






In this post, I'm going to provide a quick introduction to [Terraform](#), a tool that is used to provision and configure infrastructure. Terraform allows you to define infrastructure configurations and then have those configurations implemented/created by Terraform automatically. In this respect, you could compare Terraform to similar solutions like [OpenStack Heat](#), AWS CloudFormation, and others.

Terraform is a tool for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular service providers as well as custom in-house solutions. Multi-cloud deployments can be very challenging as many existing tools for infrastructure management are cloud-specific. Terraform is cloud-agnostic and allows a single configuration to be used to manage multiple providers, and to even handle cross-cloud dependencies. This simplifies management and orchestration, helping operators build large-scale multi-cloud infrastructures.

The [official Terraform Getting Started documentation](#) does a good job of introducing the individual elements of Terraform (i.e. resources, input variables, output variables, etc), so in this guide, we're going to focus on how to put those elements together to create a fairly real-world example. In particular, we will provision several servers on AWS in a cluster and deploy a load balancer to distribute load across that cluster. The infrastructure you'll create in this example is a basic starting point for running scalable, highly-available web services and microservices. Key features are:

	 CloudFormation	 Ansible	 Terraform
Syntax	JSON	Yaml	HCL
State Management	No	Yes	Yes
Execution Control	No	No	Yes
Manage Already Created Resources	No	Yes	Hard
Providers Support	AWS Only	+++	++

Infrastructure as Code

Infrastructure is described using a high-level configuration syntax. This allows a blueprint of your datacenter to be versioned and treated as you would any other code. Additionally, infrastructure can be shared and re-used.

Execution Plans

Terraform has a “planning” step where it generates an *execution plan*. The execution plan shows what Terraform will do when you call apply. This lets you avoid any surprises when Terraform manipulates infrastructure.

Resource Graph

Terraform builds a graph of all your resources, and parallelizes the creation and modification of any non-dependent resources. Because of this, Terraform builds infrastructure as efficiently as possible, and operators get insight into dependencies in their infrastructure.

Change Automation

Complex changesets can be applied to your infrastructure with minimal human interaction. With the previously mentioned execution plan and resource graph, you know exactly what Terraform will change and in what order, avoiding many possible human errors.

1. Install Terraform

[Download](#) terraform depending on your system. Installation is very simple. Download the

terraform zip archive and unzip it in a suitable location. Once we have unzipped the terraform, update PATH environment variable pointing to terraform. Since the folder /usr/local/bin is already set to PATH environment variable, we don't need to set it again. If you are using any other location, then specify it in the PATH environment variable either in .bash_profile or in /etc/profile.

```
$ wget https://releases.hashicorp.com/terraform/0.10.7/terraform_0.10.7_linux_386.zip
$ unzip terraform_0.10.7_linux_386.zip $ mv terraform /usr/local/bin/ $ export
PATH=$PATH:/usr/local/bin/ Check the installation: $ terraform -v
```

2. Set up AWS account

Terraform can provision infrastructure across many different types of cloud providers, including AWS, Azure, Google Cloud, DigitalOcean, and [many others](#).

You need to create an IAM user. To create a more limited user account, head over to the [Identity and Access Management \(IAM\) console](#), click "Users", and click the blue "Create New Users" button.

once with the same access type and permissions. [Learn more](#)

User name*

[+ Add another user](#)

e

cess AWS. Access keys and autogenerated passwords are provided in the last step. [Learn more](#)

Access type*



Programmatic access

Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and other tools.



AWS Management Console access

Enables a **password** that allows users to sign-in to the AWS Management Console.

Click next and in add user to group permission select ec2fullaccess. Then you should copy the Access Key ID and Secret Access Key ID.

3. Create an Instance

Create a directory for isolating terraform files. `$ mkdir ~/terraform && cd ~/terraform`

Terraform code is written in a language called HCL in files with the extension “.tf”. It is a declarative language. The first step to using Terraform is typically to configure the provider you want to use. Create a file called “example.tf” and put the following code in it:

```
provider "aws" {
  access_key = "aws_access_key_id"
  secret_key = "aws_secret_access_key_id"
  region = "ap-south-1"
}
```

In here, we’re going to be using the AWS provider and that you wish to deploy your infrastructure in the “ap-south-1” region. For each provider, there are many different kinds of “resources” you can create, such as servers, databases, and load balancers.

```
resource "aws_instance" "web" {
  ami = "${lookup(var.amis,var.region)}"
  count = "${var.count}"
  key_name = "${var.key_name}"
  vpc_security_group_ids = ["${aws_security_group.instance.id}"]
  source_dest_check = false
  instance_type = "t2.micro"
```

```
tags {
  Name = "${format("web-%03d", count.index + 1)}"
}
}
```

Create another file named “variables.tf”

```
variable "count" {
  default = 1
}
variable "region" {
  description = "AWS region for hosting our your network"
  default = "ap-south-1"
}
variable "public_key_path" {
  description = "Enter the path to the SSH Public Key to add to AWS."
  default = "/path_to_keyfile/keypair_name.pem"
}
variable "key_name" {
  description = "Key name for SSHing into EC2"
  default = "kaypair_name"
}
variable "amis" {
  description = "Base AMI to launch the instances"
  default = {
    ap-south-1 = "ami-8da8d2e2"
  }
}
```

NOTE: you need to create and download keypair using management console

In a terminal, go into the folder where you created example.tf, and run the "terraform plan" command:

```
$ terraform plan
Refreshing Terraform state in-memory prior to plan...
```

```
(...)
```

```
+ aws_instance.example
  ami: "ami-2d39803a"
  availability_zone: "<computed>"
  ebs_block_device.#: "<computed>"
  ephemeral_block_device.#: "<computed>"
  instance_state: "<computed>"
  instance_type: "t2.micro"
  key_name: "<computed>"
  network_interface_id: "<computed>"
  placement_group: "<computed>"
  private_dns: "<computed>"
  private_ip: "<computed>"
  public_dns: "<computed>"
  public_ip: "<computed>"
  root_block_device.#: "<computed>"
  security_groups.#: "<computed>"
  source_dest_check: "true"
  subnet_id: "<computed>"
  tenancy: "<computed>"
  vpc_security_group_ids.#: "<computed>"
```

```
Plan: 1 to add, 0 to change, 0 to destroy.
```

To actually create the instance, run the "terraform apply" command:

```
$ terraform apply
aws_instance.example: Creating...
.....
..... . . .
Apply complete! Resources: 1 added, 0 changed, 0 destroyed.
```

You can see your created instance in aws management console is up and running.

4. Create an AutoScaling and ELB

Running a single server is a good start, but in the real world, a single server is a single point of failure. If that server crashes, or if it becomes overwhelmed by too much traffic, users can no longer access your site. The solution is to run a cluster of servers, routing around servers that go down, and adjusting the size of the cluster up or down based on traffic. The **first step** is creating an ASG is to create a [launch configuration](#), which specifies how to configure each EC2 Instance in the ASG. From deploying the single EC2 Instance earlier, you already know exactly how to configure it. The second step is creating a load balancer that is highly available and scalable is a lot of work.

Now, reopen your `example.tf` file and copy the following

```
provider "aws" {
  access_key = "aws_access_key_id"
  secret_key = "aws_secret_access_key_id"
  region     = "ap-south-1"
}
```

```
data "aws_availability_zones" "all" {}
```

```
### Creating EC2 instance
resource "aws_instance" "web" {
  ami           = "${lookup(var.amis, var.region)}"
  count        = "${var.count}"
  key_name      = "${var.key_name}"
  vpc_security_group_ids = ["${aws_security_group.instance.id}"]
  source_dest_check = false
  instance_type = "t2.micro"
}
```

```
tags {
  Name = "${format("web-%03d", count.index + 1)}"
}
}
```

```
### Creating Security Group for EC2
resource "aws_security_group" "instance" {
  name = "terraform-example-instance"
  ingress {
    from_port = 8080
    to_port   = 8080
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 22
    to_port   = 22
    protocol = "tcp"
  }
}
```

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

```
## Creating Launch Configuration
resource "aws_launch_configuration" "example" {
  image_id          = "${lookup(var.amis,var.region)}"
  instance_type     = "t2.micro"
  security_groups   = ["${aws_security_group.instance.id}"]
  key_name          = "${var.key_name}"
  user_data = <<-EOF
    #!/bin/bash
    echo "Hello, World" > index.html
    nohup busybox httpd -f -p 8080 &
    EOF
  lifecycle {
    create_before_destroy = true
  }
}
```

```
## Creating AutoScaling Group
resource "aws_autoscaling_group" "example" {
  launch_configuration = "${aws_launch_configuration.example.id}"
  availability_zones   = ["${data.aws_availability_zones.all.names}"]
  min_size             = 2
  max_size             = 10
  load_balancers       = ["${aws_elb.example.name}"]
  health_check_type   = "ELB"
  tag {
    key = "Name"
    value = "terraform-asg-example"
    propagate_at_launch = true
  }
}
```

```
## Security Group for ELB
resource "aws_security_group" "elb" {
  name = "terraform-example-elb"
  egress {
    from_port = 0
    to_port = 0
    protocol = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port = 80
    to_port = 80
    protocol = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
```

```
### Creating ELB
resource "aws_elb" "example" {
  name = "terraform-asg-example"
  security_groups = ["${aws_security_group.elb.id}"]
  availability_zones = ["${data.aws_availability_zones.all.names}"]
  health_check {
    healthy_threshold = 2
    unhealthy_threshold = 2
    timeout = 3
    interval = 30
    target = "HTTP:8080/"
  }
  listener {
    lb_port = 80
    lb_protocol = "http"
    instance_port = "8080"
    instance_protocol = "http"
  }
}
```

In variable.tf file

```
variable "count" {
  default = 1
}
```

```
variable "region" {
  description = "AWS region for hosting our your network"
  default = "ap-south-1"
}
```

```
variable "public_key_path" {
  description = "Enter the path to the SSH Public Key to add to AWS."
  default = "/home/ratul/developments/devops/keyfile/ec2-core-app.pem"
}
```

```
variable "key_name" {
  description = "Key name for SSHing into EC2"
  default = "ec2-core-app"
}
```

```
variable "amis" {
  description = "Base AMI to launch the instances"
  default = {
    ap-south-1 = "ami-8da8d2e2"
  }
}
```


Also create a file named output.tf and copy the following

```
output "instance_ids" {
  value = ["${aws_instance.web.*.public_ip}"]
}
```

```
output "elb_dns_name" {
  value = "${aws_elb.example.dns_name}"
}
```

Now run terraform plan

```
$ **terraform plan**
Refreshing Terraform state in-memory prior to plan...
The refreshed state will be used to calculate this plan, but will not be
persisted to local or remote state storage.
```

```
aws_security_group.instance: Refreshing state... (ID: sg-54ed813c)
aws_security_group.elb: Refreshing state... (ID: sg-09ef8361)
data.aws_availability_zones.all: Refreshing state...
aws_launch_configuration.example: Refreshing state... (ID:
terraform-201710101222483117000000001)
aws_instance.web: Refreshing state... (ID: i-0c4d35b212b045d78)
aws_elb.example: Refreshing state... (ID: terraform-asg-example)
aws_autoscaling_group.example: Refreshing state... (ID:
tf-asg-201710101223154573000000002)
The Terraform execution plan has been generated and is shown below.
Resources are shown in alphabetical order for quick scanning. Green resources
will be created (or destroyed and then created if an existing resource
exists), yellow resources are being changed in-place, and red resources
will be destroyed. Cyan entries are data sources to be read.
```

Note: You didn't specify an "-out" parameter to save this plan, so when "apply" is called, Terraform can't guarantee this is what will execute.

```
\-/+ aws_instance.web (new resource required)
  ami: "ami-8da8d2e2" => "ami-8da8d2e2"
  associate_public_ip_address: "true" => "<computed>"
  availability_zone: "ap-south-1a" => "<computed>"
  ebs_block_device.#: "0" => "<computed>"
  ephemeral_block_device.#: "0" => "<computed>"
  instance_state: "running" => "<computed>"
  instance_type: "t2.micro" => "t2.micro"
  ipv6_address_count: "" => "<computed>"
  ipv6_addresses.#: "0" => "<computed>"
  key_name: "ec2-core-app" => "ec2-core-app"
  network_interface.#: "0" => "<computed>"
  network_interface_id: "eni-b523daea" => "<computed>"
  placement_group: "" => "<computed>"
```

```

    primary_network_interface_id:      "eni-b523daea" => "<computed>"
    private_dns:
"ip-172-31-24-44.ap-south-1.compute.internal" => "<computed>"
    private_ip:                        "172.31.24.44" => "<computed>"
    public_dns:
"ec2-13-126-108-226.ap-south-1.compute.amazonaws.com" => "<computed>"
    public_ip:                          "13.126.108.226" => "<computed>"
    root_block_device.#:                "1" => "<computed>"
    security_groups.#:                  "1" => "<computed>"
    source_dest_check:                  "false" => "false"
    subnet_id:                          "subnet-16814c7f" => "<computed>"
    tags.%:                              "1" => "1"
    tags.Name:                          "web-001" => "web-001"
    tenancy:                             "default" => "<computed>"
    user_data:                           "c765373c563b260626d113c4a56a46e8a8c5ca33"
=> "" (forces new resource)
    volume_tags.%:                       "0" => "<computed>"
    vpc_security_group_ids.#:            "0" => "1"
    vpc_security_group_ids.3652085476:  "" => "sg-54ed813c"

```

Plan: 1 to add, 0 to change, 1 to destroy.

Run terraform plan

```

$ **terraform plan
**aws_security_group.instance: Refreshing state... (ID: sg-54ed813c)
aws_security_group.elb: Refreshing state... (ID: sg-09ef8361)
data.aws_availability_zones.all: Refreshing state...
aws_launch_configuration.example: Refreshing state... (ID:
terraform-20171010122248311700000001)
aws_instance.web: Refreshing state... (ID: i-0c4d35b212b045d78)
aws_elb.example: Refreshing state... (ID: terraform-asg-example)
aws_autoscaling_group.example: Refreshing state... (ID:
tf-asg-20171010122315457300000002)
aws_instance.web: Destroying... (ID: i-0c4d35b212b045d78)
aws_instance.web: Still destroying... (ID: i-0c4d35b212b045d78, 10s elapsed)
aws_instance.web: Still destroying... (ID: i-0c4d35b212b045d78, 20s elapsed)
aws_instance.web: Still destroying... (ID: i-0c4d35b212b045d78, 30s elapsed)
aws_instance.web: Still destroying... (ID: i-0c4d35b212b045d78, 40s elapsed)
aws_instance.web: Destruction complete after 47s
aws_instance.web: Creating...
    ami:                                "" => "ami-8da8d2e2"
    associate_public_ip_address:        "" => "<computed>"
    availability_zone:                  "" => "<computed>"
    ebs_block_device.#:                 "" => "<computed>"
    ephemeral_block_device.#:           "" => "<computed>"
    instance_state:                     "" => "<computed>"
    instance_type:                       "" => "t2.micro"
    ipv6_address_count:                  "" => "<computed>"
    ipv6_addresses.#:                   "" => "<computed>"
    key_name:                            "" => "ec2-core-app"
    network_interface.#:                "" => "<computed>"
    network_interface_id:               "" => "<computed>"
    placement_group:                    "" => "<computed>"

```

```
primary_network_interface_id: "" => "<computed>"
private_dns: "" => "<computed>"
private_ip: "" => "<computed>"
public_dns: "" => "<computed>"
public_ip: "" => "<computed>"
root_block_device.#: "" => "<computed>"
security_groups.#: "" => "<computed>"
source_dest_check: "" => "false"
subnet_id: "" => "<computed>"
tags.%: "" => "1"
tags.Name: "" => "web-001"
tenancy: "" => "<computed>"
volume_tags.%: "" => "<computed>"
vpc_security_group_ids.#: "" => "1"
vpc_security_group_ids.3652085476: "" => "sg-54ed813c"
aws_instance.web: Still creating... (10s elapsed)
aws_instance.web: Still creating... (20s elapsed)
aws_instance.web: Still creating... (30s elapsed)
aws_instance.web: Creation complete after 32s (ID: i-019e9bf03a9d3de32)
```

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

```
Outputs:
```

```
**elb_dns_name** = terraform-asg-example-2472445669.ap-south-1.elb.amazonaws.com
instance_ids = [
  52.66.14.13
]
```

The ELB is routing traffic to your EC2 Instances. Each time you hit the URL, it'll pick a different Instance to handle the request. You now have a fully working cluster of web servers!

Copy ****elb_dns_name**** and paste in your browser



The screenshot shows the AWS Management Console interface for a Load Balancer. The left sidebar contains navigation options: LOAD BALANCING (Load Balancers, Target Groups), AUTO SCALING (Launch Configurations, Auto Scaling Groups), SYSTEMS MANAGER SERVICES (Run Command, State Manager, Configuration Compliance, Automations, Patch Compliance, Patch Baselines), and SYSTEMS MANAGER SHARED RESOURCES (Managed Instances, Activations, Documents). The main content area is titled 'Create Load Balancer' and 'Actions'. A table lists the load balancer 'terraform-asg-example' with its DNS name, VPC ID, and availability zones. Below the table, the 'Basic Configuration' section shows details for the selected load balancer.

Name	DNS name	State	VPC ID	Availability Zones	Type
terraform-asg-example	terraform-asg-example-2474...		vpc-7714d51e	ap-south-1b, ap-south-1a	classic

Load balancer: terraform-asg-example

Basic Configuration

Name:	terraform-asg-example	Creation time:	October 10, 2017 at 6:22:49 PM UTC+6
* DNS name:	terraform-asg-example-247485449.ap-south-1.elb.amazonaws.com (A Record)	Hosted zone:	ZP97RAFLXTNZK
Scheme:	internet-facing	Status:	2 of 2 instances in service
Availability Zones:	subnet-16814c7f - ap-south-1a, subnet-345e537e - ap-south-1b	VPC:	vpc-7714d51e

5. Delete the stack

Run terraform destroy

```
$ **terraform destroy
**Do you really want to destroy?
  Terraform will delete all your managed infrastructure.
  There is no undo. Only 'yes' will be accepted to confirm.
```

```
Enter a value:
```

Once you type in “yes” and hit enter, Terraform will build the dependency graph and delete all the resources in the right order, using as much parallelism as possible. In about a minute, your AWS account should be clean again.