Deploy BOSH

[**Configure Your AWS environment for BOSH**](#_bqsbenkuxbn9) **1**

[Create a Terraform Script](#_uilryoi265rw) 1

[Specify the provider](#_hm1vrewu9xyp) 2

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

[Create a Subnet](#_wve6fte6ge9r) 2

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

[Create a Route Table and associate it with the Subnet](#_2l8ysufs30jm) 3

[Create a Security Group](#_7fsrdxmjz0b6) 4

[Create an Elastic IP](#_et54bjfoeuyv) 5

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

[Add outputs](#_tnyxh68l5f20) 6

[Configuring your Terraform script](#_xy82j5uotict) 7

[Create a variable file](#_xpw4oqdyle4n) 7

[Add default values](#_buocxjuveplx) 7

[Creating the infrastructure](#_2d71o9v01vg6) 8

[**Deploy BOSH**](#_cggy1epjrjb4) **8**

[Preparing the deployment](#_32bah7xu8l5u) 8

[Accessing BOSH](#_f2ret0yl6zq6) 9

## 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.

### Create a Terraform Script

Open a file in your Jumpbox

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

#### 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 a [Security Group](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:

|  |
| --- |
| 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 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.

Create an Elastic IP

|  |
| --- |
| resource "aws\_eip" "bosh" {  } |

Now save the file.

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

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 |

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

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

#### 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 = "[${aws\_security\_group.bosh.id}]"  }  output "elastic\_ip" {  value = "${aws\_eip.bosh.public\_ip}"  }  output "default\_key\_name" {  value = "${aws\_key\_pair.bosh.key\_name}"  } |

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-west-2c"  }  variable "region" {  description = "AWS region to host the bosh environment"  default = "us-west-2"  }  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 Jumpbox. Replace the “your\_aws\*” values.

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 |

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

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

|  |
| --- |
| $ echo "director\_name: my-bosh" >> bosh-vars.yml  $ echo "access\_key\_id: $AWS\_ACCESS\_KEY\_ID" >> bosh-vars.yml  $ echo "secret\_access\_key: $AWS\_SECRET\_ACCESS\_KEY" >> bosh-vars.yml  $ echo "internal\_ip: 10.0.0.6" >> bosh-vars.yml  $ echo “internal\_gw: 10.0.0.1” >> bosh-vars.yml |

Now, we are going to need a deployment manifest to tell the CLI what to deploy. Plase create a bosh.yml file and copy/paste the following into the file:

|  |
| --- |
| name: bosh  disk\_pools: - cloud\_properties:  type: gp2  disk\_size: 32768  name: disks  networks: - name: default  subnets:  - cloud\_properties:  subnet: ((subnet\_id))  dns:  - 10.0.0.2  gateway: ((internal\_gw))  range: ((internal\_cidr))  static:  - ((elastic\_ip))  type: manual - name: public  type: vip  releases: - name: bosh  sha1: 37630c636a030454d55703cfdd495d733525353f  url: https://s3.amazonaws.com/bosh-compiled-release-tarballs/bosh-263.2.0-ubuntu-trusty-3445.7-20170915-052445-759948783-20170915052451.tgz?versionId=3kjVQuRK7JyrnbKEbKhLRi7qyo0wBXZJ  version: 263.2.0 - name: bosh-aws-cpi  sha1: 117d5518f1b1de8937c163244de8db45ad8ce1a9  url: https://bosh.io/d/github.com/cloudfoundry-incubator/bosh-aws-cpi-release?v=66  version: 66  resource\_pools: - cloud\_properties:  availability\_zone: ((az))  ephemeral\_disk:  size: 25000  type: gp2  instance\_type: m4.xlarge  env:  bosh:  mbus:  cert: ((mbus\_bootstrap\_ssl))  password: '\*'  name: vms  network: default  stemcell:  sha1: ad82e910dc07b1ef4fe475776e11adbc83470538  url: https://bosh.io/d/stemcells/bosh-aws-xen-hvm-ubuntu-trusty-go\_agent?v=3445.7  instance\_groups: - instances: 1  jobs:  - name: nats  release: bosh  - name: postgres-9.4  release: bosh  - name: blobstore  release: bosh  - name: director  release: bosh  - name: health\_monitor  release: bosh  - name: registry  release: bosh  - name: aws\_cpi  release: bosh-aws-cpi  name: bosh  networks:  - default:  - dns  - gateway  name: default  static\_ips:  - ((internal\_ip))  - name: public  static\_ips:  - ((elastic\_ip))  persistent\_disk\_pool: disks  properties:  agent:  mbus: nats://nats:((nats\_password))@((elastic\_ip)):4222  aws:  access\_key\_id: ((access\_key\_id))  default\_key\_name: ((default\_key\_name))  default\_security\_groups: ((default\_security\_groups))  region: ((region))  secret\_access\_key: ((secret\_access\_key))  blobstore:  address: ((elastic\_ip))  agent:  password: ((blobstore\_agent\_password))  user: agent  director:  password: ((blobstore\_director\_password))  user: director  port: 25250  provider: dav  director:  address: 127.0.0.1  cpi\_job: aws\_cpi  db:  adapter: postgres  database: bosh  host: 127.0.0.1  listen\_address: 127.0.0.1  password: ((postgres\_password))  user: postgres  enable\_dedicated\_status\_worker: true  enable\_nats\_delivered\_templates: true  enable\_post\_deploy: true  events:  record\_events: true  flush\_arp: true  generate\_vm\_passwords: true  name: ((director\_name))  ssl:  cert: ((director\_ssl.certificate))  key: ((director\_ssl.private\_key))  user\_management:  local:  users:  - name: admin  password: ((admin\_password))  - name: hm  password: ((hm\_password))  provider: local  workers: 4  hm:  director\_account:  ca\_cert: ((director\_ssl.ca))  password: ((hm\_password))  user: hm  resurrector\_enabled: true  nats:  address: 127.0.0.1  password: ((nats\_password))  user: nats  ntp:  - time1.google.com  - time2.google.com  - time3.google.com  - time4.google.com  postgres:  adapter: postgres  database: bosh  host: 127.0.0.1  listen\_address: 127.0.0.1  password: ((postgres\_password))  user: postgres  registry:  address: ((elastic\_ip))  db:  adapter: postgres  database: bosh  host: 127.0.0.1  password: ((postgres\_password))  user: postgres  host: ((elastic\_ip))  http:  password: ((registry\_password))  port: 25777  user: registry  password: ((registry\_password))  port: 25777  username: registry  resource\_pool: vms  variables: - name: admin\_password  type: password - name: blobstore\_director\_password  type: password - name: blobstore\_agent\_password  type: password - name: hm\_password  type: password - name: mbus\_bootstrap\_password  type: password - name: nats\_password  type: password - name: postgres\_password  type: password - name: default\_ca  options:  common\_name: ca  is\_ca: true  type: certificate - name: mbus\_bootstrap\_ssl  options:  alternative\_names:  - ((elastic\_ip))  ca: default\_ca  common\_name: ((elastic\_ip))  type: certificate - name: director\_ssl  options:  alternative\_names:  - ((elastic\_ip))  ca: default\_ca  common\_name: ((elastic\_ip))  type: certificate - name: registry\_password  type: password  cloud\_provider:  cert: ((mbus\_bootstrap\_ssl))  mbus: https://mbus:((mbus\_bootstrap\_password))@((elastic\_ip)):6868  properties:  agent:  mbus: https://mbus:((mbus\_bootstrap\_password))@0.0.0.0:6868  aws:  access\_key\_id: ((access\_key\_id))  default\_key\_name: ((default\_key\_name))  default\_security\_groups: ((default\_security\_groups))  region: ((region))  secret\_access\_key: ((secret\_access\_key))  blobstore:  path: /var/vcap/micro\_bosh/data/cache  provider: local  ntp:  - time1.google.com  - time2.google.com  - time3.google.com  - time4.google.com  ssh\_tunnel:  host: ((elastic\_ip))  port: 22  private\_key: ((private\_key))  user: vcap  template:  name: aws\_cpi  release: bosh-aws-cpi |

Let’s deploy BOSH!

|  |
| --- |
| $ bosh create-env bosh.yml --vars-file=bosh-vars.yml --vars-store=creds.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 /elastic\_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!