```
  enable_autoscaling    = false
}
```

Similarly, you can update the usage of this module in production (in *live/prod/services/webserver-cluster/main.tf*) to enable auto scaling by setting `enable_autoscaling` to `true` (make sure to also remove the custom `aws_autoscaling_schedule` resources that were in the production environment from Chapter 4):

```
module "webserver_cluster" {
  source = "../../../../modules/services/webserver-cluster"

  cluster_name           = "webservers-prod"
  db_remote_state_bucket = "(YOUR_BUCKET_NAME)"
  db_remote_state_key    = "prod/data-
stores/mysql/terraform.tfstate"

  instance_type        = "m4.large"
  min_size             = 2
  max_size             = 10
  enable_autoscaling   = true

  custom_tags = {
    Owner     = "team-foo"
    ManagedBy = "terraform"
  }
}
```

If-else-statements with the count parameter

Now that you know how to do an if-statement, what about an if-else-statement?

Earlier in this chapter, you created several IAM users with read-only access to EC2. Imagine that you wanted to give one of these users, neo, access to CloudWatch as well but allow the person applying the Terraform configurations to decide whether neo is assigned only read access or both read and write access. This is a slightly contrived example, but a useful one to demonstrate a simple type of if-else-statement.

Here is an IAM Policy that allows read-only access to CloudWatch:

```
resource "aws_iam_policy" "cloudwatch_read_only" {
  name   = "cloudwatch-read-only"
  policy = data.aws_iam_policy_document.cloudwatch_read_only.json
}

data "aws_iam_policy_document" "cloudwatch_read_only" {
  statement {
    effect    = "Allow"
    actions   = [
      "cloudwatch:Describe*",
      "cloudwatch:Get*",
      "cloudwatch:List*"
    ]
    resources = ["*"]
  }
}
```

And here is an IAM Policy that allows full (read and write) access to CloudWatch:

```
resource "aws_iam_policy" "cloudwatch_full_access" {
  name   = "cloudwatch-full-access"
  policy =
data.aws_iam_policy_document.cloudwatch_full_access.json
}

data "aws_iam_policy_document" "cloudwatch_full_access" {
  statement {
    effect    = "Allow"
    actions   = ["cloudwatch:*"]
    resources = ["*"]
  }
}
```

The goal is to attach one of these IAM Policies to `"neo"`, based on the value of a new input variable called `give_neo_cloudwatch_full_access`:

```
variable "give_neo_cloudwatch_full_access" {
  description = "If true, neo gets full access to CloudWatch"
  type        = bool
}
```

If you were using a general-purpose programming language, you might write an if-else-statement that looks like this:

```
# This is just pseudo code. It won't actually work in Terraform.
if var.give_neo_cloudwatch_full_access {
  resource "aws_iam_user_policy_attachment"
"neo_cloudwatch_full_access" {
    user       = aws_iam_user.example[0].name
    policy_arn = aws_iam_policy.cloudwatch_full_access.arn
  }
} else {
  resource "aws_iam_user_policy_attachment"
"neo_cloudwatch_read_only" {
    user       = aws_iam_user.example[0].name
    policy_arn = aws_iam_policy.cloudwatch_read_only.arn
  }
}
```

To do this in Terraform, you can use the `count` parameter and a conditional expression on each of the resources:

```
resource "aws_iam_user_policy_attachment"
"neo_cloudwatch_full_access" {
  count = var.give_neo_cloudwatch_full_access ? 1 : 0

  user       = aws_iam_user.example[0].name
  policy_arn = aws_iam_policy.cloudwatch_full_access.arn
}

resource "aws_iam_user_policy_attachment"
"neo_cloudwatch_read_only" {
  count = var.give_neo_cloudwatch_full_access ? 0 : 1

  user       = aws_iam_user.example[0].name
  policy_arn = aws_iam_policy.cloudwatch_read_only.arn
}
```

This code contains two `aws_iam_user_policy_attachment` resources. The first one, which attaches the CloudWatch full access permissions, has a conditional expression that will evaluate to 1 if `var.give_neo_cloudwatch_full_access` is `true`, and 0 otherwise (this is the if-clause). The second one, which attaches the

CloudWatch read-only permissions, has a conditional expression that does the exact opposite, evaluating to 0 if `var.give_neo_cloudwatch_full_access` is `true`, and 1 otherwise (this is the else-clause). And there you are—you now know how to do if-else-statements!

Now that you have the ability to create one resource or the other based on an if/else condition, what do you do if you need to access an attribute on the resource that actually got created? For example, what if you wanted to add an output variable called `neo_cloudwatch_policy_arn`, which contains the ARN of the policy you actually attached?

The simplest option is to use ternary syntax:

```
output "neo_cloudwatch_policy_arn" {
  value = (
    var.give_neo_cloudwatch_full_access
    ?
aws_iam_user_policy_attachment.neo_cloudwatch_full_access[0].poli
cy_arn
    :
aws_iam_user_policy_attachment.neo_cloudwatch_read_only[0].policy
_arn
  )
}
```

This will work fine for now, but this code is a bit brittle: if you ever change the conditional in the `count` parameter of the `aws_iam_user_policy_attachment` resources—perhaps in the future, it'll depend on multiple variables and not solely on `var.give_neo_cloudwatch_full_access`—there's a risk that you'll forget to update the conditional in this output variable, and as a result, you'll get a very confusing error when trying to access an array element that might not exist.

A safer approach is to take advantage of the `concat` and `one` functions. The `concat` function takes two or more lists as inputs and combines them into a single list. The `one` function takes a list as input and if the list has 0 elements, it returns `null`; if the list has 1 element, it returns that element;