Ansible

Gilles Pietri

19 septembre 2016

Introduction DevOps

L'origine du mouvement DevOps

- Terme apparu en 2009 : Développement + Opérationnel
- Implication du développeur dans le déploiement
- ... Et de l'administrateur système dans le développement

Le besoin d'industrialisation pour l'exploitation

- Méthodes agiles, quid de l'infrastructure
- Accélération du déploiement
- Assurance qualité pour l'exploitation
- Notion d'idempotence des opérations

Tour d'horizon des outils

- Vue d'ensemble
- Les outils basés sur un agent
- Sans agent... Voire juste des librairies ?

Positionnement de Ansible dans le paysage actuel

- Projet quasiment mature (premiers commits en 2012)
- du Python, pas d'agent
- Empreinte assez minimale

Présentation de Ansible

2. Présentation de Ansible
   - Origine du projet
   - Fonctionnement, rapport à SSH
   - Installation
   - installer ansible sur la machine de contrôle, préparer les nœuds.

Origine du projet

- Michael DeHaan, projet soirs et weekends
- Adoption fulgurante par la communauté
- Une entreprise bien financée au départ, rachetée par RedHat
- Projet mûr, v2 bien éprouvée

Fonctionnement, rapport à SSH

- Configuration YAML
- Construction de code Python
- Communication via SSH, transferts et exécution sur les hôtes
- Normalement : push, mais pull possible

Installation

- Pré-requis nœuds, maîtres : pas grand chose
- Versions via pip, ou dans la distribution
- Récupération des sources via GitHub

Atelier pratique

installer ansible sur la machine de contrôle, préparer les nœuds.

- Sur le maître : pip install ansible, ou jouons avec git
- Sur les nœuds : probablement rien à faire, Python 2.5 souhaitable a minima
- Préparation des machines

Configuration et commandes ≪ Ad Hoc ≫

Configuration SSH

- Cas des mots de passe (–ask-pass), de sudo (–become, option become
- Fonctionnement des clés privées (ssh-keygen, ~/.ssh/authorized_keys)
- Gestion des clés via un agent
- Configuration d'Ansible (ansible.cfg)
  `http://docs.ansible.com/intro_configuration.html`

Mise en place de l'inventaire

- Pourquoi l'inventaire ?
- Définition de l'inventaire
  - dans /etc/ansible/hosts (ini)
  - dans hosts, ou via la configuration (hostfile)
  - via un exécutable qui renvoie la liste (JSON)

```
server1

[webservers]
web[1:3]
preprod-web

[infra:children]
webservers
```

Commandes ad-hoc

- Rappel inventaire, ssh, etc.
- Format des commandes
  ansible demobox1 -a "whoami"
- Ce n'est pas du shell, mais le module command
  ansible demobox1 -m shell -a "ps aux | grep ssh"

Les modules : fichiers, packages, utilisateurs, services, ...

- Syntaxe générale :
  ansible servers -m module -a "paramètres=valeur"
- ping, service (name, state), copy, ...
- Documentation via ansible-doc module ou sur
  `http://docs.ansible.com/modules_by_category.html`

Atelier pratique

Utiliser ansible pour exécuter des commandes sur les nœuds, transférer des fichiers.

- Mise en place de 3 nœuds (machines virtuelles, Debian), configuration de l'inventaire
- Installer le paquet "apache2"
- Transférer un fichier index.html dans /var/www

Déploiement et orchestration avec les playbooks

Introduction aux playbooks et à YAML

- Première approche des playbooks, une liste de tâches
- Syntaxe YAML (ghostlol.yml)

```
- hosts: infra
  tasks:
    - name: upgrade all packages
      apt: upgrade=yes update_cache=yes
```

Encore un peu de YAML

- Listes : -
- Dictionnaires : clé : valeur
- Dictionnaire avec des listes en valeur, listes en dictionnaires
  http://docs.ansible.com/YAMLSyntax.html

```
# Liste, contenant un dictionnaire
- hosts: xxx #  Première clé
  tasks: # Deuxième clé : tasks, qui est une liste
    # Liste de dictionnaires
    - name: Nom de la tâche # Premier élément, clé "name"
      command: "SuperCommande" # Clé command
    - name: Deuxième tâche # Deuxième élément de la liste
      copy: src=toto.txt dest=titi.txt # Clé copy
```

Traduction en Python

```
# En Python :
[ { hosts: 'xxx',
    tasks: [ { name: 'Nom de la tâche',
               command: 'SuperCommande'
             },
             { name: 'Deuxième tâche',
               copy: 'src=toto.txt dest=titi.txt'
             }
           ]
  }
]
```

Les acteurs (hôtes)

La clé ≪ hosts ≫ définit pour chaque acte du playbook qui sont les cibles. Expression de ciblage, serveur unique, groupe...

```
- hosts: server1
  [...]
- hosts: webservers
  [...]
- hosts: dbservers!db-prod
  [...]
- hosts: all
  [...]
```

Servira de nom pour l'acte, mais on peut aussi utiliser une clé ≪ name ≫

Les tâches

Chaque tâche décrit un module à exploiter et ses paramètres. Une erreur arrête le traitement.

```
- hosts: all
  tasks:
    - debug: msg="hello world"
    - copy: src=hello.txt dest=/tmp/hello.txt
    - service: name="nginx" state="started" enable="yes"
```

Clé ≪ name ≫ pour identifier la tâche explicitement

Les handlers

Exécuter une opération une seule fois pour le playbook, et seulement si il y a eu changement.
Clé ≪ handlers ≫ dans le playbook, clé ≪ notify ≫ dans la tâche.

```
- hosts: all
  handlers:
    - name: reload apache
      service: name="apache2" state="reloaded"
  tasks:
    - name: Deploy configuration
      copy: src=apache2.conf dest=/etc/apache2/httpd.conf
      notify: reload apache
```

Atelier pratique : écrire un playbook, l'exécuter

- Reprendre ▸ l'exemple en ad-hoc dans un playbook webserver.yml
- L'essayer avec : ansible-playbook webserver.yml –check
- L'exécuter
- Modifier le fichier index.html, vérifier les changements occassionnés via –check –diff, puis relancer le playbook

Les variables

- Le rôle des variables : configuration paramétrée, templates
- Définition de variable, par ordre de priorité croissante
  - Dans l'inventaire, par groupe [mongroupe :vars] ou dans group_ vars/groupe
  - Dans l'inventaire, par machine : monserveur variable1=valeur ou dans host_vars/monserveur
  - Dans un playbook (vars) ou dans une tâche (register)
  - Sur la ligne de commande d'un playbook (–extra-vars)
- Réutiliser une variable : {{ variable }}

Les variables

- Le rôle des variables : configuration paramétrée, templates
- Définition de variable, par ordre de priorité croissante
  - Dans l'inventaire, par groupe [mongroupe :vars] ou dans group_ vars/groupe
  - Dans l'inventaire, par machine : monserveur variable1=valeur ou dans host_vars/monserveur
  - Dans un playbook (vars) ou dans une tâche (register)
  - Sur la ligne de commande d'un playbook (–extra-vars)
- Réutiliser une variable : {{ variable }}

Les faits

Ansible, à l'aide du module setup, peut fournir des variables sur les informations découvertes sur la machine visée.

- Récupération de tous les faits ansible :
  ansible -m setup monserveur
- Utilisation via {{ ansible_xxx }} : {{ ansible_os_family }}
- Utilisable comme variable directement dans les playbooks
- Option gather_facts des playbooks, appel du module setup
- Accès aux faits composites (hashes) via la notation pointée, ou indexation Python :

```
fait.sous-element ou fait['sous_element']
```

Les templates

Le module template permet d'étendre les variables au contenu des fichiers sur les machines distantes.

- Moteur Jinja2 (en fait utilisé directement sur les playbooks)
  Doc sur : `http://jinja.pocoo.org/docs/dev/`
- Utilisation, fichier motd.j2 :

```
Vous etes sur {{ inventory_hostname }}
J'ai l'IP : {{ ansible_default_ipv4.address }}
Je tourne sur {{ ansible_distribution }} version {{ ansible_distribution_version}}
```

Playbook :

```
- hosts: vm1
  become: true
  tasks:
    - template: src=motd.j2 dest=/etc/motd
```

Boucles et conditions

- Boucles dans les playbooks : simples, indexées, hashées
  with_items
  with_dict
- Conditions et filtres de variables
  register (et when)

Rôles et inclusions

- Inclusions de fichiers pour faciliter la ré-utilisation
- Notions de rôles
- Dépendances de rôles
- Arborescence type

Inclusions

Possibilité d'inclure quasiment n'importe quoi, n'importe où.
Intérêt ? Modularité (et éviter les indentations à outrance en YAML...)

```
- hosts: all
  tasks:
    - include tasks_part1.yml
    - debug: msg="OK, partie 1 terminée"
- include: playbook_suite.yml
```

Rôles

Ou utiliser le schéma attendu par ansible : les rôles

```
mon_role
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
└── vars
    └── main.yml
```

Et intégrer dans le playbook, avec la clé ≪ roles ≫ :

```
- hosts: all
  roles:
    - role1
    - { role: role2, param1: "valeurxx" }
```

Atelier pratique : création du rôle webserver

- Partant du playbook webserver, créer un rôle
- Vérifier la priorisation des variables, notamment dans defaults
- Adapter le playbook, et intégrer les autres comme modules

Bonnes pratiques pour la rédaction des playbooks

- Réutilisabilité, factorisation
- Gestion des incréments, mises à jour partielles et rotations
- Utilisation d'un SCM pour versionner les playbooks
- Le coffre-fort, pour la gestion des mots de passe

Pour aller plus loin

5 Pour aller plus loin

- Réutilisation et communauté : ansible-galaxy
- Intégration avec d'autres produits : vagrant, AWS, Rackspace, . . .
- Interface Web propriétaire : Ansible Tower
- exploiter des modules externes, déployer un environnement vagrant avec ansible

Réutilisation et communauté : ansible-galaxy

- Utiliser ansible-galaxy pour ne pas ré-écrire des rôles
- Exploiter ansible-galaxy dans un playbook
- Contribuer un rôle et ansible-galaxy –init pour le scaffolding

Intégration avec d'autres produits : vagrant, AWS, Rackspace, . . .

- VagrantFile pour ansible
- Manipulation des AMI de Amazon EC2
- Tour d'horizon des plugins tiers

Interface Web propriétaire : Ansible Tower

- Self-service
- Fonctionnalités principales
- Tarifs

Atelier pratique

exploiter des modules externes, déployer un environnement vagrant avec ansible

- Ecrire un VagrantFile qui exploite un playbook
- Fournir un playbook qui simule le déploiement de Drupal (répertoires, initiation db, ...)
- Tester !