statement, and deployment techniques, as well as those related to more general problems that affect Terraform as a whole:

- `count` and `for_each` have limitations.
- Zero-downtime deployment has limitations.
- Valid plans can fail.
- Refactoring can be tricky.

count and for_each Have Limitations

In the examples in this chapter, you made extensive use of the `count` parameter and `for_each` expressions in loops and if-statements. This works well, but there's an important limitation that you need to be aware of: you cannot reference any resource outputs in `count` or `for_each`.

Imagine that you want to deploy multiple EC2 Instances, and for some reason you didn't want to use an ASG. The code might look like this:

```
resource "aws_instance" "example_1" {
  count         = 3
  ami           = "ami-0fb653ca2d3203ac1"
  instance_type = "t2.micro"
}
```

Because `count` is being set to a hardcoded value, this code will work without issues, and when you run `apply`, it will create three EC2 Instances. Now, what if you want to deploy one EC2 Instance per Availability Zone (AZ) in the current AWS region? You could update your code to fetch the list of AZs using the `aws_availability_zones` data source and use the `count` parameter and array lookups to "loop" over each AZ and create an EC2 Instance in it:

```
resource "aws_instance" "example_2" {
  count             =
length(data.aws_availability_zones.all.names)
  availability_zone =
```

```
data.aws_availability_zones.all.names[count.index]
  ami               = "ami-0fb653ca2d3203ac1"
  instance_type     = "t2.micro"
}

data "aws_availability_zones" "all" {}
```

Again, this code works just fine, since `count` can reference data sources without problems. However, what happens if the number of instances you need to create depends on the output of some resource? The easiest way to experiment with this is to use the `random_integer` resource, which, as you can probably guess from the name, returns a random integer:

```
resource "random_integer" "num_instances" {
  min = 1
  max = 3
}
```

This code generates a random integer between 1 and 3. Let's see what happens if you try to use the `result` output from this resource in the `count` parameter of your `aws_instance` resource:

```
resource "aws_instance" "example_3" {
  count         = random_integer.num_instances.result
  ami           = "ami-0fb653ca2d3203ac1"
  instance_type = "t2.micro"
}
```

If you run `terraform plan` on this code, you'll get the following error:

```
Error: Invalid count argument

  on main.tf line 30, in resource "aws_instance" "example_3":
  30:   count         = random_integer.num_instances.result

The "count" value depends on resource attributes that cannot be
determined
until apply, so Terraform cannot predict how many instances will
be created.
To work around this, use the -target argument to first apply only
```