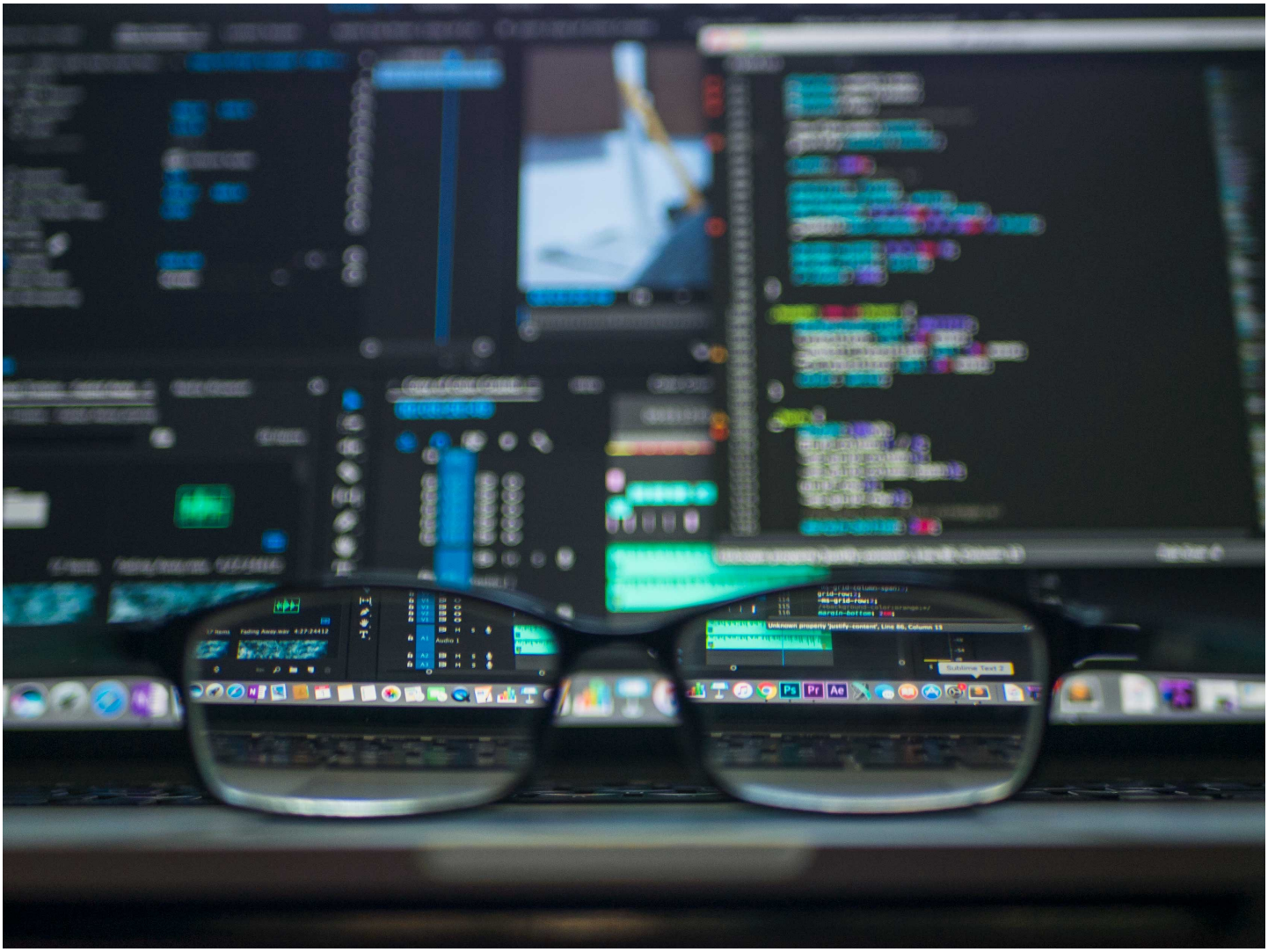
Mitesh

Follow 510 Followers About

Terraform Provisioner

Mitesh Sep 14, 2018 · 4 min read



"closeup photo of eyeglasses" by Kevin Ku on Unsplash

We have understood terraform as a tool to create and manage infrastructure with

below articles:

1. Infrastructure as a Code With Terraform
2. State management with Terraform
3. Manage AWS VPC With Terraform
4. Module in Terraform

Terraform not only helps us in infrastructure creation and management but also in provisioning them during resource creation or deletion. It uses a feature named provisioner for the same. Provisioners are used for executing scripts or shell commands on a local or remote machine as part of resource creation/deletion. They are similar to "EC2 instance user data" scripts that only run once on the creation and if it fails terraform marks it tainted. A tainted resource is one which is planned for destruction on next terraform apply. Terraform doesn't run these scripts multiple times. If we want more flexibility then we must use a configuration management tool like ansible to properly provision the instance.

There are many provisioners available. They help in:

1. local script execution on the machine from where we are running terraform
2. remote script execution on resource
3. file or directory copy on the remote resource
4. configure and run configuration management tools on the remote resource.

local-exec provisioner helps run a script on instance where we are running our terraform code, not on the resource we are creating. For example, if we want to write EC2 instance IP address to a file, then we can use below local-exec provisioner with our EC2 resource and save it locally in a file.

```
resource "aws_instance" "testInstance" {
  ami           = "${var.instance_ami}"
  instance_type = "${var.instance_type}"
  subnet_id = "${aws_subnet.subnet_public.id}"
  vpc_security_group_ids = ["${aws_security_group.sg_22.id}"]
  key_name = "${aws_key_pair.ec2key.key_name}"

tags {
        "Environment" = "${var.environment_tag}"
  }
  provisioner "local-exec" {
    command = "echo ${aws_instance.testInstance.public_ip} >>
```

```
    public_ip.txt"
      }
}
```

remote-exec provisioner helps invoke a script on the remote resource once it is created. We can provide a list of command strings which are executed in the order they are provided. We can also provide scripts with a local path which is copied remotely and then executed on the remote resource. file provisioner is used to copy files or directories to a remote resource. We can't provide any arguments to script in remote-exec provisioner. We can achieve this by copying script from file provisioner and then execute a script using a list of commands.

Provisioner which execute commands on a resource (like running a script or copying file)needs to connect to the resource which can be done through SSH. We can define connection method per resource or per provisioner if we want them to connect using different SSH parameters.

There is a special resource named "null_resource" which allow us to configure provisioners which are not directly associated with any resource. It behaves like any other resource which needs connection and provisioner details. For this resource, we need to define triggers which used to trigger a rerun for its set of provisioners.

Provisioner in action

Let's use our existing example of creating EC2 instance inside VPC and extend it to setup Nginx using provisioner and access it from port 80. We need to open port 80 using a security group and set up Nginx using remote-exec provisioner.

```
resource "aws_security_group" "sg_22_80" {
  name = "sg_22"
  vpc_id = "${aws_vpc.vpc.id}"

  ingress {
      from_port   = 22
      to_port     = 22
      protocol    = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
      from_port   = 80
      to_port     = 80
      protocol    = "tcp"
```

```
      cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }

  tags {
    "Environment" = "${var.environment_tag}"
  }
}
```

Updated our security group with port 22 and port 80 open. Port 22 is used to SSH by terraform remote-exec provisioner to setup Nginx and Port 80 is used by us when making HTTP call from our browser. Once we are ready with our setup of infrastructure, let's setup remote-exec provisioner with a connection.

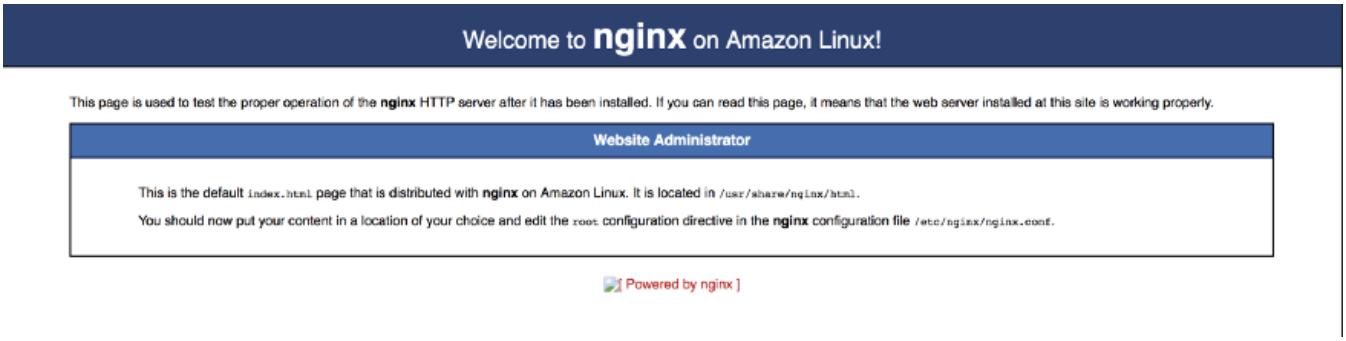
```
resource "aws_instance" "testInstance" {
  ami           = "${var.instance_ami}"
  instance_type = "${var.instance_type}"
  subnet_id = "${aws_subnet.subnet_public.id}"
  vpc_security_group_ids = ["${aws_security_group.sg_22_80.id}"]
  key_name = "${aws_key_pair.ec2key.key_name}"

  tags {
        "Environment" = "${var.environment_tag}"
    }

  provisioner "remote-exec" {
    inline = [
      "sudo amazon-linux-extras enable nginx1.12",
      "sudo yum -y install nginx",
      "sudo systemctl start nginx",
    ]
  }

  connection {
    type     = "ssh"
    user     = "ec2-user"
    password = ""
    private_key = "${file("~/.ssh/id_rsa")}"
  }
}
```

A connection is used to create an SSH connection with user “ec2-user” and ssh private key from a specified path to run remote-exec inline commands. As we are using Amazon Linux 2, so we need to enable nginx using “amazon-linux-extras”. Once this is done, we are going to install nginx and then start it. After running our terraform code, we should have nginx up and running on EC2 instance. When we try to hit public IP of our EC2 instance from the browser, we are able to get a response from nginx.



The complete code can be found in this git repository:
https://github.com/MiteshSharma/TerraformProvisioner

PS: If you liked the article, please support it with claps. Cheers

AWS Terraform Provisioner Configuration Management