191.3.7 – a single host in the ec2_instance group

The host has 2 extra parameters to be applied when used in a playbook

- **ansible_user** specifies the remote user that Ansible will use to connect to this host over SSH. Here we set it to "**ubuntu**" because we have an EC2 Ubuntu instance.
- **ansible_python_interpreter** specifies the path to the Python interpreter that Ansible should use on the remote host. Here we set it to python3 because we want to use version 3 of Python on the remote host.

This inventory files defines a host in a specific group (**ec2_instance**) and sets some parameters for the host in that group.

Next thing is to mention the **hello.html** and **default.conf** files.

- **hello.html** is just a static HTML file that will be transferred to the remote host to be used as an index page by Nginx
- **default.conf** is the Nginx configuration file that we will use to replace the default configuration after we install Nginx.

I will provide the code for these files at the end of this tutorial along with the full code for all other files as well.

Lastly, I'll explain the **playbook.yml** file and each task it does.
This file defines a set of tasks that will be executed on a specific group of hosts defined in the inventory file, in our case only our EC2 instance.

Now I will show each code block and explain them one by one:

```
---
- name: Install and Configure Nginx and MySQL
  hosts: ec2_instance
  become: true
```

- The first 3 hyphens (**---**) indicate the start of a playbook. We can have multiple playbooks in one file, each separated by 3 hyphens, like the example above.
- **name** is the name of the playbook. It is used for identification purposes
- **hosts** specifies the group of hosts defined in the inventory file on which this playbook should be run
- **become** (when set to **true**) enables privilege escalation to become root, so that tasks can be run with administrative privileges

```
tasks:
  - name: Install Python3
    apt:
      name: python3
      state: present
```

- **tasks** defines a list of tasks that should be performed on the hosts
- **name** is a description of a task. Each task must have it's own name
- **apt** defines that we'll use the apt package manager
- **name** (python3) defines the name of the package that apt will work with
- **state** defines the state we want this package to be in. Here we set it to **present** which means this task will ensure apt installs python3 or checks that it is already installed.

```
- name: Install pip
    apt:
      name: python3-pip
      state: present
```

- Pretty much like the previous task this one has it's own name and uses apt to install the **python3-pip** package or ensure that it is already installed

```
- name: Install PyMySQL
    pip:
      name: pymysql
      executable: pip3
```

- This task is a little different. It doesn't use the **apt** package manager but uses the **pip3** package manager.
- **name** specifies the package we want to install with pip (in this case **pymysql**)
- **executable** defines the command needed to run the pip package manager

```
-   - name: Install Nginx
-       apt:
-         name: nginx
-         state: present
```

- Like the previous tasks using apt this one only has a different name and installs nginx or makes sure it is installed

```
- name: Copy static HTML file
    copy:
      src: "hello.html"
      dest: /var/www/html/index.html
```

- This task copies our static html file to the appropriate location
- **copy** specifies the action to be taken
- **src** specifies the name of the file to be transferred. It is important to note that this file must be in the same directory as the playbook file in order for this to work.
- **dest** specifies the destination to which we want to transfer the file. This destination defines that our file will be transferred to the **/var/www/html** directory and will be named **index.html**

```
  - name: Copy Nginx configuration file
    copy:
      src: "default.conf"
      dest: /etc/nginx/sites-available/default
    notify: Restart Nginx
```

- This task copies our nginx configuration to the appropriate location, similar to the previous task.
- We define the source of our file and the destination on the remote host
- Here we also include this line **notify: Restart Nginx**. This line makes it so that this task will notify the **handler** named "**Restart Nginx**" when it returns a "**changed=True**" status. That means when this task completes it's actions it will notify the handler. The handler will be explained in a bit.

```
- name: Install MySQL client
    apt:
      name: mysql-client
      state: present
```

- This task is pretty much the same as the previous **apt** tasks.
- It installs the **mysql-client** or makes sure it is installed.

```
- name: Create MySQL database
    community.mysql.mysql_db:
      login_user: admin
      login_password: password
      login_host: acit-4640-rds.c5dqoalqa93e.us-west-2.rds.amazonaws.com
      login_port: 3306
      name: bookstack
      state: present
```

- This task uses the Ansible **community.mysql.mysql_db** module to create a MySQL database.
- **login_user** specifies the username to be used to connect to the database server
- **login_password** specifies the password to be used to connect to the database server
- **login_host** specifies the host that we are connecting to
- **login_port** specifies the port of the database server to connect to

Note: Here we provide the username and password in plain text. Because this is a tutorial this is fine, but otherwise I would highly advise to use either a variables file or environment variables.

- **name** specifies the name of the database we want to create
- **state** here is set to **present** which means that the database will be created if it doesn't exist.

```
- name: Create MySQL user
    community.mysql.mysql_user:
      login_user: admin
      login_password: password
      login_host: acit-4640-rds.c5dqoalqa93e.us-west-2.rds.amazonaws.com
      login_port: 3306
      name: bookstack
      password: bookstack
      priv: 'bookstack.*:ALL'
      host: '%'
      state: present
```

- This task uses the Ansible **community.mysql.mysql_user** module to create a MySQL user and give it privileges.
- **login_user** specifies the username to be used to connect to the database server
- **login_password** specifies the password to be used to connect to the database server
- **login_host** specifies the host that we are connecting to
- **login_port** specifies the port of the database server to connect to

- **name** specifies the name of the user to be created
- **password** specifies the password of the user to be created
- **priv** specifies what privileges we want the user to have. In our case we want the user to have all privileges on the **bookstack** database.
- **host** specifies which hosts the new user should be accessible from. In our case we specify it as "**%**" which means anywhere.
- **state** here is also set to **present** which means the user will be created if it doesn't exist

```
handlers:
  - name: Restart Nginx
    service:
      name: nginx
      state: restarted
```

- This is not a task, but the section that contains the handlers
- Here we only have 1 handler
- **name** is the name of the handler and is set to **Restart Nginx**
- If you remember from the previous tasks, the task to copy the new Nginx configuration had a **notify** statement that notifies exactly this handler
- **service** is the command to be executed once the handler has been notifies
- **name: nginx** is the service that we want the service command to work with
- **state** here is set to **restarted** which means we want to restart the service specified

To verify the syntax of an Ansible playbook file we can run the command:

ansible-playbook <playbook file name> --syntax-check

Also we can use the tool **ansible-lint** to check for warning or errors in our playbook:

ansible-lint <playbook file name>

This will run the ansible-lint tool against the file and display any warnings or errors that are detected. It will use the default configuration and ruleset for ansible-lint.

To run an Ansible playbook and use a specific inventory file (flag **-i**) run the command:

ansible-playbook -i inventory.ini <playbook file name>

And that is all that we needed to provision our EC2 instance with Ansible!!!

Bellow I will provide all files used in full for reference:

main.tf

```
terraform {
  cloud {
    organization = "tamim_hemat"

    workspaces {
      name = "4640-Terraform"
    }
  }

  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.16"
    }
  }

  required_version = ">= 1.2.0"
}

provider "aws" {
  region = "us-west-2"
}

variable "base_cidr_block" {
  description = "default cidr block for vpc"
  default     = "10.0.0.0/16"
}

variable "subnet1_cidr_block" {
  description = "default cidr block for subnet1"
  default     = "10.0.1.0/24"
}

variable "subnet2_cidr_block" {
  description = "default cidr block for subnet2"
  default     = "10.0.2.0/24"
}

variable "subnet3_cidr_block" {
  description = "default cidr block for subnet3"
  default     = "10.0.3.0/24"
}

resource "aws_vpc" "main" {
```

```
  cidr_block       = var.base_cidr_block
  instance_tenancy = "default"

  tags = {
    Name = "acit-4640-vpc"
  }
}

resource "aws_subnet" "sub1" {
  vpc_id                  = aws_vpc.main.id
  cidr_block              = var.subnet1_cidr_block
  availability_zone       = "us-west-2a"
  map_public_ip_on_launch = true

  tags = {
    Name = "acit-4640-pub-sub"
  }
}

resource "aws_subnet" "sub2" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = var.subnet2_cidr_block
  availability_zone = "us-west-2a"

  tags = {
    Name = "acit-4640-rds-sub1"
  }
}

resource "aws_subnet" "sub3" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = var.subnet3_cidr_block
  availability_zone = "us-west-2b"

  tags = {
    Name = "acit-4640-rds-sub2"
  }
}

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

  tags = {
    Name = "acit-4640-igw"
  }
```

```
}

resource "aws_route_table" "rt" {
  vpc_id = aws_vpc.main.id

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

  tags = {
    Name = "acit-4640-rt"
  }
}

resource "aws_route_table_association" "rta" {
  subnet_id      = aws_subnet.sub1.id
  route_table_id = aws_route_table.rt.id
}

resource "aws_security_group" "sg-ec2" {
  name        = "acit-4640-sg-ec2"
  description = "Allow SSH and HTTP traffic from anywhere"
  vpc_id      = aws_vpc.main.id

  ingress {
    description = "SSH from anywhere"
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

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

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
```

```
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_security_group" "sg-rds" {
  name        = "acit-4640-sg-rds"
  description = "Allow MySQL traffic from within the VPC"
  vpc_id      = aws_vpc.main.id

  ingress {
    description = "MySQL from within the VPC"
    from_port   = 3306
    to_port     = 3306
    protocol    = "tcp"
    cidr_blocks = [var.base_cidr_block]
  }

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

resource "aws_key_pair" "key" {
  key_name   = "acit-4640-key"
  public_key = "ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAILvf+1MwjXpuYAj0C2aUEQ/QpRGJ8wk7bK8QtEdPpy9A tamim@Tamim-
Vivobook-Flip14"
}

resource "aws_instance" "ec2" {
  ami                         = "ami-0735c191cf914754d"
  instance_type               = "t2.micro"
  key_name                    = aws_key_pair.key.key_name
  subnet_id                   = aws_subnet.sub1.id
  associate_public_ip_address = true
  vpc_security_group_ids      = [aws_security_group.sg-ec2.id]

  tags = {
    Name = "acit-4640-ec2"
  }
}
```

```
resource "aws_db_subnet_group" "rds-subnet-group" {
  name       = "acit-4640-rds-subnet-group"
  subnet_ids = [aws_subnet.sub2.id, aws_subnet.sub3.id]

  tags = {
    Name = "acit-4640-rds-subnet-group"
  }
}

resource "aws_db_instance" "rds-db" {
  identifier             = "acit-4640-rds"
  engine                 = "mysql"
  engine_version         = "8.0.28"
  instance_class         = "db.t3.micro"
  allocated_storage      = 20
  storage_type           = "gp2"
  multi_az               = false
  username               = "admin"
  password               = "password"
  vpc_security_group_ids = [aws_security_group.sg-rds.id]
  db_subnet_group_name   = aws_db_subnet_group.rds-subnet-group.name
  publicly_accessible    = false
  skip_final_snapshot    = true
  apply_immediately      = true
  availability_zone      = "us-west-2a"

  tags = {
    Name = "acit-4640-rds"
  }
}

output "ec2_public_ip" {
  value       = aws_instance.ec2.public_ip
  description = "Public IP address of the EC2 instance"
}

output "rds_endpoint" {
  value       = aws_db_instance.rds-db.endpoint
  description = "Endpoint of the RDS instance. Follows the -h flag in the mysql
command when connecting."
}

output "rds_username" {
  value       = aws_db_instance.rds-db.username
```

```
  description = "Username of the RDS instance. Follows the -u flag in the mysql
command when connecting."
  sensitive   = true
}

output "rds_password" {
  value       = aws_db_instance.rds-db.password
  description = "Password of the RDS instance. Follows the -p flag in the mysql
command when connecting."
  sensitive   = true
}
```

home.html

```
<html>
    <head>
        <title>Hello Bookstack</title>
        <style>
            .heading {
                text-align: center;
                font-size: 3rem;
            }
            .message {
                padding-top: 30px;
                text-align: center;
                font-size: 1.5rem;
            }
            .images {
                display: flex;
                flex-direction: column;
                justify-content: space-evenly;
                align-items: center;
            }
            .gap {
                padding-top: 20px;
                padding-bottom: 20px;
                font-weight: bold;
                font-size: 2rem;
            }
        </style>
    </head>
    <body>
        <h1 class="heading">Hello Bookstack</h1>
```

```
        <p class="message">My first bookstack page with Ansible and Terraform</p>
        <div class="images">
            <img src="https://i.imgflip.com/51wejc.jpg" alt="kekbuntu">
            <p class="gap">Hosted on Ubuntu.....</p>
            <img
src="https://www.bing.com/th/id/OGC.ceb2bf601a9ac079f04685a0892d8cba?pid=1.7&rurl
=https%3a%2f%2fi.imgflip.com%2f5q7ff0.gif&ehk=3JTjD4pmsryN0DqUrhYFuDBi9w%2f790VHW
R16OgoDLPs%3d" alt="bruh">
        </div>
    </body>
</html>
```

default.conf

```
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html;
    index index.html;

    server_name _;

    location / {
        try_files $uri $uri/ =404;
    }
}
```

ansible.cfg

```
[defaults]
host_key_checking = False
remote_user = ubuntu
private_key_file = /home/tamim/ACIT4640_Assignment3/acit-4640-key.pem
inventory = inventory.ini
```

inventory.ini

```
[ec2_instance]
54.191.3.7 ansible_user=ubuntu ansible_python_interpreter=/usr/bin/python3
```

playbook.yml

```yaml
---
- name: Install and Configure Nginx and MySQL
  hosts: ec2_instance
  become: true

  tasks:
  - name: Install Python3
    apt:
      name: python3
      state: present

  - name: Install pip
    apt:
      name: python3-pip
      state: present

  - name: Install PyMySQL
    pip:
      name: pymysql
      executable: pip3

  - name: Install Nginx
    apt:
      name: nginx
      state: present

  - name: Copy static HTML file
    copy:
      src: "hello.html"
      dest: /var/www/html/index.html

  - name: Copy Nginx configuration file
    copy:
      src: "default.conf"
      dest: /etc/nginx/sites-available/default
    notify: Restart Nginx

  - name: Install MySQL client
    apt:
      name: mysql-client
      state: present

  - name: Create MySQL database
    community.mysql.mysql_db:
```

```
    login_user: admin
    login_password: password
    login_host: acit-4640-rds.c5dqoalqa93e.us-west-2.rds.amazonaws.com
    login_port: 3306
    name: bookstack
    state: present

- name: Create MySQL user
  community.mysql.mysql_user:
    login_user: admin
    login_password: password
    login_host: acit-4640-rds.c5dqoalqa93e.us-west-2.rds.amazonaws.com
    login_port: 3306
    name: bookstack
    password: bookstack
    priv: 'bookstack.*:ALL'
    host: '%'
    state: present

handlers:
- name: Restart Nginx
  service:
    name: nginx
    state: restarted
```