Chef: Création et gestion des cookbooks

Table of Contents

Les cookbooks sont les unités de configuration qui permettent de configurer et d'effectuer des tâches spécifiques dans Chef sur les nœuds distants, cet article, décrit :

- les bases de la création d'un cookbook Chef.
- leur utilisation dans chef
- la mise en oeuvre d'un répertoire local de cookbooks

Bases de la création d'un cookbook

Concepts de base

Les Cookbooks

Les cookbooks constituent l'unité fondamentale des détails de configuration et de politique que Chef utilise pour amener un nœud dans un état spécifique. Cela signifie simplement que Chef utilise des cookbooks pour effectuer le travail et s'assurer que les choses sont comme il se doit sur le nœud.

Les cookbooks sont généralement utilisés pour gérer un service, une application ou une fonctionnalité spécifique. Par exemple, un livre de recettes peut être créé pour utiliser NTP pour définir et synchroniser l'heure du nœud avec un serveur spécifique. Il peut installer et configurer une application de base de données. Les cookbooks sont essentiellement des paquets pour les choix d'infrastructure.

Les recipes et les politiques décrites dans le cookbook peuvent être assignées à des nœuds dans le cadre de la «liste d'exécution» du nœud. Une liste d'exécution est une liste séquentielle de recettes et de rôles qui sont exécutés sur un nœud par chef-client afin de mettre le nœud en conformité avec la politique définie.

Les cookbooks sont organisés dans une structure de répertoires complètement autonome. Il existe de nombreux répertoires et fichiers différents utilisés à différentes fins. Passons en revue certains des plus importants maintenant.

Les recettes (Recipes)

Une recette est le cheval de trait principal du livre de recettes. Un livre de recettes peut contenir plus d'une recette ou dépendre de recettes extérieures. Les recettes sont utilisées pour déclarer l'état de différentes ressources.

Les ressources du chef décrivent une partie du système et son état souhaité. Par exemple, une ressource pourrait dire “le paquet x devrait être installé”. Une autre ressource peut dire “le service x devrait être en cours d'exécution”.

Une recette est une liste de ressources liées qui indiquent à Chef à quoi le système devrait ressembler s'il implémente la recette. Lorsque Chef exécute la recette, il vérifie la conformité de chaque ressource à l'état déclaré. Si le système correspond, il passe à la ressource suivante, sinon, il tente de déplacer la ressource dans l'état donné.

Les ressources peuvent être de différents types :

- package: Utilisé pour gérer les paquets sur un noeud
- service: utilisé pour gérer les services sur un noeud
- user: Gérer les utilisateurs sur le noeud
- group: Gérer les groupes
- template: Gérer les fichiers avec des modèles ruby incorporés
- cookbook_file: Transfère les fichiers du sous-répertoire files du cookbook vers un emplacement sur le noeud
- file: Gérer le contenu d'un fichier sur un noeud
- directory: Gérer les répertoires sur le noeud
- execute: Exécute une commande sur le noeud
- cron: Editer un fichier cron existant sur le noeud

Les attributs

Les attributs dans Chef sont essentiellement des paramètres. On peut les considérer comme de simples paires clé-valeur pour tout ce qu'ont veut utiliser dans le cookbook.

Il existe plusieurs types d'attributs pouvant être appliqués, chacun ayant un niveau de priorité différent par rapport aux paramètres définitifs utilisés par un nœud. Au niveau du cookbook, on définit généralement les attributs par défaut du service ou du système qu'on configure. Ceux-ci peuvent être remplacées plus tard par des valeurs plus spécifiques pour un nœud spécifique.

Lors de la création d'un cookbook, il est possible de définir des attributs génériques dans le sous-répertoire attributes du cookbook qui seront utilisés dans d'autres parties.

Les Files

Le sous-répertoire files du cookbook contient tous les fichiers statiques que l'on va placer sur les noeuds qui utilisent le cookbok.

Par exemple, tous les fichiers de configuration simples que l'on est pas susceptible de modifier peuvent être placés, dans leur intégralité, dans le sous-répertoire files. Une recette peut ensuite déclarer une ressource qui déplace les fichiers de ce répertoire dans leur emplacement final sur le nœud.

Les Templates

Les templates sont similaires aux fichiers, mais ils ne sont pas statiques. Les fichiers templates se terminent par l'extension .erb, ce sont des scriptes Ruby.

Ils sont principalement utilisés pour substituer des valeurs d'attribut dans le fichier pour créer la version finale du fichier qui sera placée sur le noeud.

Par exemple, si un attribut qui définit le port par défaut pour un service, le fichier template peut être appelé pour insérer l'attribut dans le fichier où le port est déclaré. En utilisant cette technique, on peut facilement créer des fichiers de configuration, tout en gardant les variables réelles que l'on souhaite changer ailleurs.

Le fichier Metadata.rb

Le fichier metadata.rb est utilisé pour gérer les métadonnées d'un package, comme le nom du paquet, une description, etc.

Il inclut également des informations sur les dépendances, par exemple on peut spécifier les cookbooks dont un cookbook a besoin pour fonctionner. Cela permettra au serveur Chef de construire correctement la liste des dépendances pour les noeuds et de s'assurer que toutes les pièces sont transférées correctement.

Création d'un cookbook

Ce cookbook très simple, installe et configure le serveur web Nginx.

Avant de commencer

Si ce n'est déjà fait installer chefdk et créer un répertoire sur la station:

```
$ chef generate repo chef-repo
```

Initialiser la structure

Créer un cookbook dans le répertoire du poste de travail:

```
$ chef generate cookbook nginx
```

```
  Generating cookbook nginx
  - Ensuring correct cookbook file content
  - Ensuring delivery configuration
  - Ensuring correct delivery build cookbook content
  Your cookbook is ready. Type `cd nginx` to enter it.
  There are several commands you can run to get started locally developing
and testing your cookbook.
  Type `delivery local --help` to see a full list.
  Why not start by writing a test? Tests for the default recipe are stored
at:
  test/integration/default/default_test.rb
  If you'd prefer to dive right in, the default recipe can be found at:
  recipes/default.rb
```

chef a construit une structure simple dans le répertoire des cookbooks pour notre nouveau cookbook dont on peut avoir un aperçu avec la commande tree.

```
nginx
├── Berksfile
├── chefignore
├── LICENSE
├── metadata.rb
├── README.md
├── recipes
│   └── default.rb
├── spec
│   ├── spec_helper.rb
│   └── unit
│       └── recipes
│           └── default_spec.rb
```

```
└── test
    └── integration
        └── default
            └── default_test.rb
```

Comme on peut le voir, cela a créé une structure de dossiers et de fichiers que l'on utilisera pour construire un cookbook.

Créer une recette (recipe)

Dans le sous-répertoire des reccettes il y a déjà un fichier appelé default.rb :

C'est la recette qui sera exécutée lorsqu'on fait référence à la recette “nginx”. C'est ici qu'il faut ajouter le code.

```
$ vi cookbooks/nginx/recipe/default.rb

#
# Cookbook:: nginx
# Recipe:: default
#
# Copyright:: 2018, The Authors, All Rights Reserved.
```

La seule chose qu'il y a actuellement dans ce fichier est un en-tête de commentaire.

Il faut planifier les opérations à conduire pour que le serveur web Nginx puisse fonctionner en configurant les “ressources”. Les ressources ne décrivent pas comment faire quelque chose; elles décrivent simplement à quoi devrait ressembler une partie du système lorsqu'il est complet.

Tout d'abord, il faut s'assurer que le logiciel est installé, en créant une ressource “package” en premier.

```
package 'nginx' do
  action :install
end
```

Ce petit morceau de code définit une ressource de paquetage pour Nginx. La première ligne commence par le type de ressource (package) et le nom de la ressource ('nginx'). Le reste est un groupe d'actions et de paramètres qui déclarent ce que l'on veut faire avec cette ressource.

Dans cette ressource, l'action: install indique à Chef que la ressource doit être installée. Le noeud qui exécute cette recette vérifiera que Nginx est installé. Si c'est le cas, il va poursuivre dans la liste des choses à faire. Sinon, il va installer le programme en utilisant les méthodes disponibles sur le système client.

Après avoir installé le service, il faut probablement ajuster son état actuel sur le nœud. Par défaut, Nginx ne démarre pas après l'installation, on va donc le démarrer :

```
service 'nginx' do
  action [ :enable, :start ]
```

```
end
```

Ici, la ressource du type “service” déclare que pour le composant de service Nginx (la partie qui permet de gérer le serveur avec init ou upstart), on veut démarrer le service dès maintenant, et lui permettre également de démarrer automatiquement lorsque la machine est redémarrée.

La ressource finale que l'on va déclarer est le fichier index.html a insérer dans le serveur. Comme il ne s'agit que d'un simple fichier qu'on ne modifiera pas, on peut simplement déclarer l'emplacement où on veut insérer le fichier et lui dire où récupérer ce fichier dans le cookbook:

```
cookbook_file "/usr/share/nginx/www/index.html" do
   source "index.html"
   mode "0644"
end
```

On utilise le type de ressource “cookbook_file” pour indiquer à Chef que ce fichier est disponible dans le cookbokk lui-même et peut être transféré tel quel à l'emplacement. Dans l'exemple, on transfert un fichier dans la racine du document de Nginx.Par défaut chef recherchera ce fichier dans le sous-répertoire “files/default” du cookbook.

Créer le fichier index.html

Comme précisé dans la recette ci-dessus, une ressource “cookbook_file” doit placer un fichier appelé “index.html” dans la racine du document sur le nœud. On doit créer ce fichier dans le sous-répertoire “files/default”.

```
$ vi  nginx/files/default/index.html
```

Ce fichier sera simplement un document HTML très simple destiné à démontrer que les ressources ont fonctionné comme attendu.

```
<html>
   <head>
     <title>Hello there</title>
   </head>
   <body>
    <h1>This is a test</h1>
     <p>Please work!</p>
   </body>
</html>
```

Utilisation d'u dépôt local pour les paquets

Avant d'aller plus loin, il faut résoudre préventivement un petit problème. Lorsque le noeud va exécuter le cookbook tel qu'il est actuellement, il y a de fortes chances qu'il échoue.

En effet, il tentera d'installer Nginx depuis les dépôts officiels, ce qui implique d'avoir les accès réseau nécessaires.

Pour résoudre ce problème, on peut :

- Stocker les RPMs dans le cookbook a le mérite simple mais cela peut trés vite consommé de l'espace, selon la taille de celui-ci.
- Utiliser une copie dans un entrepôt local ou sur un autre serveur.

Un avantage supplémentaire de ceci est que l'on peut contrôler exactement ce qui est installé sur les serveurs.

Plusieurs méthodes permettent de définir et utiliser des paquets dans un dépôt local:

Utilisation des fichiers

Installation de plusieurs paquets

```
['package1.rpm','package2.rpm'].each do |p| # A partir d'un tableau de
paquets, pourrait être un attribut comme noeud
node['my_namespace']['packages']
     package p do                           # traitement des paquets en
séquence
         source "/tmp/#{p}"                 # concaténation chemin/nom du
paquet
         action :nothing                    # ne rien faire sinon définir la
ressource
     end
     cookbook_file "/tmp/#{p}" do           # traitement en séquence des
fichiers
         source p                           # définir la ressource
         action :create
         notifies :install, "package[/tmp/#{p}]", :immediately
     end
 end
```

Le :immediately permet de lancer l'installation du paquet dès que le fichier a été placé, s'il y a des dépendances, il faut gérer l'ordre des paquets dans le tableau.

Utilisation d'une source externe

```
remote_file "#{Chef::Config[:file_cache_path]}/nom_du_paque.rpm" do
    source "https://foo.bar.baz/checkmk/nom_du_paquet.rpm";
    not_if "rpm -qa | grep -q '^nom_du_paque'"
    notifies :install, "rpm_package[mon_paquet]", :immediately
end

rpm_package "mon_paquet" do
    source "#{Chef::Config[:file_cache_path]}/nom_du_paquet.rpm"
    only_if
{::File.exists?("#{Chef::Config[:file_cache_path]}/nom_du_paque.rpm")}
```

```
    action :nothing
end
```

Utilisation d'un cookbook pour définir la ressource locale

Cette méthode nécessite de gérer les dépendances

Créer un cookbook simple

Le seul but de ce cookbook est de pouvoir utiliser les rpms du dépôt local.

```
$ chef generate cookbook yum
```

Cela créera le même type de structure de répertoires que lors de la création du cookbokk Nginx.

Créer un fichier .erb

Dans le répertoire templates, créer le fichier custom.repo.erb (format de modèle Chef) avec le contenu suivant:

```
[custom]
name=MyPackages
baseurl=http://chef.vb:8088/yum/Redhat/6/x86_64
enabled=1
gpgcheck=0
```

Note: le paramètre baseurl pointe vers un référentiel yum

Editer la recette par défaut

```
$ vi cookbooks/yum/recipe/default.rb
```

Ajouter ce qui suit:

```
template "custom.repo" do
    path "/etc/yum.repos.d/custom.repo"
    source "custom.repo.erb"
end

execute "installjdk" do
    command "yum -y --disablerepo='*' --enablerepo='bmchef' install
jdk.x86_64"
end
```

Gérer les dépendances

Maintenant que le cookbook est défini il faut s'assurer de son exécution avant le cookbook Nginx :

- soit en ajoutant ce cookbook à la liste d'exécution du nœud avant le cookbook Nginx (mais cela oblige à l'ajouter sur tous les nœuds configurés avec Nginx)
- soit en attachant cette dépendance dans le cookbook Nginx lui-même (c'est probablement la meilleure option)

Ajuster le cookbook Nginx

Intégrer la dépendance:

```
$ vi ~/chef-repo/cookbooks/nginx/recipes/default.rb
```

En haut de ce cookbook, avant les autres ressources que l'on a défini, ajouter :

```
include_recipe "yum"

package 'nginx' do
  action :install
end

service 'nginx' do
  action [ :enable, :start ]
end

cookbook_file "/usr/share/nginx/www/index.html" do
  source "index.html"
  mode "0644"
end
```

Enregistrer et fermer le fichier.

Modifier le fichier metadata.rb.

Ce fichier est vérifié lorsque le serveur Chef envoie la liste d'exécution au noeud, pour voir quelles autres recettes doivent être ajoutées à la liste d'exécution.

Au bas du fichier, ajouter les dépendances :

```
$ vi /chef-repo/cookbooks/nginx/metadata.rb
```

```
  name             'nginx'
  maintainer       'YOUR_COMPANY_NAME'
  maintainer_email 'YOUR_EMAIL'
  license          'All rights reserved'
```

```
  description      'Installs/Configures nginx'
  long_description IO.read(File.join(File.dirname(__FILE__), 'README.md'))
  version          '0.1.0'
```

```
  depends "yum"
```

Le cookbook Nginx s'appuie maintenant sur le cookbook yum pour ajouter l'entrepôt local de paquet.

Utilisation de yum_repository

On peut également ajouter un référentiel en mettant ce qui suit dans la recette

```
yum_repository "custom" do
  description "MyRepo"
  url "http://chef.vb:8088/yum/Redhat/6/x86_64"
  repo_name "custom"
  action :add
end

yum_package "mypackage" do
  action :install
  flush_cache [:before]
end
```

Utilisation des cookbooks

Déploiement des cookboks sur le serveur

On peut déployer les cookbooks sur le serveur chef:

Soit individuellement en tapant:

```
$ knife cookbook upload apt
$ knife cookbook upload nginx
```

Soit globalement en tapant:

```
$ knife cookbook upload -a
```

De toute façon, les recettes seront téléchargées sur le serveur Chef.

Préparation de la liste de lancement des noeuds

Pour trouver le nom des nœuds disponibles, taper:

```
$ knife node list
```

```
  dgfip-base
```

Pour modifier la liste de lancement des noeuds taper:

```
$ knife node edit name_of_node
```

Par exemple pour le node dgfip base le fichier de lancement ressemble à ceci:

```
$ knife node edit dgfip-base
```

```
  {
    "name": "dgfip-base",
    "chef_environment": "_default",
    "normal": {
      "tags": [
      ]
    },
    "policy_name": null,
    "policy_group": null,
    "run_list": [
      ]
  }
```

Note:/ il faut définir la variable d'environnement EDITOR avant que cela fonctionne en tapant:

```
$ export EDITOR=nom_d'éditeur
```

Il s'agit d'un simple document JSON qui décrit certains aspects du nœud dont le tableau “run_list”, est actuellement vide.

On peut ajouter les cookbooks à ce tableau en utilisant le format suivant:

```
"recipe [nom_de_recette]"
```

Par exemple le même fichier avec la recette nginx devrait ressembler à ceci:

```
 {
  "name": "dgfip-base",
  "chef_environment": "_default",
  "normal": {
    "tags": [

    ]
  },
  "policy_name": null,
  "policy_group": null,
```

```
  "run_list": [
    "recipe[nginx]"
  ]
}
```

Enregistrer et fermer le fichier pour implémenter les nouveaux paramètres.

Exécution des listes de déploiement

Se connecter en SSH dans le noeud et exécuter le logiciel client Chef

```
$ sudo chef-client
```

```
  Démarrage de Chef Client, version 11.8.2
  résoudre des livres de recettes pour la liste de diffusion: ["nginx"]
  Synchronisation des livres de cuisine:
    - apt
    - nginx
  Compiler des livres de cuisine ...
  Convergence de 4 ressources
  Recette: apt :: default
    * exécuter [apt-get update] action exécuter
      - Exécuter apt-get mise à jour
  Recette: nginx :: par défaut
    * package [nginx] action installer (à jour)
    * service [nginx] action active
      - activer le service de service [nginx]
    * service [nginx] début de l'action (à jour)
    * cookbook_file [/usr/share/nginx/www/index.html] action créer (à jour)
  Chef Client terminé, 2 ressources mises à jour
```

Le cookbook apt a été envoyé et exécuté, même s'il ne figurait pas dans la liste de lancement appelée. Chef a résolu les dépendances et modifié la liste des tâches avant de l'exécuter sur le noeud.

Il existe plusieurs méthodes pour s'assurer qu'un livre de recettes ou une recette est exécuté avant un autre. L'ajout d'une dépendance n'est qu'un choix et d'autres méthodes peuvent être préférées.

Vérification du déploiement

Vérifier que cela fonctionne en allant à l'adresse IP ou au nom de domaine du nœud:

```
http://node_domain_or_IP
```