반복문 count

테라폼은 몇가지 반복문 구성을 제공한다.

- count 매개변수: 리소스를 반복
- for_each 표현식: 리소스 내에서 리소스 및 인라인 블록을 반복
- for 표현식: 리스트와 맵을 반복
- for 문자열 지시어: 문자열 내에서 리스트와 맵을 반복

count 매개변수를 이용한 반복

aws_iam_user 리소스를 사용하여 새 개별 IAM 사용자를 생성한다.

```
provider "aws" {
  region = "ap-northeast-2"
}

resource "aws_iam_user" "example" {
  name = "neo"
}
```

```
# 이 코드는 의사 코드이므로 테라폼에서는 작동하지 않는다.
for (i = 0, i < 3, i++) {
  resource "aws_iam_user" "example" {
    name = "neo"
  }
}
```

테라폼에서는 for 반복문 또는 언어에 내장된 기존의 절차 논리가 없기 때문에 이 구문은 동작하지 않는다.

count를 사용하여 3명의 IAM 사용자를 생성하는 방법은 다음과 같다.

```
resource "aws_iam_user" "example" {
  count = 3
  name  = "neo"
}
```

각 사용자 이름이 유일해야 하는데 이 코드에서는 3명의 IAM 사용자 모두 같은 이름을 가지므로 오류가 발생한다.

일반적인 for 반복문에서는 인덱스를 사용하여 각 사용자에게 고유한 이름을 지정할 수 있다.

```
for (i = 0, i < 3, i++) {
  resource "aws_iam_user" "example" {
    name = "neo.${i}"
  }
}
```

테라폼에서 이와 같은 작업을 수행하려면 count.index를 사용하여 반복문 안에 있는 각각의 반복을 가리키는 인덱스를 얻을 수 있다.

```
resource "aws_iam_user" "example" {
  count = 3
  name  = "neo.${count.index}"
}
```

실행

```
$ terraform init
...
Terraform will perform the following actions:

  # aws_iam_user.example[0] will be created
  + resource "aws_iam_user" "example" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "neo.0"
      + path          = "/"
      + tags_all      = (known after apply)
      + unique_id     = (known after apply)
    }

  # aws_iam_user.example[1] will be created
  + resource "aws_iam_user" "example" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "neo.1"
      + path          = "/"
      + tags_all      = (known after apply)
      + unique_id     = (known after apply)
    }

  # aws_iam_user.example[2] will be created
  + resource "aws_iam_user" "example" {
      + arn           = (known after apply)
      + force_destroy = false
      + id            = (known after apply)
      + name          = "neo.2"
      + path          = "/"
      + tags_all      = (known after apply)
```

```
      + unique_id     = (known after apply)
    }
...
```

서로 다른 이름을 가지는 사용자를 생성하려면 다음과 같은 코드를 사용한다.

```
resource "aws_iam_user" "example" {
  count = 3
  name  = "neo.${count.index}"
}

variable "user_names" {
  description = "Create IAM users with these names"
  type        = list(string)
  default     = ["neo", "trinity", "morpheus"]
}
```

배열 조회 구문

테라폼에서 배열의 배열 요소를 찾는 구문은 다른 대부분 프로그래밍 언어에서 사용하는 구조와 유사하다.

ARRAY[<INDEX>]

예를 들어 다음은 var.user_names의 인덱스 1에서 요소를 찾는 방법이다.

var.user_names[1]

length함수

주어진 ARRAY의 항목 수를 반환하는 함수이다. 문자열 및 맵을 대상으로 동작한다.

length(<ARRAY>)

main.tf

```
provider "aws" {
  region = "ap-northeast-2"
}

resource "aws_iam_user" "example" {
  count = length(var.user_names)
  name  = var.user_names[count.index]
}
```

variable.tf

```
variable "user_names" {
  description = "Create IAM users with these names"
  type        = list(string)
  default     = ["neo", "trinity", "morpheus"]
}
```

실행

```
$ terraform apply
...
Terraform will perform the following actions:

  # aws_iam_user.example[0] will be updated in-place
  ~ resource "aws_iam_user" "example" {
        id            = "neo.0"
      ~ name          = "neo.0" -> "neo"
        tags          = {}
        # (5 unchanged attributes hidden)
    }

  # aws_iam_user.example[1] will be updated in-place
  ~ resource "aws_iam_user" "example" {
        id            = "neo.1"
      ~ name          = "neo.1" -> "trinity"
        tags          = {}
        # (5 unchanged attributes hidden)
    }

  # aws_iam_user.example[2] will be updated in-place
  ~ resource "aws_iam_user" "example" {
        id            = "neo.2"
      ~ name          = "neo.2" -> "morpheus"
        tags          = {}
        # (5 unchanged attributes hidden)
    }
...
```

리소스에 count를 사용한 후에는 하나의 리소스가 아니라 리소스의 배열이 된다.
aws_iam_user.example은 이제 IAM 사용자의 배열이므로 표준 구문을 사용하여 해당 리소스인 <PROVIDER>_<TYPE>.<NAME>.<ATTRIBUTE>에서 속성을 읽는 대신 동일한 배열 조회 구문을 이용해 배열에서 인덱스를 지정함으로써 IAM 사용자를 명시해야 한다.

<PROVIDER>_<TYPE>.<NAME>[INDEX].ATTRIBUTE

예를 들어 IAM 사용자 중 하나의 ARN을 출력 변수로 제공하려면 다음과 같이 작성한다.

```
output "neo_arn" {
  value       = aws_iam_user.example[0].arn
  description = "The ARN for user neo"
}
```

모든 사용자의 ARN을 원하면 인덱스 대신 스플랫연산자인 “*”를 사용한다.

```
output "all_arns" {
  value       = aws_iam_user.example[*].arn
  description = "The ARNs for all users"
}
```

실행

```
$ tf apply
...
Outputs:

all_arns = [
  "arn:aws:iam::123456789012:user/neo",
  "arn:aws:iam::123456789012:user/trinity",
  "arn:aws:iam::123456789012:user/morpheus",
]
neo_arn = "arn:aws:iam::123456789012:user/neo"
```

count에는 유용성을 저해하는 두 가지 제약이 있다.

첫 번째로 count를 사용하여 전체 리소스를 반복할 수는 있지만 리소스 내에서 인라인 블록을 반복할 수는 없다. 인라인 블록은 다음 형식의 리소스 내에서 설정한 인수이다.

```
resource "xxx" "yyy" {
  <NAME> {
    [CONFIG...]
  }
}
```

여기서 NAME은 tag와 같은 인라인 블록의 이름이고 CONFIG는 key 또는 value 같이 해당 인라인 블록에 특정한 하나 이상의 인수로 구성된다. 예를 들어 aws_autoscaling_group 리소스에서 태그를 설정하는 방법은 다음과 같다.

```
resource "aws_autoscaling_group" "example" {
  launch_configuration = aws_launch_configuration.example.name
  vpc_zone_identifier  = data.aws_subnet_ids.default.ids
  target_group_arns    = [aws_lb_target_group.asd.arn]
  health_check_type    = "ELB"

  min_size = var.min_size
  max_size = var.max_size

  tag {
```

```
    key                 = "Name"
    value               = var.cluster_name
    propagate_at_launch = true
  }
}
```

각각의 tag를 사용하려면 key, value, propagate_at_launch에 대한 값으로 새 인라인 블록을 만들어야 한다. count 매개변수를 사용하여 이러한 태그를 반복하고 동적인 인라인 tag블록을 생성하려고 시도할 수도 있지만 인라인 블록 내에서 count 사용은 지원하지 않는다.

두 번째 제약은 변경하려고 할 때 발생한다. 앞서 생성한 IAM 사용자 목록을 확인한다.

```
variable "user_names" {
  description = "Create IAM users with these names"
  type        = list(string)
  default     = ["neo", "trinity", "morpheus"]
}
```

이 목록에서 "trinity"를 제거했다고 가정하고 terraform plan을 실행하면 다음과 같은 결과가 나온다.

```
$ terraform plan                                                                [14:30:23]
aws_iam_user.example[1]: Refreshing state... [id=trinity]
aws_iam_user.example[2]: Refreshing state... [id=morpheus]
aws_iam_user.example[0]: Refreshing state... [id=neo]

Terraform used the selected providers to generate the following execution plan. Resource actions are
indicated with the following symbols:
  ~ update in-place
  - destroy

Terraform will perform the following actions:

  # aws_iam_user.example[1] will be updated in-place
  ~ resource "aws_iam_user" "example" {
        id            = "trinity"
      ~ name          = "trinity" -> "morpheus"
        tags          = {}
        # (5 unchanged attributes hidden)
    }

  # aws_iam_user.example[2] will be destroyed
  # (because index [2] is out of range for count)
  - resource "aws_iam_user" "example" {
      - arn           = "arn:aws:iam::257307634175:user/morpheus" -> null
      - force_destroy = false -> null
      - id            = "morpheus" -> null
      - name          = "morpheus" -> null
      - path          = "/" -> null
      - tags          = {} -> null
      - tags_all      = {} -> null
      - unique_id     = "AIDATX2F25H7RHWFL3BLB" -> null
    }

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

plan의 출력 내용은 테라폼이 "trinity" IAM 사용자를 삭제하는 대신 “trinity” IAM 사용자의 이름을 "morpheus"로 바꾸고 원래 “morpheus”였던 사용자를 삭제할 것이라고 나타낸다.

리소스에 count 매개 변수를 사용하면 해당 리소스는 리소스 리스트 또는 리소스 배열이 된다. 테라폼은 해당 배열의 위치(인덱스)로 배열 내의 각 리소스를 식별한다. 즉, 처음 3명의 사용자 이름으로 apply를 실행한 수 이러한 IAM 사용자에 대한 테라폼의 내부 표현은 다음과 같다.

```
aws_iam_user.example[0]: neo
aws_iam_user.example[1]: trinity
aws_iam_user.example[2]: morpheus
```

배열의 중간에서 항목을 제거하면 모든 항목이 1칸씩 앞으로 당겨진다. 따라서 2개의 버킷 이름으로 plan 명령을 실행한 후의 테라폼 내부 표현은 다음과 같다.

```
aws_iam_user.example[0]: neo
aws_iam_user.example[1]: morpheus
```