```
  deploy:
    jobs:
      - terraform_apply
    # Only run this workflow on commits to the main branch
    filters:
      branches:
        only:
          - main
    # Expose secrets in the CircleCI context as environment
variables
    context:
      - example-context
```

When your build runs, CircleCI will automatically expose the secrets in that Context as environment variables—in this case, as the environment variables `AWS_ACCESS_KEY_ID` and `AWS_SECRET_ACCESS_KEY`—and `terraform apply` will automatically use those environment variables to authenticate to your provider.

The biggest drawbacks to this approach are that (a) you have to manually manage credentials, and (b) as a result, you have to use permanent credentials, which once saved in CircleCI, rarely (if ever) change. The next two examples show alternative approaches.

EC2 Instance running Jenkins as a CI server, with IAM roles

If you're using an EC2 Instance to run Terraform code—e.g., you're running Jenkins on an EC2 Instance as a CI server—the solution I recommend for machine user authentication is to give that EC2 Instance an IAM role. An *IAM role* is similar to an IAM user, in that it's an entity in AWS that can be granted IAM permissions. However, unlike IAM users, IAM roles are not associated with any one person and do not have permanent credentials (password or access keys). Instead, a role can be *assumed* by other IAM entities: for example, an IAM user might assume a role to temporarily get access to different permissions than they normally have; many AWS services, such as EC2 Instances, can assume IAM roles to grant those services permissions in your AWS account.

For example, here's code you've seen many times to deploy an EC2 Instance:

```
resource "aws_instance" "example" {
  ami           = "ami-0fb653ca2d3203ac1"
  instance_type = "t2.micro"
}
```

To create an IAM role, you must first define an *assume role policy*, which is an IAM Policy that defines who is allowed to assume the IAM role. You could write the IAM Policy in raw JSON, but Terraform has a convenient `aws_iam_policy_document` data source that can create the JSON for you. Here's how you can use an `aws_iam_policy_document` to define an assume role policy that allows the EC2 service to assume an IAM role:

```
data "aws_iam_policy_document" "assume_role" {
  statement {
    effect  = "Allow"
    actions = ["sts:AssumeRole"]

    principals {
      type        = "Service"
      identifiers = ["ec2.amazonaws.com"]
    }
  }
}
```

Now, you can use the `aws_iam_role` resource to create an IAM role and pass it the JSON from your `aws_iam_policy_document` to use as the assume role policy:

```
resource "aws_iam_role" "instance" {
  name_prefix        = var.name
  assume_role_policy =
data.aws_iam_policy_document.assume_role.json
}
```

You now have an IAM role, but by default, IAM roles don't give you any permissions. So, the next step is to attach one or more IAM policies to the IAM role that specify what you can actually do with the role once you've assumed it. Let's imagine that you're using Jenkins to run Terraform code that deploys EC2 Instances. You can use the

`aws_iam_policy_document` data source to define an IAM Policy that gives admin permissions over EC2 Instances as follows:

```
data "aws_iam_policy_document" "ec2_admin_permissions" {
  statement {
    effect    = "Allow"
    actions   = ["ec2:*"]
    resources = ["*"]
  }
}
```

And you can attach this policy to your IAM role using the `aws_iam_role_policy` resource:

```
resource "aws_iam_role_policy" "example" {
  role   = aws_iam_role.instance.id
  policy =
data.aws_iam_policy_document.ec2_admin_permissions.json
}
```

The final step is to allow your EC2 Instance to automatically assume that IAM role by creating an *instance profile*:

```
resource "aws_iam_instance_profile" "instance" {
  role = aws_iam_role.instance.name
}
```

And then tell your EC2 Instance to use that instance profile via the `iam_instance_profile` parameter:

```
resource "aws_instance" "example" {
  ami           = "ami-0fb653ca2d3203ac1"
  instance_type = "t2.micro"

  # Attach the instance profile
  iam_instance_profile = aws_iam_instance_profile.instance.name
}
```

Under the hood, AWS runs an *instance metadata endpoint* on every EC2 Instance at *http://169.254.169.254*. This is an endpoint that can only be