

Terraform Count

Terraform's count feature is used to create multiple instances of a resource or module based on a numeric value or Boolean condition. It allows you to define multiple copies of a resource or module using a single configuration block.

The count attribute can be set to a number or a Boolean expression that evaluates to either true or false. Here's how you can use count:

```
resource "aws_instance" "web" {  
  ami           = "ami-0ded8326293d3201b"  
  instance_type = "t2.micro"  
  count         = 3  
  tags = {  
    Name = "OS ${count.index}"  
  }  
}
```

If count is set to 3, it will create three EC2 instances, and if it is set to 0, no instances will be created.

And while launching the Instance It will name the Instance as

OS 1

OS 2

OS 3

```
terraform.exe apply -auto-approve
```

```
PublicIP = [  
  "43.205.208.208",  
  "13.235.45.14",  
  "13.235.18.20",  
]
```

Terraform Locals

In Terraform, locals is a block used to define named local values within your configuration. Local values are similar to variables, but they are calculated and defined within your Terraform configuration and are not meant to be exposed as inputs or outputs.

The locals block allows you to create intermediate values that can be used within the configuration to simplify expressions, avoid repetition, or perform calculations. These values are evaluated only once during the Terraform execution and do not change over the course of the configuration run.

Using the locals block helps to keep your configuration DRY (Don't Repeat Yourself), reduces duplication, and makes it easier to manage and update values consistently across your Terraform code.

Remember that local values are not meant to be changed dynamically during the execution of your configuration. If you need values that can change during execution, you should use input variables.

```
variable "Instance-Environment" {
  type = list(string)
  default = [ "Dev", "Test", "Prod" ]
}
variable "Enter-Number" {
  type = number
}
variable "user_input-Greeting" {
  type = bool
}
# Initializing Local Variables
locals {
  instance_count = var.user_input-Greeting ? var.Enter-Number : 0
  Welcome = "Hello User, Infrastructure is Ready"
  Terminate = "By User, Come Back Soon 😊"
}
```

1. **instance_count**: This local value calculates the number of instances based on the condition provided by **user_input-Greeting** and the value of **Enter-Number**.
2. **Welcome**: A string local value that holds a welcome message for the user.
3. **Terminate**: A string local value that holds a message indicating termination or farewell.

Remember, these local values will be calculated once during the Terraform execution and will remain constant throughout the configuration run. You can reference them in your resources, outputs, or other parts of the configuration to ensure consistency and avoid repetition.

```
instance_count = var.user_input-Greeting ? var.Enter-Number : 0
```

Here I am Taking User Input as a boolean (**True or False**). If user enter True, then it will take number as input from user. And launch that much Instance in AWS Cloud.

```
PS C:\Users\Himanshu Kumar\Documents\Terraform\Day 3\INSTANCE> .\terraform.exe apply -auto-approve
var.Enter-Number
  Enter a value: 3

var.user_input-Greeting
  Enter a value: true
```

Here I have mentioned 3 as Number.

```
Apply complete! Resources: 3 added, 0 changed, 0 destroyed.
```

```
Outputs:
```

```
PublicIP = [
  "43.205.129.6",
  "3.110.178.26",
  "43.205.112.215",
]
Value-IS = 3
greeting = "Hello User, Infrastructure is Ready"
```

Now Terraform Launched 3 EC2 Instance, and, I Print their Public IP and also a greeting Message.

If my requirement is over, then we can destroy the infrastructure, also print the Message for the same.

```
Changes to Outputs:
- PublicIP = [] -> null
- Value-IS = 0 -> null
- greeting = "By User, Come Back Soon 😊" -> null
```

Terraform Taint

When we are using Terraform then we need to mention our Desired State. In which we need to mention what we are looking for. For that we need to write the code like I want 1 EC2 Instance 2 Gb of Storage, T2.micro as instance type, 1 GB of RAM.

But after this If we have manually change something in the Instance. For example, Added Security Group in the instance manually. And if any problem occurs in the instance, then it is harder to find the challenges all the issue. Then in this case, the Desired State is not matched with the current state. For solving this issue, we have the concept of terraform taint.

Here Terraform will destroy the complete environment and launch everything from scratch.

```
resource "aws_instance" "web" {  
  ami          = "ami-0ded8326293d3201b"  
  instance_type = "t2.micro"  
  tags = {  
    Name = "OS"  
  }  
}
```

Here I have Just Launched a Simple Instance. And after launching it have the default security group. So I am going to change the security group manually

Associated security groups
Add one or more security groups to the network interface. You can also remove security groups.

Security groups associated with the network interface (eni-028eddefb5a00be06)

Security group name	Security group ID	
default	sg-0f87a60e53bac819b	<input type="button" value="Remove"/>

I have removed the default security group and attached mine

Associated security groups

Add one or more security groups to the network interface. You can also remove security groups.

Q Select security groups

Add security group

Security groups associated with the network interface (eni-028eddefb5a00be06)

Security group name

Security group ID

SGforTF

sg-010f683d9801ca435

Remove

and when I run the terraform taint command for this resource

```
PS C:\Users\Himanshu Kumar\Documents\Terraform\Day 3\INSTANCE> .\terraform.exe taint aws_instance.web
Resource instance aws_instance.web has been marked as tainted.
PS C:\Users\Himanshu Kumar\Documents\Terraform\Day 3\INSTANCE>
```

EC2 Instance
Storage 8 GB
T2.micro
Ram 1 Gb

Desire State

EC2 Instance
Storage 8 GB
T2.micro
Ram 1 Gb
Security Group

Current State

Terraform apply -auto-approve

Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since the last "terraform apply" which may have affected this plan:

```
# aws_instance.web has changed
~ resource "aws_instance" "web" {
  id = "i-05bd7a1a1cfe0c8f9"
  ~ security_groups = [
    - "default",
    + "SGforTF",
  ]
}
```

Terraform detects the changes and destroying the current environment and launching the new environment.

Associated security groups

Add one or more security groups to the network interface. You can also remove security groups.

Q Select security groups

Add security group

Security groups associated with the network interface (eni-024dd6627b65a1274)

Security group name

Security group ID

default

sg-0f87a60e53bac819b

Remove

Terraform plan

Sometimes we have written our entire code for launching the environment. But if someone will make the changes in that code for example.

EC2 Instance Storage 8 GB T2.micro Ram 1 Gb
My Requirement

EC2 Instance Storage 30 GB T2.large Ram 4 Gb
Code Change

So, it is better to create the output of plan while writing your requirement. So, if someone make any changes to your code then we can have the option to launch the code from using that plan.

Directory: C:\Users\Himanshu Kumar\Documents\Terraform\Day 3\INSTANCE

Mode	LastWriteTime	Length	Name
d----	03-08-2023 18:46		.terraform
-a----	03-08-2023 18:46	1377	.terraform.lock.hcl
-a----	04-08-2023 15:22	427	main.tf
-a----	04-08-2023 15:22	235	output.tf
-a----	04-08-2023 13:47	332	Providers.tf
-a----	20-07-2023 12:44	63506088	terraform.exe
-a----	04-08-2023 15:30	181	terraform.tfstate
-a----	04-08-2023 15:30	5066	terraform.tfstate.backup
-a----	04-08-2023 15:22	406	variable.tf

Not Created any plan

Terraform plan -out EC2Instance

```
Saved the plan to: EC2Instance

To perform exactly these actions, run the following command to apply:
terraform apply "EC2Instance"
```

The File is stored here. And when we try to open the file the it will be on binary form.

Mode	LastWriteTime	Length	Name
d----	03-08-2023 18:46		.terraform
-a----	03-08-2023 18:46	1377	.terraform.lock.hcl
-a----	04-08-2023 15:32	4142	EC2Instance
-a----	04-08-2023 15:22	427	main.tf
-a----	04-08-2023 15:22	235	output.tf
-a----	04-08-2023 13:47	332	Providers.tf
-a----	20-07-2023 12:44	63506088	terraform.exe
-a----	04-08-2023 15:30	181	terraform.tfstate
-a----	04-08-2023 15:30	5066	terraform.tfstate.backup
-a----	04-08-2023 15:22	406	variable.tf

Terraform LOG

Remember that detailed logs can be useful for troubleshooting but can also generate a lot of information, so it's recommended to use them when necessary and for specific debugging purposes.

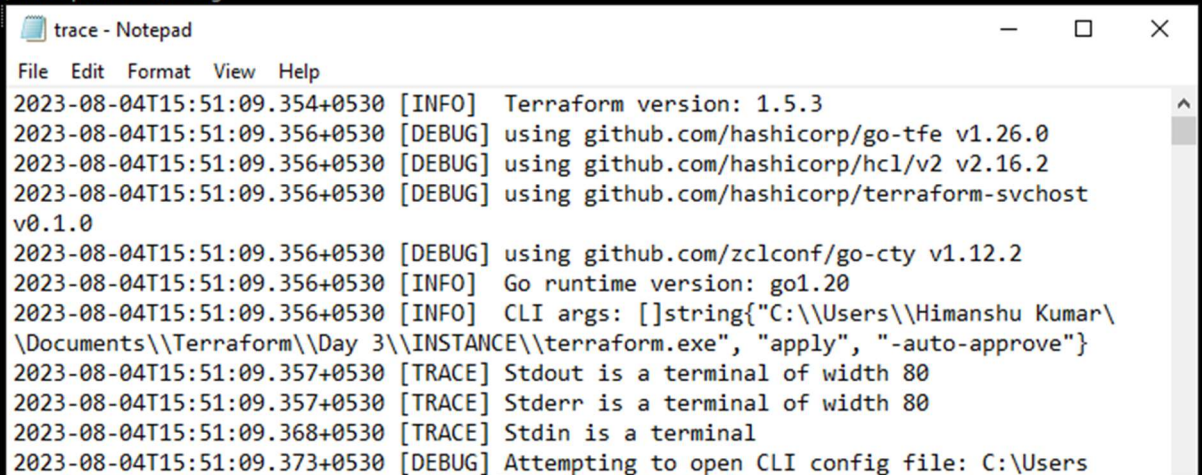
```
Himanshu Kumar@AICPL-D010 MINGW64 ~/Documents/Terraform/Day 3/INSTANCE (main)
$ export TF_LOG_PATH=trace.log
```

Export the Log in the Specific file.

When we run the terraform apply command then it will create the detailed log.

```
Himanshu Kumar@AICPL-D010 MINGW64 ~/Documents/Terraform/Day 3/INSTANCE (main)
$ ./terraform.exe apply -auto-approve
aws_instance.web: Refreshing state... [id=i-011c2d6bc59b45020]
```

```
Himanshu Kumar@AICPL-D010 MINGW64 ~/Documents/Terraform/Day 3/INSTANCE (main)
$ notepad trace.log
```



```
File Edit Format View Help
2023-08-04T15:51:09.354+0530 [INFO] Terraform version: 1.5.3
2023-08-04T15:51:09.356+0530 [DEBUG] using github.com/hashicorp/go-tfe v1.26.0
2023-08-04T15:51:09.356+0530 [DEBUG] using github.com/hashicorp/hcl/v2 v2.16.2
2023-08-04T15:51:09.356+0530 [DEBUG] using github.com/hashicorp/terraform-svchost
v0.1.0
2023-08-04T15:51:09.356+0530 [DEBUG] using github.com/zclconf/go-cty v1.12.2
2023-08-04T15:51:09.356+0530 [INFO] Go runtime version: go1.20
2023-08-04T15:51:09.356+0530 [INFO] CLI args: []string{"C:\\Users\\Himanshu Kumar\\
\\Documents\\Terraform\\Day 3\\INSTANCE\\terraform.exe", "apply", "-auto-approve"}
2023-08-04T15:51:09.357+0530 [TRACE] Stdout is a terminal of width 80
2023-08-04T15:51:09.357+0530 [TRACE] Stderr is a terminal of width 80
2023-08-04T15:51:09.368+0530 [TRACE] Stdin is a terminal
2023-08-04T15:51:09.373+0530 [DEBUG] Attempting to open CLI config file: C:\\Users
```

If we facing any issues, then it is very easy to solve those issues by seeing into log file.

TERRAFORM CONSOLE

Terraform Console is an interactive command-line tool that provides an environment for evaluating and testing expressions within your Terraform configuration. It allows you to experiment with Terraform's expression language, query the values of variables and resources, and perform calculations or transformations before applying them in your actual configurations.

Here's how you can use the Terraform Console:

1. Open a terminal or command prompt.
2. Navigate to the directory containing your Terraform configuration files.
3. Run the following command to enter the Terraform Console:

terraform console

Once inside the Terraform Console, you can perform various tasks:

max Function

max takes one or more numbers and returns the greatest number from the set.

```
PS C:\Users\Himanshu Kumar\Documents\Terraform\Day 3\INSTANCE> .\terraform.exe console
> max(5,12,20)
20
```

element Function

The index is zero-based. This function produces an error if used with an empty list. The index must be a non-negative integer.

```
> element(["a", "b", "c"],0)
"a"
> |
```

If the given index is greater than the length of the list then the index is "wrapped around" by taking the index modulo the length of the list

Join Function

join produces a string by concatenating all of the elements of the specified list of strings with the specified separator.

```
> join(", ", ["Foo", "Boo", "Too"])
"Foo,Boo,Too"
> |
```

Split Function

split produces a list by dividing a given string at all occurrences of a given separator.

```
> split(".", "192.168.130.65")[2]
"130"
> |
```

```
> split(".", "192.168.130.65")
tolist([
  "192",
  "168",
  "130",
  "65",
])
> |
```

Like this Terraform have more function you can explore it from the mentioned link

<https://developer.hashicorp.com/terraform/language/functions>

www.linkedin.com/in/kumar--himanshu/