I. Infrastructure as Code

Le pourcentage des activités opérationnelles manuelles en data center va tomber en-dessous des 50% d'ici 2021, contre 80% seulement aujourd'hui. (Gartner Magic Quadrant for Data Center Networking 2018)

The CLI Is Dead; the API Is Cool
(Gartner Magic Quadrant for Data Center Networking 2017)

1. Principes Iac

Cette section développe les principes de base liés au concept IaC, Infrastructure as Code :

- Définition de l'Infrastructure en tant que Code (IaC)
- IaC et DevOps
- Modèles et méthodes de communication des outils IaC
- Type d'approches des outils IaC
- Évaluation des outils IaC

1.1. Définition de l'Infrastructure en tant que Code (IaC)

"L'Infrastructure en tant que Code (IaC) est un type d'infrastructure informatique que les équipes d'exploitation peuvent automatiquement gérer et approvisionner via du code, plutôt que d'utiliser un processus manuel ou interactif. L'infrastructure en tant que code est parfois appelée infrastructure programmable."[1]

[1] Paradigme Infrastructure as Code : "*Infrastructure as Code (IAC) is a type of IT infrastructure that operations teams can automatically manage and provision through code, rather than using a manual process. Infrastructure as Code is sometimes referred to as programmable infrastructure.*" (What is Infrastructure as Code (IAC)? - Definition from WhatIs.com)

IaC est donc une approche de conception et de gestion des infrastructures et des services qu'elles suportent. Alors que cette approche IaC est souvent intégrée à la gestion des ressources dans le *nuage* (ce qui n'est pas exclusif) et qu'elle permet la gestion de services *IaaS* (Infrastructure as a Service), ces termes ne doivent pas être confondus.

L'IaC suppose que l'infrastructure soit écrite dans des fichiers sous forme de code et qu'ils soient maintenus avec un logiciel de contrôle de sources (SCM) tel que Git. Dans ce cadre, ce code pourrait l'objet de n'importe quelle pratique **DevOps** comme du test et de l'intégration avant d'être mis en production comme élément d'infrastructure dans un chaînage DevOps plus large.

On imagine aussi une maintenance **automatisée** des infrastructures ainsi déployées avec des capacités à atteindre un "état", une "intention".

L'IaC se fonde aussi sur l'usage d'outils spécifiques qui se différencient en termes de fonctionnement, d'objectif (de départ), de capacités ou de support. On trouvera donc dans une infrastructure IaC des outils d'automation (continue) comme *Ansible*, mais aussi d'autres très populaires comme Chef, Puppet ou encore Terraform. Beaucoup de ces logiciels sont disponibles sous licence Open Source. Les outils d'automatisation de l'infrastructure sont souvent inclus comme composants d'une chaîne d'outils DevOps comme on l'aura compris.

1.2. IaC et DevOps

L'IaC peut être un élément clé de la mise en oeuvre des meilleures pratiques dans DevOps : intégration des équipes, automatisation, collaboration, reporting, réduction de la complexité, performance/ rentabilité, ...

- Les développeurs s'impliquent davantage dans la définition de la configuration et les équipes d'exploitation s'impliquent plus tôt dans le processus de développement.
- Les outils IaC apportent une visibilité sur l'état et la configuration des serveurs et, en conséquence, fournissent la visibilité aux utilisateurs de l'entreprise en vue de réunir les équipes pour maximiser leurs efforts.
- L'automatisation en général a pour but de rendre les processus manuels plus efficaces et productifs, tout en évitant les risques de confusion et les erreurs.
- L'IaC permet la création de meilleurs logiciels et applications avec de la flexibilité, moins de temps d'arrêt et une solution globale rentable pour l'entreprise.
- L'IaC est destiné à réduire la complexité qui empêche l'efficacité de la configuration manuelle.

1.3. Modèles et méthodes de communication des outils IaC

On trouvera principalement deux modèles de communication entre la solution IaC et ses cîbles de gestion :

- Modèles **Agentless** versus Agent-Based
- Modèle **Push** versus Pull

Le premier critère *Agentless/Agent-Based* (sans ou avec agent) signifie la contrainte d'un approvisionnement préalbale des cibles avec l'installation d'un logiciel spécifique (l'agent). Bien souvent, en conséquence des protocoles et des mécanismes sous-jacents comme DNS, NTP et TLS sont aussi des péalables qui peuvent être difficile à contrôller au regard de la solution de gestion. On peut alors considérer que l'IaC dépendra toujours d'une autre infrastructure qui n'est pas nécessairement gérée de manière aussi "agile".

Le second critère Push/Pull concerne la direction des communciations entre la solution IaC et ses cîbles. Un modèle Push "pousse" les informations auprès des cîbles alors qu'un modèle Pull demande aux cîbles de la solution de gestion d'aller chercher leurs paramètres.

1.4. Type d'approches des outils IaC

L'Infrastructure as Code connait aussi différents types d'approches [2] :

- **Déclarative** (fonctionnelle)
- **Impérative** (procédurale)
- Intelligent (en connaissance de son environnement)

[2] https://en.wikipedia.org/wiki/Infrastructure*as*Code#Types*of*approaches

Ces approches répondent différemment aux exigences de déploiement et de gestion des infgrastructures.

Déclarative	**Impérative**	**Intelligente**
Quoi ?	*Comment ?*	*Pourquoi ?*
Ce que la configuration devrait être.	*Comment* l'infrastructure devrait être changée.	Pourquoi l'infrastructure doit atteindre un certain état compte tenu de toutes les relations et dépendances
Cette approche définit un état désiré et le système fait le nécessaire pour atteindre cet état	Cette approche définit des commandes spécifiques à exécuter pour obtenir l'état désiré	Cette approche détermine le bon état désiré avant de faire le nécessaire pour l'atteindre sans impacter les applications co-dépendantes

1.5. Évaluation des outils IaC

On peut donc évaluer les outils IaC en fonction de différents crièteres :

- Agentless ou Agent-Based
- Push ou Pull
- Approche déclarative, impérative ou basée contexte

Dans les points suivants nous verrons d'autres critères de choix :

- Selon son objectif
- Selon sa popularité
- Selon la courbe d'apprentissage
- Selon le public visé

2. Objectifs Outils Iac

Il n'y aurait pas un seul outil IaC qui se suffirait à lui-même. En effet, si certains peuvent tout faire, ils ne sont pas nécessairement les plus appropriés pour certaines tâches. De plus, chaque solution a une histoire avec un point de départ qui correspondait à des besoins spécifiques dans lesquels elle pouvait exceller. Ce n'est que dans l'évolution des fonctionnalités, avec le temps et les besoins du marché que ces produits peuvent s'imposer à une époque puis diminuer en popularité ... C'est comme cela que l'on peut catégoriser les outils IaC selon un point de départ, une fonction originale dans lesquels ils sont les meileurs alors que l'on pourrait probablement les inclure dans d'autres catégories.

Voici la nomenclature proposée :

- Configuration Management tool
- Deployment tool
- Orchestration of Deployment tool
- Provisioning tool

2.1. Configuration Management tool

En génie logiciel, la gestion de la configuration logicielle (SCM ou S / W CM) consiste à suivre et à contrôler les modifications apportées au logiciel, dans le cadre plus large du domaine multidisciplinaire de la gestion de la configuration. Les pratiques SCM incluent le contrôle des révisions et l'établissement de "baselines". Si quelque chose ne fonctionne pas, SCM peut déterminer ce qui a été changé et qui l'a modifié. Si une configuration fonctionne correctement, SCM peut déterminer comment le répliquer sur de nombreux hôtes. [3]

---

[3] Software configuration management

2.2. Deployment tool

Le déploiement de logiciels est l'ensemble des activités qui permettent de disposer d'un système logiciel. Le processus de déploiement comprend plusieurs activités interdépendantes avec des transitions possibles entre elles. Ces activités peuvent avoir lieu du côté du producteur ou du consommateur ou des deux. Étant donné que chaque système logiciel est unique, les procédures ou les processus précis au sein de chaque activité peuvent difficilement être définis. Par conséquent, le "déploiement" doit être interprété comme un processus général qui doit être personnalisé en fonction d'exigences ou de caractéristiques spécifiques. [4]

---

[4] Software deployment

2.3. Orchestration of Deployment tool

L'orchestration décrit le processus automatique d'organisation, de coordination, et de gestion de systèmes informatiques complexes, de middleware et de services.[5]

[5] Orchestration informatique

2.4. Provisioning tool

Le "provisionnement" - calque français du mot provisioning, mot anglais signifiant l'approvisionnement, est un terme utilisé dans le monde de l'informatique désignant l'allocation automatique de ressources. Les outils de *provisioning* sont des outils de gestion de configuration (ou de « gestion de paramétrage ») permettant d'installer et de configurer des logiciels à distance (télédistribution), ou encore d'allouer de l'espace disque, de la puissance ou de la mémoire. Dans le monde des télécommunications, le provisioning consiste à adapter un service aux besoins d'un client. Dans certains cas, l'utilisateur peut même effectuer lui-même certaines opérations : il s'agit alors d'auto-provisionnement, en anglais « self-provisioning ». Au sens large, le provisioning est l'affectation plus ou moins automatisée de ressources à un utilisateur (poste de travail, téléphonie). [6]

---

[6] Provisionnement

3. Marchés des Outils Iac

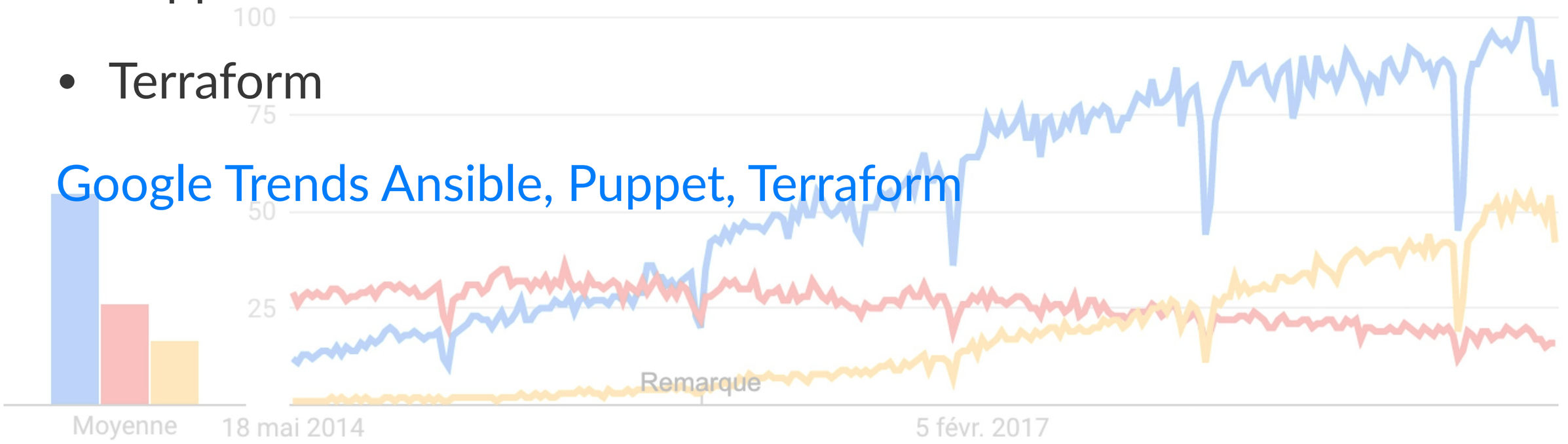
Homogénéité/hétérogénité des outils, adoption du marché, courbe d'apprentissage, public visé ... sont d'autres critères de choix d'une solution Infrastruicture as Code et pourrrait faire son succès. D'avance, un produit comme Ansible rencontre pas mal de ces critères.

3.1. Google Trends Ansible, Puppet, Terraform

Google Trends :

- Ansible
- Puppet
- Terraform

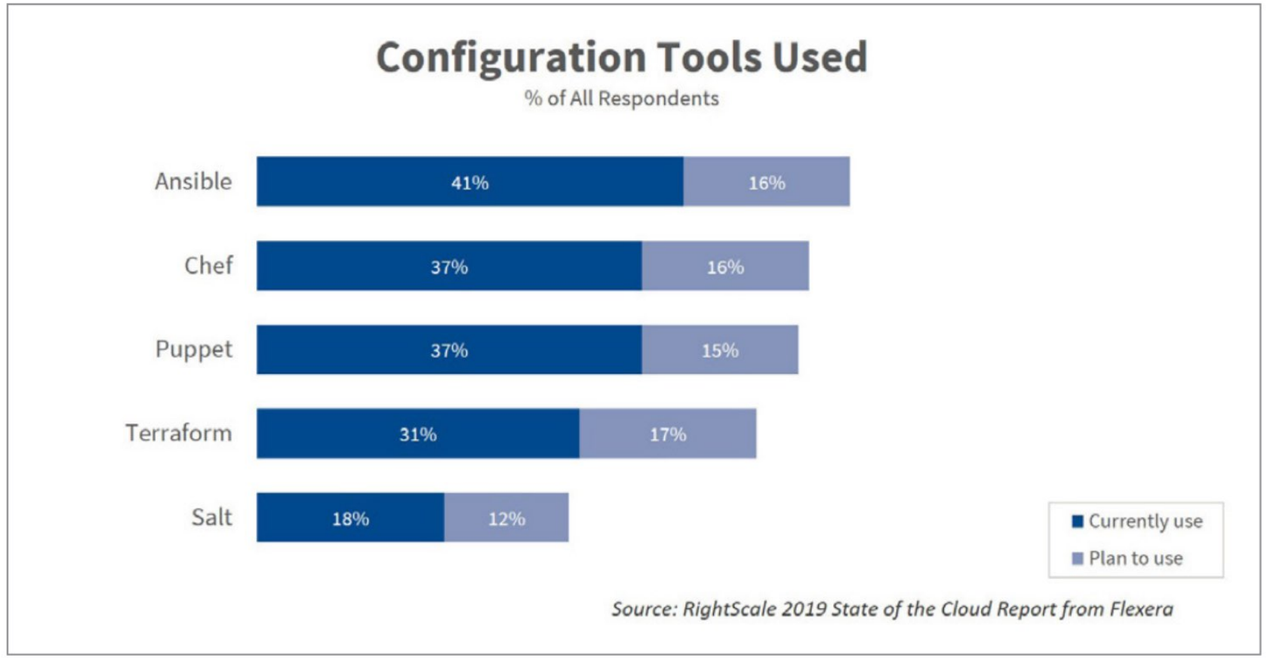
Google Trends Ansible, Puppet, Terraform

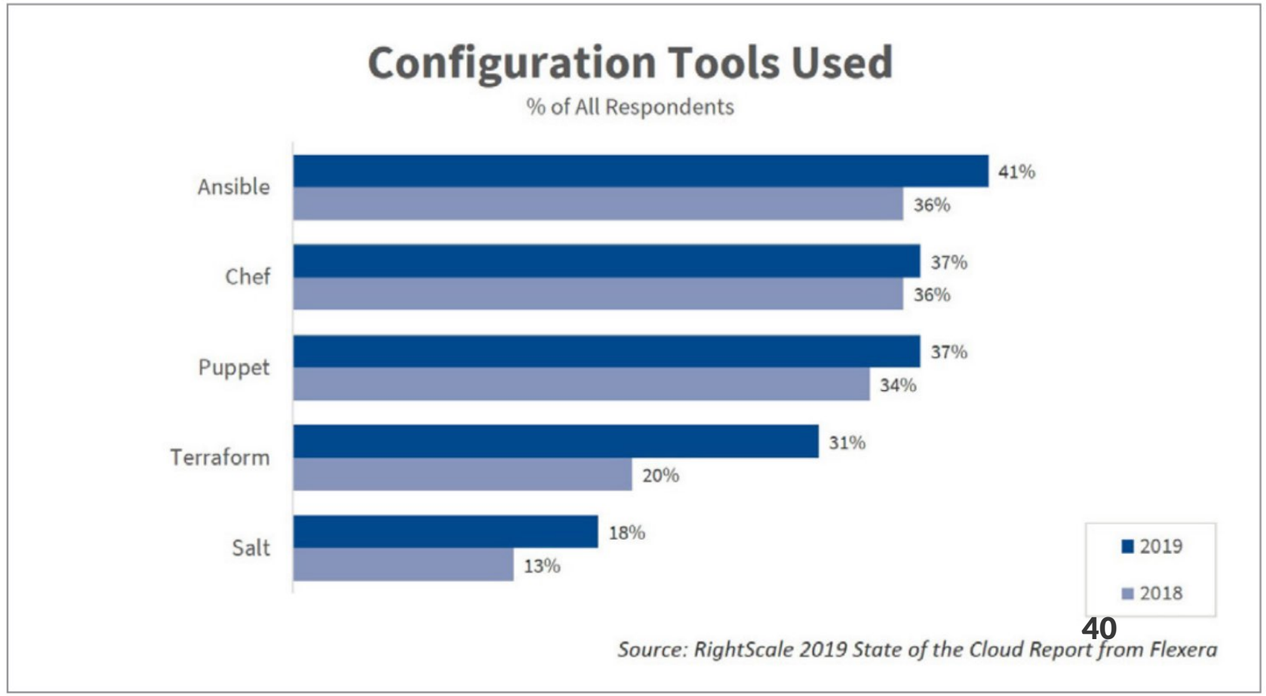


3.2. Adoption d'Ansible dans la gestion du nuage

- Dans le cadre des technologies en nuage, RightScale a mené une enquête sur leur usage.
- Ansible prend la première place parmi les outils de configuration utilisés dans le nuage.
- Parmi tous les répondants, Ansible est à 41 % d'adoption, suivi de Chef et Puppet à 37 % d'adoption.
- Ansible dispose d'un taux d'adoption encore plus élevé parmi les entreprises (53 %), tout comme Chef et Puppet, à égalité avec 50 % d'adoption chacun.
- Ansible a été adopté par 26 % des PME en 2018 suivi par Terraform et Chef.
- Terraform affiche la plus forte croissance depuis l'an dernier, soit une hausse de 55 %, passant de 20 à 31 % d'adoption.

Source : Copyright © 2019, RightScale 2019 State of the Cloud-Report from Flexera.





3.3. Comparatif des principaux outils IaC

La page Wikipedia EN intitulée Comparison of open-source configuration management software cite bon nombre d'outils de "configuration management" open source parmi Ansible.

Tool	Released by	Method	Approach	Written in
Ansible Tower	RedHat	Push	Declarative and imperative	Python
CFEngine	CFEngine	Pull	Declarative	-
Chef	Chef	Pull	Imperative	Ruby
Otter	Inedo	Push	Declarative and imperative	-
Puppet	Puppet	Pull	Declarative	Ruby
SaltStack	SaltStack	Push and Pull	Declarative and imperative	Python
Terraform	HashiCorp	Push	Declarative	Go

4. Présentation des Outils IaC

Enfin, on terminera ce chapitre sur l'Infastructure as Code en présentant quelques produits IaC bien connus :

- Ansible (Red Hat)
- Chef
- SaltStack
- CFEngine
- Puppet Labs
- Terraform (HashiCorp)

4.1. Ansible

Ansible --> *Playbooks* : Ansible combine le déploiement multi-noeuds, l'exécution de tâches ad hoc et la gestion de la configuration en un seul logiciel. Il gère les noeuds avec SSH et requiert l'installation de python (2.6+ ou 3.5+). Les modules fonctionnent avec JSON et les sorties standard. Ils peuvent être écrits dans n'importe quel langage. Ansible utilise YAML pour exprimer des descriptions réutilisables de systèmes.

Ansible peut être considéré comme un :

- "configuration management tool"
- "deployment tool"
- "orchestration of deployment tool"
- "provisioning tool"

4.2. Chef

Chef --> *Cookbook* : Chef est un outil de gestion de configuration écrit en Erlang qui utilise un DSL Ruby pur pour la rédaction de "recettes" de configuration. Ces recettes contiennent des ressources qui doivent être placées dans l'état déclaré. Chef peut être utilisé comme un outil client-serveur ou utilisé en mode "solo".

4.3. SaltStack

SaltStack --> *Salt State* : Salt a commencé comme un outil de gestion de serveur à distance. Au fur et à mesure de son utilisation, il a acquis un certain nombre de fonctionnalités étendues, notamment un mécanisme plus complet de configuration de l'hôte. Elle est une fonctionnalité relativement nouvelle facilitée par le composant Salt States.

4.4. CFEngine

CFEngine --> *Policy* : CFEngine est un système d'agents légers. Il gère la configuration d'un grand nombre d'ordinateurs à l'aide du paradigme client-serveur ou autonome. Tout état client différent de la description de la politique est rétabli à l'état souhaité. L'état de configuration est spécifié via un langage déclaratif. Le paradigme de CFEngine est la «immunologie informatique» convergente.

4.5. Puppet Labs

Puppet Labs --> *Manifest* : Puppet offre un langage déclaratif personnalisé pour décrire la configuration du système, distribué à l'aide du paradigme client-serveur (utilisant le protocole XML-RPC dans les versions antérieures, avec un commutateur récent pour REST) et une bibliothèque pour réaliser la configuration. La couche d'abstraction des ressources permet aux administrateurs de décrire la configuration en termes de haut niveau, tels que les utilisateurs, les services et les packages. Puppet s'assurera alors que l'état du serveur correspond à la description.

4.6. Terraform

Cette partie est en cours de développement.

5. Notes

Quelques éléments à garder en mémoire qui trouvent difficielement leur place.

- Immutabilité de l'infrastructure
- Intent Based Network (Cisco)

https://iac.goffinet.org/infrastructure-as-code/infrastructure-as-code-introduction/