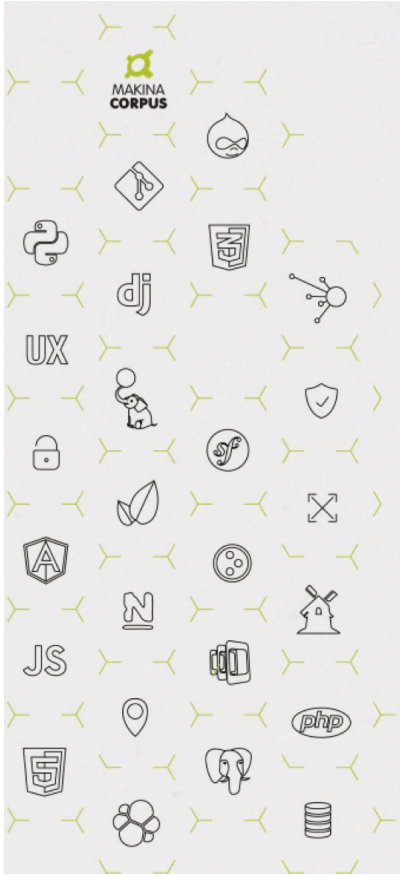
Gérer son site Drupal avec Ansible

Spécialisée dans l'ingénierie logicielle open source

Développements web, applications métier, cartographique et mobile


Des audits et des accompagnements techniques
Migration
Sécurité
Performance
SEO
UX
...
UX
MAKINA CORPUS



Des formations avec des experts sur des domaines maîtrisés

- Drupal
- Plone
- Django / Symfony
- PhoneGap Cordova
- Front-End (AngularJS / Responsive...)
- Sécurité Web
- Webmapping / PostGIS
- Python de l'initiation au scientifique
- Ergonomie / UX

Nous sommes basé
Sur Toulouse, Nantes et Paris

Ils nous font confiance

Programme

- Introduction
- Les termes à connaître
- Execution
- Démo

Ansible - Qu'est-ce que c'est ?

- Déploiement multi-node
- Exécution de tâches
- Gestion de configuration

Et puis

D'où ça vient

- Créé en 2012
- Racheté par Redhat en 2015
- GPLv3

C'est fait comment

- Python
- YAML
- Jinja2

Ça s'installe sur Linux, Mac, Windows (parce que Python) et ça communique en SSH

Le provisionning, l'orchestration ?

2017 : on administre plus son(ses) serveur(s) à la main, genaoueg !

On est humain, donc :

- on oublie
- on fait des fautes
- on s'enmêle les pinceaux
- on fait les choses différement à chaque fois
- on prend plus de temps qu'un script

UN ORCHESTRATEUR ?

Puppet ?

Chef ?

Salt ?

Petite comparaison

	Puppet	Chef	Salt	Ansible
Date de création	2005	2009	2011	2012
Langage	DSL	Ruby/DSL	YAML	YAML
Templates	ERB	ERB	Jinja2	Jinja2
Agent requis	Oui	Oui	Oui	Non
Éxecution ad-hoc	Non	Non	Oui	Oui
Vocabulaire				
Directive	Resource	Resource	State	Task
Plusieurs directives	-	Recipe	-	Role
Script	Manifest	Cookbook	SLS	Playbook
Celui qui contrôle	Master	Master	Serveur	Master
Ceux où on déploie	Agents	Clients	Minions	Hôtes

“Ansible est jeune mais simple, et a une communauté très active”

Rappels

- Jinja2 ≈ Twig
- YAML utilisé dans Drupal 8 notamment

Les termes à connaître

Inventaires, utilisateurs, modules, roles, variables, templates, handlers

Inventaire

Décrit le(s) serveur(s) sur lequel(lesquels) on va éxecuter des tâches.

```
[beta]
192.168.1.1
# host2 ...

[beta:vars]
git_branch=dev

[production]
monsupersite.com

[production:vars]
git_branch=master
```

Chaque section est un groupe de serveurs (IP ou nom de domaine)

Une section `nom_du_groupe:vars` permet de définir des variables uniquement pour ce groupe

Playbook

Un playbook est un(des) ensemble(s) de tâches qu'on execute sur un(des) host(s)

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root

  # Tâches à effectuer
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest

  - name: write the apache config file
    template: src="/srv/httpd.j2" dest="/etc/httpd.conf"

  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes
```

Ici on installe Apache, on lui met un fichier de config et on le démarre

Utilisateurs

Forcément quand on administre un serveur, il y a des utilisateurs Unix en face, et potientiellement un utilisateur `root`.

Pour pouvoir fonctionner, Ansible devra se connecter en SSH avec un utilisateur (`root` ou non) qui par la suite devra avoir les droits pour effectuer toutes les tâches. Pour définir l'utilisateur qui se connectera :

```
remote_user: sebastien
```

Si besoin d'un autre utilisateur pour executer ces tâches, on peut pour un playbook ou une tâche spécifier :

```
become: yes
become_user: postgres  # si non renseigné, ce sera root
become_method: su  # si non rensigné, ce sera sudo
```

Tâches

Chacun des *play* d'un playbook

```
- name: Ensure the httpd service is running
  service:
    name: httpd
    state: started
  become: true
```

Composé de :

- `name` : le nom de la tâche
- commande qui execute la tâche
- paramètres du module
- variables si besoin
- `become` : changement d'utilisateur si besoin

Execution des tâches

- Exécutées une à la fois, les unes à la suite des autres
- Idempotentes (même effet si exécutée plusieurs fois)
- Peuvent avoir ces statuts : ok / changed / fail / skipped
- Un playbook s'arrête au premier fail

Modules

http://docs.ansible.com/ansible/latest/modules_by_category.html

1378 modules dans le cœur

En vrac : apt, yum, copy, replace, cron, postgres, redis, shell, rabbitmq, npm, composer, group, ping service, user, ...

Possibilité d'ajouter ses propres modules (en écrivant des plugins python)

Python c'est simple

Parce que c'est installé de base dans toutes les distributions, il suffit de coder son module en python

```
#!/usr/bin/python
from ansible.module_utils.basic import AnsibleModule

def main():
    module_args = {'name': {'type': 'str', 'required': True}}
    result = {'message': '', 'changed': False, 'original_message': ''}

    module = AnsibleModule(
        argument_spec=module_args,
        supports_check_mode=True
    )

    # Votre métier ici ...

    if module.params['name'] == 'fail me':
        module.fail_json(msg='You requested this to fail', **result)
    module.exit_json(**result)

if __name__ == '__main__':
    main()
```

Role

Un *role* est un ensemble de tâches dont le tout a un *rôle*...

- Installer et configurer **Nginx** pour le web
- Installer et configurer **PHP-FPM**
- Installer et configurer **MySQL** pour la BDD
- Installer **Composer** pour Drupal 8 et autre
- Installer **Drush**
- Installer et configurer **Varnish** pour cache front
- Installer et configurer **LetsEncrypt** pour le SSL
- Récupérer notre repo **Git** et faire un déploiement

Plusieurs roles vont ainsi constituer notre playbook.

Pourquoi les roles ?

Les rôles sont en général **génériques** et configurables via des **variables**, qui ont une **valeur par défaut**.

On peut surcharger les variables de ces rôles, avec un ordre d'importance (non expliqué ici, sinon je vous embrouille).

Variables

Un playbook

```
---
- hosts: webservers
  vars:
    - http_port: 80
    - max_clients: 200
  vars_files:
    - env_vars/mesvariables.yml
```

Un inventaire

```
[production:vars]
git_branch=master
```

Syntaxe des variables dans un .yaml

```
---
# Used only for Redhat installation, enables source Nginx repo.
nginx_yum_repo_enabled: true

# Use the official Nginx PPA for Ubuntu, and the version to use if so.
nginx_ppa_use: false
nginx_ppa_version: stable

# avec du jinja2
processes: "{{processor_vcpus|default(processor_count)}}"

# des tableaux
mysql_users:
   - name: example
     host: 127.0.0.1
     password: secret
     priv: "*.*:USAGE"

# du texte sur plusieurs lignes
un_texte_tres_long: |
  "la c'est pour voir si vous suivez bien, le/la"
  "premier(e) à crier \"Makina Corpus c'est génial\""
  "aura mon estime éternelle"
```

Utilisation des variables

Soit dans les tâches :

```
- name: Define nginx_user.
  set_fact:
    nginx_user: "{{ nginx_user }}"
  when: nginx_user is defined
```

Peut être utilisée dans les **paramètres** ou même avec le mot clé `when` pour **passer la tâche** en fonction d'une variable

Soit dans les templates...

Les templates

```
- name: Add managed vhost config files.
  template:
    src: vhost.j2
    dest: "{{ nginx_vhost_path }}/vhost.conf"
    with_items:
      server_name: "{{ server_name }}"
      root: "{{ project_root }}"
    notify: reload nginx
```

Arborescence

```
monrole
├── tasks
|   └── vhosts.yml
└── templates
    └── vhost.j2
```

Fichier

```
server {
    listen {{ listen | default('80') }};
    server_name {{ server_name }};
    root {{ root }};
    index {{ index | default('index.html index.htm') }};

{% if error_page is defined %}
    error_page {{ error_page }};
{% endif %}
}
```

Les handlers

Tâches devant être réalisées une fois une autre faite.

Exemple : redémarrer nginx une fois un site ajouté

Arborescence

```
monrole
├── handlers
|   └── main.yml
└── templates
    └── vhost.j2
```

Fichier

```
---
- name: restart nginx
  service: name=nginx state=restarted
```

Utilisation

```
- name: Ensure nginx_vhost_path exists.
  file: path="{{ nginx_vhost_path }}" state=directory
  notify: reload nginx
```

Finie la théorie !

Place à la pratique

Pourquoi réinventer la roue ?

Grâce aux rôles **configurables** via les variables et donc **réutilisables**, il y a toute une **galaxie** de roles provenant de la communauté, avec gestion de dépendances & co.

https://galaxy.ansible.com/

Mise en situation

- J'ai un serveur (vagrant)
- J'ai un site jaimelesponeys.dev
- Il doit être déployé en Drupal 8
- J'ai la flemme*

*ces slides ont été terminées ~~hier~~ aujourd'hui à [renseigner l'heure]

Préparation

Créer le fichier `Vagrantfile`

```
Vagrant.configure("2") do |config|
  config.vm.box = "debian/jessie64"
  config.ssh.insert_key = false
  config.vm.network :forwarded_port, guest: 80, host: 4567
end
```

Créer le fichier `inventory`

```
[myserver]
10.0.2.15 ansible_ssh_host=127.0.0.1 ansible_ssh_port=2222
```

Installer la VM et les roles

```
$ vagrant up
$ ansible-galaxy install --roles-path . geerlingguy.nginx \
    geerlingguy.mysql geerlingguy.php geerlingguy.php-mysql \
    geerlingguy.composer geerlingguy.drush geerlingguy.drupal
$ tree -d -L 1
.
├── geerlingguy.composer
├── geerlingguy.drupal
├── geerlingguy.drush
├── geerlingguy.mysql
├── geerlingguy.nginx
├── geerlingguy.php
└── geerlingguy.php-mysql
```

Playbook

```
---
- hosts: myserver
  user: vagrant
  become: yes
  roles:
    - geerlingguy.nginx
    - geerlingguy.mysql
    - geerlingguy.php
    - geerlingguy.php-mysql
    - geerlingguy.git
    - geerlingguy.composer
    - geerlingguy.drush
    - geerlingguy.drupal
  vars_file:
    my_vars.yml
```

Variables de configuration utilisées

```
drupal_install_site: true
drupal_build_composer_project: true
drupal_deploy_dir: "/var/www/jaimelesponeys"
drupal_db_backend: mysql
drupal_domain: "jaimelesponeys.dev"
drupal_db_user: root
drupal_db_password: root
php_enable_php_fpm: true
php_fpm_listen: /var/run/php5-fpm.sock
nginx_vhosts:
  - server_name: "{{ drupal_domain }}"
    root: "{{ drupal_deploy_dir }}/web"
    index: "index.php"
    extra_parameters: |
      location / {
          try_files $uri /index.php?$query_string;
      }
      location @rewrite {
          rewrite ^/(.*)$ /index.php?q=$1;
      }
      location ~ \.php$ {
          fastcgi_split_path_info ^(.+\.php)(/.+)$;
          fastcgi_pass unix:/var/run/php5-fpm.sock;
          fastcgi_index index.php;
          fastcgi_param SCRIPT_FILENAME $document_root$fastcgi_script_name;
          include fastcgi_params;
      }
```

Problèmes rencontrés

- PPA PHP-Dotdeb à ajouter

```
- name: Install DotDeb repo.
  apt_repository: repo="deb http://packages.dotdeb.org jessie all"

- name: Import DotDeb GPG key.
  apt_key: url="https://www.dotdeb.org/dotdeb.gpg"
```

- Mauvais nommage de paquet : `php-acpu` -> `php7.0-acpu`
- Il manquait `geerlingguy.git` sinon impossbile d'installer composer
- Il manquait `php7.0-zip` sinon composer téléchargeait tout via git
- "this may take a while": oui, composer est très lent
- le chmod 777 de `sites/default/files` a faire à la main
- un flush des caches manuel est requis

Aller plus loin : Drupal VM

Projet fait par geerlinguy, un mec qui doit avoir beaucoup de temps libre

http://drupalvm.com/

Requiert :

- Vagrant
- Virtualbox
- Ansible

Peut installer :

- sur Redhat/CentOS, Debian ou Ubuntu
- Nginx ou Apache
- MySQL ou Postgres
- En option : Drupal Console, Drush, Varnish, Apache Solr, Elasticsearch, Node.js, Selenium (for Behat testing), Memcached, Redis, Blackfire, XHProf, XDebug

Retour d'XP

Les tags

Permet de passer des étapes :

```
ansible-playbook -i ansible/beta ansible/site.yml --tags=deploy
```

Utile pour le déploiement pur (mise à jour)

Ne pas faire confiance

- Prendre un rôle
- Se l'approprier
- Le modifier pour ses besoins

Versionner

- Mettre tout ça dans Git (dans un autre repo)

Faire de la veille

- C'est jeune donc, il y a des nouveau rôles tous les jours
- TODO list personnelle : regarder le provisioning de containers Docker

Merci !

Des questions ?