DevOps and Cloud Based Software

Lab 1-4: Software deployment and runtime adaptation (Ansible and Docker Swarm)

University of Amsterdam

Objectives

You will learn how to use Ansible to deploy and configure software on multiple remote hosts, and how to adapt an application in a Docker Swarm cluster at runtime.
In this assignment, you will create several playbooks and use Ansible to set up and configure a Docker swarm cluster on top of a set of VMs. You will also use the deployed Docker swarm cluster to practice the service scaling.

Background

Ansible

Ansible is an open-source, command-line automation software that is typically used to configure systems, deploy software, and orchestrate advanced workflows to support application deployment and system updates.
Ansible uses the following terms:

- **Control node**: the machine where Ansible is installed. It manages the execution of the Playbook. It can be any machine on the internet
- **Managed node**: A remote system, or host, that Ansible controls
- **Inventory**: provides a complete list of all the target machines on which various modules are run by making an ssh connection and install the necessary software's
- **Playbook**: consists of steps that the control machine will perform on the managed nodes defined in the inventory file
- **Task**: a block that defines a single procedure to be executed, e.g., install a package
- **Module**: the main building blocks of Ansible and are reusable scripts that are used by Ansible playbooks. Ansible comes with many reusable modules. These include functionality for controlling services, software package installation, working with files and directories, etc.
- **Role**: a way for organizing playbooks and other files to facilitate sharing and reusing portions of a provisioning
- **Facts**: global variables containing information about the system, like network interfaces or operating system
- **Handlers**: used to trigger service status changes, like restarting or stopping a service.

You can find a short technical explanation here https://www.youtube.com/watch?v=fHO1X93e4WA

Docker Swarm

Docker is a tool used to automate the deployment of an application as a lightweight container so that the application can run in different environments.
Swarm Mode is Docker's built-in orchestration system for scaling containers across a cluster.

Prepare your Development Environment

Install Ansible

Start a control node t2.micro Ubuntu Linux. Log in the newly created VM run:

IMPORTANT

The default username for Ubuntu instances is 'ubuntu'. So the ssh command should look like this

```
ssh ubuntu@ec2-XX-XXX-XXX-XX.compute-1.amazonaws.com -i PRIVATE_KEY.pem
```

```
sudo apt update
sudo apt install ansible
```

Check the installation:

```
ansible --version
```

Copy the ssh private key from the sandbox to the newly created VM and save it at /home/ubuntu/lab_user.pem
Make sure that the permissions are set to "Read Only". To do that type

```
chmod 600 /home/ubuntu/lab_user.pem
```

Introduction to Ansible

Make sure Ansible is working in your control node by executing the following command:

```
ansible all --inventory "localhost," --module-name debug --args "msg='Hello'"
```

Here is a break-down of Ansible the command:

- **all**: this means do run the module on all machines that are listed in the "inventory" file, which is the next part of the command
- **--inventory "localhost,"**: The inventory is where all details of the managed nodes are listed such as IP addresses, usernames, etc. In this case, we only use our local computer. This may also be a file
- **--module-name debug**: Specify which module to use. In this case the "debug" module, prints statements during execution and can be useful for debugging variables
- **--args "msg='Hello'"**: Part of the debug module. In this case 'Hello' is the customized massage that is printed. If omitted, prints a generic message.

Controlling Hosts

Start 2 t2.micro Ubuntu Linux VMs and **allow all inbound traffic** in the security groups.

NOTE

Use the same key you created for the control node.

Create an inventory file named 'aws_hosts1' that looks like this:
aws_hosts1

Change the hostnames with the names of your VMs.

NOTE

The **ansible_ssh_private_key_file** has to correspond to the private key that you used to ssh to the provisioned VMs.

You will notice that this file has two headings in brackets **[aws]** and **[aws:vars]**.
The first heading in brackets is a group name. You can have more than one group name, which is used to classify systems and decide what systems you are controlling at what times and for what purpose. So, in this case, we only have a specified [aws] group.

To assign variables to hosts, you can use the [aws:vars] group variables. In this case, we set the VMs username and the location of the key. For more information on inventories, see here:
https://docs.ansible.com/ansible/latest/user_guide/intro_inventory.html

Next, run:

```
ansible aws --inventory aws_hosts1 -m setup
```

The setup module will gather information about the target machines.

The output should look like this:

```
    ec2-18-204-5-138.compute-1.amazonaws.com | SUCCESS => {
        "ansible_facts": {
            "ansible_all_ipv4_addresses": [
                "172.31.57.85"
            ],
            "ansible_all_ipv6_addresses": [
                "fe80::4f9:30ff:fef2:d5a9"
            ],
            "ansible_apparmor": {
                "status": "enabled"
            },
            "ansible_architecture": "x86_64",
            "ansible_bios_date": "08/24/2006",
            "ansible_bios_vendor": "Xen",
            "ansible_bios_version": "4.11.amazon",
            "ansible_board_asset_tag": "NA",
            "ansible_board_name": "NA",
            "ansible_board_serial": "NA",
            "ansible_board_vendor": "NA",
            "ansible_board_version": "NA",
            "ansible_chassis_asset_tag": "NA",
            "ansible_chassis_serial": "NA",
            "ansible_chassis_vendor": "Xen",
            "ansible_chassis_version": "NA",
            "ansible_cmdline": {
                "BOOT_IMAGE": "/boot/vmlinuz-5.15.0-1028-aws",
                "console": "ttyS0",
                "nvme_core.io_timeout": "4294967295",
                "panic": "-1",
                "ro": true,
                "root": "PARTUUID=d4fad590-6773-46cd-91f5-4ee9c1ba24ba"
            },
    }
 ...
```

The setup module gathers facts about the managed nodes and prints it the terminal.

Using Playbooks

Ansible Playbooks are like a to-do list for Ansible that contains a list of tasks. They are written in YAML format and run sequentially.

Playbook Structure

Each playbook is an aggregation of one or more plays, and there can be more than one play inside the playbook. A play maps a set of instructions defined against a particular managed node.

Create a Playbook

Create a playbook that will install in both VMs Python:
playbook_example1.yml

Execute the playbook:

```
ansible-playbook -i aws_hosts1 playbook_example1.yml
```

The output should look like this:

```
PLAY [all] ********************************************************************************************************************************

TASK [Gathering Facts] ********************************************************************************************************************
ok: [ec2-54-90-167-82.compute-1.amazonaws.com]
ok: [ec2-54-91-92-164.compute-1.amazonaws.com]

TASK [ansible.builtin.package] ************************************************************************************************************
ok: [ec2-54-90-167-82.compute-1.amazonaws.com]
ok: [ec2-54-91-92-164.compute-1.amazonaws.com]

PLAY RECAP ********************************************************************************************************************************
ec2-54-90-167-82.compute-1.amazonaws.com : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
ec2-54-91-92-164.compute-1.amazonaws.com : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

In this output you can see:

- PLAY [all]: The group of host the play will run.
- TASK [Gathering Facts]: The Gather Facts task runs implicitly. By default, Ansible gathers information about your inventory that it can use in the playbook.
- TASK [ansible.builtin.package]: The name the of module to run the task.
- ok: [ec2-54-90-167-82.compute-1.amazonaws.com], ok: [ec2-54-91-92-164.compute-1.amazonaws.com]The status of each task. Each task has a status of ok which means it ran successfully.
- PLAY RECAP : The play recap that summarizes results of all tasks in the playbook per managed node. In this example, there are two tasks so ok=2 indicates that each task ran successfully.

Execute plays on different hosts

If we want to execute different plays on different hosts, if for example, we need to install Apache server on one host and Nginx server on another we need to specify that in the playbook by setting the - hosts: web-server1

Create the following inventory in the control node :
aws_hosts2

Change the hostnames in the file with the names of your VMs.

In the control node create the following playbook:
playbook_example2.yml

If we change the inventory and execute:

```
ansible-playbook -i aws_hosts2 playbook_example2.yml
```

If we open a browser to **[web-server1]** and **[web-server2]** that we set in the inventory files we should see Apache and Nginx running.

If you cannot connect, verify that you are using HTTP, and check the VMs' inbound rules in its Security Groups.

Make sure to stop the servers:
Get the playbook to stop the servers:
playbook_example2-1.yml

and run:

```
ansible-playbook -i aws_hosts2 playbook_example2-1.yml
```

Pass Variables Between Plays

Sometimes it is necessary to pass variables between plays. Consider the following playbook:
playbook_example3.yml

if we execute:

```
ansible-playbook -i aws_hosts2 playbook_example3.yml
```

The play with the 'web-server2' hosts will fail.

```
[WARNING]: Invalid characters were found in group names but not replaced, use -vvvv to see details

PLAY [web-server1] ************************************************************************************************************************

TASK [Gathering Facts] ********************************************************************************************************************
ok: [ec2-54-90-167-82.compute-1.amazonaws.com]

TASK [generate secret] ********************************************************************************************************************
changed: [ec2-54-90-167-82.compute-1.amazonaws.com]

TASK [debug] ******************************************************************************************************************************
ok: [ec2-54-90-167-82.compute-1.amazonaws.com] => {
    "msg": "Secret password is OGE1YTdmYWY5NmIwZTRkM2ZiZDY0N2I3"
}

PLAY [web-server2] ************************************************************************************************************************

TASK [Gathering Facts] ********************************************************************************************************************
ok: [ec2-54-91-92-164.compute-1.amazonaws.com]

TASK [print paswd] ************************************************************************************************************************
fatal: [ec2-54-91-92-164.compute-1.amazonaws.com]: FAILED! => {"msg": "The task includes an option with an undefined variable. The error was: 'c

PLAY RECAP ********************************************************************************************************************************
ec2-54-90-167-82.compute-1.amazonaws.com : ok=3    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
ec2-54-91-92-164.compute-1.amazonaws.com : ok=1    changed=0    unreachable=0    failed=1    skipped=0    rescued=0    ignored=0
```

As stated by the line:

```
fatal: [ec2-54-91-92-164.compute-1.amazonaws.com]: FAILED! => {"msg": "The task includes an option with an undefined variable. The error was: 'c
```

The variable stored in the play of web-server1 is not visible by web-server2. Instead, we need to use 'hostvars':
playbook_example4.yml

If we execute this playbook:

```
ansible-playbook -i aws_hosts2 playbook_example4.yml

[WARNING]: Invalid characters were found in group names but not replaced, use -vvvv to see details

PLAY [web-server1] ************************************************************************************************************************

TASK [Gathering Facts] ********************************************************************************************************************
ok: [ec2-54-90-167-82.compute-1.amazonaws.com]

TASK [generate secret] ********************************************************************************************************************
changed: [ec2-54-90-167-82.compute-1.amazonaws.com]

TASK [debug] ******************************************************************************************************************************
ok: [ec2-54-90-167-82.compute-1.amazonaws.com] => {
    "msg": "Secret password is MzFiMGQxMTc1ZDIwNzMyZTZjZmU0MWVj"
}
```

```
TASK [Add command_output to dummy host] ********************************************************************************************************
changed: [ec2-54-90-167-82.compute-1.amazonaws.com]

TASK [debug] ***********************************************************************************************************************************
ok: [ec2-54-90-167-82.compute-1.amazonaws.com] => {
    "msg": "Secret password is MzFiMGQxMTc1ZDIwNzMyZTZjZmU0MWVj"
}

PLAY [web-server2] *****************************************************************************************************************************

TASK [Gathering Facts] *************************************************************************************************************************
ok: [ec2-54-91-92-164.compute-1.amazonaws.com]

TASK [print paswd] *****************************************************************************************************************************
ok: [ec2-54-91-92-164.compute-1.amazonaws.com] => {
    "msg": "paswd is MzFiMGQxMTc1ZDIwNzMyZTZjZmU0MWVj"
}

PLAY RECAP *************************************************************************************************************************************
ec2-54-90-167-82.compute-1.amazonaws.com : ok=5    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
ec2-54-91-92-164.compute-1.amazonaws.com : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

we'll see that the variable is now available to the 'web-server2' hosts as well. This is archived with the use of the module 'add_host' which adds a host during the play execution. More infomrtaion about the module can be found here: https://docs.ansible.com/ansible/latest/collections/ansible/builtin/add_host_module.html

More information about variables and 'hostvars' can be found here: https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#accessing-information-about-other-hosts-with-magic-variables.

Exercises

Create a playbook that will install and configure a Docker swarm cluster

Use the following inventory: ansible_cluster_hosts
And the following playbook: configure-cluster.yml

Fill in the tasks in the playbook provided:

- Go to task named 'docker swarm init' and save the output of the 'join_cmd' so it is accessible by the worker
- Go to task named 'join worker' and on the worker play execute the command to join the cluster
- Advanced (optional) create a new playbook to:
  - create a new worker VM
  - add new VM workers on the existing running cluster

When you have your playbooks, ready execute the Docker swarm setup playbook. Note, if you have problems with initializing the cluster on the master node or joining the cluster on the workers you may need to open all traffic between the VMs of the cluster.

Docker swarm scale benchmark

If your playbook is installed successfully you should be able to see the docker swarm visualizer at http://MATER-IP:5000/

Now we can benchmark Nginx. To do that, run the benchmark-cluster.yml :

```
ansible-playbook -i ansible_cluster_hosts benchmark-cluster.yml
```

Fill in the tasks in the playbook provided:

- Look in the end of the file and add your plays/tasks to repeat the process for 2, 4, and 8 instances
- Record the results for the Avg 'Req/Sec'
- Create a histogram graph where in the x-axis you will have the number of instances i.e. 1,2,4,8 and in the y-axis the 'Req/Sec' for each run.
- Comment on the results do you get increased performance as you add more instances? If not explain why and how would you achieve more requests per second.

Questions

Ansible Play Failure

If we execute:

```
ansible-playbook -i aws_hosts1 playbook_example2.yml
```

We get this output:

```
[WARNING]: Could not match supplied host pattern, ignoring: web-server1

PLAY [web-server1] *****************************************************************************************************************************
skipping: no hosts matched
[WARNING]: Could not match supplied host pattern, ignoring: web-server2

PLAY [web-server2] *****************************************************************************************************************************
skipping: no hosts matched

PLAY RECAP *************************************************************************************************************************************
```

Explain in a few lines why this play failed.

Ansible Play Development

In playbook_example2 if we want to run only the 'start nginx' play how would achieve that?
Provide the ansible-playbook command to do that.

Ansible in DevOps

Discuss how Ansible can be used during the DevOps lifecycle, e.g., which stages? What are the advantages and alternatives of Ansible?