Deploy BOSH

[**Configure your AWS access**](#_sq8a6nz4hw64) **1**

[Generate an Access Key and Secret Key](#_bhqvzlesoxgs) 1

[**Configure Your AWS environment for BOSH**](#_bqsbenkuxbn9) **2**

[Create a Terraform Script](#_uilryoi265rw) 2

[Specify the provider](#_hm1vrewu9xyp) 2

[Create a Virtual Private Cloud](#_d3ivvtz4f55) 2

[Create a Subnet](#_wve6fte6ge9r) 3

[Create an Internet Gateway and attach it to the VPC](#_xs0tpi8qfd3q) 3

[Create a Route Table and associate it with the Subnet](#_2l8ysufs30jm) 4

[Create the Security Groups](#_7fsrdxmjz0b6) 4

[Create a Security Group for BOSH](#_3x7bz7cjl2gb) 5

[Create a Security Group for the Jumpbox](#_42o04bka8r22) 6

[Create a Key Pair.](#_f1xhfl7eewtk) 6

[Create your Jumpbox](#_4gytvykr7cgw) 7

[Create an Elastic IP](#_et54bjfoeuyv) 7

[Add outputs](#_tnyxh68l5f20) 8

[Configuring your Terraform script](#_xy82j5uotict) 9

[Create a variable file](#_xpw4oqdyle4n) 9

[Add default values](#_buocxjuveplx) 9

[Creating the infrastructure](#_2d71o9v01vg6) 9

[**Deploy BOSH**](#_cggy1epjrjb4) **10**

[Preparing the deployment](#_32bah7xu8l5u) 10

[Accessing BOSH](#_f2ret0yl6zq6) 11

## Configure your AWS access

#### Generate an Access Key and Secret Key

1. Go to your AWS Console
2. Click in your username at the menu
3. Select “Security credentials”
4. If asked, click on “Continue to Security Credentials”
5. If you find previous credentials, just delete them.
6. Click on the “+” symbol at the “Access Keys” row
   1. Click on “Create a New Access Key”
   2. Click on “Download Key File”
   3. Click on “Close”
7. Have it readily available, you will need this file later.

## Configure Your AWS environment for BOSH

There are several options to configure an AWS account for a BOSH deployment - using [AWS web interface](https://bosh.io/docs/init-aws.html) or manually via CLI. Below we provide an explanation on how to configure your AWS account for BOSH.

In order to do this, we will use Terraform, a great tool by Hashicorp that will help us create a repeatable, easily tweakable Infrastructure As Code script.

Download the Terraform CLI for your OS [here](https://www.terraform.io/downloads.html). Once you have the CLI executable, put it accessible in your PATH.

### Create a Terraform Script

Create a directory called “deployment”. Our example is based in Linux, but you can adapt it to any OS with little effort.

|  |
| --- |
| $ nano ~/deployment/bosh.tf  # OR  $ vim ~/deployment/bosh.tf  # OR the editor of your choice |

#### Specify the provider

|  |
| --- |
| provider "aws" {  region = "${var.region}" } |

#### Create a Virtual Private Cloud

[Amazon Virtual Private Cloud (Amazon VPC)](https://aws.amazon.com/vpc/) allows you to provision a logically isolated section of the Amazon Web Services (AWS) cloud, where you can launch AWS resources in a virtual network that you define.

To add a VPC to your Terraform script, type the following code block into your bosh.tf file:

|  |
| --- |
| resource "aws\_vpc" "bosh" {  cidr\_block = "${var.bosh\_vpc\_cidr}"  enable\_dns\_hostnames = true  tags {  Name = "training\_vpc"  }  } |

#### Create a [Subnet](http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Subnets.html)

After creating a VPC, we can add one or more subnets in each Availability Zone (AVZ). When we create a subnet, we specify the Classless Inter-Domain routing (CIDR) block for the subnet, which is a subset of the VPC CIDR block. Each subnet must reside entirely within one Availability Zone and cannot span zones. Availability Zones are distinct locations that are engineered to be isolated from failures in other Availability Zones. Each subnet gets a unique ID.

#### 

|  |
| --- |
| resource "aws\_subnet" "bosh" {  vpc\_id = "${aws\_vpc.bosh.id}"  cidr\_block = "${var.bosh\_subnet\_cidr}"  availability\_zone = "${var.default\_az}"  map\_public\_ip\_on\_launch = true  tags {  Name = "training\_subnet"  }  } |

#### Create an [Internet Gateway](http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Internet_Gateway.html) and attach it to the VPC

An Internet gateway is a horizontally scaled, redundant, and highly available VPC component that allows for communication between instances in your VPC and the Internet. It, therefore, imposes no availability risks or bandwidth constraints on your network traffic.

An Internet gateway serves two purposes: to provide a target in your VPC route tables for Internet-routable traffic, and to perform network address translation (NAT) for instances that have been assigned public IPv4 addresses.

Create an new Internet Gateway:

|  |
| --- |
| resource "aws\_internet\_gateway" "bosh" {  vpc\_id = "${aws\_vpc.bosh.id}"  } |

#### Create a [Route Table](http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Route_Tables.html) and associate it with the Subnet

A route table contains a set of rules, called routes, that are used to determine where network traffic is directed.

Each subnet in your VPC must be associated with a route table; the table controls the routing for the subnet. A subnet can only be associated with one route table at a time, but you can associate multiple subnets with the same route table.

Create a Route Table:

|  |
| --- |
| resource "aws\_route\_table" "bosh" {  vpc\_id = "${aws\_vpc.bosh.id}"  route {  cidr\_block = "0.0.0.0/0"  gateway\_id = "${aws\_internet\_gateway.bosh.id}"  }  tags {  Name = "training\_route\_table"  }  }  resource "aws\_route\_table\_association" "bosh" {  subnet\_id = "${aws\_subnet.bosh.id}"  route\_table\_id = "${aws\_route\_table.bosh.id}"  } |

#### Create the [Security Groups](http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_SecurityGroups.html)

A security group acts as a virtual firewall for your instance to control inbound and outbound traffic. When you launch an instance in a VPC, you can assign the instance to up to five security groups. Security groups act at the instance level, not the subnet level. Therefore, each instance in a subnet in your VPC could be assigned to a different set of security groups. If you don't specify a particular group at launch time, the instance will be automatically assigned to the default security group for the VPC.

##### Create a Security Group for BOSH

|  |
| --- |
| resource "aws\_security\_group" "bosh" {  name = "bosh"  description = "Security group for bosh deployment"  vpc\_id = "${aws\_vpc.bosh.id}"  /\*\* Allow ICMP (pings) packages \*\*/  ingress {  from\_port = 0  to\_port = 0  protocol = "icmp"  cidr\_blocks = ["0.0.0.0/0"]  }  /\*\* Allow SSH connections \*\*/  ingress {  from\_port = 22  to\_port = 22  protocol = "tcp"  cidr\_blocks = ["0.0.0.0/0"]  }  /\*\* Allow BOSH Agent access from the CLI \*\*/  ingress {  from\_port = 6868  to\_port = 6868  protocol = "tcp"  cidr\_blocks = ["0.0.0.0/0"]  }  /\*\* Allow BOSH Director access from the CLI \*\*/  ingress {  from\_port = 25555  to\_port = 25555  protocol = "tcp"  cidr\_blocks = ["0.0.0.0/0"]  }  /\*\* Allow all protocols, all addresses, all ports inside the group \*\*/  ingress {  from\_port = "0"  to\_port = "0"  protocol = "-1"  self = true  }  /\*\* Allow egress for every port, every protocol \*\*/  egress {  from\_port = "0"  to\_port = "0"  protocol = "-1"  cidr\_blocks = ["0.0.0.0/0"]  }  tags {  Name = "bosh"  }  } |

##### 

##### Create a Security Group for your Jumpbox

|  |
| --- |
| resource "aws\_security\_group" "jumpbox" {  name = "cf\_training\_jumpbox"  description = "Security group for the jumpbox"  vpc\_id = "${aws\_vpc.bosh.id}"  /\*\* Allow ICMP (pings) packages \*\*/  ingress {  from\_port = 0  to\_port = 0  protocol = "icmp"  cidr\_blocks = ["0.0.0.0/0"]  }  /\*\* Allow SSH connections \*\*/  ingress {  from\_port = 22  to\_port = 22  protocol = "tcp"  cidr\_blocks = ["0.0.0.0/0"]  }  /\*\* Allow egress for every port, every protocol \*\*/  egress {  from\_port = "0"  to\_port = "0"  protocol = "-1"  cidr\_blocks = ["0.0.0.0/0"]  }  tags {  Name = "cf\_trainig\_jumpbox"  }  } |

##### 

#### Create a [Key Pair](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-key-pairs.html) for your BOSH Director and your Jumpbox.

Amazon EC2 uses public–key cryptography to encrypt and decrypt login information. Public–key cryptography uses a public key to encrypt a piece of data, such as a password, then the recipient uses the private key to decrypt the data. The public and private keys are known as a key pair.

To log in to your instance, you must create a key pair, specify its name when you launch the instance, and provide the private key when you connect to the instance. Linux instances have no password, and you use a key pair to log in using SSH.

Create a new KeyPair:

* Run in the command line:

|  |
| --- |
| mkdir ~/deployment/ssh  ssh-keygen -t rsa -C "bosh" -P '' -f ~/deployment/ssh/bosh.pem -b 4096  ssh-keygen -t rsa -C "bosh" -P '' -f ~/deployment/ssh/jumpbox.pem -b 4096 |

* Edit the bosh.tf file and add at the end:

|  |
| --- |
| resource "aws\_key\_pair" "jumpbox" {  key\_name = "jumpbox-training\_key"  public\_key = "${file("ssh/jumpbox.pem.pub")}" }  resource "aws\_key\_pair" "bosh" {  key\_name = "bosh-training\_key"  public\_key = "${file("ssh/bosh.pem.pub")}" } |

#### Create your Jumpbox

In order to safely access BOSH from inside the VPC, we need to create a Jumpbox that will help us execute the CLI commands from it without compromising external access.

|  |
| --- |
| resource "aws\_instance" "jumpbox" {  ami = "ami-da05a4a0"  instance\_type = "t2.micro"  subnet\_id = "${aws\_subnet.bosh.id}"  key\_name = "${aws\_key\_pair.jumpbox.key\_name}"  vpc\_security\_group\_ids = [  "${aws\_security\_group.jumpbox.id}"  ]  tags {  Name = "cf\_training\_jumpbox"  }  root\_block\_device {  volume\_type = "gp2"  volume\_size = 30  }  } |

#### Create an [Elastic IP](http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html)

An Elastic IP address is a static, public IPv4 address designed for dynamic cloud computing. Elastic IP address could be associated with any instance or network interface for any VPC in your account. With an Elastic IP address, you can mask the failure of an instance by rapidly remapping the address to another instance in your VPC.

Add the following to the end of the file:

|  |
| --- |
| resource "aws\_eip" "jumpbox" {  instance = "${aws\_instance.jumpbox.id}"  vpc = true  } |

Now save the file.

#### Add outputs

Terraform offers a way to view relevant data about our deployment thru the “output” key.  
In this way we will be able to have all the necessary deployment data at hand.

|  |
| --- |
| output "region" {  value = "${var.region}"  }  output "az" {  value = "${aws\_subnet.bosh.availability\_zone}"  }  output "subnet\_id" {  value = "${aws\_subnet.bosh.id}"  }  output "internal\_cidr" {  value = "${aws\_subnet.bosh.cidr\_block}"  }  output "default\_security\_groups" {  value = "[${aws\_security\_group.bosh.id}]"  }  output "default\_key\_name" {  value = "${aws\_key\_pair.bosh.key\_name}"  }  output "jumpbox\_ip" {  value = "${aws\_eip.jumpbox.public\_ip}"  } |

Save the file and exit

### Configuring your Terraform script

Terraform offers a great way to separate configuration values from the actual infrastructure itself.

#### Create a variable file

|  |
| --- |
| $ nano ~/deployment/variables.tf  # OR  $ vim ~/deployment/variables.tf |

#### Add default values

|  |
| --- |
| variable "default\_az" {  description = "Default availability zone which is used during the deployment"  default = "us-east-1a"  }  variable "region" {  description = "AWS region to host the bosh environment"  default = "us-east-1"  }  variable "bosh\_vpc\_cidr" {  description = "CIDR for VPC"  default = "10.0.0.0/16"  }  variable "bosh\_subnet\_cidr" {  description = "CIDR for bosh subnet"  default = "10.0.0.0/24"  } |

Save the file and exit.

### Creating the infrastructure

Terraform will need the Access Key and the Secret Key to manage our AWS infrastructure. It is not recommended to save them into a file, since this can lead to accidentally pushing that file to a repo, therefore exposing our infrastructure to non-authorized people.

You will need the Access Key and Secret Key you generated while configuring your AWS secret access keys.

In the command line:

|  |
| --- |
| echo "export AWS\_ACCESS\_KEY\_ID=your\_aws\_access\_key\_id" >> ~/.bashrc  echo "export AWS\_SECRET\_ACCESS\_KEY=your\_aws\_secret\_access\_key" >> ~/.bashrc  source ~/.bashrc |

Now you can test if your Terraform script is ok:

|  |
| --- |
| cd ~/deployment  terraform plan |

If no error is shown, go ahead and create the infrastructure:

|  |
| --- |
| terraform apply |

Once Terraform has created the infrastructure, you can go into your AWS Console and check out all the resources created for you.

## Deploy BOSH

Now that we have our BOSH infrastructure in place, let’s deploy BOSH!

With BOSH CLI V1, we needed [bosh-init](https://github.com/cloudfoundry/bosh-init) to deploy BOSH itself. This additional tool that deployed BOSH in our IaaS is no longer necessary, since its functionality was merged into the BOSH CLI V2.

### Preparing the deployment

One of the many advantages of Terraform is that it can output the data we need to proceed with the deployment of our platform. When you run terraform output, you will see the output variables declared in our bosh.tf file. But, the format of this file is not compatible with our BOSH CLI, since the BOSH CLI is expecting a YAML file, and Terraform outputs the values as a list of variable=value lines. The good news is that Linux provides a great tool to change this, called sed.

So, we can run:

|  |
| --- |
| terraform output > bosh-vars.yml  sed -i "s/ =/:/g" bosh-vars.yml  ssh -i ~/deployment/ssh/jumpbox.pem ubuntu@$(terraform output jumpbox\_ip) "mkdir -p ~/deployment/ssh"  scp -i ~/deployment/ssh/jumpbox.pem bosh-vars.yml ubuntu@$(terraform output jumpbox\_ip):/home/ubuntu/deployment/bosh-vars.yml  scp -i ~/deployment/ssh/jumpbox.pem ~/deployment/ssh/bosh.pem ubuntu@$(terraform output jumpbox\_ip):/home/ubuntu/deployment/ssh/bosh.pem |

And we will have almost all the variables required for our BOSH deployment ready.

The rest of the required variables, we are going to input them by hand:

|  |
| --- |
| ssh -i ~/deployment/ssh/jumpbox.pem ubuntu@$(terraform output jumpbox\_ip) << EOF  echo "director\_name: my-bosh" >> ~/deployment/bosh-vars.yml  echo "internal\_ip: 10.0.0.6" >> ~/deployment/bosh-vars.yml  echo "internal\_gw: 10.0.0.1" >> ~/deployment/bosh-vars.yml  # Replace with the placeholders with the corresponding values  echo "access\_key\_id: $AWS\_ACCESS\_KEY\_ID" >> ~/deployment/bosh-vars.yml  echo "secret\_access\_key: $AWS\_SECRET\_ACCESS\_KEY" >> ~/deployment/bosh-vars.yml  EOF |

Now, we are going to need a deployment manifest to tell the CLI what to deploy.

We are going to use the new bosh-deployment style, and deploy BOSH using the vanilla, official templates.

|  |
| --- |
| # Log in into our Jumpbox  ssh -i ~/deployment/ssh/jumpbox.pem ubuntu@$(terraform output jumpbox\_ip)  cd ~/deployment  # Get the AWS CPI configurator  curl -o cpi.yml -J -L https://raw.githubusercontent.com/cloudfoundry/bosh-deployment/master/aws/cpi.yml  # Get the Cloud Config for later use  curl -o cloud-config.yml -J -L https://raw.githubusercontent.com/cloudfoundry/bosh-deployment/master/aws/cloud-config.yml  # Get the BOSH deployment manifest  curl -o bosh.yml -J -L https://raw.githubusercontent.com/cloudfoundry/bosh-deployment/master/bosh.yml |

Before deploying BOSH, we need to install BOSH’s dependencies for creating an environment:

|  |
| --- |
| #Install basic packages  sudo apt-get update  sudo apt-get -y install gnupg2 git unzip tree  # Install RVM, a Ruby version management software  gpg2 --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3  \curl -sSL https://get.rvm.io | bash -s stable  source /home/ubuntu/.rvm/scripts/rvm  # Test that RVM is running correctly  rvm help  # Install Ruby 2.2.0  rvm install 2.2.0  # Install bundler package manager  gem install bundler  # Install the BOSH CLI V2  curl -o bosh\_cli\_v2 -J -L https://s3.amazonaws.com/bosh-cli-artifacts/bosh-cli-2.0.40-linux-amd64  sudo install -m0755 bosh\_cli\_v2 /usr/local/bin/bosh  rm bosh\_cli\_v2  # Install the CF CLI  curl -o cf\_cli.deb -J -L 'https://cli.run.pivotal.io/stable?release=debian64&source=github'  sudo dpkg -i cf\_cli.deb  rm cf\_cli.deb |

Now we are ready. Let’s deploy BOSH!

|  |
| --- |
| bosh create-env bosh.yml -o cpi.yml --state=bosh.state --vars-store creds.yml --vars-file=bosh-vars.yml --var-file private\_key=./ssh/bosh.pem |

### Accessing BOSH

When BOSH finishes deploying, we need to add the credentials to our environment to have a smooth and secure workflow:

|  |
| --- |
| echo "export BOSH\_CLIENT=admin" >> ~/.bashrc  echo "export BOSH\_ENVIRONMENT=my-bosh" >> ~/.bashrc  echo "export BOSH\_CLIENT\_SECRET=`bosh int ./creds.yml --path /admin\_password`" >> ~/.bashrc  source ~/.bashrc |

Once the client ID and client secret are set, we can create an alias for our environment:

|  |
| --- |
| $ bosh alias-env my-bosh -e `bosh int ./bosh-vars.yml --path /internal\_ip` --ca-cert <(bosh int ./creds.yml --path /director\_ssl/ca) |

And now, we can login and get information about our new BOSH environment:

|  |
| --- |
| $ bosh -e my-bosh login  $ bosh -e my-bosh env |

Done! You have deployed BOSH!