Using the Terraform Map Function

How many times have you had something you're building in Terraform where you use dynamic counting or things that need you to map one value to another?

One clear example I had very recently for my fully automated EKS lab using Terraform for AWS was the need to distribute my nodes across availability zones. How can I easily have each node be assigned to a specific AZ without doing a bunch of crazy code inside Terraform?

Terraform makes this easy with two simple features which are **count** parameter and **map** function. Let's dig in to see how these work together!

Map All the Things!

Well, how about we start with mapping 3 things. I have a simple situation that we are going to use which is to map 3 integers to 3 text strings. We call this our **az_map** because it will become an Availability Zone map within an AWS Region.

Each integer for 0, 1, and 2, are mapped to default values which are a, b, and c, respectively. Simple enough with this code:

```
variable "az_map" {
        type = map

        default = {
                0 = "a"
                1 = "b"
                2 = "c"
        }
}
```

Now the fun is where we make this value mapping work in our Terraform code. The above can go anywhere but I like to keep in it a **vars.tf** file just so I know where all my variables are.

Let me COUNT the Ways

Our use-case is that we want 3 nodes for a cluster (EKS nodes in this case) and we want them to the distributed for resiliency across 3 availability zones.

We could statically assign the AZ list…but that's no fun (and not good practice). We know that each AWS Region has at least 3 AZs. The 3-node deployment only needs a simple mapping of a counter to the A/B/C Availability Zone lists for our subnets and then our node deployment.

e.g. US-east-2 is our Region and contains US-east-2a, US-east-2b, and US-east-2c (and more)

You should use variables wherever possible, and our Region is defined by **var.aws_region** which will be US-east-2 using our example.

Just create your AZ configuration to pull in the variable and

```
availability_zone = "${var.aws_region}${var.az_map[count.index]}"
```

Next we wrap this all inside a count parameter of 3 so that the code block is iterated over 3 times.

The full code of this code block is here (and also on Github):

```
resource "aws_subnet" "eks-lab-pub" {
    count = 3
    vpc_id = aws_vpc.eks-lab-vpc.id
    cidr_block = "10.0.${count.index}.0/24"
    map_public_ip_on_launch = "true"
    availability_zone = "${var.aws_region}${var.az_map[count.index]}"

    tags = {
        Name = "eks-lab"
        Terraform = "true"
        Turbonomic = "true"
        "kubernetes.io/cluster/eks-lab" = "shared"
        "kubernetes.io/role/elb" = "1"
        net = "public"
        }
```

```
}
```

The result will be AZs defined for 0,1,2 as a,b,c, which creates US-east-2a, US-east-2B, and US-east-2C. We even assign the networks using the **${count.index}** to define the third octet of the IPv4 range. How cool is that?!

Now you can iterate over those same 3 subnets for your nodes, security groups, or anything that required the mapping across Availability Zones.

Hopefully you find this example helpful. Try out a few different methods as needed and you will be able to put **map** as a staple for your Terraform configurations in a variety of use-cases.