Finally, pass through the important outputs from the `asg-rolling-deploy` and `alb` modules as outputs of the `hello-world-app` module:

```
output "alb_dns_name" {
  value       = module.alb.alb_dns_name
  description = "The domain name of the load balancer"
}

output "asg_name" {
  value       = module.asg.asg_name
  description = "The name of the Auto Scaling Group"
}

output "instance_security_group_id" {
  value       = module.asg.instance_security_group_id
  description = "The ID of the EC2 Instance Security Group"
}
```

This is function composition at work: you're building up more complicated behavior (a "Hello, World" app) from simpler parts (ASG and ALB modules).

Testable Modules

At this stage, you've written a whole lot of code in the form of three modules: `asg-rolling-deploy`, `alb`, and `hello-world-app`. The next step is to check that your code actually works.

The modules you've created aren't root modules meant to be deployed directly. To deploy them, you need to write some Terraform code to plug in the arguments you want, set up the `provider`, configure the `backend`, and so on. A great way to do this is to create an *examples* folder that, as the name suggests, shows examples of how to use your modules. Let's try it out.

Create *examples/asg/main.tf* with the following contents:

```
provider "aws" {
  region = "us-east-2"
```

```
}

module "asg" {
  source = "../../modules/cluster/asg-rolling-deploy"

  cluster_name  = var.cluster_name
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t2.micro"

  min_size           = 1
  max_size           = 1
  enable_autoscaling = false

  subnet_ids        = data.aws_subnets.default.ids
}

data "aws_vpc" "default" {
  default = true
}

data "aws_subnets" "default" {
  filter {
    name   = "vpc-id"
    values = [data.aws_vpc.default.id]
  }
}

data "aws_ami" "ubuntu" {
  most_recent = true
  owners      = ["099720109477"] # Canonical

  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-focal-20.04-amd64-
server-*"]
  }
}
```

This bit of code uses the `asg-rolling-deploy` module to deploy an ASG of size 1. Try it out by running `terraform init` and `terraform apply` and checking to see that it runs without errors and actually spins up an ASG. Now, add in a *README.md* file with these instructions, and suddenly this tiny little example takes on a whole lot of power. In just several files and lines of code, you now have the following:

A manual test harness

> You can use this example code while working on the `asg-rolling-deploy` module to repeatedly deploy and undeploy it by manually running `terraform apply` and `terraform destroy` to check that it works as you expect.

An automated test harness

> As you will see in Chapter 9, this example code is also how you create automated tests for your modules. I typically recommend that tests go into the *test* folder.

Executable documentation

> If you commit this example (including *README.md*) into version control, other members of your team can find it, use it to understand how your module works, and take the module for a spin without writing a line of code. It's both a way to teach the rest of your team and, if you add automated tests around it, a way to ensure that your teaching materials always work as expected.

Every Terraform module you have in the *modules* folder should have a corresponding example in the *examples* folder. And every example in the *examples* folder should have a corresponding test in the *test* folder. In fact, you'll most likely have multiple examples (and therefore multiple tests) for each module, with each example showing different configurations and permutations of how that module can be used. For example, you might want to add other examples for the `asg-rolling-deploy` module that show how to use it with auto scaling policies, how to hook up load balancers to it, how to set custom tags, and so on.

Putting this all together, the folder structure for a typical *modules* repo will look something like this:

```
modules
 └ examples
```

```
    └ alb
    └ asg-rolling-deploy
      └ one-instance
      └ auto-scaling
      └ with-load-balancer
      └ custom-tags
    └ hello-world-app
    └ mysql
  └ modules
    └ alb
    └ asg-rolling-deploy
    └ hello-world-app
    └ mysql
  └ test
    └ alb
    └ asg-rolling-deploy
    └ hello-world-app
    └ mysql
```

As an exercise for the reader, I leave it up to you to add lots of examples for the `alb`, `asg-rolling-deploy`, `mysql`, and `hello-world-app` modules.

A great practice to follow when developing a new module is to write the example code *first*, before you write even a line of module code. If you begin with the implementation, it's too easy to become lost in the implementation details, and by the time you resurface and make it back to the API, you end up with a module that is unintuitive and difficult to use. On the other hand, if you begin with the example code, you're free to think through the ideal user experience and come up with a clean API for your module and then work backward to the implementation. Because the example code is the primary way of testing modules anyway, this is a form of *Test-Driven Development* (TDD); I'll dive more into this topic in Chapter 9, which is entirely dedicated to testing.

In this section, I'll focus on creating *self-validating modules*: that is, modules that can check their own behavior to prevent certain types of bugs. Terraform has two ways of doing this built in:

- Validations