Lab: Introduction to the Terraform Data Block

Data sources are used in Terraform to load or query data from APIs or other Terraform workspaces. You can use this data to make your project's configuration more flexible, and to connect workspaces that manage different parts of your infrastructure. You can also use data sources to connect and share data between workspaces in Terraform Cloud and Terraform Enterprise.

To use a data source, you declare it using a `data` block in your Terraform configuration. Terraform will perform the query and store the returned data. You can then use that data throughout your Terraform configuration file where it makes sense.

Data Blocks within Terraform HCL are comprised of the following components:

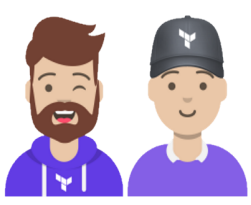
- Data Block - "resource" is a top-level keyword like "for" and "while" in other programming languages.
- Data Type - The next value is the type of the resource. Resources types are always prefixed with their provider (aws in this case). There can be multiple resources of the same type in a Terraform configuration.
- Data Local Name - The next value is the name of the resource. The resource type and name together form the resource identifier, or ID. In this lab, one of the resource IDs is `aws_instance.web`. The resource ID must be unique for a given configuration, even if multiple files are used.
- Data Arguments - Most of the arguments within the body of a resource block are specific to the selected resource type. The resource type's documentation lists which arguments are available and how their values should be formatted.

Example: A data block requests that Terraform read from a given data source ("aws_ami") and export the result under the given local name ("example"). The name is used to refer to this resource from elsewhere in the same Terraform module.

Template

```
data "<DATA TYPE>" "<DATA LOCAL NAME>"    {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}
```

- Task 1: Add a new data source to query the current AWS region being used
- Task 2: Update the Terraform configuration file to use the new data source



- Task 3: View the data source used to retrieve the availability zones within the region
- Task 4: Validate the data source is being used in the Terraform configuration file
- Task 5: Create a new data source for querying a different Ubuntu image
- Task 6: Make the aws_instance web_server use the Ubuntu image returned by the data source

Task 1: Add a new data source to query the current AWS region being used

Modify the main.tf file to include a new data source block. This simple data source will use the AWS credentials and determine the AWS region that you are using.

```
#Retrieve the AWS region
data "aws_region" "current" { }
```

Task 2: Update the Terraform configuration file to use the new data source

Now that we know Terraform will query AWS for the current region, we can now use that information within our configuration file. In our main.tf file, you'll find the resource block where we are creating the VPC. Within that block, you'll see that we are adding tags to the VPC for easily identification. Let's add a "Region" tag and use the results of our data source we just added.

Modify the VPC resource block to add another tag and get the value from our data source. Terraform will query AWS, grab the current region, and add the value to our new tag.

```
#Define the VPC
resource "aws_vpc" "vpc" {
  cidr_block = var.vpc_cidr

  tags = {
    Name        = var.vpc_name
    Environment = "demo_environment"
    Terraform   = "true"
    Region      = data.aws_region.current.name

  }
}
```

Notice the formatting of our reference to the data block. We'll go into more detail when we learn about Variables, but the syntax to refer to a data block is as follows:

```
data.<type>.<name>.<attribute>
```


In our example, `data` is used since we're referring to a data block. The "**type**" is `aws_region`, the "**name**" is `current`, and the "**attribute**" is `name`. You can simply look at the data block and see how this is all put together. Finally, `name` is one of the attributes that is exported by the `aws_region` data source. Make sure to check out the official documentation for a data source to learn about the different attributes available for the one you're using.

Task 3: View the data source used retrieve the availability zones within the region

If you didn't noticed already, we were already using a data source to retrieve the availability zones in the region we are using. Towards the top of the main.tf file, look for the following code:

```
#Retrieve the list of AZs in the current AWS region
data "aws_availability_zones" "available" {}
```

It looks very similar to our first task, but this is retrieving different data for us. Obtaining this data might be helpful if you are building a fleet of web servers and you want to ensure each one is deployed in a different AZ. The benefit of using a data source, as opposed to manually typing in the availability zones, is that you can run the exact same Terraform in multiple regions and it would work without modifications, since the AZs are dynamically obtained by the data source.

Task 4: Validate the data source is being used in the Terraform configuration filee

Let's take a look at how we're already using this data source in our deployment. Look in the main.tf and find the resource block that is creating our private subnets. When you create a subnet, one of the required parameters is choosing an availability zone where the subnet will be created. Rather than hard code this, we can use our data source to dynamically choose an availability zone.

```
#Deploy the private subnets
resource "aws_subnet" "private_subnets" {
  for_each          = var.private_subnets
  vpc_id            = aws_vpc.vpc.id
  cidr_block        = cidrsubnet(var.vpc_cidr, 8, each.value)
  availability_zone = tolist(data.aws_availability_zones.available.names)[
     each.value]

  tags = {
    Name      = each.key
    Terraform = "true"
  }
}
```


Task 5: Create a new data source for querying a different Ubuntu image

Step 5.1.1

Add an aws_ami data source called “ubuntu_22_04” in `main.tf`. It will find the most recent instance of a Ubuntu 22.04 AMI from Canonical (owner ID 099720109477).

```
# Terraform Data Block - Lookup Ubuntu 22.04
data "aws_ami" "ubuntu_22_04" {
  most_recent = true

  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"]
  }

  owners = ["099720109477"]
}
```

Task 6: Modify the aws_instance so it uses the returned AMI

Edit the aws_instance in `main.tf` so that its ami argument uses the AMI returned by the data source.

```
resource "aws_instance" "web_server" {
  ami                         = data.aws_ami.ubuntu_22_04.id
  instance_type               = "t2.micro"
  subnet_id                   = aws_subnet.public_subnets["public_subnet_1
     "].id
  security_groups             = [aws_security_group.vpc-ping.id]
  associate_public_ip_address = true
  tags = {
    Name = "Web EC2 Server"
  }
}
```

Step 6.1.1

Run `terraform apply` one last time to apply the changes you made.

```
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
```

```
aws_instance.web_server: Destroying... [id=i-007601704642290cc]
aws_instance.web_server: Still destroying... [id=i-007601704642290cc, 10s
    elapsed]
aws_instance.web_server: Still destroying... [id=i-007601704642290cc, 20s
    elapsed]
aws_instance.web_server: Still destroying... [id=i-007601704642290cc, 30s
    elapsed]
aws_instance.web_server: Destruction complete after 40s
aws_instance.web_server: Creating...
aws_instance.web_server: Still creating... [10s elapsed]
aws_instance.web_server: Still creating... [20s elapsed]
aws_instance.web_server: Still creating... [30s elapsed]
aws_instance.web_server: Creation complete after 31s [id=i-0566
    fade24f7cd155]

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