

Terraform resource meta arguments

[Terraform](#) resource Meta-Arguments can be useful while setting up your cloud infrastructure. The resource arguments `depends_on`, `count`, `for_each`, `provider`, `lifecycle` has some features such as -

1. You can create multiple `aws_resource` using the `count`
2. `for_each` can be used for iteration and can also help you to create multiple `aws_resource` using the same block
3. `provider` is used for overriding terraform default behavior using the `alias`
4. With `lifecycle`, you can **prevent destroy**, **create resource after destroy** and **ignore changes to be saved inside state**

We will look into each resource meta arguments in a bit more detail along with the example -

Table of Content

- `count`
- `for_each`
- `provider`
- `lifecycle`

1. Count

As the name suggests `count` can be used inside the `aws_instance` block to specify how many resources you would like to create.

Here is an example in which we are going to spin 2 `aws_instance` -

```
provider "aws" {
  region      = "eu-central-1"
  access_key  = "AKIATQ37NXB2HS7IVM5R"
  secret_key  = "MJy5JX6HIqHwP9gLA+22kffS/jiDsMo2XLP9mZn"
}

resource "aws_instance" "ec2_example" {

  count          = 2
  ami            = "ami-0767046d1677be5a0"
  instance_type = "t2.micro"

  tags = {
    Name = "Terraform EC2"
  }
}
```

The benefit of Count: -

1. You do not need to write the same **resource block** again if you want to create more than one **resource**.
2. It can also be used with modules and any kind of resource type available in terraform.

2. for_each

`for_each` can also be used for creating similar kinds of resources instead of creating a writing duplicate terraform block.

Here is one more example of a terraforming block with `for_each` -

```

provider "aws" {
  region      = "eu-central-1"
  access_key  = "AKIATQ37NXB2HS7IVM5R"
  secret_key  = "MJy5JX6HIqHwP9gLAv+22kffS/jiDsMo2XLp9mZn"
}

resource "aws_instance" "ec2_example" {

  for_each = {
    instance1 = "t2.micro"
    instance2 = "t2.micro"
  }

  ami          = "ami-0767046d1677be5a0"
  instance_type = each.value

  tags = {
    Name = "Terraform ${each.key}"
  }
}

```

As you can see in the above terraform block we have created 2 key-value pair `instance1 = t2.micro` and `instance2 = t2.micro` inside the `for_each` block.

The next question is **how to use key-value pair defined inside for_each?**

The answer - It is very simple **you can just simply type each.value** and it will iterate over the values.

Here is a screenshot from aws after starting the `aws_instance`

The screenshot shows the AWS Management Console 'Instances' page. It displays two instances in a 'running' state, both of type 't2.micro'. The instances are named 'Terraform inst...' and have unique IDs. The console includes a search bar, a filter for 'Instance state: running', and a 'Clear filters' button.

	Name	Instance ID	Instance state	Instance type
<input type="checkbox"/>	Terraform inst...	i-0502f5b0a04a42abd	Running	t2.micro
<input type="checkbox"/>	Terraform inst...	i-00d9eefbf5bff5128	Running	t2.micro

3. provider

This meta argument is one of my favorite because it lets you override Terraform's default behavior. It can help you to create multiple configurations for a single cloud service provider (.e.g - [AWS](#), [GCP](#)).

One simple example would be - "Suppose you want to create two `aws_instance` one in `eu-central-1` and another one in `eu-nort-1` region, would it be possible for you to create in single `main.tf` file?"

Well, **YES** you can do that but to achieve this you need to use `provider` inside your terraform file along with the `alias`.

Here are the steps for using provider meta argument -

Step 1 - First create a simple `provider` block in your terraform file -

```
provider "aws" {  
  region      = "eu-central-1"  
  access_key  = "AKIATQ37NXB2HS7IVM5R"  
  secret_key  = "MJy5JX6HIqHwP9gLAv+22kffS/jiDsMo2XLP9mZn"  
}
```

Step 2 - Create one more provider block but with an additional argument `alias`

```
provider "aws" {  
  alias      = "north"  
  region     = "eu-north-1"  
  access_key = "AKIATQ37NXB2HS7IVM5R"  
  secret_key = "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"  
}
```

Step 3 - Here is the final terraform file in which we are going to create 2 aws instances with one in eu-north-1 region and another in eu-central-1 region -

```

provider "aws" {
  alias    = "north"
  region   = "eu-north-1"
  access_key = "AKIATQ37NXB2HS7IVM5R"
  secret_key = "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"
}

provider "aws" {
  region      = "eu-central-1"
  access_key  = "AKIATQ37NXB2HS7IVM5R"
  secret_key  = "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"
}

resource "aws_instance" "ec2_eu_north" {
  provider      = aws.north
  ami           = "ami-0ff338189efb7ed37"
  instance_type = "t3.micro"
  count         = 1
  tags = {
    Name = "Terraform EC2"
  }
}

resource "aws_instance" "ec2_eu_central" {
  ami           = "ami-0767046d1677be5a0"
  instance_type = "t2.micro"
  count         = 1
  tags = {
    Name = "Terraform EC2"
  }
}

```

So as you can see a provider with an alias can be useful in case you want to spin/start multiple instances in the different regions of your cloud service provider.

4. lifecycle

This meta argument is a lifesaver if you are working in the production environment where you have to be very careful so that you do not accidentally destroy any resource.

With `lifecycle` meta tag, you can make sure that certain resources should not be deleted and you can also create a new similar resource after the `terraform destroy` command.

There are three arguments that you can pass inside the `lifecycle` block -

1. **create_before_destroy** - Once you set this argument the resource will be created once again after you issue the `terraform destroy` command
2. **prevent_destroy** - It prevents from destroying your terraform resource, once you set this terraform argument then the resource can not be destroyed

3. **ignore_changes** - Suppose you have manually made some changes on AWS or [GCP](#) but you want to prevent those changes to be saved inside your terraform `terraform.tfstate` file then you can use `ignore_changes` arguments.

Here is the sample code snippet (*Please uncomment the arguments as per your need*) -

```
provider "aws" {
  region      = "eu-central-1"
  access_key  = "AKIATQ37NXB2HS7IVM5R"
  secret_key  = "MJy5JX6HIqHwP9gLAv+22kffS/jiDsMo2XLP9mZn"
}

resource "aws_instance" "ec2_example" {

  count          = 2
  ami            = "ami-0767046d1677be5a0"
  instance_type = "t2.micro"

  tags = {
    Name = "Terraform EC2"
  }

  lifecycle {
    create_before_destroy = true
    #prevent_destroy       = true
    #ignore_changes        = [tags]
  }
}
```