Network and Service Operations Project OpenStack with Ansible

SHIVAKUMAR YADAV JABBU
Electrical Engineering with a focus on Telecommunication Systems
BLEKINGE INSTITUTE OF TECHNOLOGY
Karlskrona, Sweden
SHJA20@student.bth.se

***Abstract*—OpenStack is a distributed cloud platform that is incredibly complex to set up. In this project, I will construct an Ansible playbook (automatic deployment script) to deploy my own OpenStack infrastructure, and I will use bash script to run the ansible files.**

***Index Terms*—OpenStack,Ansible**

I. Introduction

The purpose of this project is to create and construct an installation script using ansible-playbook that can install simple OpenStack deployments while also providing flexibility in setting more complicated situations by reusing the script's separate components [1].On the networking front, I did set up a network, router, and subnet. Then created a security key, build the required number of instances, a Bastion host, and Haproxy.To generate the hosts and configuration files, I utilized Python scripts. We will deploy flask apps to webservers (backend servers) and loadbalance them using the Haproxy (frontend server) device, which will be setup with haproxy. The backend servers should also reply to the udp service, and the frontend server should use nginx to loadbalance the backend servers.

II.

A. Design of project

OpenStack is cloud platform which is incredibly complex to set up. I have used shell script which needs openstack rc file. It can be downloaded from the openstack users in cloud. It specifies required information of the cloud. A sample openstack rc file is shown

```
export OS_USERNAME=****
export OS_PASSWORD=***
export OS_AUTH_URL=https://*****:****
export OS_USER_DOMAIN_NAME=**
export OS_PROJECT_DOMAIN_NAME=***
export OS_REGION_NAME=****
export OS_PROJECT_NAME="****"
export OS_TENANT_NAME="****"
export OS_AUTH_VERSION=3
export OS_IDENTITY_API_VERSION=3
```

The project is divided into three stages, each of which is controlled by three commands: install, operate, and cleanup. The commands are extended with rc file, a tag name and ssh key file. It looks like

```
install <openrc> <tag> <ssh_key>
operate <openrc> <tag> <ssh_key>
cleanup <openrc> <tag> <ssh_key>
```

1) Install: During the installation step, I create a temporary folder in the same running folder to hold files essential for the project to run, the directory and files are later erased. I'm making the hosts file without any hosts, so we'll later update the hosts file with instances in the playbook and perform the tasks in the instances. Export openstack rc file in the first step. Then we run Ansible-playbook, which produces a keyname, network, router, and subnet with names that will be extended with the tagname that given in the command. Then from file servers.conf , we check required numbers instances (backend servers) to build. The network id and key name that were produced in the previous step will be used all backend servers. "Ubuntu 20.04 Focal Fossa 20200423" is the image used to create the servers. Only fixed ip is given to backend servers. Example for creation an backend servers

```
- name : Create server nodes
    os_server:
      state: present
      name: dev{{ item }}
      image: Ubuntu 20.04 Focal Fossa 20200423
      flavor: flavorname
      key_name: ansible_key
      auto_floating_ip: no
      security_groups: default
      nics:
        - net-id: "{{ testnet_network.id }}"
    with_sequence: start=1 end={{lookup('file')}}
```

In the same network that we developed, I built Bastion and Haproxy servers with floating IP addresses. The bastion host's floating IP address is used to connect ssh to all servers, including the haproxy server. Haproxy's floating IP address is used to validate http and udp requests.
Now I have created hosts file and config file in the same folder running two python scripts. The hosts are added into webservers and haproxy.
Example of hosts file

```
[haproxy]
net_haproxy

[webservers]
dev3
dev2
dev1

[all:vars]
ansible_user=ubuntu
```

example of config file

```
host dev1
port 22
user ubuntu
IdentityFile /.ssh/id_rsa
hostname 192.168.0.29

host net_haproxy
port 22
user ubuntu
IdentityFile /.ssh/id_rsa
hostname 192.168.0.102

host net_bastion
port 22
user ubuntu
IdentityFile /.ssh/id_rsa
hostname 91.106.195.140
```

Then pinging the newly formed hosts to see if they are accessible or not. As of here the install part is completed.

2) Operate: In the first step of this part, we check to see whether there are any more servers existing other than those built during the installation stage. Ansible is used to eliminate any excess servers that are present. Then I verified the active instances that were deployed during the installation step and created additional instances as required. Then modify the hosts file and config file. Ping all the hosts and check whether all the hosts are reachable. I've installed the uWSGI application server, launched the flak application, and configured Nginx to serve. I have configured haproxy in frontend server(Haproxy) which listens to http requests and send requests to backend servers in roundrobin loadbalancing. I have checked http request in haproxy server using curl on command line and printed the output. The example of output

```
TASK [test-1] ******
ok: [haproxy]

TASK [debug] ************************
ok: [haproxy] => {"msg":
"21:24:17 Serving from dev5 (*.*.*.*)\n"
}

TASK [test-2] ***
ok: [haproxy]

TASK [debug] ********************
ok: [jyoo_haproxy] => {"msg":
"21:24:19 Serving from dev4(*.*.*.*)\n"
}
```

I have also configured snmpd in the backend servers which can respond to udp requests and frontend servers is configured with nginx to listen to udp request and send the request to backend servers. then udp request is done from the command line of server haproxy with command snmpwalk -t 1 -r 1 -v2c -c public localhost:6000. The output looks like

```
TASK [checking snmpwalk -4] ***
changed: [jyoo_haproxy]

TASK [debug] *****************
ok: [jyoo_haproxy] => {
    "output.stdout_lines": [
        "iso.3.6.1.2.1.1.1.0 = STRING:
        \"Linux dev2 5.4.0-26-generic
        #30-Ubuntu SMP Mon Apr 20 16:58:30
        UTC 2020 x86_64\""
    ]
}

TASK [checking snmpwalk -5] *************
changed: [jyoo_haproxy]

TASK [debug] ******************
ok: [jyoo_haproxy] => {
    "output.stdout_lines": [
        "iso.3.6.1.2.1.1.1.0 = STRING:
        \"Linux dev1 5.4.0-26-generic
        #30-Ubuntu SMP Mon Apr 20 16:58:30
        UTC 2020 x86_64\""
    ]
}
```

3) Cleanup: In the last stage I am using ansible to do all the tasks. In this stage I deleted all the floating ip's that are allocated, deleting all the servers that are created in stage one and two. I deleted the subnet,router,network respectively and key-name that are created in the first stage.

4) Motivation of design: OpenStack delivers infrastructure blueprints, API endpoints, and Heat to enable infrastructure-as-a-service delivery. After the infrastructure is in place, Ansible provides an agentless framework for configuring, deploying, and automating complex application software stacks in the cloud. Ansible and OpenStack Heat work together to deliver end-to-end solutions for deploying even the most complex application stacks. The advantages of Ansible extend beyond OpenStack deployment. After you've installed OpenStack, you may utilize the various Ansible OpenStack modules to control how your cloud operates. The cloud's apps and services may then be provisioned, configured, and deployed using the same simple and powerful Ansible capabilities that were used to

deploy and manage OpenStack. Ansible is a one-tool cloud operator, developers, and users.Every layer is managed by a module in Ansible. Even if some modules doesn't we can use cli to execute or debug the task.

Ansible is a system and application deployment automation framework. Ansible manages systems using Secure Shell (SSH) rather than proprietary protocols that require the deployment of remote daemons or agents.Ansible is a straightforward yet powerful orchestration tool that is suited for deploying OpenStack-powered clouds.

In my design I can create any number servers using ansible loops. I can manage more number of servers at a time using playbooks. Using roles I can run the tasks multiple times. I can unite different tasks into roles and re use them. Ansible files are human readable and easy to write. Ansible makes complex openstack services easy with modules.

5) Alternatives: The alternatives of this project for IaaS are python and shell script. shell scripting python can be used to create instances and networks for the instances to use. To check ping tests and for serving flask app we use ansible files.

B. Performance

To investigate the performance I have installed "apache2-utils" in ubuntu. I am running 10000 requests with a concurrency of 1000 requests at a time. The command i used for doing the test is "ab - n 10000 -c 100 http://*.*.*.*:5000/". I have used the floating ip of haproxy to do this test. The mean of the values is almost similar with some fluctuation but the standard deviation got decreased as the number of nodes increases.I am showing the results of 1 node, 4 nodes and 5 nodes respectively in figures labeled with fig1,fig2 and fig3

```
Concurrency Level:      10
Time taken for tests:   53.931 seconds
Complete requests:      10000
Failed requests:        3333
   (Connect: 0, Receive: 0, Length: 3333, Exceptions: 0)
Total transferred:      1766667 bytes
HTML transferred:       426667 bytes
Requests per second:    185.42 [#/sec] (mean)
Time per request:       53.931 [ms] (mean)
Time per request:       5.393 [ms] (mean, across all concurrent requests)
Transfer rate:          31.99 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       23   26   5.1     25     104
Processing:    24   28   4.9     27     106
Waiting:       23   27   4.8     27     106
Total:         47   54   7.2     52     134
```

Fig. 1. number of nodes is 1

```
Server Software:        gunicorn
Server Hostname:        91.106.193.108
Server Port:            5000

Document Path:          /
Document Length:        43 bytes

Concurrency Level:      10
Time taken for tests:   52.146 seconds
Complete requests:      10000
Failed requests:        5000
   (Connect: 0, Receive: 0, Length: 5000, Exceptions: 0)
Total transferred:      1765000 bytes
HTML transferred:       425000 bytes
Requests per second:    191.77 [#/sec] (mean)
Time per request:       52.146 [ms] (mean)
Time per request:       5.215 [ms] (mean, across all concurrent requests)
Transfer rate:          33.05 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       23   25   4.7     25      99
Processing:    24   27   4.7     26     108
Waiting:       24   27   4.6     26     108
Total:         47   52   6.9     51     134
```

Fig. 2. number of nodes is 4

```
Server Software:        gunicorn
Server Hostname:        91.106.195.82
Server Port:            5000

Document Path:          /
Document Length:        43 bytes

Concurrency Level:      10
Time taken for tests:   54.314 seconds
Complete requests:      10000
Failed requests:        4000
   (Connect: 0, Receive: 0, Length: 4000, Exceptions: 0)
Total transferred:      1766000 bytes
HTML transferred:       426000 bytes
Requests per second:    184.11 [#/sec] (mean)
Time per request:       54.314 [ms] (mean)
Time per request:       5.431 [ms] (mean, across all concurrent requests)
Transfer rate:          31.75 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:       23   26   4.6     25     100
Processing:    24   28   5.5     27     107
Waiting:       24   28   5.4     27     107
Total:         48   54   7.3     53     132
```

Fig. 3. number of nodes is 5

C. Solution Operation

Opestack administration with ansible can be used to create large number of networks, instances and can be used to manage large number of resources. While running it from the multiple locations we have to consider networking security, baremetal security, security requirements and Multi-site security [2].

REFERENCES

[1] ADAM ŠAMALÍK. Extension of openstack modules for ansi-ble platform.

[2] Factors affecting OpenStack deployment — operations-guide 2013.2.1.dev1189 documentation.