TP part 03 - Ansible

⚠️`Checkpoint: call us to check your results`
`(don't stay blocked on a checkpoint if we are busy, we can check ⅔ checkpoints at the same time)`

❓`Point to document/report`

ℹ️`Interesting information`

Goals

Install and deploy your application automatically with ansible.

Intro

Inventories

By default, Ansible's inventory is saved in the location /etc/ansible/hosts where you already defined your server. The headings between brackets (eg: [webservers]) are used to group sets of hosts together, they are called, surprisingly, groups. You could regroup them by roles like database servers, front-ends, reverse proxies, build servers…

Let's create a project specific inventory, in your project create an ansible directory, then create a new directory called inventories and in this folder a new file (my-project/ansible/inventories/setup.yml):

```
all:
 vars:
   ansible_user: centos
   ansible_ssh_private_key_file: /path/to/private/key
 children:
   prod:
     hosts: hostname or IP
```

Test your inventory with the ping command:

```
$ ansible all -i inventories/setup.yml -m ping
```

Facts

Let's get information about hosts: these kinds of variables, not set by the user but discovered are called **facts**. Facts are prefixed by *ansible_* and represent information derived from speaking with your remote systems.

You will request your server to get your OS distribution, thanks to the setup module.

```
$ ansible all -i inventories/setup.yml -m setup -a
"filter=ansible_distribution*"
```

Earlier you installed Apache httpd server on your machine, let's remove it:

```
$ ansible all -i inventories/setup.yml -m yum -a "name=httpd state=absent"
--become
```

With ansible, you just describe the state of your server and let ansible automatically update it for you. If you run this command another time you won't have the same output as httpd would have been removed.

? `3-1 Document your inventory and base commands`

Playbooks

First playbook

Let's create a first very simple playbook in my-project/ansible/playbook.yml:

```
- hosts: all
  gather_facts: false
  become: yes

  tasks:
   - name: Test connection
     ping:
```

Just execute your playbook:

```
$ ansible-playbook -i inventories/setup.yml playbook.yml
```

You can check your playbooks before playing them using the option: --syntax-check

Advanced playbook

Let’s create a playbook to install docker on your server, follow the documentation and create the corresponding tasks: https://docs.docker.com/install/linux/docker-ce/centos/.

```
- hosts: all
  gather_facts: false
  become: yes

# Install Docker
  tasks:
  - name: Clean packages
    command:
      cmd: dnf clean -y packages

   - name: Install device-mapper-persistent-data
    dnf:
      name: device-mapper-persistent-data
      state: latest

   - name: Install lvm2
    dnf:
      name: lvm2
      state: latest

   - name: add repo docker
    command:
      cmd: sudo dnf config-manager --add-repo=https://download.docker.com/linux/centos/docker-ce.repo

   -  name: Install Docker
    dnf:
      name: docker-ce
      state: present

   - name: install python3
    dnf:
      name: python3

   - name: Pip install
    pip:
      name: docker

   - name: Make sure Docker is running
    service: name=docker state=started
    tags: docker
```

Good news, we now have docker installed on our server. One task was created to be sure docker was running, you could check this with an ad-hoc command or by connecting to the server until you really trust ansible.

Using role

Our docker install playbook is nice and all but it will be cleaner to have in a specific place, in a role for example. Create a docker role and move the installation task there:

```
$ ansible-galaxy init roles/docker
```

Call the docker role from your playbook to check your refactor and your installation.

Initialized role has a couple of directories, keep only the one you will need:

- tasks - contains the main list of tasks to be executed by the role.
- handlers - contains handlers, which may be used by this role or outside.

`3-2 Document your playbook`

Deploy your app

Time has come to deploy your application to your Ansible managed server.

1 - Create specific roles for each part of your application and use the Ansible module: docker_container to start your dockerized application. Here is what a docker_container task should look like :

```
- name: Run HTTPD
 docker_container:
   name: httpd
   image: jdoe/my-httpd:1.0
```

You must have at least this roles :

- install docker
- create network
- launch database
- launch app
- launch proxy

Help:

- you will need to add env variables on app and database tasks. Ansible is able to modify the variables either in the .env for the db or in the application.yml for the app.
- don't forget to use existing module for example to create the network

Useful links:

- docker_container module documentation : https://docs.ansible.com/ansible/2.6/modules/docker_container_module.html#docker-container-module
- docker_network module documentation : https://docs.ansible.com/ansible/2.4/docker_network_module.html

⚠ Checkpoint: You should be able to access your API on your server.

❓ `3-3` Document your docker_container tasks configuration.

Front

If you have reached the end of each TP, you are able to access your api through your server.

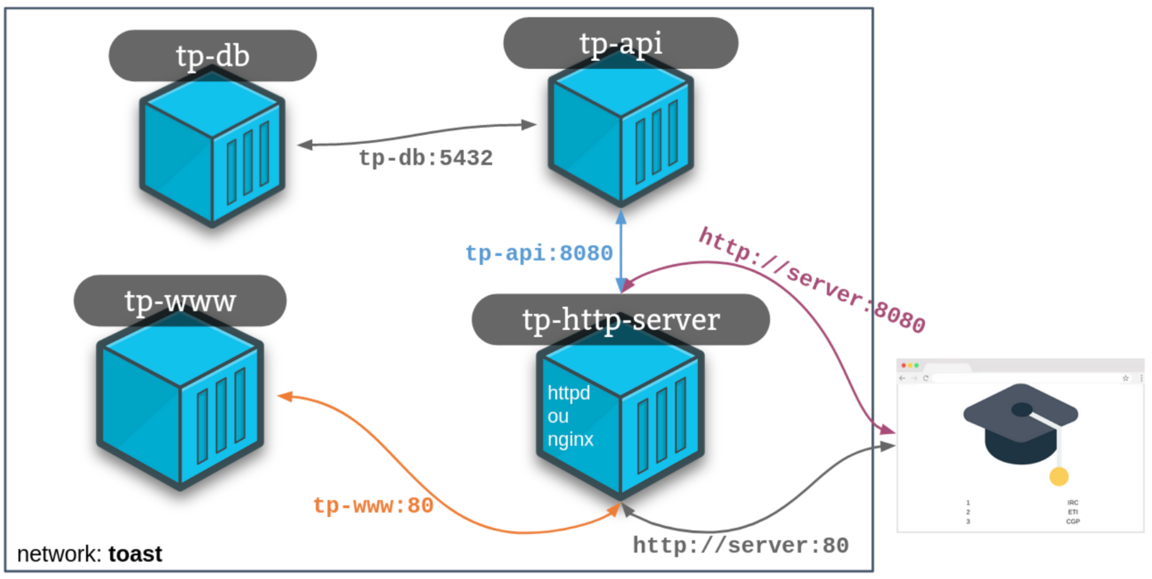
Your database, api and httpd must be up on your server and deployed with your Github action.

Everything under the hood of docker-compose.

Usually when we have an API we also have something called a front part to display our information.

That's your bonus part to do, you can find the code of the front ready https://github.com/Mathilde-lorrain/devops-front

You have to customize your httpd server to make the redirection correct between the API and the front. The httpd server is a proxy within your system.



Checkpoint: Front

Continuous deployment

Do this part in a separate workflow
Configure Github action to automatically deploy your application when you release it on the **production** branch of your github repository.

- It is a little bit overkilled to launch a Ansible job for deploying on one unique server. Therefore you ssh to your machine with your encrypted private key and only relaunch your http api backend application.
- You like challenges and overkilled solutions, you run your Ansible script through a Docker image (that provides Ansible, of course) and you use a VAULT to encrypt your private data.

Checkpoint: Full CI/CD pipeline in action