Lab: Terraform Resource Blocks

Terraform uses resource blocks to manage infrastructure, such as virtual networks, compute instances, or higher-level components such as DNS records. Resource blocks represent one or more infrastructure objects in your Terraform configuration. Most Terraform providers have a number of different resources that map to the appropriate APIs to manage that particular infrastructure type.

```
# Template
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>"   {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}
```

Resource	AWS Provider	AWS Infrastructure
Resource 1	aws_instance	EC2 Instance
Resource 2	aws_security_group	Security Group
Resource 3	aws_s3_bucket	Amazon S3 bucket
Resource 4	aws_key_pair	EC2 Key Pair

When working with a specific provider, like AWS, Azure, or GCP, the resources are defined in the provider documentation. Each resource is fully documented in regards to the valid and required arguments required for each individual resource. For example, the `aws_key_pair` resource has a "Required" argument of `public_key` but optional arguments like `key_name` and `tags`. You'll need to look at the provider documentation to understand what the supported resources are and how to define them in your Terraform configuration.

Important - Without `resource` blocks, Terraform is not going to create resources. All of the other block types, such as `variable`, `provider`, `terraform`, `output`, etc. are essentially supporting block types for the `resource` block.

- Task 1: View and understand an existing resource block in Terraform
- Task 2: Add a new resource to deploy an Amazon S3 bucket
- Task 3: Create a new AWS security group
- Task 4: Configure a resource from the random provider
- Task 5: Update the Amazon S3 bucket to use the random ID

Task 1: View and understand an existing resource block in Terraform

Using the main.tf created in previous labs, look for a resource block that deploys a route table. The code should look something like this:

```
resource "aws_route_table" "public_route_table" {
  vpc_id = aws_vpc.vpc.id

  route {
    cidr_block     = "0.0.0.0/0"
    gateway_id     = aws_internet_gateway.internet_gateway.id
    #nat_gateway_id = aws_nat_gateway.nat_gateway.id
  }
  tags = {
    Name      = "demo_public_rtb"
    Terraform = "true"
  }
}
```

Let's look at the details in this resource block. First, all resource blocks will be declared using the resouce block type. Next, you'll find the type of resource that is going to be deployed. In this case, it's `aws_route_table` which is part of the AWS provider. Finally, we gave this resource a local name of `public_route_table`.

> Note: Your resource blocks must have a unique resource id (combination of resource type along with resource name). In our example, our resource id is `aws_route_table.public_route_table`, which is the combination of our resource type `aws_route_table` and resource name `public_route_table`. This naming and interpolation nomenclature is powerful part of HCL that allows us to reference arguments from other resource blocks.

Within our resouce block, we have arguments that are specific to the type of resource we are deploying. In our case, when we deploy a new AWS route table, there are certain arguments that are either `required` or `optional` that will apply to this resource. In our example, an AWS route table must be tied to a VPC, so we are providing the `vpc_id` argument and providing a value of our VPC using interpolation. Check out the `aws_route_table` documentation and see all the additional arguments that are available.

Task 2: Add a new resource to deploy an Amazon S3 bucket

Ok, so now that we understand more about a resource block, let's create a new resource that will create an Amazon S3 bucket. In your main.tf file, add the following resource block:

```
resource "aws_s3_bucket" "my-new-S3-bucket" {
  bucket = "my-new-tf-test-bucket-bryan"
  acl    = "private"

  tags = {
    Name    = "My S3 Bucket"
    Purpose = "Intro to Resource Blocks Lab"
  }
}
```

Task 2.1.1

Run a `terraform plan` to see that this new Amazon S3 bucket will be added to our account. Don't worry, S3 buckets don't incur any fees unless you upload data to the bucket.

```
terraform plan
```

```
Terraform used the selected providers to generate the following execution plan. Resource
  + create

Terraform will perform the following actions:

  # aws_s3_bucket.my-new-S3-bucket will be created
  + resource "aws_s3_bucket" "my-new-S3-bucket" {
      + acceleration_status         = (known after apply)
      + acl                         = "private"
      + arn                         = (known after apply)
      + bucket                      = "my-new-tf-test-bucket-bryan"
      + bucket_domain_name          = (known after apply)
      + bucket_regional_domain_name = (known after apply)
      + force_destroy               = false
      + hosted_zone_id              = (known after apply)
      + id                          = (known after apply)
      + region                      = (known after apply)
      + request_payer               = (known after apply)
      + tags                        = {
          + "Name"    = "My S3 Bucket"
          + "Purpose" = "Intro to Resource Blocks Lab"
        }
      + tags_all                    = {
          + "Name"    = "My S3 Bucket"
          + "Purpose" = "Intro to Resource Blocks Lab"
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)
```


```
      + versioning {
          + enabled    = (known after apply)
          + mfa_delete = (known after apply)
        }
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Once the plan looks good, let's apply our configuration to the account by issuing a `terraform apply`

```
terraform apply

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

You will be prompted to confirm the changes before they're applied. Respond with `yes`.

Task 2.2.1

Log into your AWS account and validate that the new Amazon S3 bucket exists.

Task 3: Configure an AWS security group

Let's add one more resource block just to make sure you've got the hang of it. In this example, we're going to create a new Security Group that could be used for a web server to allow secure inbound connectivity over 443. If you are deploying EC2 instances or other resources in AWS, you'll probably need to define a custom security group.

In your main.tf file, add the following resource block:

```
resource "aws_security_group" "my-new-security-group" {
  name        = "web_server_inbound"
  description = "Allow inbound traffic on tcp/443"
  vpc_id      = aws_vpc.vpc.id

  ingress {
    description = "Allow 443 from the Internet"
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
```


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

  tags = {
    Name    = "web_server_inbound"
    Purpose = "Intro to Resource Blocks Lab"
  }
}
```

Notice that this resource, the `aws_security_group` requires completely different arguments. We're providing a name, a description, a VPC, and most importantly, the rules that we want to permit or restrict traffic. We also providing tags that will be added to the resource.

Task 3.1.1

Run a `terraform plan` to see that this new Amazon S3 bucket will be added to our account. Don't worry, S3 buckets don't incur any fees unless you upload data to the bucket.

```
terraform plan
```

```
Terraform used the selected providers to generate the following execution plan. Resource
  + create

Terraform will perform the following actions:

  # aws_security_group.my-new-security-group will be created
  + resource "aws_security_group" "my-new-security-group" {
      + arn                    = (known after apply)
      + description            = "Allow inbound traffic on tcp/443"
      + egress                 = (known after apply)
      + id                     = (known after apply)
      + ingress                = [
          + {
              + cidr_blocks      = [
                  + "0.0.0.0/0",
                ]
              + description      = "Allow 443 from the Internet"
              + from_port        = 443
              + ipv6_cidr_blocks = []
              + prefix_list_ids  = []
              + protocol         = "tcp"
              + security_groups  = []
              + self             = false
              + to_port          = 443
            },
        ]
```


```
      + name                   = "web_server_inbound"
      + name_prefix            = (known after apply)
      + owner_id               = (known after apply)
      + revoke_rules_on_delete = false
      + tags                   = {
          + "Name"    = "web_server_inbound"
          + "Purpose" = "Intro to Resource Blocks Lab"
        }
      + tags_all               = {
          + "Name"    = "web_server_inbound"
          + "Purpose" = "Intro to Resource Blocks Lab"
        }
      + vpc_id                 = "vpc-0407edc1d4962b00f"
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Once the plan looks good, let's apply our configuration to the account by issuing a `terraform apply`

```
terraform apply

Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
```

You will be prompted to confirm the changes before they're applied. Respond with `yes`.

Log into AWS, browse to the EC2 or VPC console and verify the newly created Security Group.

Task 4: Configure a resource from the random provider

Ok, one more resource. But this time, it's not an AWS resource. Terraform supports many resources that don't interact with any other services. In fact, there is a provider that you can use just to create random data to be used in your Terraform.

Let's add a new resource block to Terraform using the `random` provider. In your main.tf file, add the following resource block:

```
resource "random_id" "randomness" {
  byte_length = 16
}
```


Task 4.1.1

Run a `terraform plan` to see that this new Amazon S3 bucket will be added to our account. Don't worry, S3 buckets don't incur any fees unless you upload data to the bucket.

```
terraform plan
```

You're probably going to get an output like this:

```
Error: Could not load plugin


 Plugin reinitialization required. Please run "terraform init".

 Plugins are external binaries that Terraform uses to access and manipulate
 resources. The configuration provided requires plugins which can't be located,
 don't satisfy the version constraints, or are otherwise incompatible.

 Terraform automatically discovers provider requirements from your
 configuration, including providers used in child modules. To see the
 requirements and constraints, run "terraform providers".

 failed to instantiate provider "registry.terraform.io/hashicorp/random" to obtain schem
```

Whoops….what happened? Well, we added a new resource that requires a provider that we haven't downloaded yet. Up until now, we've just been using the AWS provider, so adding AWS resources has worked fine. But now we need the hashicorp/random provider to use the `random_id` resource.

Task 4.1.2

Let's run a `terraform init` so Terraform will download the provider we need. Notice in the output of the `terraform init` that you will see Terraform download the `hashicorp/random` provider.

```
terraform init
```

```
Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Finding latest version of hashicorp/random...
- Using previously-installed hashicorp/aws v3.63.0
- Installing hashicorp/random v3.1.0...
- Installed hashicorp/random v3.1.0 (signed by HashiCorp)

Terraform has made some changes to the provider dependency selections recorded
```


```
in the .terraform.lock.hcl file. Review those changes and commit them to your
version control system if they represent changes you intended to make.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Task 4.1.3

Run a `terraform plan` again to see that this new Amazon S3 bucket will be added to our account. Don't worry, S3 buckets don't incur any fees unless you upload data to the bucket.

```
terraform plan
```

Notice that Terraform is going to create a new resource for us now:

```
Terraform used the selected providers to generate the following execution plan. Resource
  + create

Terraform will perform the following actions:

  # random_id.randomness will be created
  + resource "random_id" "randomness" {
      + b64_std     = (known after apply)
      + b64_url     = (known after apply)
      + byte_length = 16
      + dec         = (known after apply)
      + hex         = (known after apply)
      + id          = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
```

Task 4.1.4

Run a `terraform apply -auto-approve` to create the new resource. Remember that we're not actually creating anything on AWS, but generating a random 16 byte id.

```
terraform apply -auto-approve
```

You will see in the output that Terraform created a random id for us, and the value can be found in the output as shown below. Mine is `Htd2k6vC5PrbOxGeCBxAcQ` but yours will be different.

```
Terraform used the selected providers to generate the following execution plan. Resource
  + create

Terraform will perform the following actions:

  # random_id.randomness will be created
  + resource "random_id" "randomness" {
      + b64_std     = (known after apply)
      + b64_url     = (known after apply)
      + byte_length = 16
      + dec         = (known after apply)
      + hex         = (known after apply)
      + id          = (known after apply)
    }

Plan: 1 to add, 0 to change, 0 to destroy.
random_id.randomness: Creating...
random_id.randomness: Creation complete after 0s [id=Htd2k6vC5PrbOxGeCBxAcQ]

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

Task 5: Update the Amazon S3 bucket to use the random ID

Now that we have a `random_id` being created by Terraform, let's see how we can use that for other resources. If you're not aware, Amazon S3 bucket names have to be globally unique, meaning nobody in the world can create a bucket with the same name as an existing bucket. That means if I create a bucket named "my-cool-s2-bucket", nobody else can create a bucket with the same name. This is where the random_id might come in handy, so let's update the name of our bucket to use a random ID.

In your main.tf file, find the resource block where you created a new Amazon S3 bucket. It's the code we added in Task 2 above. Modify the `bucket` argument to include the following:

```
resource "aws_s3_bucket" "my-new-S3-bucket" {
  bucket = "my-new-tf-test-bucket-${random_id.randomness.id}"
  acl    = "private"

  tags = {
    Name    = "My S3 Bucket"
```


```
    Purpose = "Intro to Resource Blocks Lab"
  }
}
```

Task 5.1.1

Run a `terraform plan` again to see that Terraform is going to replace our Amazon S3 bucket in our account because the name is changing. The name will now end with our random id that we created using the `random_id` resource.

```
terraform plan
```

```
Terraform used the selected providers to generate the following execution plan. Resource
-/+ destroy and then create replacement

Terraform will perform the following actions:

  # aws_s3_bucket.my-new-S3-bucket must be replaced
-/+ resource "aws_s3_bucket" "my-new-S3-bucket" {
      + acceleration_status         = (known after apply)
      ~ arn                         = "arn:aws:s3:::my-new-tf-test-bucket-bryan" -> (kno
      ~ bucket                      = "my-new-tf-test-bucket-bryan" -> "my-new-tf-test-b
      ~ bucket_domain_name          = "my-new-tf-test-bucket-bryan.s3.amazonaws.com" ->
      ~ bucket_regional_domain_name = "my-new-tf-test-bucket-bryan.s3.amazonaws.com" ->
      ~ hosted_zone_id              = "Z3AQBSTGFYJSTF" -> (known after apply)
      ~ id                          = "my-new-tf-test-bucket-bryan" -> (known after appl
      ~ region                      = "us-east-1" -> (known after apply)
      ~ request_payer               = "BucketOwner" -> (known after apply)
      ~ tags                        = {
          + "Random"  = "Htd2k6vC5PrbOxGeCBxAcQ"
            # (2 unchanged elements hidden)
        }
      ~ tags_all                    = {
          + "Random"  = "Htd2k6vC5PrbOxGeCBxAcQ"
            # (2 unchanged elements hidden)
        }
      + website_domain              = (known after apply)
      + website_endpoint            = (known after apply)
        # (2 unchanged attributes hidden)

      ~ versioning {
          ~ enabled    = false -> (known after apply)
          ~ mfa_delete = false -> (known after apply)
        }
    }
```


```
Plan: 1 to add, 0 to change, 1 to destroy.
```

Task 5.1.2

Now that we're done with going over the Resource block, feel free to delete the resources that we created in this lab. These resources include:

- Amazon S3 bucket
- Security Group
- Random ID

Once you deleted them from your Terraform configuration file, go ahead and run a `terraform plan` and `terraform apply` to delete the resources from your account.