Lab: Dynamic Blocks

A dynamic block acts much like a for expression, but produces nested blocks instead of a complex typed value. It iterates over a given complex value, and generates a nested block for each element of that complex value. You can dynamically construct repeatable nested blocks using a special dynamic block type, which is supported inside resource, data, provider, and provisioner blocks.

- Task 1: Create a Security Group Resource with Terraform
- Task 2: Look at the state without a dynamic block
- Task 3: Convert Security Group to use dynamic block
- Task 4: Look at the state with a dynamic block
- Task 5: Use a dynamic block with Terraform map
- Task 6: Look at the state with a dynamic block using Terraform map

Task 1: Create a Security Group Resource with Terraform

Add an AWS security group resource to our `main.tf`

```
resource "aws_security_group" "main" {
  name   = "core-sg"
  vpc_id = aws_vpc.vpc.id

  ingress {
    description = "Port 443"
    from_port   = 443
    to_port     = 443
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    description = "Port 80"
    from_port   = 80
    to_port     = 80
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```


Task 2: Look at the state without a dynamic block

Run a `terraform apply` followed by a `terraform state list` to view how the security groups are accounted for in Terraform's State.

```
terraform state list

aws_security_group.main
```

```
terraform state show aws_security_group.main
```

```
# aws_security_group.main:
resource "aws_security_group" "main" {
    arn                    = "arn:aws:ec2:us-east-1:508140242758:security-group/sg-00157
    description            = "Managed by Terraform"
    egress                 = []
    id                     = "sg-00157499a6de61832"
    ingress                = [
        {
            cidr_blocks      = [
                "0.0.0.0/0",
            ]
            description      = "Port 443"
            from_port        = 443
            ipv6_cidr_blocks = []
            prefix_list_ids  = []
            protocol         = "tcp"
            security_groups  = []
            self             = false
            to_port          = 443
        },
        {
            cidr_blocks      = [
                "0.0.0.0/0",
            ]
            description      = "Port 80"
            from_port        = 80
            ipv6_cidr_blocks = []
            prefix_list_ids  = []
            protocol         = "tcp"
            security_groups  = []
            self             = false
            to_port          = 80
        },
    ]
    name                   = "core-sg"
    owner_id               = "508140242758"
```


```
    revoke_rules_on_delete = false
    tags_all               = {}
    vpc_id                 = "vpc-0e3a3d76e5feb63c9"
}
```

Task 3: Convert Security Group to use dynamic block

Refactor the `aws_security_group` resource block created above to utilize a dynamic block to built out the repeatable `ingress` nested block that is a part of this resource. We will supply the content for these repeatable blocks via local values to make it easier to read and update moving forward.

```
locals {
  ingress_rules = [{
      port        = 443
      description = "Port 443"
    },
    {
      port        = 80
      description = "Port 80"
    }
  ]
}

resource "aws_security_group" "main" {
  name   = "core-sg"
  vpc_id = aws_vpc.vpc.id

  dynamic "ingress" {
    for_each = local.ingress_rules

    content {
      description = ingress.value.description
      from_port   = ingress.value.port
      to_port     = ingress.value.port
      protocol    = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
  }
}
```

Task 4: Look at the state with a dynamic block

Run a `terraform apply` followed by a `terraform state list` to view how the servers are accounted for in Terraform's State.

```
terraform apply
terraform state list
```

```
aws_security_group.main
```

```
terraform state show aws_security_group.main
```

```
# aws_security_group.main:
resource "aws_security_group" "main" {
    arn                    = "arn:aws:ec2:us-east-1:508140242758:security-group/sg-00157
    description            = "Managed by Terraform"
    egress                 = []
    id                     = "sg-00157499a6de61832"
    ingress                = [
        {
            cidr_blocks      = [
                "0.0.0.0/0",
            ]
            description      = "Port 443"
            from_port        = 443
            ipv6_cidr_blocks = []
            prefix_list_ids  = []
            protocol         = "tcp"
            security_groups  = []
            self             = false
            to_port          = 443
        },
        {
            cidr_blocks      = [
                "0.0.0.0/0",
            ]
            description      = "Port 80"
            from_port        = 80
            ipv6_cidr_blocks = []
            prefix_list_ids  = []
            protocol         = "tcp"
            security_groups  = []
            self             = false
            to_port          = 80
        },
    ]
    name                   = "core-sg"
    owner_id               = "508140242758"
    revoke_rules_on_delete = false
    tags                   = {}
    tags_all               = {}
    vpc_id                 = "vpc-0e3a3d76e5feb63c9"
```


```
}
```

Task 5: Use a dynamic block with Terraform map

Rather then using the local values, we can refactor our dynamic block to utilize a variable named `web_ingress` which is of map. Let's first create the variable of type map, specifying some default values for our ingress rules inside our `variables.tf` file.

```
variable "web_ingress" {
  type = map(object(
    {
      description = string
      port        = number
      protocol    = string
      cidr_blocks = list(string)
    }
  ))
  default = {
    "80" = {
      description = "Port 80"
      port        = 80
      protocol    = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
    "443" = {
      description = "Port 443"
      port        = 443
      protocol    = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
  }
}
```

Then we will refactor our security group to use this variable rather then using local values.

```
resource "aws_security_group" "main" {
  name = "core-sg"

  vpc_id = aws_vpc.vpc.id

  dynamic "ingress" {
    for_each = var.web_ingress
    content {
      description = ingress.value.description
      from_port   = ingress.value.port
      to_port     = ingress.value.port
```


```
      protocol    = ingress.value.protocol
      cidr_blocks = ingress.value.cidr_blocks
    }
  }
}
```

Task 6: Look at the state with a dynamic block using Terraform map

Run a `terraform apply` followed by a `terraform state list` to view how the servers are accounted for in Terraform's State.

```
terraform state list
```

```
terraform state show aws_security_group.main
```

```
# aws_security_group.main:
resource "aws_security_group" "main" {
    arn                    = "arn:aws:ec2:us-east-1:508140242758:security-group/sg-00157
    description            = "Managed by Terraform"
    egress                 = []
    id                     = "sg-00157499a6de61832"
    ingress                = [
        {
            cidr_blocks      = [
                "0.0.0.0/0",
            ]
            description      = "Port 443"
            from_port        = 443
            ipv6_cidr_blocks = []
            prefix_list_ids  = []
            protocol         = "tcp"
            security_groups  = []
            self             = false
            to_port          = 443
        },
        {
            cidr_blocks      = [
                "0.0.0.0/0",
            ]
            description      = "Port 80"
            from_port        = 80
            ipv6_cidr_blocks = []
            prefix_list_ids  = []
            protocol         = "tcp"
            security_groups  = []
            self             = false
```


```
            to_port           = 80
        },
    ]
    name                   = "core-sg"
    owner_id               = "508140242758"
    revoke_rules_on_delete = false
    tags                   = {}
    tags_all               = {}
    vpc_id                 = "vpc-0e3a3d76e5feb63c9"
}
```

Best Practices

Overuse of dynamic blocks can make configuration hard to read and maintain, so it is recommend to use them only when you need to hide details in order to build a clean user interface for a re-usable module. Always write nested blocks out literally where possible.