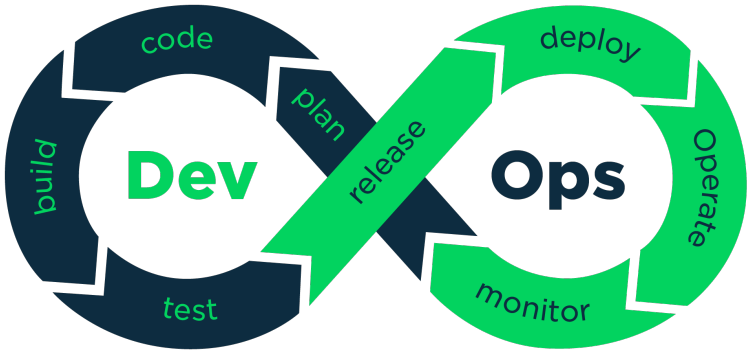
B4 - DevOps

B-DOP-400

Octopus - bootstrap

Dive into the configuration management sea and mind-control machines



Octopus - bootstrap

Let us get straight to the point: configuring machines is tedious to do manually.
And when is comes to configuring 5, 10, or even 100 or 1000 machines, “tedious” becomes an euphemism.

Thus comes back the notion of automation to save the day! But not in the same way as with the my_marvin project.
Here, we are talking about a way to facilite the configuration of machines, and a way to manage all of that: *configuration management*.

During this bootstrap, you are going to become familiar with one of the leading configuration management tool: *Ansible*, and its associated tool *Ansible Playbook*.

Step 0 – Ansible aims at the incredible!

A little quiz to start with

Traditions are traditions, one may not skip them! Here is a little quiz that you need to answer with your friends before going on an adventure:

- What is Ansible?
- What is an Ansible *playbook*?
- Why using Ansible instead of just bare shell scripts?
- In what way is Ansible different than Jenkins?
- Can octopuses help you in your DevOps journey? (octocats do not count)

Alright, have you answered all the questions above? For real? Then, onto the practical stuff!

Setup time

During this bootstrap, as well as with the Octopus project itself, you will need multiple virtual machines. You can run them locally, or use a cloud provider.

Install Ansible and Ansible Playbook.

Start 3 virtual machines running on Debian Buster, and upload your public SSH key to them (so you can connect without entering a password).

Step 1 – Taking inventory

Before describing your infrastructure, you will need to list targeted servers into an inventory file.
We often create an inventory for each environment: development, staging, production… For easier manipulation, it is recommended to organize servers in groups.

Create your first inventory:

- Group `web` -> 2 hosts `web-1` and `web-2`.
- Group `redis`-> 1 host `cache-1`.

You can now check if hosts are up and responding:

Terminal

```
~/B-DOP-400> ansible -i my-inventory web -m ping
~/B-DOP-400> ansible -i my-inventory redis -m ping
~/B-DOP-400> ansible -i my-inventory all -m ping
```

1. Here, `ping` is an Ansible module. Can you find a way to execute the `uptime` Linux command on all 3 hosts at the same time?
2. Can you reboot these machines simultaneously?
3. Using the `apt` Ansible module, can you install `htop` everywhere, then `nginx` only on web servers?
4. We do not need `htop` anymore, can you uninstall it using the `apt` module?
5. Can you ensure that Nginx will be automatically started at system start, using the `service` module?

Step 2 – Role-playing development

Ansible will help you configure a lot of machines and ensure that the expected state is reached. But executing tons of Ansible commands is not very funny! Starting from now, you will set in stone your configuration in YAML files (yes, YAML again!).

In the Ansible architecture, a *playbook* calls multiple *roles*, that call multiple *tasks*. Playbooks must be as simple as possible, and call reusable roles.

Roles must do one thing and do it well. For example: installing and configuring a database.
If you need multiple databases on different machines, this Ansible role will use *variables* to configure different usernames and passwords.

1. Create your first playbook, and a role `base`, which must:
    - be executed on every host;
    - update the OS if necessary;
    - install some essential packages: `apt-transport-https`, `ca-certificates`, `emacs-nox`, `git`, `curl`, `unzip`, `zsh`, ...
    - set the timezone (e.g.: `Europe/Paris` if your server is based in France);
    - set `zsh` as the root shell.
2. Create a `redis` role that installs... well... Redis. ;)
3. Create a `nodejs` role, responsible for installing Node.js and NPM.
4. Create a `deploy` role that:
    - uploads the application provided in this bootstrap to both web servers;
    - unzips application;
    - installs dependencies.
5. Right now, the Node.js application is still not running. Write your own SystemD service and upload it to `/etc/systemd/system/my-awesome-app.service`.

None of those tasks need the `shell` module !

Step 3 – Variables, for even more laziness

In the last step, your Node.js app has started. But as you probably figured out, the application configuration file has been hardcoded.

1. Can you create a "template" of `config.json`, using Ansible and Jinja2, then upload it on the server from the `deploy` role?
2. Configure the HTTP port and Redis hostname in `group_vars/all.yml`. This is the right place to write your (global) variables when using Ansible.
3. Now, try to set the Redis hostname dynamically, based on the inventory.
4. The SystemD service must be restarted **only** when the application is updated. If you deploy the app many times without changing anything, the service should not restart.

Step 4 – Keep secrets... secrets

Ansible comes with a tool that encrypts passwords: *Vault*.

Can you add an encrypted variable `password` containing `octopus-secrets` to the `group_vars/all.yml` file?

Try now to print it into `/tmp/octoleak.txt` on the `redis` machine.

Step 5 – A long time ago in a galaxy far, far away...

Red Hat, who leads the Ansible project, built a registry for roles: Ansible Galaxy.
It is analogous to Docker Hub for Docker images.

On this platform, find a role for installing both Docker and Docker Compose. Add then this role to your playbook.

Have you ever heard of the wise man *geerlingguy*?

What a long and dense bootstrap! You should have gained more or less control over your (Ansible) tentacles now. You are ready to dive into the An-sea-ble, and automate, from a single computer, an entire fleet of machines for you to control! Good luck with that! :)

STEP -1 – A SECRET WORLD

Nothing to do here, but here is a little tip to help you. ;)

The following task prints the value of the `name` variable.

```
- name: Create variable
  set_fact:
    name: world

- name: Print debug
  debug:
    msg: "Hello {{ name }}"
```