Évolution de l'outil de gestion de configuration et d'orchestration à l'université de Strasbourg

Ludovic Hutin
Université de Strasbourg - Direction du Numérique
Département Infrastructures - Pôle Plateforme Cloud et Intégration
14 rue René Descartes
67 081 Strasbourg Cedex

François Ménabé
Université de Strasbourg - Direction du Numérique
Département Infrastructures - Pôle Plateforme Cloud et Intégration
14 rue René Descartes
67 081 Strasbourg Cedex

Résumé

Ansible est un outil libre de gestion de configuration et d'orchestration multi-systèmes. Simple et accessible, il s'est progressivement imposé comme un standard du marché. Utilisé à l'Université de Strasbourg, Ansible est devenu un prérequis au déploiement des nouvelles applications.

Historiquement, plusieurs outils ont été utilisés pour gérer les configurations (voire l'orchestration) de nos différentes infrastructures et applications. Depuis deux ans, nous faisons converger l'ensemble de ces solutions vers un outil unique : Ansible. Nous montrerons tout d'abord la démarche projet utilisée pour mener cette transformation complexe et chronophage qui nécessite d'impliquer tous les acteurs, de les former, et surtout d'uniformiser les pratiques.

Nous montrerons ensuite les prérequis nécessaires au déploiement et à la mise en production cohérente des VMs, intégrées au système d'information et sécurisées. Pour cela, nous avons développé plusieurs modules Ansible et nous supportons désormais un nombre conséquent d'OS (Windows, CentOS, RedHat, Ubuntu). Notre playbook de déploiement de VM a été adapté afin de l'intégrer à Ansible Tower, ce qui permet à tous nos collègues de créer des VMs disposant de toutes les configurations de base.

Enfin, nous présenterons Ansible Tower. Cet outil nous a fait gagner en qualité, en efficacité et nous a permis de déléguer à tous nos collègues des tâches complexes. Nous avons développé un plugin d'inventaire dynamique, nous permettant de l'interfacer avec GLPI, ce qui nous offre la possibilité d'exécuter en masse des actions sur un ensemble de serveurs en fonction de critères de recherche avancés.

Mots-clefs

Gestion de configuration, orchestration, administration système, Ansible, Tower

1 Introduction

La *Direction du Numérique* (*DNum*) de l'*Université de Strasbourg* s'occupe de fournir la totalité des moyens informatiques et numériques à l'ensemble de ses usagers mais aussi à ses partenaires (CNRS, INSA, ENGEES, ENA, etc.). En termes de population, cela représente 35 composantes, 51 000 étudiants, 6 000 enseignants et personnels administratifs (environ 80 000 comptes dans l'annuaire LDAP). Pour assurer ces missions, la *DNum* gère une infrastructure d'environ 150 machines physiques et plus de 1000 machines virtuelles permettant d'héberger environ 350 applications.

Les problématiques d'un service tel que le nôtre sont assez particulières: nous devons fournir beaucoup d'applications pour de nombreux besoins métiers/fonctionnels disposant d'une criticité variable et d'une charge d'exploitation différente. Le tout est hébergé sur un périmètre technologique large et mouvant. Il est donc nécessaire de s'outiller et d'évoluer dans nos pratiques. C'est pourquoi nous avons choisi de converger vers l'outil *Ansible*.

En première partie, nous présenterons la démarche projet et l'accompagnement, important et nécessaire, au déploiement d'un outil d'automatisation au sein d'un service de 150 personnes. En deuxième partie, nous présenterons quelques-unes de nos réalisations afin d'illustrer les possibilités offertes par *Ansible*. Enfin, en dernière partie, nous parlerons d'*Ansible Tower* qui est une surcouche web permettant notamment de déléguer des tâches d'exploitations à l'ensemble de la *DNum*.

2 Démarche projet et accompagnement

Avant de commencer à migrer vers Ansible il y a environ deux ans, nous avions officiellement deux outils permettant d'automatiser les déploiements d'applications : *Chef* et *Pydiploy*. Chef est un outil bien connu qui fait d'ailleurs l'actualité au moment où ces lignes sont écrites [1] et qui a aussi fait l'objet d'une présentation de notre part aux JRES 2013 [2]. *Pydiploy* [3], qui est conceptuellement proche d'*Ansible*, est un outil développé en interne par les développeurs de la *DNum* pour répondre au besoin spécifique du déploiement des applications internes développées en *Python/Django*.

Nous nous sommes retrouvés avec ces deux outils pour différentes raisons historiques. Mais avoir deux outils pour faire la même chose n'était plus pertinent et tenable, surtout dans un contexte tendu en ressources humaines et compétences techniques, et où le temps passé à automatiser le déploiement de nos applications, pourtant nécessaire, reste extrêmement limité par rapport à celui passé à l'exploitation courante du parc (« à l'ancienne » et en « mode pompier »)!

2.1 Étude, ComArc et projet

Un consensus officieux a commencé à apparaître entre les différentes personnes issues de plusieurs départements de la *DNum* sur l'outil *Ansible*. *Chef* était jugé « trop compliqué » et son adoption sur cinq ans a été limitée à une poignée de personnes. De plus, développer et maintenir son propre logiciel d'automatisation, dans le contexte cité précédemment, n'était plus réaliste.

La *DNum* a donc décidé de fédérer cette initiative autour d'un projet. Au-delà d'officialiser l'outil, les principaux objectifs étaient d'avoir un cadrage clair mais surtout des objectifs viables pour l'ensemble des équipes. Nous avons rédigé une étude qui a :

- rappelé la nécessité d'automatiser le déploiement des applications ;
- fait un bilan des outils existant ;
- présenté Ansible et les avantages de l'outil ;
- rappelé la nécessité de mettre en place un programme d'accompagnement.

Cette étude a été présentée au *Comité d'Architecture* de la *DNum* (*ComArc*) pour valider l'ajout d'*Ansible* à notre socle technique qui définit les technologies et langages officiellement supportés par la *DNum*. L'étape suivante fut de rédiger la note de cadrage du projet et de le lancer.

2.2 Ateliers et guide de bonnes pratiques

Avant même le démarrage du projet, plusieurs personnes utilisaient déjà *Ansible*. Il nous a paru nécessaire d'organiser des ateliers où chacun présentait ses besoins et ses réalisations avec comme objectif d'homogénéiser et de normer nos pratiques.

Ces ateliers ont abouti à la rédaction d'un guide de bonnes pratiques qui définit :

- des règles générales (suivre au maximum les versions d'*Ansible*, un rôle ne doit pas mélanger les genres, faire un inventaire par environnement, etc.) ;
- à minima des règles de *coding-style* et notamment l'indentation des fichiers YAML (cela peut paraître anecdotique, mais c'est la base du travail en équipe !) ;
- l'obligation d'historiser avec *Git* ;
- l'organisation dans notre plateforme *GitLab* des différents sous-groupes et des dépôts.

2.3 Formation

Une fois le guide de bonnes pratiques rédigé, nous devions former l'ensemble de nos collègues. Après avoir organisé une première formation à destination de plusieurs responsables d'applications, nous nous sommes vite rendu compte que celle-ci n'avait pas de sens si les personnes n'avaient pas pour objectif de l'utiliser dans les semaines suivantes.

Nous avons abandonné les formations collectives génériques et nous les avons remplacées par des formations personnalisées. Ainsi, lorsqu'un(e) collègue souhaite utiliser *Ansible* pour le déploiement de son application, une personne du pôle *PCI* l'accompagne dans sa réalisation.

Nous avons identifié de bien meilleurs résultats en utilisant cette méthode de formation. Elle a, au premier abord, un coût bien plus élevé mais s'est avérée bien plus rentable pour standardiser et emmener l'ensemble de la *DNum* sur des outils communs.

2.4 Ansible games

En complément des formations proposées, nous avions envie de former nos utilisateurs de façon ludique.

Un jeu, en cours de construction, est réalisé sur notre plateforme pédagogique *Moodle*. Cette activité se déroule sur une journée. Les équipes doivent répondre à un nombre conséquent de questions, de niveaux variés, autour d'*Ansible*. Chaque équipe est constituée de personnes issues de différents départements (2-3 personnes maximum) permettant :

- de se mélanger à des personnes que nous ne côtoyons pas forcément (aspect *team building*) ;
- d'avoir une complémentarité technique cohérente par équipe.

Pour la notation, chaque exercice soumis sera corrigé par l'organisateur et rapportera un certain nombre de points à l'équipe. Elle pourra voir son classement en temps réel. La direction nous sponsorisera dans cette expérimentation et nous fournira des lots pour motiver les troupes.

Cette épreuve se déroulera avant les JRES (28/11/2019), mais après le dépôt de ce résumé. De ce fait, nous n'avons pas connaissance des résultats au moment de l'écriture de cet article.

2.5 Planning de migration

Le remplacement d'un outil de déploiement par un autre, implique généralement de migrer l'intégralité de l'existant. Cette activité étant chronophage, nous avons décidé de ne pas le faire. Nous avons concentré nos efforts sur l'automatisation de nouvelles applications et nous migrerons les anciennes plus tard.

3 Présentation de quelques réalisations

Maintenant que nous avons présenté la démarche interne et l'accompagnement nécessaire à la mise en place d'un outil comme *Ansible*, nous allons illustrer nos propos par quelques-unes de nos réalisations afin de vous montrer ce qu'il est possible de faire avec cet outil.

3.1 Déploiement de nouvelles machines virtuelles

Le premier exemple est le déploiement de nouvelles machines virtuelles correctement intégrées au système d'information (inventaire/CMDB, monitoring, logs…) dans notre plateforme principale de virtualisation *OpenNebula*. L'objectif est d'automatiser complètement le déploiement des systèmes, y compris Windows, sans intervention manuelle à l'exception de la configuration décrivant les différents paramètres de la machine (type de système, nombre de CPU, etc.).

3.1.1 Création des machines et gestion des DNS

La première opération à effectuer est de créer les machines dans la plateforme de virtualisation. Nous avons écrit un rôle `opennebula` pour cela, qui gère :

- la création de la machine ;
- la modification des entrées DNS (pour remplacer le nom généré par l'IPAM par un nom plus parlant) ;
- l'installation des prérequis nécessaires pour faire du *Ansible* (via le module `raw`) ;
- une partie de la configuration de base des systèmes (hostname, machine-id, formatage et montage des disques, etc.).

Pour écrire ce rôle, nous avons notamment dû développer deux modules : un pour *OpenNebula* et un pour *Netmagis* qui est notre application DNS. La création de modules *Ansible* est simple. Le code générique correspondant à la structure d'un module est minime et la complexité dépend de ce que doit faire le module et de la gestion de l'idempotence.

Pour donner un exemple de configuration (inventaire *Ansible* au format *YAML)* permettant le déploiement d'une machine :

```
---
all:
  hosts:
    prometheus-test.di.unistra.fr:
      one_ostype: linux
      one_template_id: 302 # centos7-infra-ssd
      one_cpu: 4
      one_vcpu: 4
      one_memory: 8g
      one_nics: [26, 8] # Vlan 302 sup, Vlan 896 SAN
      one_disks:
      - datastore_id: 104
        size: "200g"
        fstype: xfs
        device: /dev/vdb
        mountpoint: /var/lib/prometheus
```

3.1.2 Rôle « base »

Pour l'installation de certains éléments des systèmes (EPEL, SELinux, firewall local, etc.) et l'intégration aux outils du système d'information, nous avons implémenté une douzaine de rôles « base ». Ces rôles sont génériques et ne font des actions que lorsqu'elles sont nécessaires ou souhaitées (par exemple, un rôle à destination d'un système *Linux* peut être appliqué sur un système *Windows* sans erreur).

Ansible repose en grande partie sur des variables qu'il est possible de passer à plusieurs niveaux [5]. Pour ces rôles, nous avons fait le choix d'utiliser trois « types » de variables (d'un point de vue usage et non *Ansible*):

- celles qui décrivent la configuration et qui sont à mettre de préférence dans l'inventaire ;
- celles qui permettent de gérer des comportements à passer de préférence en « extravar » (faut-il mettre à jour le système ? Le redémarrer ?)
- les « facts » récupérés par *Ansible*, notamment pour différencier les types de systèmes.

3.1.3 « Meta role »

Avoir une douzaine de rôles à inclure dans un playbook puis devoir configurer les variables de ces rôles pour chaque serveur est lourd (Et nous ne nous sommes pas encore attelés à la partie « applicative »). En parallèle, nous avons plusieurs profils d'administration (il arrive que certains serveurs soient « infogérés » par des entreprises externes) qui ne nécessitent pas l'utilisation des mêmes rôles et la même configuration. Pour remédier à cela, nous avons créé des « méta-rôles » qui vont inclure statiquement et « tagger » les rôles génériques en fonction des besoins et définir dans les `defaults` du rôle la configuration des rôles « enfants ».

Au final, le playbook pour la création de nouvelles machines est très simple :

```
- hosts: all
  remote_user: root
  # Required as guests may not exist yet
  gather_facts: no
  tasks:
  - name: Provision hosts
    import_role: { name: opennebula }
  - name: Integrate hosts to the SI
    import_role: { name: base-dnum }
```

3.1.4 Quelques exemples d'exploitation

Le premier exemple concerne la gestion de crise mise en place lors de la publication de l'alerte sécurité « TCP SACK PANIC [6] ». Lors de la diffusion de l'alerte, toutes les distributions ne proposaient pas de correctif pour le noyau. Nous avons alors décidé d'appliquer le correctif suivant à l'ensemble de nos machines directement exposées sur Internet.

```
sysctl -w net.ipv4.tcp_sack=0
```

Ce qui génère le playbook suivant :

```
- name: Disable selective acknowledgments for the running system
  hosts: all
  remote_user: root
  tasks:
    - name: Set net.ipv4.tcp_sack to 0
      sysctl:
        name: net.ipv4.tcp_sack
        value: '0'
        sysctl_set: yes
        state: present
        reload: yes
        sysctl_file: /etc/sysctl.d/99-tcpsack.conf
```

Qu'on exécute de la manière suivante sur l'ensemble de nos serveurs :

```
ansible-playbook -i hosts disable_tck_sack.yml
```

Le deuxième exemple proposé est l'extinction de l'ensemble de nos 1250 VMs (hébergées sur *OpenNebula*) dans le cadre d'une opération de maintenance. L'objectif est d'éteindre l'ensemble des VMs dans l'ordre et dans le délai imposé (1h).

Dans un premier temps, nous « taggons » chacune des VMs dans une catégorie (« STOP1 », « STOP2 », etc.) au sein de la plateforme de virtualisation.

Puis, grâce à notre module « OpenNebula » et un inventaire dynamique écrit pour l'occasion, nous envoyons un signal d'extinction propre à l'ensemble des serveurs d'une catégorie.

```
- hosts: "{{ label|mandatory }}"
  remote_user: root
  gather_facts: no
  strategy: free
  tasks:
  - name: Manage stage #{{ stage }}
    when: label in one.USER_TEMPLATE.LABELS|default([])
    block:
    - name: Power-off guests through the API
      delegate_to: localhost
      onevm:
          endpoint: "{{ one_endpoint }}"
          username: "{{ one_admin_username }}"
          password: "{{ one_admin_password }}"
          id: "{{ one.ID|int }}"
          state: shutdown
  etc.
```

Puis on exécute le script de la façon suivante :

```
ansible-playbook shutdown.yml -i one.py -e @~/extravars/ids.yml
--vault-id perso@prompt -e label=STOP1
```

Pour l'anecdote, nous avons mis à genoux la plateforme de virtualisation, celle-ci n'a pas apprécié l'arrêt de 200 VMs en même temps et nous avons mis 1h30 pour éteindre proprement et dans l'ordre l'ensemble de nos VMs.

3.2 Inventaire dynamique pour GLPI

Un autre exemple est l'inventaire dynamique que nous avons écrit pour interagir avec notre plateforme d'inventaire *GLPI*. Un inventaire dynamique de type « script» [7] se contente d'utiliser du JSON comme source de données. Nous avons écrit une CLI en *Python* [8], interagissant avec l'API *GLPI,* qui nous permet de récupérer les données et de les formater en JSON. Le *mapping* entre l'arborescence des groupes et les requêtes *GLPI* est géré par un fichier de configuration au format *YAML*.

Par exemple, un fichier de configuration qui permet de générer des groupes par constructeur de serveurs physiques :

```
servers:
  children: [dell, hp]
  itemtype: Computer
  fields:
    - 1   # name
    - 4   # ComputerType.name
    - 33  # Domain.name
    - 23  # Manufacturer.name
    - 31  # State.completename
  criteria:
  - link: AND,
    field: 4,
    searchtype: contains,
    value: '^Rack Mount Chassis$'
    hostname: $1.$33
  hostvars:
    type: $4
    manufacturer: $23
    state: $31
    domain: $33

dell:
  criteria:
  - { link: AND, field: 23, searchtype: contains, value: 'Dell' }

hp:
  criteria:
  - { link: AND, field: 23, searchtype: contains, value: 'HP' }
```

Astuce : Comme le résultat est au format *JSON*, il est possible d'utiliser la commande `jq` [9] pour rechercher dedans.

Pour afficher l'état des contrôleurs raid de nos serveurs HP avec la simple commande *Ansible* :

```
ansible -i /usr/local/bin/ansbile-glpi.py hp -m command -a
"hpacucli ctrl all show status"
serveur1.u-strasbg.fr | CHANGED | rc=0 ”

Smart Array P410 in Slot 3
   Controller Status: OK
   Cache Status: OK
   Battery/Capacitor Status: OK

server2.u-strasbg.fr | CHANGED | rc=0 ”

Smart Array P410i in Slot 0 (Embedded)
   Controller Status: OK
   Cache Status: OK
   Battery/Capacitor Status: OK
…
```

3.3 Autres exemples

Le *provisioning* et l'inventaire dynamique sont assez spécifiques, nous allons vous présenter rapidement quelques exemples de déploiements « applicatifs ».

La première application déployée avec *Ansible* a été notre plateforme d'inventaire. Celle-ci étant obsolète et dans notre périmètre, elle faisait un candidat idéal. Ainsi, avec un fichier d'inventaire correctement renseigné, il est désormais possible de redéployer en un temps record une nouvelle instance de pré-production avec les données de production et d'y tester la mise à jour applicative.

La deuxième application, disposant d'une architecture un peu plus complexe et à être entièrement redéployée, a été notre plateforme de supervision. Nous avions la garantie de l'exacte homogénéité dans la configuration de nos collecteurs de métriques de supervision. Aujourd'hui, il est possible, à très faible coût, de redéployer l'intégralité de notre plateforme de supervision à partir d'un fichier d'inventaire différent.

Exemple d'inventaire pour notre plateforme de supervision :

```
[bdd]
centreon-bdd.mondomaine.fr
[central]
centreon.mondomaine.fr
[poller]
centreon-poller-1.mondomaine.fr
centreon-poller-2.mondomaine.fr
```

Nouvelle plateforme de supervision :

```
[bdd]
centreon-bdd-new.unistra.fr
[central]
centreon-new.di.unistra.fr
[poller]
centreon-poller-1-new.mondomaine.fr
centreon-poller-2-new.mondomaine.fr
```

Ces deux premières expériences validées ont permis de démontrer à nos collègues l'intérêt d'un tel outil. Même les plus réfractaires au changement ont été convaincus.

4 Ansible Tower

Il nous a paru utile de profiter de la surcouche graphique proposée autour d'*Ansible* pour déléguer l'exécution de scripts à des niveaux différents.

Après avoir testé la version libre d'*AWX* à partir des sources disponibles depuis le dépôt *GitHub*, nous avons constaté qu'en utilisation courante, cette application est compliquée à maintenir à jour. Après plusieurs tentatives de mise à jour avortées et face au faible coût de la licence *Ansible Tower* dans le cadre du marché logiciel, nous avons opté pour l'achat de celle-ci.

Après avoir affiné les droits en fonction des périmètres de chacun des départements et des pôles, nous avons ouvert la plateforme à l'ensemble de nos collègues. Nous sommes désormais en mesure, grâce à cette plateforme, de répondre à diverses problématiques :

- le support utilisateur (niveau 1) dispose d'un accès à des procédures automatisées permettant d'effectuer des actions dont le mécanisme est strictement encadré et contrôlé. (Redémarrer une application).
- un gestionnaire de parc peut lancer le processus de réinstallation d'une salle de TP sans maîtriser les outils et les mécanismes sous-jacents.

4.1 Création de la VM à l'ensemble de la *DNum*

Nous avons fait le choix d'avoir des modèles correspondants à une installation de base sans personnalisations pour répondre à trois problématiques :

- le service de machine « infogéré » à destination de nos partenaires doit refléter l'installation standard d'un système d'exploitation et ne doit pas être intégré au SI de la *DNum ;*
- l'hébergement de plusieurs plateformes de virtualisation et le maintien de l'ensemble de ces images à jour ;
- le support d'un nombre de systèmes d'exploitation conséquent (Ubuntu 14/16/18, CentOS 6/7 et RedHat 6/7).

La *DNum* gère un parc de serveurs hétérogènes très important (environ 1250 VM). Après avoir écrit l'ensemble des rôles *Ansible* nécessaires au déploiement de base d'un serveur, nous avons très vite convergé vers leur intégration dans *Ansible Tower*.

La création d'une VM intégrée dans le SI, opération précédemment longue et fastidieuse, prend désormais moins de 10 minutes. Le processus le plus long consiste à attendre la mise à jour des DNS et de mettre à jour le système d'exploitation.

L'utilisateur, après s'être authentifié, remplit le formulaire suivant :

Figure 1: Formulaire de création de VM

Une fois les informations saisies, le processus de déploiement s'exécute et quand l'opération est terminée, un mail est envoyé au demandeur avec toutes les informations nécessaires.

4.2 Scripts d'exploitation

Nous souhaitons relancer le processus *Tomcat* qui héberge l'application *ScenariServer* lorsque celui ne répond plus. Nous avons créé un playbook qui s'occupe de cela. Le responsable de l'application *ScenariServer*, qui n'est pas informaticien, est en mesure de redémarrer son application sans faire appel aux exploitants des serveurs. Il doit simplement se connecter à *Ansible Tower* et démarrer le modèle de job en cliquant sur la « fusée ».

Figure 2: Redémarrage de l'application Scenari

Voici un autre exemple que nous avons mis en œuvre à la *DNum*. Pour des raisons que nous ne maîtrisons pas, il arrive que des services fonctionnant sous Windows s'arrêtent. Dans ce cas-là, uniquement les personnes habilitées sont en capacité de redémarrer le service en panne.

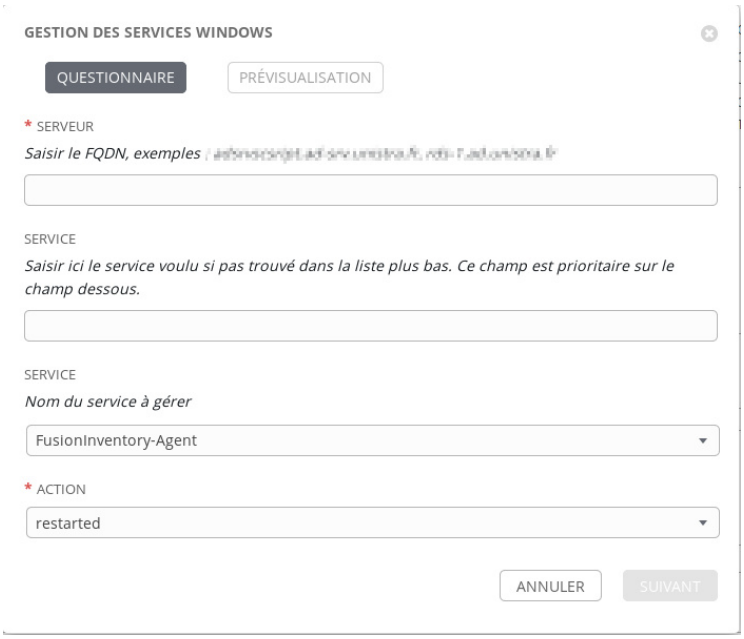

Figure 3: Redémarrage d'un service Windows

Afin de pouvoir déléguer cette possibilité, sans pour autant donner plus d'autorisations, nous avons créé un modèle dans *Tower* qui permet à n'importe qui disposant d'habilitations de redémarrer un service en panne.

Cela permet de les déléguer simplement.

Ces exemples illustrent, très succinctement, les possibilités de délégations offertes par *Ansible Tower*.

Actuellement, nous expérimentons l'intégration continue offerte par notre plateforme *GitLab*, qui pour chaque « commit » passant les tests avec succès, appelle une URL externe (un job *Tower*) qui s'occupe de déployer directement en pré-production la nouvelle fonctionnalité. Nous pouvons enfin commencer à rêver et passer automatiquement en production et s'intégrer pleinement dans la démarche de déploiement continue…

5 Conclusion

Nous avons fait le choix de rationaliser nos outils existants de gestion de configuration (*Chef* et *Pydiploy*) vers un outil unique : *Ansible*. Après deux ans d'utilisation, l'équipe projet est entièrement convaincue de l'outil. La démarche mise en œuvre à la *DNum* ainsi que la surcouche graphique nous ont permis de convaincre les derniers réfractaires. Ainsi, chacun s'approprie l'outil en fonction des missions qu'il doit accomplir. La possibilité de déléguer des tâches nous a permis de gagner en productivité, en sécurité et en qualité dans le déploiement de nos infrastructures.

Bibliographie

[1] https://techcrunch.com/2019/09/23/programmer-who-took-down-open-source-pieces-over-chef-ice-contract-responds/

[2] https://conf-ng.jres.org/2013/planning.html#article_166

[3] https://github.com/unistra/pydiploy

[4] https://github.com/unistra/ansible-modules-opennebula

[5] https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable

[6] https://access.redhat.com/security/vulnerabilities/tcpsack

[7] https://docs.ansible.com/ansible/latest/plugins/inventory/script.html

[8] https://github.com/unistra/ansible-inventory-glpi

[9] https://stedolan.github.io/jq/