Terraform Resource Addressing

Resource Addressing

- A resource address is a unique way to target a specific resource
  - Acts as an identifier
  - Every resource in Terraform is assigned a unique resource address
  - What is a resource? Everything you create or define in Terraform, is a resource
- When to use a resource address?
  - In code - to create internal dependencies or get attributes from specific resources
  - Terraform CLI - running commands targeted to a specific resource
- A resource address consists of: resource_type.resource_name[instance_index]
  - resource_type - the type of resource, like an ec2 instance, an s3 bucket, depends completely on what Provider you are using (if using GCP provider, obviously the resource type is different)
  - resource_name - the name you've provided to that resource
  - instance_index - if the resource is created in an array or map (optional)
  - Not all resources follow this convention
    - Modules, variables, locals - are slightly different

Resource Addressing

- Example: We are using an AWS Provider, and creating an EC2 instance named "main"
  - Always consult documentation for resources available by Provider type
- resource_type = aws_instance
- resource_name = main
- instance_index = None, this is a standalone object and not part of any array or map
- To target this specific resource in your Workflow commands, the resource address is: aws_instance.main

```
resource "aws_instance" "main" {
  provider = alias.west
  ami = data.aws_ami.aws_linux_2.id

  instance_type               = "t3.small"
  associate_public_ip_address = true

  vpc_security_group_ids = [aws_security_group.allow_ssh.id]

  ebs_block_device {
    device_name = "/dev/sdf"
    volume_type = "gp2"
    volume_size = "2"
  }

  tags = merge(local.tags, {
    Name : "levelup_with_terraform"
  })
}
```

Resource Addressing

```
data "aws_ami" "example" {
  most_recent = true
```

data.aws_ami.example

```
variable "image_id" {
  type = string
}
```

var.image_id

```
locals {
  # Ids for multiple sets o
  instance_ids = concat(aws
}
```

local.instance_ids

```
module "vpc" {
  source = "./modules/vpc"
}
```

module.vpc

```
resource "aws_instance" "example" {
  # ...
```

aws_instance.example

Resource Addressing

```hcl
module "vpc" {
  source = "./modules/vpc"
}
```

```hcl
resource "aws_vpc" "main" {
  cidr_block       = "10.0.0.0/16"
  instance_tenancy = "default"

  tags = {
    Name = "main"
  }
}
```

```hcl
output "vpc_id" {
  value = aws_vpc.main.id
}
```

module.vpc.vpc_id

module.vpc.aws_vpc.main

terraform plan -target module.vpc.aws_vpc.main

HashiCorp Terraform

Resource Addressing

```
resource "azurerm_resource_group" "rg" {
  for_each = {
    a_group = "eastus"
    another_group = "westus2"
  }
  name     = each.key
  location = each.value
}
```

azurem_resource_group.rg["a_group"]

```
resource "aws_instance" "server" {
  count = 4 # create four similar EC2 instances

  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
```

aws_instance.server[0]

Why Is This Important?

- When writing Terraform code, resources may depend on other resources to be created first, or depend on their attributes
  - Example: Creating a subnet resource depends on your VPC resource id attribute

```
resource "aws_subnet" "main" {
  vpc_id     = aws_vpc.main.id
  cidr_block = "10.0.1.0/24"

  tags = {
    Name = "Main"
  }
}
```

Why Is This Important?

- When executing a terraform plan or terraform apply, you may want to run it for a particular resource, instead of the entire configuration
  - Example: terraform apply -target aws_subnet.main

```
resource "aws_subnet" "main" {
  vpc_id     = aws_vpc.main.id
  cidr_block = "10.0.1.0/24"

  tags = {
    Name = "Main"
  }
}
```

HashiCorp Terraform