Deploying NixOS using Terraform

Contents

Assuming you're familiar with the basics of Terraform, by the end of this tutorial you will have provisioned an Amazon Web Services (AWS) instance with Terraform, and will be able to use Nix to deploy incremental changes to NixOS running on the instance.

We'll look at how to boot a NixOS machine and how to deploy the incremental changes.

Booting NixOS image

1. Start by providing the Terraform executable:

```
$ nix-shell -p terraform
```

2. We are using Terraform Cloud as a state/locking backend:

```
$ terraform login
```

3. Make sure to create an organization, like `myorganization`, in your Terraform Cloud account.
4. Inside `myorganization`, create a workspace by choosing **CLI-driven workflow** and pick a name, like `myapp`.
5. Inside your workspace, under `Settings / General`, change Execution Mode to `Local`.
6. Inside a new directory, create a `main.tf` file with the following contents. This will start

Skip to main content

group:

```
terraform {
    backend "remote" {
        organization = "myorganization"

        workspaces {
            name = "myapp"
        }
    }
}

provider "aws" {
    region = "eu-central-1"
}

module "nixos_image" {
    source  = "git::https://github.com/tweag/terraform-nixos.git//aws_image_nixos?r
    release = "20.09"
}

resource "aws_security_group" "ssh_and_egress" {
    ingress {
        from_port   = 22
        to_port     = 22
        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"]
    }
}

resource "tls_private_key" "state_ssh_key" {
    algorithm = "RSA"
}

resource "local_file" "machine_ssh_key" {
    sensitive_content = tls_private_key.state_ssh_key.private_key_pem
    filename          = "${path.module}/id_rsa.pem"
    file_permission   = "0600"
}

resource "aws_key_pair" "generated_key" {
    key_name   = "generated-key-${sha256(tls_private_key.state_ssh_key.public_key_o
    public_key = tls_private_key.state_ssh_key.public_key_openssh
}

resource "aws_instance" "machine" {
    ami             = module.nixos_image.ami
    instance_type   = "t3.micro"
    security_groups = [ aws_security_group.ssh_and_egress.name ]
```

Skip to main content

```
    root_block_device {
        volume_size = 50 # GiB
    }
}

output "public_dns" {
    value = aws_instance.machine.public_dns
}
```

The only NixOS specific snippet is:

```
module "nixos_image" {
  source = "git::https://github.com/tweag/terraform-nixos.git/aws_image_nixos?ref=5
  release = "20.09"
}
```

Note

The `aws_image_nixos` module will return a NixOS AMI given a NixOS release number so that the `aws_instance` resource can reference the AMI in instance_type argument.

5. Make sure to configure AWS credentials.
6. Applying the Terraform configuration should get you a running NixOS:

```
$ terraform init
$ terraform apply
```

Deploying NixOS changes

Once the AWS instance is running a NixOS image via Terraform, we can teach Terraform to always build the latest NixOS configuration and apply those changes to your instance.

1. Create `configuration.nix` with the following contents:

```
{ config, lib, pkgs, ... }: {
  imports = [ <nixpkgs/nixos/modules/virtualisation/amazon-image.nix> ];

  # Open https://search.nixos.org/options for all options
}
```

Skip to main content

```
module "deploy_nixos" {
    source = "git::https://github.com/tweag/terraform-nixos.git//deploy_nixos?ref=5
    nixos_config = "${path.module}/configuration.nix"
    target_host = aws_instance.machine.public_ip
    ssh_private_key_file = local_file.machine_ssh_key.filename
    ssh_agent = false
}
```

3. Deploy:

```
$ terraform init
$ terraform apply
```

Caveats

- The `deploy_nixos` module requires NixOS to be installed on the target machine and Nix on the host machine.
- The `deploy_nixos` module doesn't work when the client and target architectures are different (unless you use distributed builds).
- If you need to inject a value into Nix, there is no elegant solution.
- Each machine is evaluated separately, so note that your memory requirements will grow linearly with the number of machines.

Next steps

- It's possible to switch to Google Compute Engine.
- The **`deploy_nixos`** module supports a number of arguments, for example to upload keys.