and if the list has more than 1 element, it shows an error. Putting these two together, and combining them with a splat expression, you get the following:

```
output "neo_cloudwatch_policy_arn" {
  value = one(concat(

aws_iam_user_policy_attachment.neo_cloudwatch_full_access[*].poli
cy_arn,

aws_iam_user_policy_attachment.neo_cloudwatch_read_only[*].policy
_arn
  ))
}
```

Depending on the outcome of the if/else conditional, either `neo_cloudwatch_full_access` will be empty and `neo_cloudwatch_read_only` will contain one element or vice versa, so once you concatenate them together, you'll have a list with one element, and the `one` function will return that element. This will continue to work correctly no matter how you change your if/else conditional.

Using `count` and built-in functions to simulate if-else-statements is a bit of a hack, but it's one that works fairly well, and as you can see from the code, it allows you to conceal lots of complexity from your users so that they get to work with a clean and simple API.

Conditionals with for_each and for Expressions

Now that you understand how to do conditional logic with resources using the `count` parameter, you can probably guess that you can use a similar strategy to do conditional logic by using a `for_each` expression.

If you pass a `for_each` expression an empty collection, the result will be zero copies of the resource, inline block, or module where you have the `for_each`; if you pass it a nonempty collection, it will create one or more copies of the resource, inline block, or module. The only question is, how do you conditionally decide if the collection should be empty or not?

The answer is to combine the `for_each` expression with the `for` expression. For example, recall the way the `webserver-cluster` module in *modules/services/webserver-cluster/main.tf* sets tags:

```
dynamic "tag" {
  for_each = var.custom_tags

  content {
    key                 = tag.key
    value               = tag.value
    propagate_at_launch = true
  }
}
```

If `var.custom_tags` is empty, the `for_each` expression will have nothing to loop over, so no tags will be set. In other words, you already have some conditional logic here. But you can go even further, by combining the `for_each` expression with a `for` expression as follows:

```
dynamic "tag" {
  for_each = {
    for key, value in var.custom_tags:
    key => upper(value)
    if key != "Name"
  }

  content {
    key                 = tag.key
    value               = tag.value
    propagate_at_launch = true
  }
}
```

The nested `for` expression loops over `var.custom_tags`, converts each value to uppercase (perhaps for consistency), and uses a conditional in the `for` expression to filter out any `key` set to `Name` because the module already sets its own `Name` tag. By filtering values in the `for` expression, you can implement arbitrary conditional logic.