Manage Resource Drift

The Terraform state file is a record of all resources Terraform manages. You should not make manual changes to resources controlled by Terraform, because the state file will be out of sync, or "drift," from the real infrastructure. If your state and configuration do not match your infrastructure, Terraform will attempt to reconcile your infrastructure, which may unintentionally destroy or recreate resources.

In this tutorial, you will detect and fix differences between your state file and real infrastructure. First, you will create an EC2 instance and security group with Terraform. Then, you will manually edit them via the AWS CLI. Next, you will identify and resolve the discrepancies between Terraform state and your infrastructure.

Tip: Terraform Cloud for Business offers health assessments, which include daily checks that compare Terraform state and the real infrastructure that it tracks.

Prerequisites

This tutorial assumes you are familiar with the standard Terraform workflow. If you are unfamiliar with Terraform, complete the Get Started tutorials first.

For this tutorial, you will need:

- The Terraform CLI, version 0.15.4 or later.
- AWS Credentials configured for use with Terraform.
- The `awscli` configured

Create infrastructure

Start by cloning the example repository. This configuration builds an EC2 instance, an SSH key pair, and a security group rule to allow SSH access to the instance.

```
$ git clone https://github.com/hashicorp/learn-terraform-drift-management.git
```

Change into the repository directory.

```
$ cd learn-terraform-drift-management
```

Create an SSH key pair in your current directory, replacing `your_email@example.com` with your email address. Use an empty passphrase.

```
$ ssh-keygen -t rsa -C "your_email@example.com" -f ./key
Generating public/private rsa key pair.

Enter passphrase (empty for no passphrase):
```

Confirm your AWS CLI region.

```
$ aws configure get region
us-east-2
```

Open the `terraform.tfvars` file and edit the region to match your AWS CLI configuration.

```
region = "us-east-2"
```

Open the `main.tf` file and review your configuration. The main resources are your EC2 instance, your key pair, and the SSH security group.

```
##...
resource "aws_key_pair" "deployer" {
  key_name   = "deployer-key"
  public_key = file("${path.module}/key.pub")
}

resource "aws_instance" "example" {
  ami                    = data.aws_ami.ubuntu.id
  key_name               = aws_key_pair.deployer.key_name
  instance_type          = "t2.micro"
  vpc_security_group_ids = [aws_security_group.sg_ssh.id]
  user_data              = <<-EOF
              #!/bin/bash
              apt-get update
              apt-get install -y apache2
              sed -i -e 's/80/8080/' /etc/apache2/ports.conf
              echo "Hello World" > /var/www/html/index.html
              systemctl restart apache2
              EOF
  tags = {
    Name          = "terraform-learn-state-ec2"
    drift_example = "v1"
  }
}


resource "aws_security_group" "sg_ssh" {
  name = "sg_ssh"
  ingress {
    from_port   = "22"
    to_port     = "22"
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  // connectivity to ubuntu mirrors is required to run `apt-get update` and `apt-
get install apache2`
  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

Initialize your configuration.

```
$ terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/aws from the dependency lock file
- Installing hashicorp/aws v3.26.0...
- Installed hashicorp/aws v3.26.0 (signed by HashiCorp)

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

Apply your configuration. Enter `yes` when prompted to accept your changes.

```
$ terraform apply


## ...

Apply complete! Resources: 3 added, 0 changed, 0 destroyed.

Outputs:

instance_id = "i-0250e3c625858c3ee"
public_ip = "18.224.17.153"
security_groups = [
  toset([
    "sg-0bf5fe7b3b54df9c3",
  ]),
]
```

When your apply operation completes, run `terraform state list` to review the resources managed by Terraform in your state file.

```
$ terraform state list
data.aws_ami.ubuntu
aws_instance.example
aws_key_pair.deployer
aws_security_group.sg_ssh
```

Introduce drift

To introduce a change to your configuration outside the Terraform workflow, create a new security group with the AWS CLI and export that value as an environment variable.

```
$  export SG_ID=$(aws ec2 create-security-group --group-name "sg_web" --
description "allow 8080" --output text)
```

Confirm you created the environment variable. This will return the security group you just created.

```
$ echo $SG_ID
sg-04c74100cc8b9fc8c
```

```
$  $env:SG_ID = "$(aws ec2 create-security-group --group-name "sg_web" --
description "allow 8080" --output text)"
```

Confirm you created the environment variable. This will return the security group you just created.

```
$ $env:SG_ID
sg-04c74100cc8b9fc8c
```

Next, create a new rule for your group to provide TCP access to the instance on port 8080.

```
$ aws ec2 authorize-security-group-ingress --group-name "sg_web" --protocol tcp --
port 8080 --cidr 0.0.0.0/0
```

Associate the security group you created manually with the EC2 instance provisioned by Terraform.

```
$ aws ec2 modify-instance-attribute --instance-id $(terraform output -raw
instance_id) --groups $SG_ID
```

Now, you have replaced your instance's SSH security group with a new security group that is not tracked in the Terraform state file.

Run a refresh-only plan

By default, Terraform compares your state file to real infrastructure whenever you invoke `terraform plan` or `terraform apply`. The refresh updates your state file in-memory to reflect the actual configuration of your infrastructure. This ensures that Terraform determines the correct changes to make to your resources.

If you suspect that your infrastructure configuration changed outside of the Terraform workflow, you can use a `-refresh-only` flag to inspect what the changes to your state file would be. This is safer than the `refresh` subcommand, which automatically overwrites your state file without displaying the updates.

Tip: The `-refresh-only` flag was introduced in Terraform 0.15.4, and is preferred over the `terraform refresh` subcommand.

Run `terraform plan -refresh-only` to determine the drift between your current state file and actual configuration.

```
$ terraform plan -refresh-only
aws_key_pair.deployer: Refreshing state... [id=deployer-key]
aws_security_group.sg_ssh: Refreshing state... [id=sg-0b318a348a4a4e391]
aws_instance.example: Refreshing state... [id=i-008bef01721ee7f7c]

Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since the last
"terraform apply":

  # aws_instance.example has been changed
  ~ resource "aws_instance" "example" {
        id                          = "i-008bef01721ee7f7c"
        tags                        = {
            "Name"          = "terraform-learn-state-ec2"
            "drift_example" = "v1"
        }
      ~ vpc_security_group_ids       = [
          + "sg-0226a51361bf1497a",
          - "sg-0b318a348a4a4e391",
        ]
        # (27 unchanged attributes hidden)




        # (4 unchanged blocks hidden)
    }

This is a refresh-only plan, so Terraform will not take any actions to undo
these. If you were expecting these changes then you can apply this plan to
record the updated values in the Terraform state without changing any remote
objects.

─────────────────────────────────────────────────────────────────────────────────
──

Changes to Outputs:
  ~ security_groups = [
      - [
          - "sg-0b318a348a4a4e391",
        ],
      + [
          + "sg-0226a51361bf1497a",
        ],
    ]

You can apply this plan to save these new output values to the Terraform state,
without changing any real infrastructure.

─────────────────────────────────────────────────────────────────────────────────
───

Note: You didn't use the -out option to save this plan, so Terraform can't
guarantee to take exactly these actions if you run "terraform apply" now.
```

As shown in the output, Terraform has detected differences between the infrastructure and the current state, and sees that your original security group allowing access on port 80 is no longer attached to your EC2 instance. The refresh-only plan output indicates that Terraform will update your state file to modify the configuration of your EC2 instance to reflect the new security group with access on port 8080.

Apply these changes to make your state file match your real infrastructure, but not your Terraform configuration. Respond to the prompt with a `yes`.

```
$ terraform apply -refresh-only
aws_key_pair.deployer: Refreshing state... [id=deployer-key-rita]
aws_security_group.sg_ssh: Refreshing state... [id=sg-0b318a348a4a4e391]
aws_instance.example: Refreshing state... [id=i-008bef01721ee7f7c]

Note: Objects have changed outside of Terraform

Terraform detected the following changes made outside of Terraform since the last
"terraform apply":

  # aws_instance.example has been changed
##...
Would you like to update the Terraform state to reflect these detected changes?
  Terraform will write these changes to the state without modifying any real
infrastructure.
  There is no undo. Only 'yes' will be accepted to confirm.

  Enter a value: yes


Apply complete! Resources: 0 added, 0 changed, 0 destroyed.

Outputs:

instance_id = "i-008bef01721ee7f7c"
public_ip = "35.163.80.243"
security_groups = [
  toset([
    "sg-0226a51361bf1497a",
  ]),
]
```

A refresh-only operation does not attempt to modify your infrastructure to match your Terraform configuration -- it only gives you the option to review and track the drift in your state file.

If you ran `terraform plan` or `terraform apply` without the `-refresh-only` flag now, Terraform would attempt to revert your manual changes. Instead, you will update your configuration to associate your EC2 instance with both security groups.

Add the security group to configuration

Import the `sg_web` security group resource to your state file to bring it under Terraform management.

First, add the resource definition to your configuration by adding a new security group resource and rule resource to your `main.tf` file.

```
resource "aws_security_group" "sg_web" {
  name        = "sg_web"
  description = "allow 8080"
}

resource "aws_security_group_rule" "sg_web" {
  type      = "ingress"
  to_port   = "8080"
  from_port = "8080"
  protocol  = "tcp"
  cidr_blocks = ["0.0.0.0/0"]
  security_group_id = aws_security_group.sg_web.id
}
```

Add the security group ID to your instance resource.

```
resource "aws_instance" "example" {
  ami                    = data.aws_ami.ubuntu.id
  key_name               = aws_key_pair.deployer.key_name
  instance_type          = "t2.micro"
- vpc_security_group_ids = [aws_security_group.sg_ssh.id]
+ vpc_security_group_ids = [aws_security_group.sg_ssh.id,
aws_security_group.sg_web.id]
  user_data              = <<-EOF
              #!/bin/bash
              apt-get update
              apt-get install -y apache2
              sed -i -e 's/80/8080/' /etc/apache2/ports.conf
              echo "Hello World" > /var/www/html/index.html
              systemctl restart apache2
              EOF
  tags = {
    Name          = "terraform-learn-state-ec2"
    drift_example = "v1"
  }
}
```

Import the security group

Run `terraform import` to associate your resource definition with the security group created in the AWS CLI.

```
$ terraform import aws_security_group.sg_web $SG_ID

aws_security_group.sg_web: Importing from ID "sg-04c74100cc8b9fc8c"...
aws_security_group.sg_web: Import prepared!
  Prepared aws_security_group for import
aws_security_group.sg_web: Refreshing state... [id=sg-04c74100cc8b9fc8c]

Import successful!

The resources that were imported are shown above. These resources are now in
your Terraform state and will henceforth be managed by Terraform.
```

Import your security group rule.

```
$ terraform import aws_security_group_rule.sg_web
"$SG_ID"_ingress_tcp_8080_8080_0.0.0.0/0

aws_security_group_rule.sg_web: Importing from ID "sg-
04c74100cc8b9fc8c_ingress_tcp_8080_8080_0.0.0.0/0"...
aws_security_group_rule.sg_web: Import prepared!
  Prepared aws_security_group_rule for import
aws_security_group_rule.sg_web: Refreshing state... [id=sg-
04c74100cc8b9fc8c_ingress_tcp_8080_8080_0.0.0.0/0]

Import successful!

The resources that were imported are shown above. These resources are now in
your Terraform state and will henceforth be managed by Terraform.
```

Run `terraform state list` to return the list of resources Terraform is managing, which now includes the imported resources.

```
$ terraform state list

data.aws_ami.ubuntu
aws_instance.example
aws_key_pair.deployer
aws_security_group.sg_web
aws_security_group.sg_ssh
aws_security_group_rule.sg_web
```

Terraform successfully associated both security groups with the instance in state. However, your instance still only allows port 8080 access because the `modify-instance-attribute` AWS CLI command detached the SSH security group.

Update your resources

Now that the `sg_web` security group is represented in state, re-run `terraform apply` to associate the SSH security group with your EC2 instance.

Notice how this updates your EC2 instance's security group to include *both* the security groups allowing SSH and `8080`. Enter `yes` when prompted to confirm your changes.

```
$ terraform apply
aws_security_group.sg_ssh: Refreshing state... [id=sg-09d0b575577f258d5]
aws_key_pair.deployer: Refreshing state... [id=deployer-key]
aws_security_group.sg_web: Refreshing state... [id=sg-0acc6237c67c07e4b]
aws_security_group_rule.sg_web: Refreshing state... [id=sgrule-4278118923]
aws_instance.example: Refreshing state... [id=i-092c09eed28bdb2f7]

Terraform used the selected providers to generate the following execution plan.
Resource actions
are indicated with the following symbols:
  ~ update in-place

Terraform will perform the following actions:

  # aws_instance.example will be updated in-place
  ~ resource "aws_instance" "example" {
        id                           = "i-092c09eed28bdb2f7"
        tags                         = {
            "Name"          = "terraform-learn-state-ec2"
            "drift_example" = "v1"
        }
      ~ vpc_security_group_ids       = [
          + "sg-09d0b575577f258d5",
            # (1 unchanged element hidden)
        ]
        # (27 unchanged attributes hidden)




        # (4 unchanged blocks hidden)
    }

Plan: 0 to add, 1 to change, 0 to destroy.
Do you want to perform these actions?
  Terraform will perform the actions described above.
  Only 'yes' will be accepted to approve.

  Enter a value: yes
##...

Apply complete! Resources: 0 added, 1 changed, 0 destroyed.

Outputs:

instance_id = "i-092c09eed28bdb2f7"
public_ip = "3.142.238.150"
security_groups = [
  toset([
    "sg-09d0b575577f258d5",
    "sg-0acc6237c67c07e4b",
  ]),
]
```

Access the instance

Confirm your instance allows SSH. Enter `yes` when prompted to connect to the instance.

```
$ ssh ubuntu@$(terraform output -raw public_ip) -i key
The authenticity of host '3.142.238.150 (3.142.238.150)' can't be established.
ECDSA key fingerprint is SHA256:7PCkol+dVFps8YkOPMVZ7zG9sKXq0tnzqRENB7FTodM.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '3.142.238.150' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-1041-aws x86_64)
##...
ubuntu@ip-172-31-20-193:~$
```

Exit the SSH connection by typing `exit` in the SSH prompt.

Confirm your instance allows port 8080 access.

```
$ curl $(terraform output -raw public_ip):8080
Hello, World
```

Clean up your resources

When you are finished with this tutorial, destroy the resources you created. Enter `yes` when prompted to confirm your changes.

```
$ terraform destroy
##...
Destroy complete! Resources: 5 destroyed.
```

Next steps

In this tutorial, you created an EC2 instance and security group deployment with Terraform. Then, you introduced drift by editing your security groups outside the Terraform workflow. Finally, you learned how to detect your drifted configuration with `-refresh-only` operations and how to reconcile your state file and configuration with the Terraform CLI refresh and import commands.

For more information about Terraform state and drift management, review the resources below:

- The Terraform `refresh` documentation
- The `refresh-only` planning mode documentation
- The State of Infrastructure Drift Presentation from Stephane Jourdan on the HashiCorp Blog
- Learn Terraform Import
- Manipulate Terraform State documentation