PROVISIONERS VERSUS USER DATA

You've now seen two different ways to execute scripts on a server using Terraform: one is to use a `remote-exec` provisioner, and the other is to use a User Data script. I've generally found User Data to be the more useful tool for the following reasons:

- A `remote-exec` provisioner requires that you open up SSH or WinRM access to your servers, which is more complicated to manage (as you saw earlier with all the security group and SSH key work) and less secure than User Data, which solely requires AWS API access (which you must have anyway when using Terraform to deploy to AWS).
- You can use User Data scripts with ASGs, ensuring that all servers in that ASG execute the script during boot, including servers launched due to an auto scaling or auto recovery event. Provisioners take effect only while Terraform is running and don't work with ASGs at all.
- The User Data script can be seen in the EC2 console (select an Instance, click Actions → Instance Settings → View/Change User Data), and you can find its execution log on the EC2 Instance itself (typically in */var/log/cloud-init*.log*), both of which are useful for debugging and neither of which is available with provisioners.

The only real advantage to using a provisioner to execute code on an EC2 Instance is that User Data scripts are limited to a length of 16 KB, whereas provisioner scripts can be arbitrarily long.

Provisioners with null_resource

Provisioners can be defined only within a resource, but sometimes, you want to execute a provisioner without tying it to a specific resource. You can do this using something called the `null_resource`, which acts just like a normal Terraform resource, except that it doesn't create anything. By

defining provisioners on the `null_resource`, you can run your scripts as part of the Terraform lifecycle but without being attached to any "real" resource:

```
resource "null_resource" "example" {
  provisioner "local-exec" {
    command = "echo \"Hello, World from $(uname -smp)\""
  }
}
```

The `null_resource` even has a handy argument called `triggers`, which takes in a map of keys and values. Whenever the values change, the `null_resource` will be re-created, therefore forcing any provisioners within it to be reexecuted. For example, if you want to execute a provisioner within a `null_resource` every single time you run `terraform apply`, you could use the `uuid()` built-in function, which returns a new, randomly generated UUID each time it's called, within the `triggers` argument:

```
resource "null_resource" "example" {
  # Use UUID to force this null_resource to be recreated on every
  # call to 'terraform apply'
  triggers = {
    uuid = uuid()
  }

  provisioner "local-exec" {
    command = "echo \"Hello, World from $(uname -smp)\""
  }
}
```

Now, every time you call `terraform apply`, the `local-exec` provisioner will execute:

```
$ terraform apply

(...)

null_resource.example (local-exec): Hello, World from Darwin
x86_64 i386
```