Week 7

Session 1

MicroServices

Olga Chernukhina

Please briefly explain Ansible templates and Ansible Galaxy

Templates:

A template is a file containing all your **configuration parameters**, but dynamic values are set as variables in Ansible. During playbook execution, this depends on conditions such as the cluster being used, and the variables will be replaced with the appropriate values.

You can do more than just replace variables using the **Jinja2 template engine**. You can have loops, conditional operators, write macros, filters to transform data, perform arithmetic calculations, and so on.

Template files usually have an extension .j2, which indicates the Jinja2 template engine used.

Double curly brackets will indicate **variables** in the template file " {{variables}}".

When using the Ansible template module, we need to have two parameters, such as:

src: the source of the template file. This can be a relative or absolute path.
dest: dest is the destination path on the remote server.

Galaxy:

The **Ansible galaxy** is essentially a large **public repository** of Ansible **roles**. Roles ship with a README detailing the role and use variables. Ansible Galaxy contains a large number of roles that are constantly evolving and increasing.

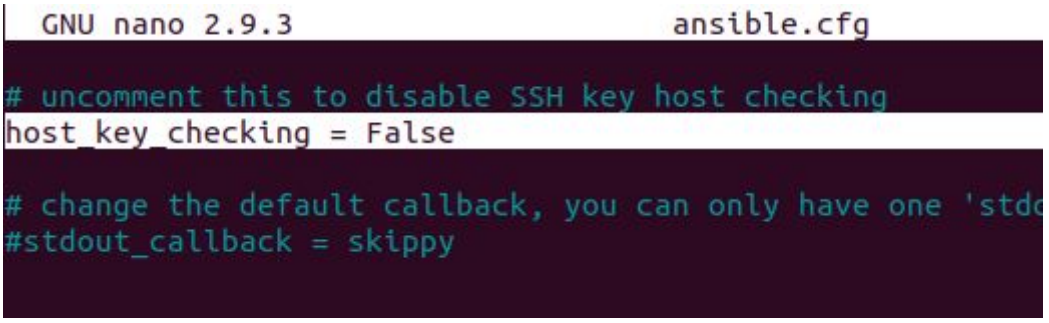
Galaxy can use **Git** to add other role sources, such as GitHub. You can initialize a new galaxy role using Ansible-galaxy init, or install the role directly from the Ansible galaxy role store by running the ansible-galaxy install <name of role>command.

To create an Ansible role using Ansible Galaxy, you need to use the ansible-galaxy command and its templates. Roles must be uploaded before they are used in the playbook. They are placed in the **default directory** called /etc/ansible/roles.

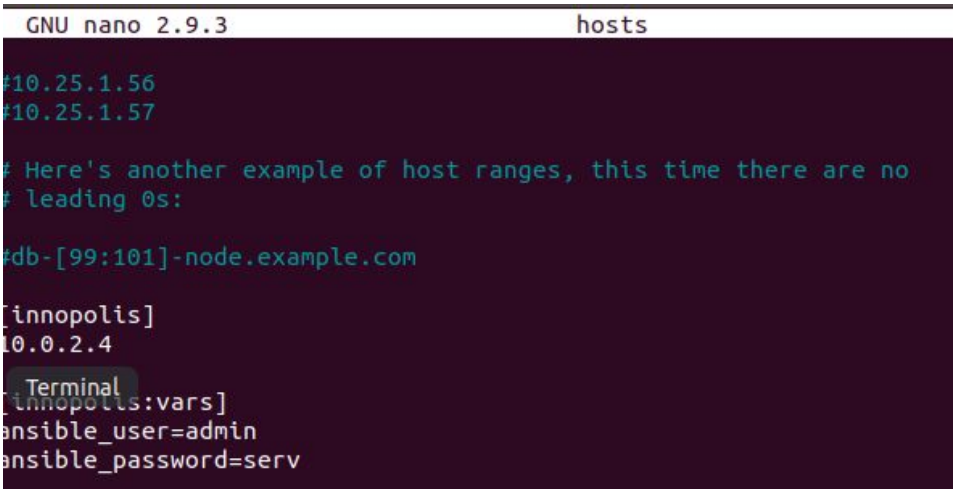
Using Ansible playbook, Install docker on the client and run the "r3dw0lf/secondapp" image on the client (SecondApp is a web server running on port 85). Open the new website.

On ansible master:
sudo apt install ansible sshpass

then configure Ansible not to ask for ssh public keys



Create a group and users in this group:



On the server:
add a new user following the instructions given on the lab

Check the connection between a server and an ansible-master works:

```
cli2@cli2-VirtualBox:/etc/ansible$ sudo ansible innopolis -m ping
10.0.2.4 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Create a playbook according to the task:

```
  GNU nano 2.9.3                        lab.yml

- name: docker
  hosts: innopolis
  become: yes
  become_user: root
  tasks:
    - name: install Docker
      command: 'bash -c "curl -sLS get.docker.com | sh" creates=/usr/bin/docke$
    - name: run server image
      command: docker run -d -p 85:85 r3dw0lf/secondapp
```

Because of errors I modified my playbook, so on the last run previous stages were already completed (and not counted in “changed”), and only the last task changed the state of the machine:

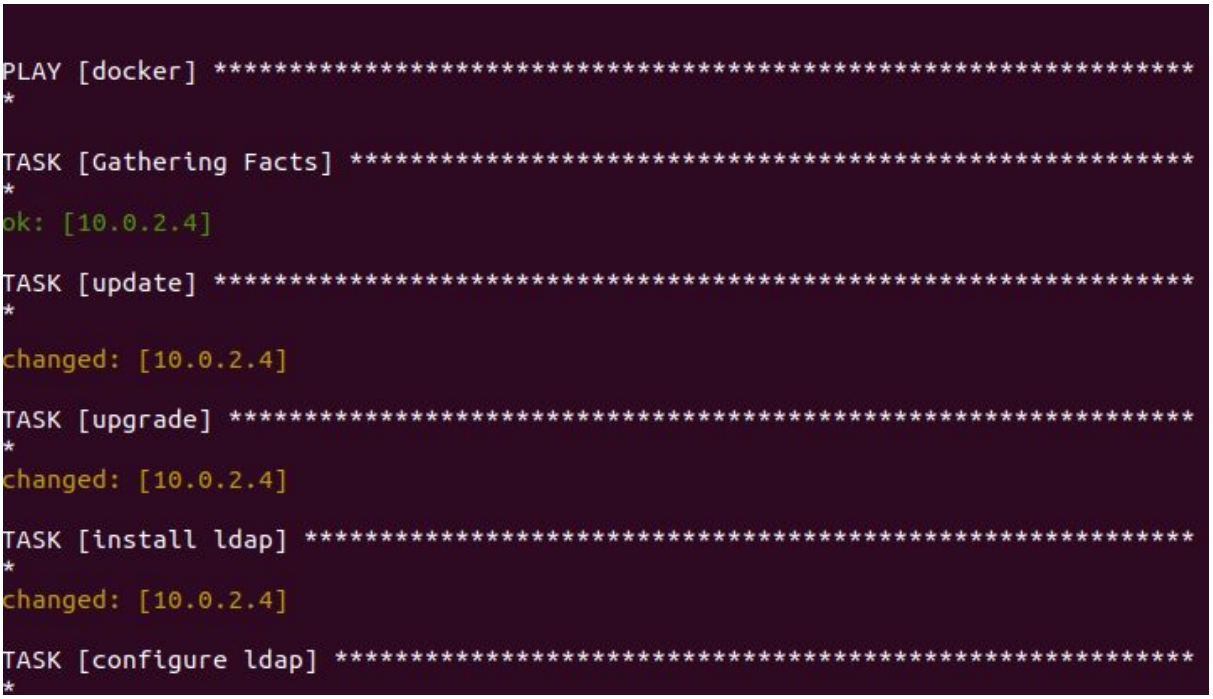
```
cli2@cli2-VirtualBox:~$ sudo ansible-playbook /etc/ansible/lab.yml

PLAY [docker] ******************************************************************
*

TASK [Gathering Facts] *********************************************************
*
ok: [10.0.2.4]

TASK [install Docker] **********************************************************
*
ok: [10.0.2.4]

TASK [run server image] ********************************************************
*
changed: [10.0.2.4]

PLAY RECAP *********************************************************************
*
10.0.2.4                   : ok=3    changed=1    unreachable=0    failed=0
```

Check the website is working:

```
serv@server:~$ curl localhost:85
<!doctype html>
<title>Hello from Flask</title>
<body style="background: #130f40;"></body>
<div style="color: #e4e4e4;
    text-align:  center;
    height: 90px;
    vertical-align:  middle;">

  <h1>Hello from 373efb52a3e0!</h1>




</div>serv@server:~$ _
```

Create an Ansible playbook which can update the client machine (apt update, apt upgrade), prepares the client's machine to join OpenLDAP (from graphical interface).

```
- name: docker
  hosts: innopolis
  become: yes
  become_user: root
  tasks:
    - name: update
      apt:
        update_cache: yes
    - name: upgrade
      apt:
        name: "*"
        state: latest
    - name: install ldap
      apt: name=slapd state=present
      apt: name=ldap-utils state=present
    - name: configure ldap
      lineinfile: dest=/etc/ldap/ldap.conf regexp="*BASE*" line="BASE  dc=inno$
      lineinfile: dest=/etc/ldap/ldap.conf regexp="*URI*" line="URI ldap://inn$
```



Bonus: Explain Terraform with small example.

Terraform is a tool from Hashicorp that helps you declaratively manage your infrastructure. In this case, you don't have to manually create instances, networks, and so on in your cloud provider's console; just write a configuration that explains how you see your future infrastructure. This configuration is created in a human-readable text format. If you want to change your infrastructure, edit the configuration and run terraform apply. Terraform will direct API calls to your cloud provider to bring the infrastructure in line with the configuration specified in this file. If you move infrastructure management to text files, you can arm yourself with all your favorite tools for managing source code and processes, and then reorient them to work with the infrastructure. Now the infrastructure is subject to version control systems, just like the source code, and it can be reviewed in the same way or rolled back to an earlier state if something goes wrong.

```
resource "aws_instance" "example" {
  ami = "ami-lab15"
  instance_type = "t2.small"

  ebs_block_device {
    device_name = "/dev/lab15"
    volume_type = "io1"
  }
}
```