



**Student Name:** Vinod Kumar Dhanavath  
**Course:** CSCL1030 –CloudOps Tools and Techniques  
**Instructor:** Joba Hassan  
**Date:** 4-02-2025

## I. Introduction:

This lab exercise involved leveraging Terraform and Ansible to automate the provisioning and configuration of AWS infrastructure. The goal was to deploy a control node and four web servers, organizing them into distinct groups. This report provides a step-by-step account of the process, modifications made to the provided code, and evidence of successful execution through screenshots.

## II. Infrastructure Deployment

## 1. Cloning the Repository or Forked Repositories

The following repositories were cloned into my GitHub account:

- ProvisionControlNode
  - ProvisionManagedNodes
  - ControlNodeFiles

## 2. Setting Up the Control Node

Following the instructions from Class 6, the Control Node was provisioned using Terraform. The SSH key output was extracted and utilized to generate an `id_rsa.pub` key in the **ProvisionManagedNodes** repository. Following is the Control Node output with **Public Ip="3.84.189.198"**

### 3. Deploying Managed Nodes

Four web servers were created and categorized into groups in the hosts.ini file.

**[einstein]** – The first two web servers

**54.211.73.69**

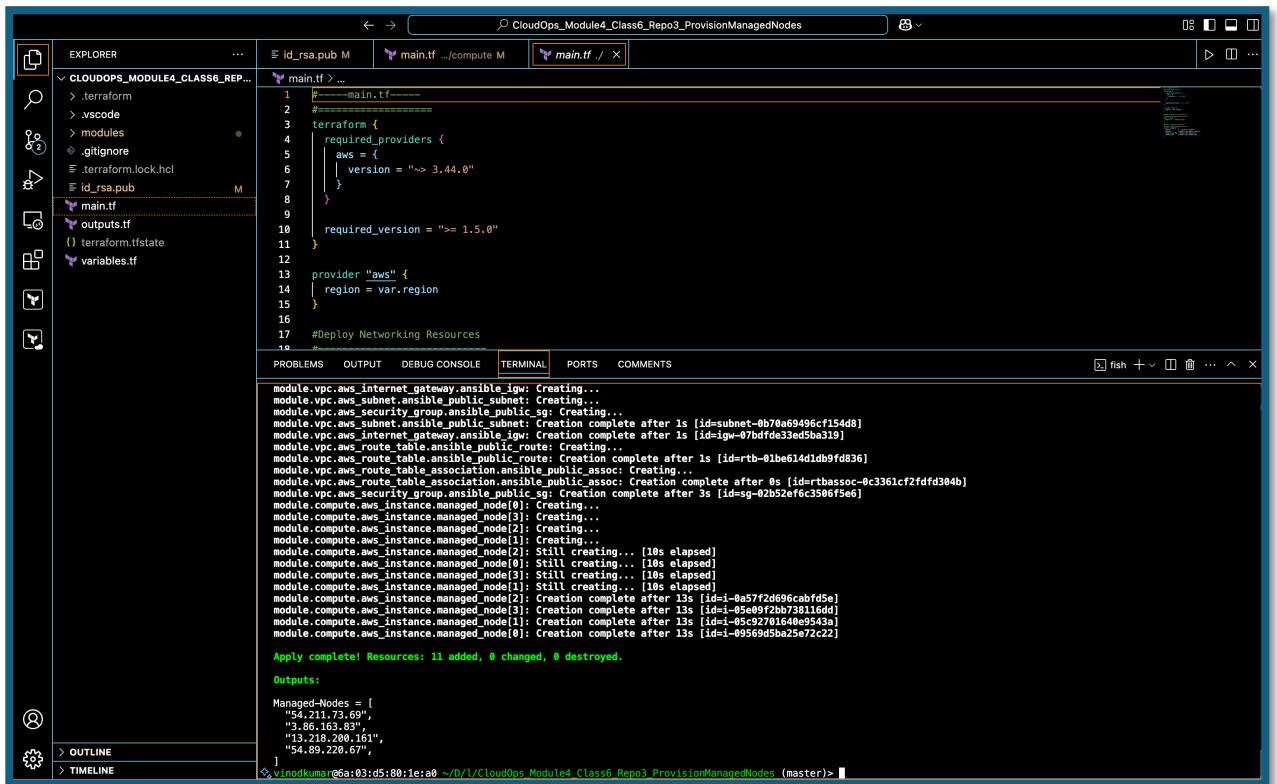
**3.86.163.83**

**[Cloud]** – The last two web servers

**13.218.200.161**

**54.89.220.67**

Following is the screenshot to demonstrate the all the managed nodes and their “Public Ips”.



```
1  -----main.tf-----
2  =====
3  terraform {
4      required_providers {
5          aws = {
6              | version = "> 3.44.0"
7          }
8      }
9      required_version = ">= 1.5.0"
10 }
11
12 provider "aws" {
13     region = var.region
14 }
15
16 #Deploy Networking Resources
17
18 module.vpc.aws_internet_gateway.igw: Creating...
19 module.vpc.aws_subnet.ansible_public_subnet: Creating...
20 module.vpc.aws_security_group.ansible_public_sg: Creating...
21 module.vpc.aws_subnet.ansible_public_subnet: Creation complete after 1s [id=subnet-0b70a69496cf1394d8]
22 module.vpc.aws_internet_gateway.igw: Creation complete after 1s [id=igw-070fdde33ed5ba319]
23 module.vpc.aws_route_table.ansible_public_route: Creating...
24 module.vpc.aws_route_table_association.ansible_public_assoc: Creating...
25 module.vpc.aws_route_table_association.ansible_public_assoc: Creation complete after 0s [id=rtbassoc-0c3361cf2fd836]
26 module.vpc.aws_security_group.ansible_public_sg: Creation complete after 3s [id=sg-02b52ef6c3506f5e6]
27 module.compute.aws_instance.managed_node[0]: Creating...
28 module.compute.aws_instance.managed_node[1]: Creating...
29 module.compute.aws_instance.managed_node[2]: Creating...
30 module.compute.aws_instance.managed_node[1]: Creating...
31 module.compute.aws_instance.managed_node[0]: Still creating... [10s elapsed]
32 module.compute.aws_instance.managed_node[0]: Still creating... [10s elapsed]
33 module.compute.aws_instance.managed_node[1]: Still creating... [10s elapsed]
34 module.compute.aws_instance.managed_node[1]: Still creating... [10s elapsed]
35 module.compute.aws_instance.managed_node[2]: Creation complete after 13s [id=1-0a57f2d696cabfd5e]
36 module.compute.aws_instance.managed_node[3]: Creation complete after 13s [id=1-05e0972b738116dd]
37 module.compute.aws_instance.managed_node[1]: Creation complete after 13s [id=1-05c92701640e9543a]
38 module.compute.aws_instance.managed_node[0]: Creation complete after 13s [id=1-09569d5ba25e72c22]
39
40 Apply complete! Resources: 11 added, 0 changed, 0 destroyed.
41
42 Outputs:
43
44 Managed-Nodes = [
45     "54.211.73.69",
46     "3.86.163.83",
47     "13.218.200.161",
48     "54.89.220.67",
49 ]
50
```

Following is the screenshot for the AWS EC2 instance list showing all deployed servers with Public IPs.

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability zone	Public IP	Public IPv4
ansible_controller	i-0ca26cad293ae301e	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	ec2-3-84-1...	3.84.189.198
host_node_2	i-0a57f2d696cabfd5e	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	ec2-13-218...	13.218.200.161
host_node_0	i-09569d5ba25e72c22	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	ec2-54-211...	54.211.73.69
host_node_3	i-05e09f2bb738116dd	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	ec2-54-89-...	54.89.220.67
host_node_1	i-05c92701640e9543a	Running	t2.micro	2/2 checks passed	View alarms	us-east-1a	ec2-3-86-1...	3.86.163.83

### III. Configuration with Ansible

#### 4. Updating the Hosts File

The `hosts.ini` file was configured as follows:

```
[ec2-user@ip-172-31-89-167 ~]$ vi hosts.ini
[ec2-user@ip-172-31-89-167 ~]$ ls
hosts.ini install_control_node.yaml
[ec2-user@ip-172-31-89-167 ~]$ cat hosts.ini
[einstein]
54.211.73.69
3.86.163.83

[Cloud]
13.218.200.161
54.89.220.67
[ec2-user@ip-172-31-89-167 ~]$
```

Connected to all the managed nodes in the `hosts.ini` file resolving the python error adding the following line in the `ansible.cfg` on the default section.

`interpreter_python = auto_silent`

Following is the screenshot that establishes the connection to all the managed nodes also with group without Python error.

```

[ec2-user@ip-172-31-89-167 ~]$ ansible all -m shell -a "free -m" -i hosts.ini
54.211.73.69 | CHANGED | rc=0 >>
      total        used        free      shared  buff/cache   available
Mem:       970         73        341          0      554        756
Swap:          0          0          0          0          0          0
13.218.200.161 | CHANGED | rc=0 >>
      total        used        free      shared  buff/cache   available
Mem:       970         88        275          0      605        734
Swap:          0          0          0          0          0          0
3.86.163.83 | CHANGED | rc=0 >>
      total        used        free      shared  buff/cache   available
Mem:       970         73        343          0      553        757
Swap:          0          0          0          0          0          0
54.211.73.69 | CHANGED | rc=0 >>
      total        used        free      shared  buff/cache   available
Mem:       970         88        276          0      605        734
Swap:          0          0          0          0          0          0
[ec2-user@ip-172-31-89-167 ~]$ ansible einstein -m shell -a "free -m" -i hosts.ini
3.86.163.83 | CHANGED | rc=0 >>
      total        used        free      shared  buff/cache   available
Mem:       970         73        343          0      553        757
Swap:          0          0          0          0          0          0
54.211.73.69 | CHANGED | rc=0 >>
      total        used        free      shared  buff/cache   available
Mem:       970         73        341          0      554        756
Swap:          0          0          0          0          0          0
[ec2-user@ip-172-31-89-167 ~]$ ansible Cloud -m shell -a "free -m" -i hosts.ini
54.211.73.69 | CHANGED | rc=0 >>
      total        used        free      shared  buff/cache   available
Mem:       970         88        276          0      605        734
Swap:          0          0          0          0          0          0
13.218.200.161 | CHANGED | rc=0 >>
      total        used        free      shared  buff/cache   available
Mem:       970         88        275          0      605        734
Swap:          0          0          0          0          0          0
[ec2-user@ip-172-31-89-167 ~]$
```

## IV. Setting up Web Servers

Next, the code from the **ControlNodeFiles** repository has been successfully cloned and securely copied (SCP) to the home directory of the ec2-user on the control node.

The `install_httpd.yaml` playbook was renamed to `install_einstein.yaml` and updated to target the `einstein` group.

Executed using `ansible-playbook -i /home/ec2-user/hosts.ini install_einstein.yaml`

Execution output of `install_einstein.yaml`

```

[ec2-user@ip-172-31-89-167 CloudOps_Module4_Class6_Repo4_ControlNodeFiles]$ ansible-playbook -i /home/ec2-user/hosts.ini install_einstein.yaml
PLAY [Install httpd] ****
TASK [Gathering Facts] ****
ok: [54.211.73.69]
ok: [3.86.163.83]

TASK [install yum httpd package] ****
ok: [3.86.163.83]
ok: [54.211.73.69]

TASK [copy zip file to remote server] ****
changed: [3.86.163.83]
changed: [54.211.73.69]

TASK [Unarchive a file that is already on the remote machine] ****
changed: [54.211.73.69]
changed: [3.86.163.83]

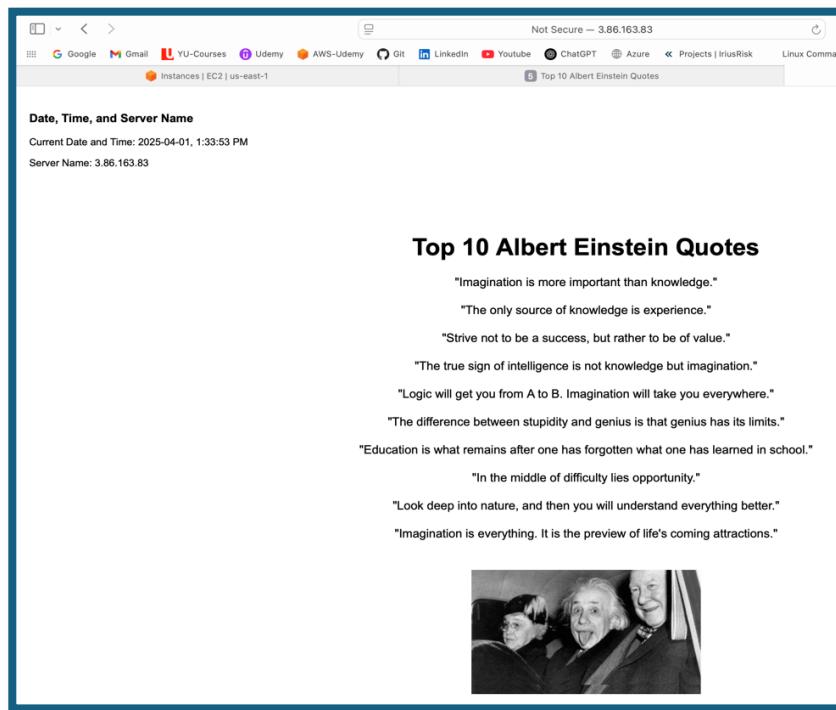
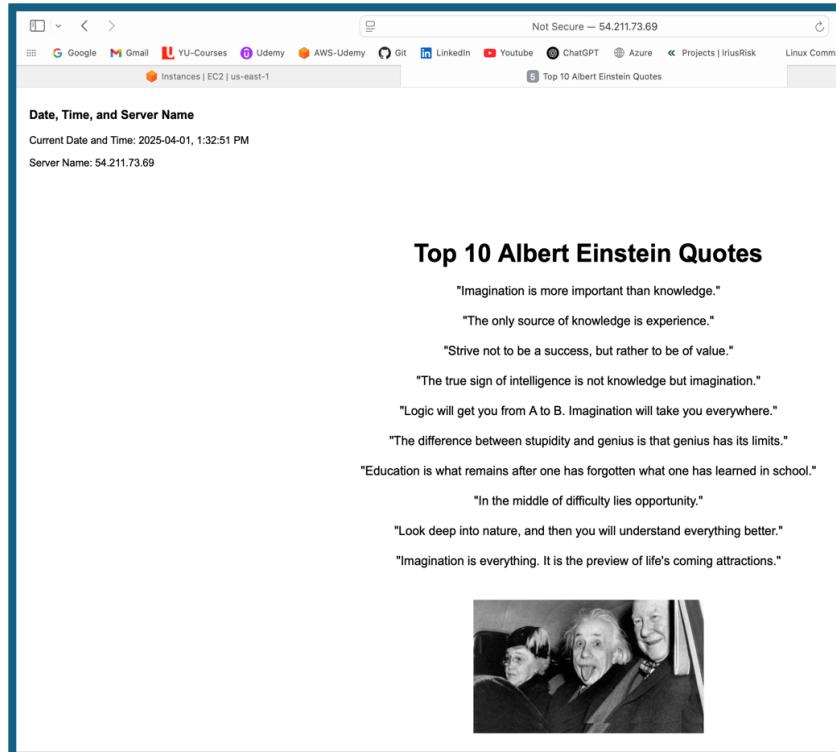
TASK [Copy files] ****
changed: [3.86.163.83] => (item=index.html)
changed: [54.211.73.69] => (item=index.html)
changed: [54.211.73.69] => (item=einstein.jpg)
changed: [3.86.163.83] => (item=einstein.jpg)

TASK [Start and Enable httpd] ****
changed: [3.86.163.83]
changed: [54.211.73.69]

PLAY RECAP ****
3.86.163.83 : ok=6    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
54.211.73.69 : ok=6    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-89-167 CloudOps_Module4_Class6_Repo4_ControlNodeFiles]$
```

Browser verification of Einstein web servers



## 5. Configuring the Cloud Group

**Note:** I have successfully cloned the CSCC1030 repository and used SCP to transfer it to my control node's home directory. I then moved it to the ControlNodeFiles folder within the same directory to execute the .yaml file.

A new playbook **install\_Cloud.yaml** was created to deploy the York University Cloud computing page to the **Cloud group**.

Executed using: **ansible-playbook -i ~/hosts.ini install\_Cloud.yaml**

Execution output of **install\_Cloud.yaml**

```
[ec2-user@ip-172-31-89-167 CloudOps_Module4_Class6_Repo4_ControlNodeFiles]$ vi install_Cloud.yaml
[ec2-user@ip-172-31-89-167 CloudOps_Module4_Class6_Repo4_ControlNodeFiles]$ ansible-playbook -i ~/hosts.ini install_Cloud.yaml

PLAY [Install and Deploy York U Cloud Page] *****

TASK [Gathering Facts] *****
ok: [54.89.228.67]
ok: [13.218.200.161]

TASK [Install yum httpd package] *****
ok: [54.89.228.67]
ok: [13.218.200.161]

TASK [Copy CSCC1030 repo zip to remote server] *****
changed: [54.89.228.67]
changed: [13.218.200.161]

TASK [Unarchive CSCC1030 repo] *****
changed: [13.218.200.161]
changed: [54.89.228.67]

TASK [Copy Cloud Computing website files to /var/www/html/] *****
changed: [54.89.228.67]
changed: [13.218.200.161]

TASK [Start and Enable httpd] *****
changed: [54.89.228.67]
changed: [13.218.200.161]

PLAY RECAP *****
13.218.200.161 : ok=6    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
54.89.228.67   : ok=6    changed=4    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ec2-user@ip-172-31-89-167 CloudOps_Module4_Class6_Repo4_ControlNodeFiles]$
```

Browser verification of Cloud web servers





## V. Final Confirmation

- Terraform successfully provisioned all required infrastructure components.
- Ansible playbooks were executed correctly, configuring the web servers as intended.
- The hosted web pages loaded successfully in browser tests.

## VI. Conclusion

This lab demonstrated the seamless integration of Terraform and Ansible in deploying and managing AWS infrastructure. By automating provisioning and configuration, the workflow ensured efficiency and accuracy in setting up the control node and web servers. The results validate the correctness of the implementation, as evidenced by successful server deployment and web page accessibility.

## VII. References:

[https://learn.continue.yorku.ca/pluginfile.php/1142963/mod\\_lesson/page\\_contents/133502/CSCL1030%20Class%206%20Infrastructure%20as%20Code%20%20Configuration%20Management%20Part%202%20.pdf](https://learn.continue.yorku.ca/pluginfile.php/1142963/mod_lesson/page_contents/133502/CSCL1030%20Class%206%20Infrastructure%20as%20Code%20%20Configuration%20Management%20Part%202%20.pdf)  
<https://learn.continue.yorku.ca/mod/lesson/view.php?id=603346&pageid=133502>

**Thank you.**