# Code Cube

# Assignment 1: Ansible

# Members:

| Name | SJSU ID |
| --- | --- |
| Neel Desai | 017436575 |
| Nick Kornienko | 014284023 |
| Kelly Nguyen | 014161277 |
| Mansi Vekariya | 016996291 |

# 

# Aim

To configure the Ansible server on a virtual machine and use it, launch 2 other VMs running a webserver on port 8080 that returns the message: “Hello World from SJSU-X '', where X is 1 or 2 depending on the web server on receiving any request.

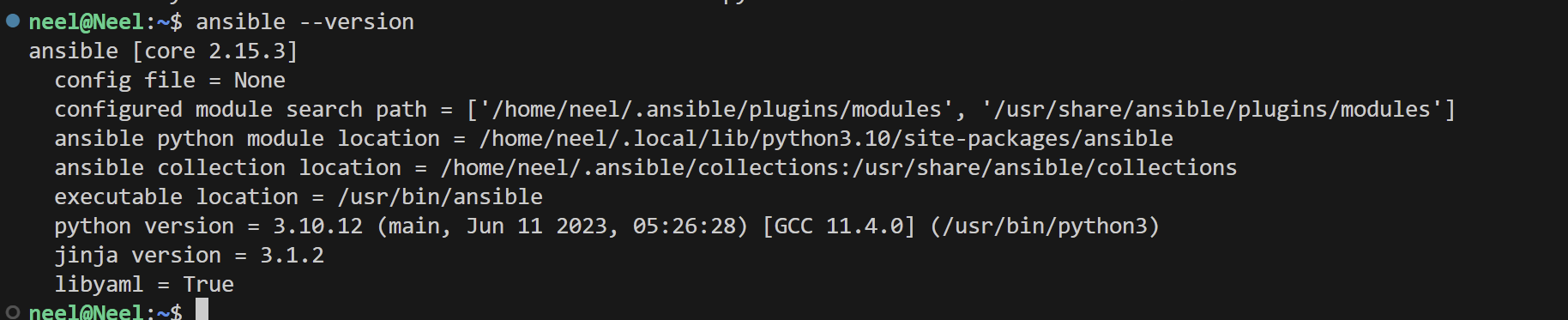
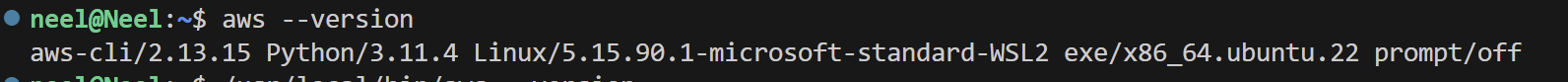
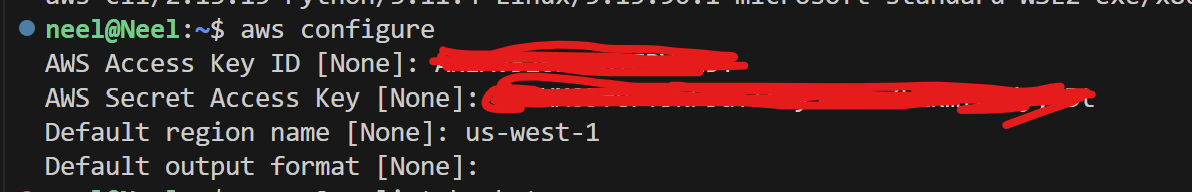
# Approach

To accomplish this we would be doing the following:

1. Use WSL on a Windows device to install the Ansible server (you can use any Linux-based VM).
2. Use 2 EC2 instances as the other 2 VMs which will host our 2 webservers running on port 8080 of each instance. Along with EC2 instances, we’ll also create a security group which can control the type of network traffic that can be allowed in and out of these EC2 instances.
3. Use Flask for making the web server and use tmux for running the server so that the Flask app doesn’t hoard the stdout of the VM and deploy it in development mode.
4. Each of the steps will be divided into its own playbook for maintenance, readability and deployment convenience.

Git Repository: https://github.com/neel26desai/cmpe272\_assignment1

# Prerequisites

1. Ansible and AWS module should be installed
   1. To Install Ansible
      1. sudo apt install ansible
   2. To install AWS modules, needed for deploying resources on AWS
      1. ansible-galaxy collection install community.aws
      2. ansible-galaxy collection install amazon.aws
   3. You can check the successful installation
      1. ansible –version
      2. 
2. Install boto3 python package
   1. pip install boto3
3. Install and configure AWS CLI
   1. ****
4. Generate AWS\_ACCESS\_KEY & AWS\_ACCESS\_SECRET\_KEY from your AWS account, and configure them in your dev environment using aws configure
   1. 
5. Inside AWS>EC2>Network and Security> Generate a new RSA public-private key pair, we will use this pair for remote login and perform operations on our EC2 instances. The public key will be stored inside the instances and you’ll have to download the private key and save it somewhere safe as a PEM file.
6. Change access on the PEM file as if such that only owner of the file can read and write to it
   1. chmod 400 /path/to/your/privatekey.pem

# Steps

1. Create 2 EC2 Instances
2. Add Instances to Inventory
3. Ping Instances
4. Install Dependencies
5. Move Code
6. Run WebServer
7. Remove Everything

### **Create 2 EC2 Instances**

In this step we will be running the “ec2-creation.yml” playbook. This includes plays for creating a security group called “allow-all”, which we will attach to our EC2 instances, to allow network traffic to and fro on port 8080. It also includes 2 plays to create 2 EC2 instances, each having a different instance name (in our case WebServer1 & WebServer2). Within the play, we include a logic which checks if an instance with a given name exists, and only runs the EC2 instance creation code if the instances do not exist.

To run the play:

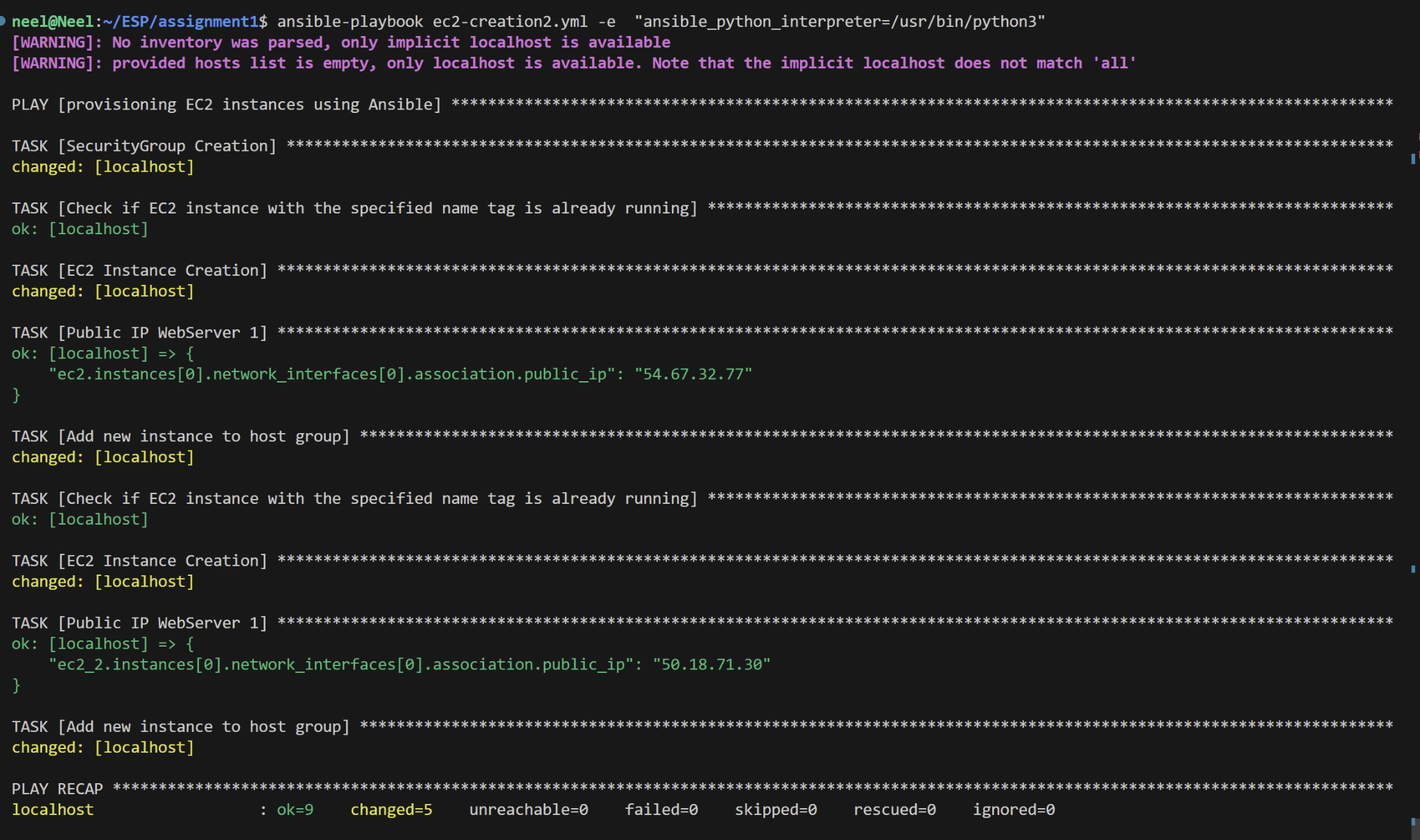
ansible-playbook ec2-creation.yml -e "ansible\_python\_interpreter=/usr/bin/python3"

Additional Info about the command:

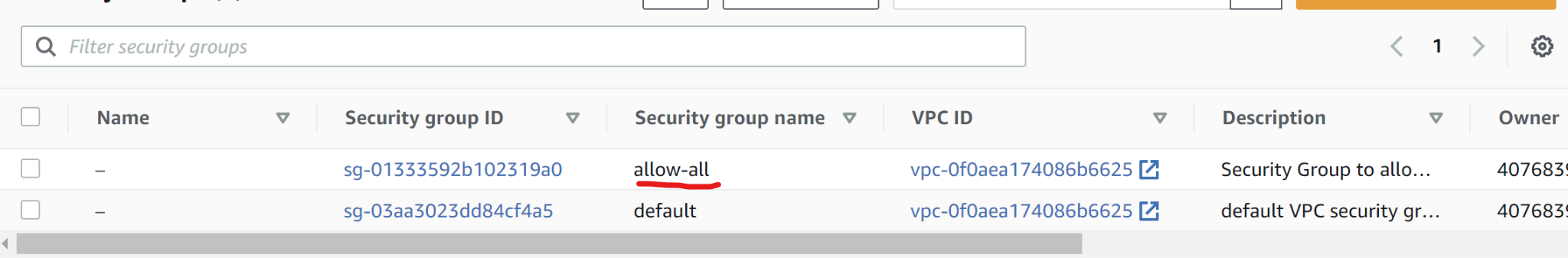
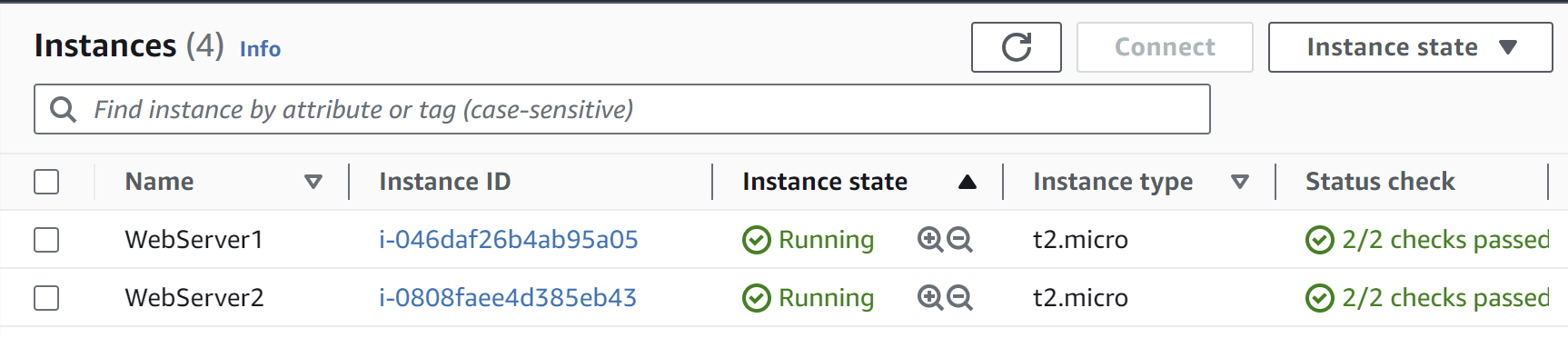
When you have more than one version of Python installed on your system you have to specify which version you want Ansible to use. To accomplish this we set “ansible\_python\_interpreter=/path/to/python3” after the -e flag.

Output:

As we have 5 different things happening in the play we see that 5 things were changed successfully.



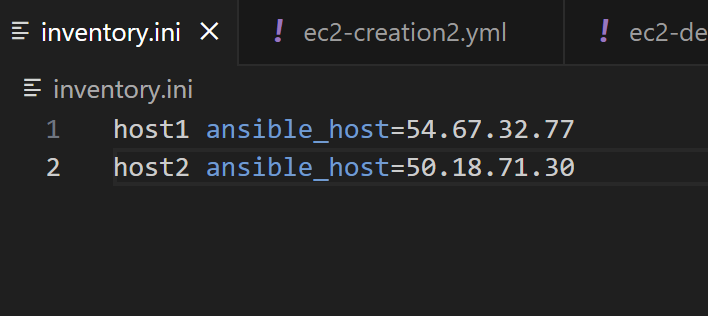
Checking if the resources are created in AWS

1. Check for “allow-all” security group
   1. 
2. Check for 2 EC2 instances with the name WebServer1 and WebServer2
   1. 

### **Add Instances to Inventory**

Once we have our EC2 instances created we need to add them to our inventory file. To accomplish this, note the public ip address of both instances, shown in the output of the previous step (you can also find this from your AWS console). Create a new file in your current directory called ”inventory.ini. Inside the file add each instance as “hostX ansible\_host=public\_ip”

Sample:



Here “host1” is our EC2 instance named WebServer1 and “host2” is our EC2 instance named WebServer2

### **Ping Instances**

Once we have added the hosts to our inventory we would like to check if we can connect with them at all. To do this, we will ping each instance/host one by one.

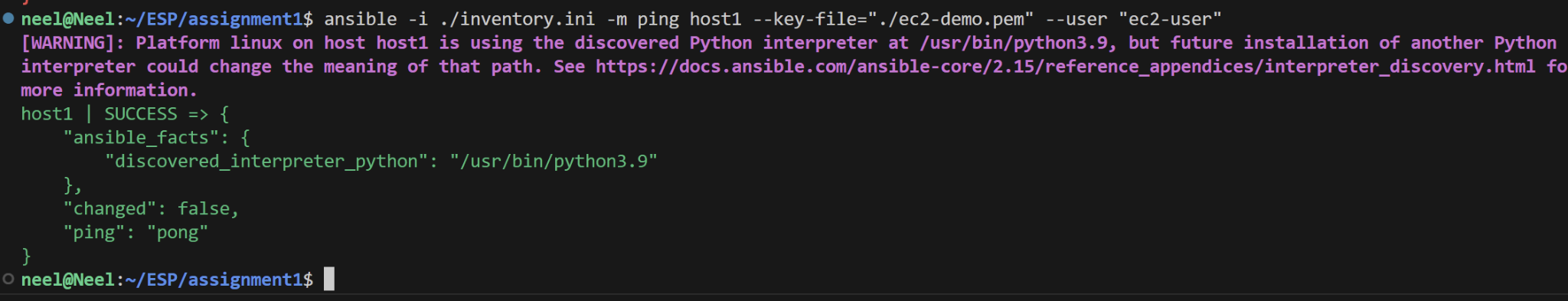
Command:

ansible -i ./inventory.ini -m ping host1 --key-file="./ec2-demo.pem" --user "ec2-user"

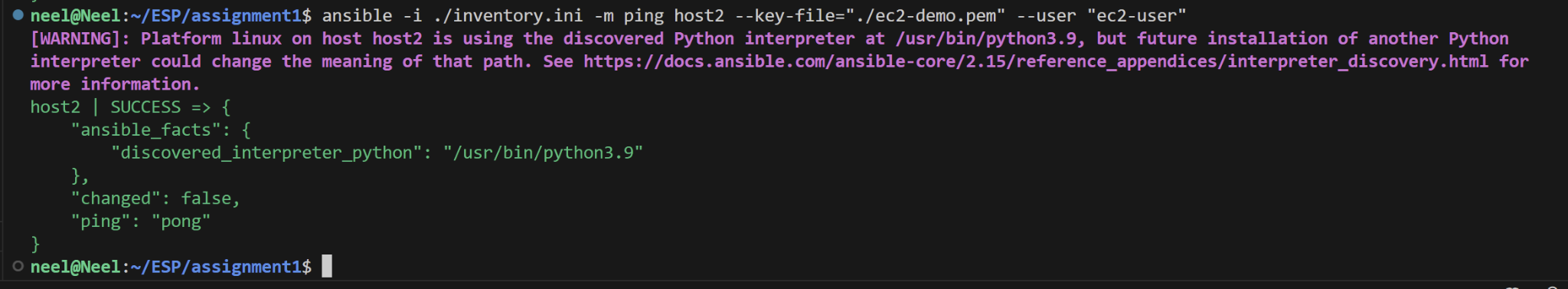
Addition Info about the command:

The path after -i is the path to your inventory.ini file, we need to specify this as we have created an inventory file in our current working directory and are not using the default path. The --key-file includes the path to the pem file previously saved. We are using an Amazon Linux Instance image, which has the default user "ec2-user" hence we specify that as our user.

Output on pinging host1:



Output on pinging host2:

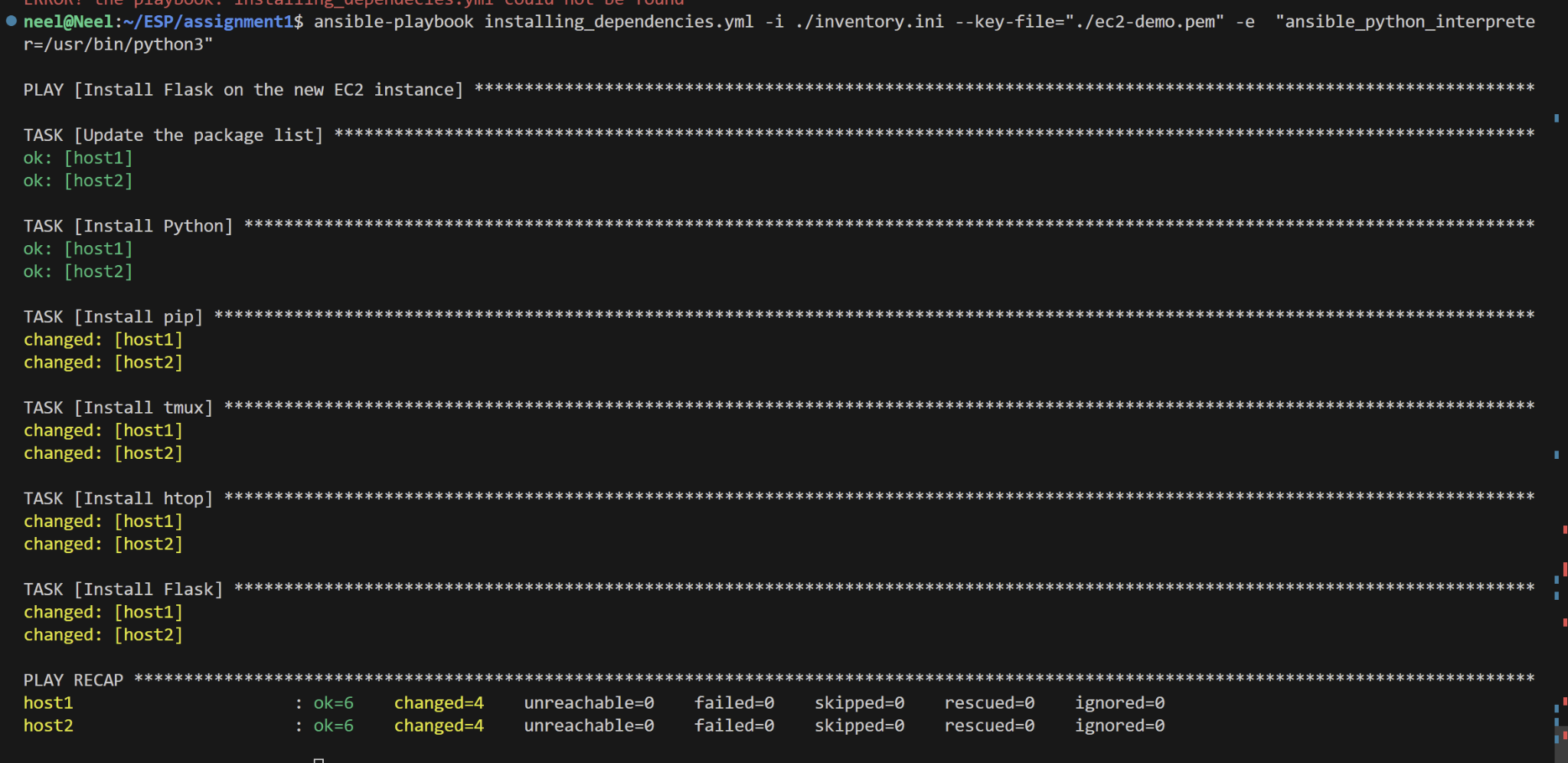


### **Install Dependencies**

Next we will install all our dependencies on the EC2 instance.We will run the playbook “ installing\_dependencies.yml”. It includes plays to install python, tmux, htop, pip3 and Flask on both instances. We do not write separate plays for both hosts, instead we specify that a single play should run on both hosts.

Command:

ansible-playbook installing\_dependencies.yml -i ./inventory.ini --key-file="./ec2-demo.pem" -e "ansible\_python\_interpreter=/usr/bin/python3"

Output: 

### **Move Code**

In this step, we copy the Flask server code from our current working directory to the EC2 instances. We use the playbook “move\_code.yml”.It has 2 plays the first will copy the file app1.py from local to WebServer1(host1) at /home/ex2-user/app.py and the second will copy app2.py from local to WebServer2(host2) at /home/ex2-user/app.py .

Command:

ansible-playbook move\_code.yml -i ./inventory.ini --key-file="./ec2-demo.pem" -e "ansible\_python\_interpreter=/usr/bin/python3"

Output:



You can remote login to the instance to check if the new file exists. For now, we can skip that as our command’s output shows that it pushed 1 change on each host.

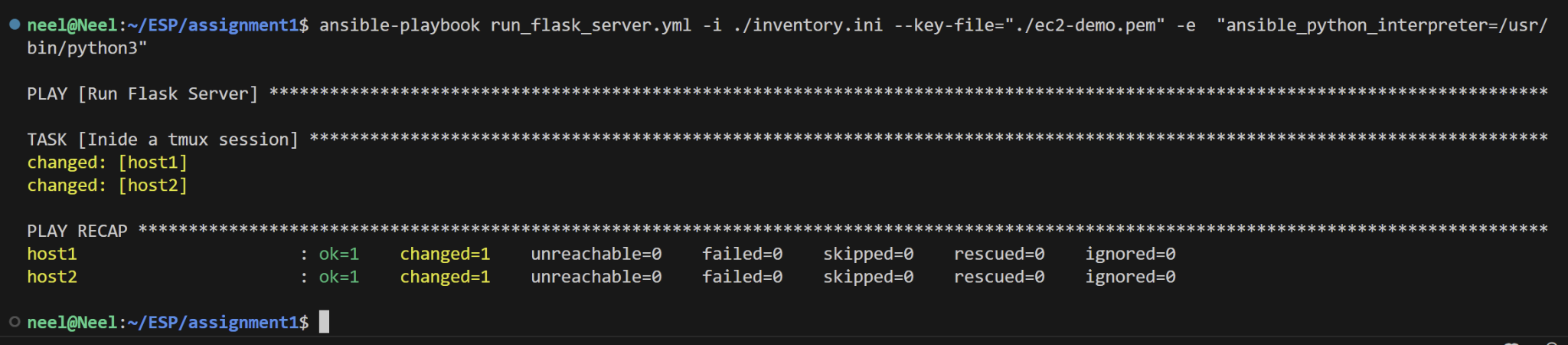
### **Run WebServer**

In this step we will run the flask servers on both the ec2 instances. We use the “run\_flask\_server.yml” playbook, where we ask it to run a single play on both hosts.

Command:

ansible-playbook run\_flask\_server.yml -i ./inventory.ini --key-file="./ec2-demo.pem" -e "ansible\_python\_interpreter=/usr/bin/python3"

Output:

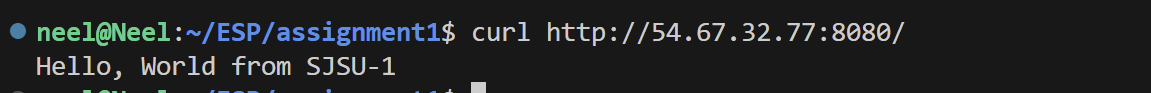


Verifying the implementation:

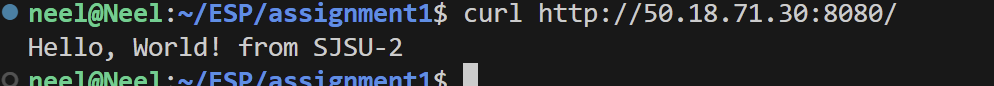
At this stage, we have a webserver running on port 8080 of both the hosts. Now if we send a request to these web servers we should receive “Hello World! From SJSU-X”.

To send the requests will use the “curl” command and to route will be public\_ip:8080.

Sending request to WebServer1(host1):



Sending request to WebServer2(host2):



**Remove Everything**

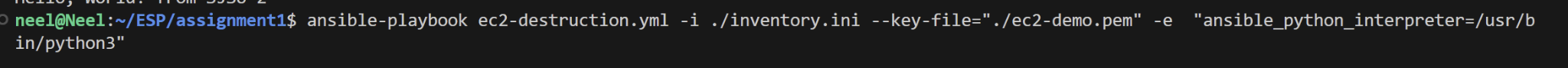
Now that we have achieved our name the EC2 instances and the security group which we created are of no use to us, so we will delete them. We use the playbook

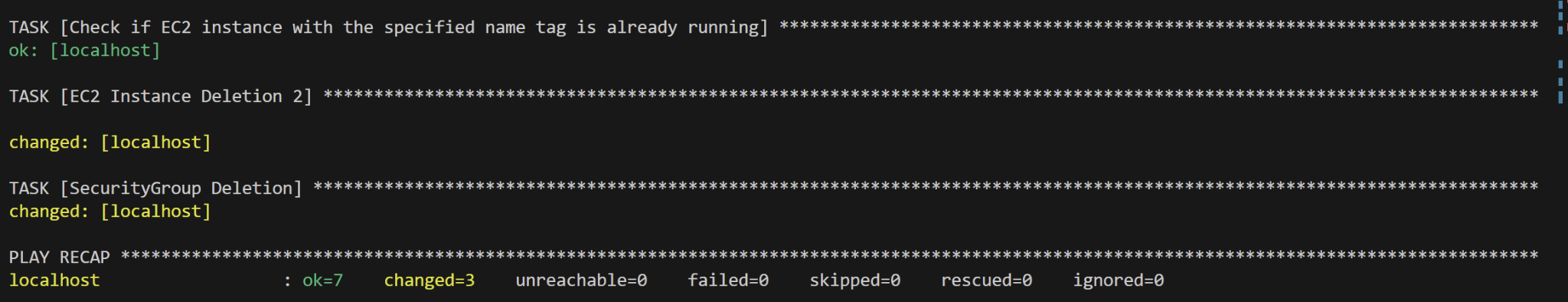
“ec2-destruction.yml”. In this, we first check if a EC2 instance exists (we use the instance name for this identification purpose) if it does we terminate it, by setting the state as terminated. Once both the EC2 instances are terminated we proceed to delete the security group using the name which we gave when creating it.

Command:

ansible-playbook ec2-destruction.yml -i ./inventory.ini --key-file="./ec2-demo.pem" -e "ansible\_python\_interpreter=/usr/bin/python3"

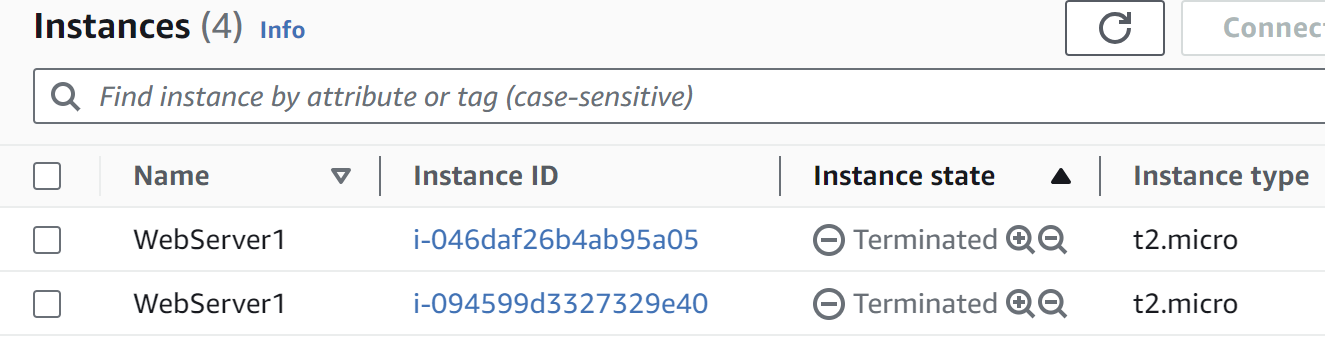
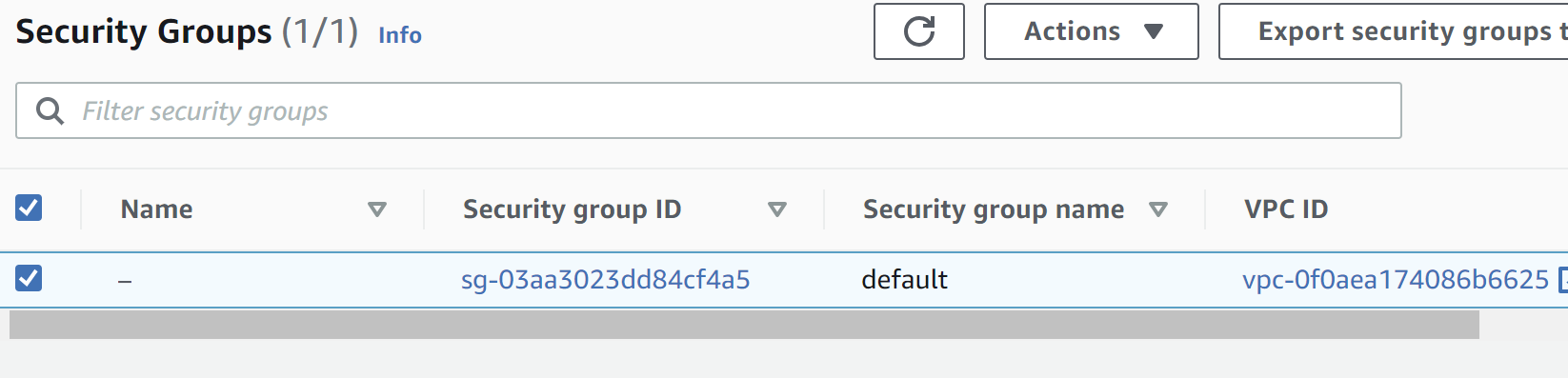
Output:





We see 3 changes were committed that correspond to our 3 tasks, 2 for terminating the EC2 instances and 1 for removing the security group.

Verifying the deletion tasks on AWS:

1. Checking if both the instances are terminated
   1. 
2. Check if the “allow-all” security group is deleted, we can see the security group is no longer visible
   1. 

Additionally remove the hosts from the inventory file, as the public ip are dynamically assigned and once the instance is terminated the public IPfor it will be released so those IPs won't be of use to us the next time we want to deploy it.

# References:

[1] https://www.middlewareinventory.com/blog/ansible-aws-ec2/

[2] https://www.coachdevops.com/2021/07/ansible-playbook-for-provisioning-new.html

[3] https://docs.ansible.com/ansible/latest/collections/amazon/aws/ec2\_security\_group\_module.html

[4] https://docs.ansible.com/ansible/latest/collections/amazon/aws/ec2\_instance\_module.html

[5] https://medium.com/@dharmesh.gangwar/use-ansible-to-create-terminate-ec2-instances-d57743bfb4cc

[6] <https://www.twilio.com/blog/deploy-flask-python-app-aws>

[7] https://chat.openai.com/