reached by processes running on the instance itself, and those processes can use that endpoint to fetch metadata about the instance. For example, if you SSH to an EC2 Instance, you can query this endpoint using `curl`:

```
$ ssh ubuntu@<IP_OF_INSTANCE>
Welcome to Ubuntu 20.04.3 LTS (GNU/Linux 5.11.0-1022-aws x86_64)
(...)

$ curl http://169.254.169.254/latest/meta-data/
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
events/
hibernation/
hostname
identity-credentials/
(...)
```

If the instance has an IAM role attached (via an instance profile), that metadata will include AWS credentials that can be used to authenticate to AWS and assume that IAM role. Any tool that uses the AWS SDK, such as Terraform, knows how to use these instance metadata endpoint credentials automatically, so as soon as you run `terraform apply` on the EC2 Instance with this IAM role, your Terraform code will authenticate as this IAM role, which will thereby grant your code the EC2 admin permissions it needs to run successfully.[2]

For any automated process running in AWS, such as a CI server, IAM roles provide a way to authenticate (a) without having to manage credentials manually, and (b) the credentials AWS provides via the instance metadata endpoint are always temporary, and rotated automatically. These are two big advantages over the manually managed, permanent credentials with a tool like CircleCI that runs outside of your AWS account. However, as you'll see in the next example, in some cases, it's possible to have these same advantages for external tools, too.

GitHub Actions as a CI server, with OIDC

GitHub Actions is another popular managed CI/CD platform you might want to use to run Terraform. In the past, GitHub Actions required you to manually copy credentials around, just like CircleCI. However, as of 2021, GitHub Actions offers a better alternative: *Open ID Connect (OIDC)*. Using OIDC, you can establish a trusted link between the CI system and your cloud provider (GitHub Actions supports AWS, Azure, and Google Cloud) so that your CI system can authenticate to those providers without having to manage any credentials manually.

You define GitHub Actions workflows in YAML files in a *.github/workflows* folder, such as the *terraform.yml* file shown here:

```
name: Terraform Apply
# Only run this workflow on commits to the main branch
on:
  push:
    branches:
      - 'main'
jobs:
  TerraformApply:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v2

      # Run Terraform using HashiCorp's setup-terraform Action
      - uses: hashicorp/setup-terraform@v1
          with:
            terraform_version: 1.1.0
            terraform_wrapper: false
        run: |
          terraform init
          terraform apply -auto-approve
```

If your Terraform code talks to a provider such as AWS, you need to provide a way for this workflow to authenticate to that provider. To do this using OIDC,[3] the first step is to create an *IAM OIDC identity provider* in your AWS account, using the `aws_iam_openid_connect_provider` resource, and to configure it to trust the GitHub Actions thumbprint, fetched via the `tls_certificate` data source:

```
# Create an IAM OIDC identity provider that trusts GitHub
resource "aws_iam_openid_connect_provider" "github_actions" {
  url             = "https://token.actions.githubusercontent.com"
  client_id_list  = ["sts.amazonaws.com"]
  thumbprint_list = [
    data.tls_certificate.github.certificates[0].sha1_fingerprint
  ]
}

# Fetch GitHub's OIDC thumbprint
data "tls_certificate" "github" {
  url = "https://token.actions.githubusercontent.com"
}
```

Now, you can create IAM roles exactly as in the previous section—e.g., an IAM role with EC2 admin permissions attached—except the assume role policy for those IAM roles will look different:

```
data "aws_iam_policy_document" "assume_role_policy" {
  statement {
    actions = ["sts:AssumeRoleWithWebIdentity"]
    effect  = "Allow"

    principals {
      identifiers =
[aws_iam_openid_connect_provider.github_actions.arn]
      type        = "Federated"
    }

    condition {
      test     = "StringEquals"
      variable = "token.actions.githubusercontent.com:sub"
      # The repos and branches defined in
var.allowed_repos_branches
      # will be able to assume this IAM role
      values = [
        for a in var.allowed_repos_branches :

"repo:${a["org"]}/${a["repo"]}:ref:refs/heads/${a["branch"]}"
      ]
    }
  }
}
```

This policy allows the IAM OIDC identity provider to assume the IAM role via federated authentication. Note the `condition` block, which ensures that only the specific GitHub repos and branches you specify via the `allowed_repos_branches` input variable can assume this IAM role:

```
variable "allowed_repos_branches" {
  description = "GitHub repos/branches allowed to assume the IAM
role."
  type = list(object({
    org    = string
    repo   = string
    branch = string
  }))
  # Example:
  # allowed_repos_branches = [
  #   {
  #     org    = "brikis98"
  #     repo   = "terraform-up-and-running-code"
  #     branch = "main"
  #   }
  # ]
}
```

This is important to ensure you don't accidentally allow *all* GitHub repos to authenticate to your AWS account! You can now configure your builds in GitHub Actions to assume this IAM role. First, at the top of your workflow, give your build the `id-token: write` permission:

```
permissions:
  id-token: write
```

Next, add a build step just before running Terraform to authenticate to AWS using the `configure-aws-credentials` action:

```
      # Authenticate to AWS using OIDC
      - uses: aws-actions/configure-aws-credentials@v1
        with:
          # Specify the IAM role to assume here
          role-to-assume: arn:aws:iam::123456789012:role/example-
role
          aws-region: us-east-2
```