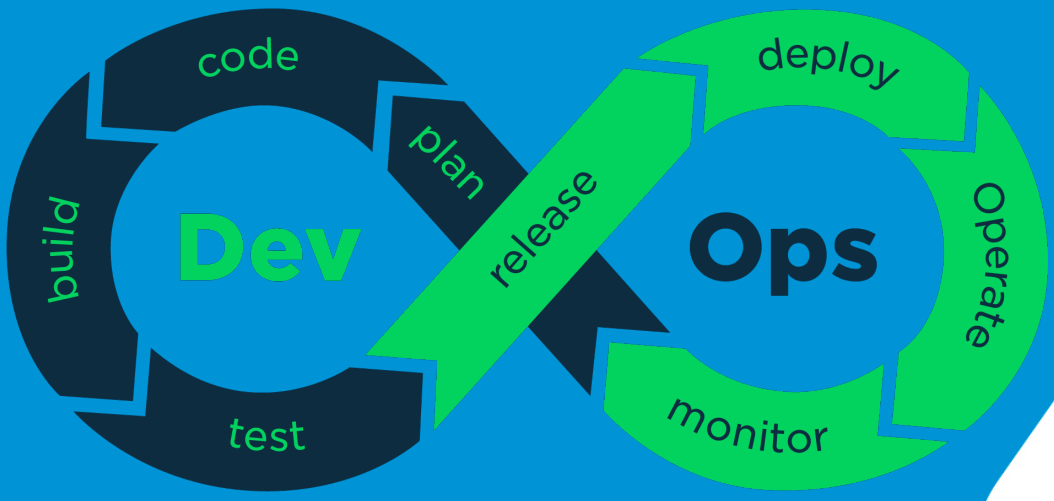
{EPITECH}

OCTOPUS

GROW TENTACLES AND CONTROL MACHINES



OCTOPUS

Configuring a machine (virtual or not) is easy, you can do what you want manually.

Configuring a few machines can be tedious to do manually.

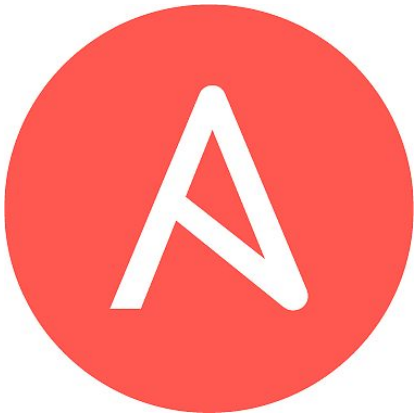
Configuring manually a lot of machines is exhausting, and not worth of investing that much time into it.

There comes the solution, for us perfectly lazy DevOps engineers (in the making): Ansible!

Ansible is an automation tool that allows one to automate an other part of the duty of operators and developers: *configuration management*.

Configuration management refers to the tools and practices used to manage the state in which machines needs to be, in a given set of machines.

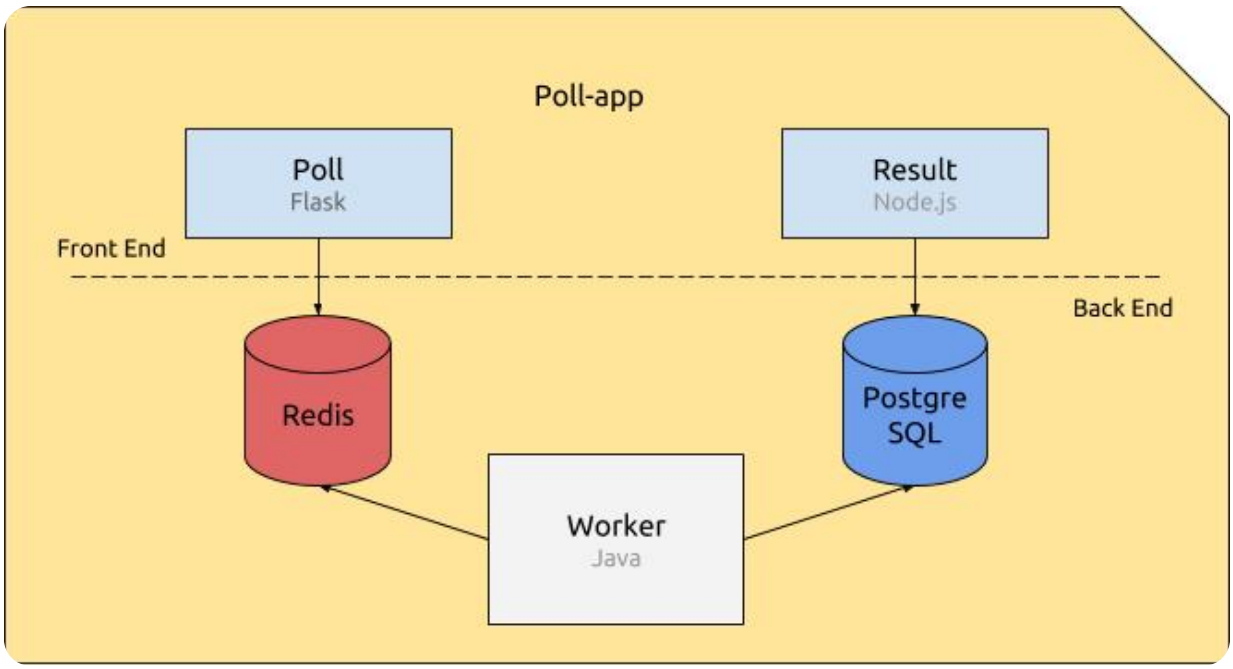
Ansible is an incredibly handy tool that allows you to do that in an efficient and scalable way.



General objective

You are going to work again with the poll application you worked with during the Popeye project.

As a reminder, the poll application starts with a Python Flask web client that gathers the votes, and then pushes them into a Redis queue.
Afterwards, the Java worker consumes the votes stored in the Redis queue, and pushes them into a PostgreSQL database.
Finally, the Node.js web client fetches the votes from the PostgreSQL database and displays the results.



This time, you are going to *deploy* the application onto 5 different machines, without using containers, by using Ansible.

Technical details

For this project, you have to turn in a `playbook.yml` file and a directory named `roles` that will hold your Ansible roles (see section *Repository structure* below).

Your project will be manually evaluated.

It will be tested using the following commands:

```
Terminal
~/B-DOP-400> export ANSIBLE_VAULT_PASSWORD_FILE=/tmp/.vault_pass
B-DOP-400> echo verySecretPassword > /tmp/.vault_pass
B-DOP-400> ansible-playbook -i production playbook.yml
```

We will use our own inventory file named `production`, which will ensure that your playbook can access the machines on which we will test your project.
It will be in compliance with the requirements described in the *Environment* section below.
The inventory file will be configured to connect to the machines as a normal user (i.e.: not `root`) which itself will be able to use the `sudo` command.

Docker, Podman, other containerization technologies, and Ansible Galaxy are strictly **forbidden in all their forms** for this project.
You may however use the remaining Community namespace's collections.

Security cannot be ignored when working in a DevOps environment.
As such, any clear-text password found in your repository will cause the evaluation to stop right away, and the entire project will be considered as failed.
You have been warned.

Features

Roles

You have to write 6 roles.

`base`

This role is associated to all machines.

- ✓ Installs useful packages (such as Git) using `apt`.
- ✓ Configures instance.

Think about what is necessary to do to ensure a properly configured instance.
What about useful ***commands*** *or* ***system tools****? ;)*

Do not install useless or too big packages.

`redis`

- ✓ Installs Redis.
- ✓ Sets up Redis.

`postgresql`

✓ Installs PostgreSQL 16.
✓ Creates a user `paul` with the password `democracyIsFragile` and limited permissions.
✓ Creates the schema of the database `paul`.

Be careful!
The `paul` ***user*** and the `paul` ***database*** are two different things!
The `paul` user is a user created in the database, not on the operating system itself.

`poll`

✓ Uploads `poll` service.
✓ Installs dependencies.
✓ Runs the poll web client.

`worker`

✓ Uploads `worker` service.
✓ Installs dependencies.
✓ Builds the worker.
✓ Runs the worker.

`result`

✓ Uploads `result` service.
✓ Installs dependencies.
✓ Runs the result web client.

Avoid using the `command` / `shell` / `raw` modules as much as possible.
Have a look at dedicated and cleaner modules instead, such as `apt_key`, `apt_repository`, or `pip`.
Use the `command` module only if no dedicated module exists.

All services must be managed by `systemd` and start automatically on boot.

In order to respect the 12 factor best practices, services must be configured with environment variables, such as: host, port, user, password, database name, etc.

After applying an Ansible playbook twice, you should have as few `changed` tasks as possible in your `PLAY RECAP` (see the example below).
This is the famous notion of *idempotence* that Ansible is famous for, and which you should strive for.

```
PLAY RECAP *********************************************************************
1.1.1.1    : ok=49 changed=0 unreachable=0 failed=0 skipped=12 rescued=0 ignored=0
2.2.2.2    : ok=25 changed=0 unreachable=0 failed=0 skipped=16 rescued=0 ignored=0
3.3.3.3    : ok=42 changed=1 unreachable=0 failed=0 skipped=22 rescued=0 ignored=0
```

Environment

Your inventory **must** have the following 5 groups, each containing 1 instance:

- ✓ `redis` with a `redis-1` instance;
- ✓ `postgresql` with a `postgresql-1` instance;
- ✓ `poll` with a `poll-1` instance;
- ✓ `result` with a `result-1` instance;
- ✓ and `worker` with a `worker-1` instance.

You will need 5 virtual machines based on Debian 12.
You can run it locally, it is however recommended to use a cloud platform.

Do not spend too much credit however, you may also need cloud platforms for the next DevOps projects.

Repository structure

In the end, your repository must at least contain the following files:

```
|-- playbook.yml
|-- poll.tar
|-- result.tar
|-- worker.tar
|-- group_vars
|   `-- all.yml
`-- roles
    |-- base
    |   `-- tasks
    |       `-- main.yml
    |-- postgresql
    |   |-- files
    |   |   |-- pg_hba.conf
    |   |   `-- schema.sql
    |   `-- tasks
    |       `-- main.yml
    |-- redis
    |   |-- files
    |   |   `-- redis.conf
    |   `-- tasks
    |       `-- main.yml
    |-- poll
    |   |-- files
    |   |   `-- poll.service
    |   `-- tasks
    |       `-- main.yml
    |-- result
    |   |-- files
    |   |   `-- result.service
    |   `-- tasks
    |       `-- main.yml
    `-- worker
        |-- files
        |   `-- worker.service
        `-- tasks
            `-- main.yml
```


Bonus

You want to go further? Great! Here is an idea:

- ✓ make Redis more secure by adding a password to it (you are allowed to update the applications' code in this case).

v1.4