

1 Big Data Platform Cloud Deployment

1.1 Tools Installation

This section explains the installation of the tools Terraform for the infrastructure deployment and Ansible for the provisioning.

1.1.1 Ansible

The Ansible installation is explained in the “Install Ansible” section of [\(Tools Installation\)](#).

1.1.2 Terraform

The Terraform installation is explained in the “Install Terraform” section of [\(Tools Installation\)](#).

1.2 Data Center deployment

We can find the Data Center deployment for the AWS cloud inside the directory “.../code/despliegue-plataforma/cloud”

1.2.1 Configure AWS account access

To avoid writing AWS access keys in terraform files, the AWS they have to be exported to the machine itself as an environment variable.

For that it is necessary to export the “AWS_ACCESS_KEY_ID” and “AWS_SECRET_ACCESS_KEY”.

```
$ export AWS_ACCESS_KEY_ID=AKIAIFM5VPP5BESYIG5A
$ export
AWS_SECRET_ACCESS_KEY=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
```

1.2.2 Configure number of nodes

In the directory “.../Kubernetes-Data-Platform /aws-infrastructure”, we can find ‘*terraform.tfvars*’ file, where we can configure how many nodes we want in the cluster, like how many masters, private-slaves or public-slaves.

1.2.3 Deploy Infrastructure + Data Platform

To automate the deployment, the scale, the delete of the infrastructure and the provision we can use the following scripts.

The first script **cluster-setup.sh** deploys the infrastructure in AWS using Terraform and provisions it with Ansible. To run this script we can use:

```
$ ./cluster-setup.sh
```

To scale the AWS platform, we only have to change the number of nodes explained on [\(Configure number of nodes\)](#) , and then run the **cluster-setup.sh** script. So it would add the new nodes to the actual cluster. To run this script we can use:

```
$ ./cluster-setup.sh
```

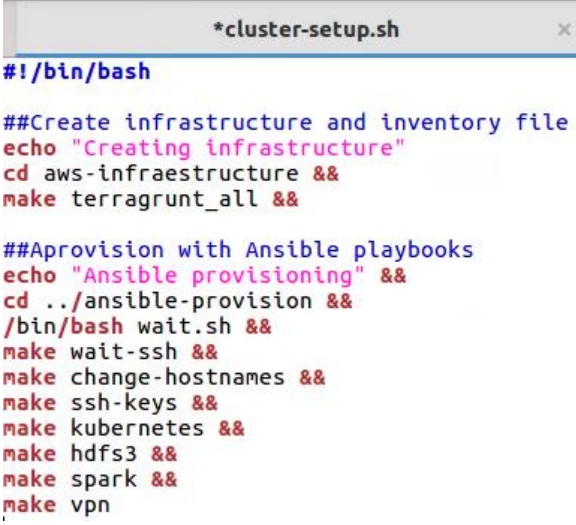
To destroy completely the platform, we can use the **cluster-destroy.sh** script. This script destroys the infrastructure of AWS deployed by Terraform. To run this script we can use:

```
$ ./cluster-destroy.sh
```

1.3 Used Scripts

This section explains the different scripts used for deploying the Big Data Platform in the cloud.

Cluster-setup.sh



```
#!/bin/bash

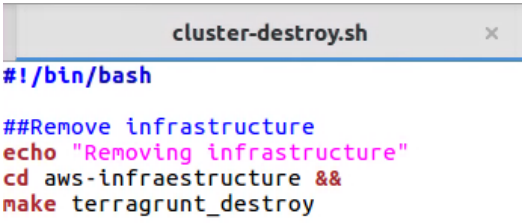
##Create infrastructure and inventory file
echo "Creating infrastructure"
cd aws-infrastructure &&
make terragrunt_all &&

##Aprovision with Ansible playbooks
echo "Ansible provisioning" &&
cd ../ansible-provision &&
/bin/bash wait.sh &&
make wait-ssh &&
make change-hostnames &&
make ssh-keys &&
make kubernetes &&
make hdfs3 &&
make spark &&
make vpn
```

The first thing this script does is access the directory where terraform files are located, and then it executes "make terragrunt_all" to deploy the infrastructure in the cloud.

Once deployed, the script runs the "wait.sh" script to wait until the instances are up and accessible so finally it deploys the Big Data platform with the ansible playbooks.

Cluster-destroy.sh



```
#!/bin/bash

##Remove infrastructure
echo "Removing infrastructure"
cd aws-infrastructure &&
make terragrunt_destroy
```

The only function of this script is to destroy the deployed platform.

Wait.sh

```
wait.sh x Makefile x README.md
#!/bin/bash

user="centos"
List="34.244.222.71 34.250.77.11 34.244.151.122 34.245.93.169 34.244.117.82"
hosts=($List)
maxConnectionAttempts=200
up=false
instance_up=false
index=1

while [[ $up == false && $index -le $maxConnectionAttempts ]]
do
    sleep 2
    up=true
    echo -e "\n\nWaiting for hosts up"
    echo "Try" $index
    echo "Hosts up?"
    for host in "${hosts[@]}"
    do
        ssh -oStrictHostKeyChecking=no -q -o "BatchMode=yes" $user@$host "exit" && instance_up=true || instance_up=false
        if test $instance_up = false; then
            up=false;
        fi
        echo "$host: $instance_up";
    done
    ((index++))
done

echo -e "\n\nAll Instances are up and running"
echo -e "\n"
```

When the cloud infrastructure is deployed, terraform automatically writes the Public-IPs of the instances in the “List” variable of this script.

The script try to make SSH connections every 2 seconds and with a maximum of 200 attempts (about 6-7 minutes), when it detect that all the instances are up the script ends, otherwise it keeps trying until the maximum of attempts.

The output of the script would be the next:

```
Waiting for hosts up
Try 1
Hosts up?
34.244.222.71: false
34.250.77.11: true
34.244.151.122: true
34.245.93.169: true
34.244.117.82: true

Waiting for hosts up
Try 2
Hosts up?
34.244.222.71: true
34.250.77.11: true
34.244.151.122: true
34.245.93.169: true
34.244.117.82: true
```

1.3.1 VPN connection

To access HDFS and Spark it is necessary to do it through the private IP of the AWS instance, so you have to configure the VPN. On the cluster-setup.sh script, the VPN have been installed in the Bootstrap node.

So we only have to access to the OpenVPNServer, configure the public IP and download the VPN access file.

First, we have to Access to the OpenVPNServer using a Web browser.

```
https://Bootstrap-Public-IP:943/admin
```

Once inside, we will access through the configured username and password, by default those are configured has (user: openvpn, password: ikerlanopenvpn).

Then, we will access to the “Server Network Settings” section and change the private IP with the public IP. We can see in the next illustration (Illustration 1: OpenVPN Server Network Settings). Once configured, we save the configuration.

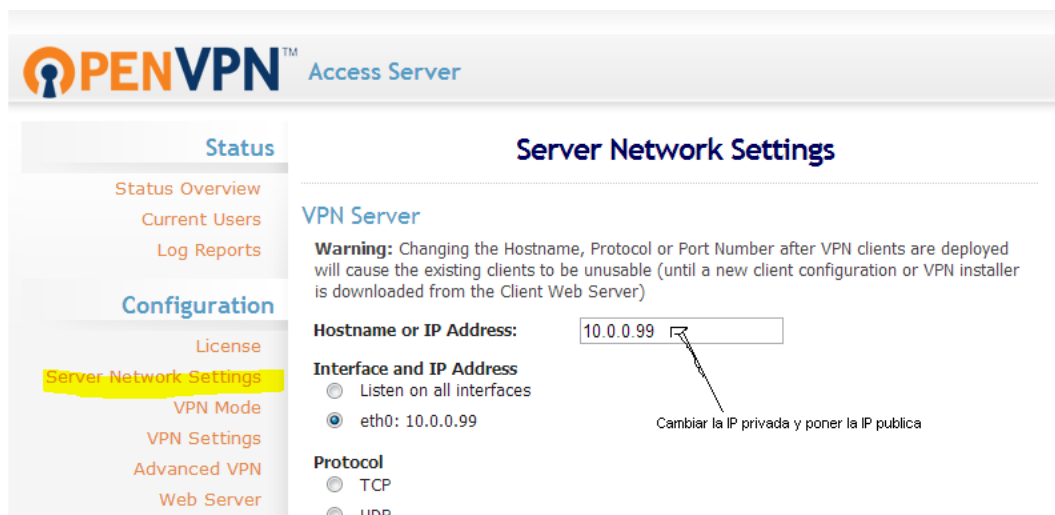


Illustration 1: OpenVPN Server Network Settings

Now we will proceed to connect to the VPN, for this we must first download the configuration file from the OpenVPN client interface:

```
https://Bootstrap-Public-IP:943/
```

In this interface, we have to select “**Yourself (user-locked profile)**”, and the file will start downloading.

Once the file has been downloaded, we can use the next command to connect to the VPN:

```
$ sudo openvpn --config Descargas/client.ovpn
```