CHEF: Les attributs

Table of Contents

Présentation

Un attribut est un détail spécifique concernant un nœud. Les attributs sont utilisés par le chef-client pour comprendre:

- L'état actuel du noeud
- Quel était l'état du nœud à la fin de la précédente exécution chef-client
- Quel doit être l'état du nœud à la fin de l'exécution actuelle client-chef?

Les attributs sont définis par:

- L'état du noeud lui-même
- Cookbooks (dans les fichiers d'attributs et / ou les recettes)
- Rôles
- Environnements

À chaque exécution chef-client, le chef-client construit la liste d'attributs en utilisant:

- Données sur le noeud recueillies par Ohai
- L'objet nœud qui a été enregistré sur le serveur Chef à la fin de la précédente exécution chef-client
- L'objet de noeud reconstruit de l'exécution chef-client actuelle, après avoir été mis à jour pour les modifications des livres de recettes (fichiers d'attributs et / ou recettes), des rôles et/ou des environnements, et mis à jour pour toute modification de l'état du noeud lui-même.

Une fois que l'objet nœud est reconstruit, tous les attributs sont comparés, puis le nœud est mis à jour en fonction de la priorité des attributs. À la fin de chaque exécution chef-client, l'objet nœud qui définit l'état actuel du nœud est chargé sur le serveur Chef afin qu'il puisse être indexé pour la recherche.

Alors, comment le chef-client détermine-t-il quelle valeur doit être appliquée? Continuez votre lecture pour en savoir plus sur le fonctionnement des attributs, notamment sur les types d'attributs, sur les emplacements où les attributs sont enregistrés et sur la manière dont le chef-client choisit l'attribut à appliquer.

Persistance d'attribut

Au début d'une exécution chef-client, tous les attributs, à l'exception des attributs normaux, sont réinitialisés. Le chef-client les reconstruit en utilisant les attributs automatiques collectés par Ohai au début de l'exécution du chef-client, puis en utilisant les attributs par défaut et de substitution spécifiés dans les livres de recettes ou par les rôles et les environnements. Tous les attributs sont ensuite fusionnés et appliqués au nœud en fonction de la priorité des attributs. À la fin de l'exécution de chef-client, les attributs appliqués au nœud sont enregistrés sur le serveur Chef en tant que partie de l'objet nœud.

Types d'attributs

Le chef-client utilise six types d'attributs pour déterminer la valeur appliquée à un nœud lors de l'exécution du chef-client. En outre, les sources chef-client attribuent des valeurs provenant de cinq emplacements maximum. La combinaison de types d'attributs et de sources permet au chef-client de disposer de 15 valeurs différentes et concurrentes au cours de son exécution:

Type d'attribut	La description
default	Un attribut default est automatiquement réinitialisé au début de chaque exécution chef-client et a la priorité la plus basse sur les attributs. Utilisez les attributs default aussi souvent que possible dans les livres de recettes.
force_default	Utiliser l'attribut force_default pour s'assurer qu'un attribut défini dans un livre de recettes (par un fichier d'attribut ou par une recette) a priorité sur un attribut par default défini par un rôle ou un environnement.

Type d'attribut	La description
normal	Un attribut normal est un paramètre qui persiste dans l'objet nœud. Un attribut normal a une priorité d'attribut plus élevée qu'un attribut default .
override	Un attribut de override est automatiquement réinitialisé au début de chaque exécution chef-client et a une priorité sur les attributs supérieure à celle des attributs default , force_default et normal . Un attribut de override est le plus souvent spécifié dans une recette, mais peut être spécifié dans un fichier d'attribut, pour un rôle et / ou pour un environnement. Un livre de recettes doit être créé de manière à ce qu'il utilise les attributs de override uniquement lorsque cela est nécessaire.
force_override	Utiliser l'attribut force_override pour s'assurer qu'un attribut défini dans un livre de recettes (par un fichier d'attribut ou par une recette) a priorité sur un attribut de override défini par un rôle ou un environnement.
automatic	Un attribut automatic contient des données identifiées par Ohai au début de chaque exécution chef-client. Un attribut automatic ne peut pas être modifié et a toujours la priorité d'attribut la plus élevée.

Sources d'attributs

Les attributs sont fournis au chef-client depuis les emplacements suivants:

- Nœuds (collectés par Ohai au début de chaque parcours client-chef)
- Fichiers d'attributs (dans les livres de cuisine)
- Recettes (dans les livres de cuisine)
- Environnements
- Rôles

- De nombreux attributs sont conservés dans chef-repo pour les environnements, les rôles et les livres de recettes (fichiers d'attributs et recettes).
- De nombreux attributs sont collectés par Ohai sur chaque noeud individuel au début de chaque exécution chef-client.
- Les attributs conservés dans le chef-repo sont téléchargés sur le serveur Chef à partir du poste de travail, périodiquement.
- Le chef-client extrait l'objet noeud du serveur Chef (qui contient les données d'attribut de la précédente exécution chef-client), après quoi tous les attributs (sauf les attributs normal sont réinitialisés).
- Le chef-client mettra à jour les livres de recettes sur le nœud (si nécessaire), ce qui met à jour les attributs contenus dans les fichiers d'attributs et les recettes.
- Le chef-client mettra à jour les données de rôle et d'environnement (si nécessaire)
- Le chef-client reconstruira la liste d'attributs et appliquera la priorité d'attribut lors de la configuration du noeud.
- Le chef-client pousse l'objet noeud vers le serveur Chef à la fin de l'exécution du chef-client; l'objet de noeud mis à jour sur le serveur Chef est ensuite indexé pour la recherche et est stocké jusqu'à la prochaine exécution chef-client

Les attributs automatiques (Ohai)

Un attribut automatique est un détail spécifique concernant un nœud, tel qu'une adresse IP, un nom d'hôte, une liste de modules du noyau chargés, etc. Les attributs automatiques sont détectés par Ohai et sont ensuite utilisés par le chef-client pour s'assurer qu'ils sont gérés correctement lors de chaque exécution du chef-client.

Les attributs automatiques les plus couramment utilisés sont:

Attribut	**Description**
node['platform']	La plate-forme sur laquelle un nœud est en cours d'exécution. Cet attribut aide à déterminer quels fournisseurs seront utilisés.
node['platform_version']	La version de la plateforme. Cet attribut aide à déterminer quels fournisseurs seront utilisés.
node['ipaddress']	L'adresse IP d'un noeud. Si le noeud a une route par défaut, il s'agit de l'adresse IPV4 de l'interface. Si le nœud n'a pas d'itinéraire par défaut, la valeur de cet attribut doit être nil . L'adresse IP de la route par défaut est la valeur par défaut recommandée.
node['macaddress']	L'adresse MAC d'un nœud, déterminée par la même interface que celle qui détecte le node['ipaddress'] .
node['fqdn']	Le nom de domaine complet pour un nœud. Ceci est utilisé comme nom d'un nœud, sauf indication contraire.
node['hostname']	Le nom d'hôte du noeud.
node['domain']	Le domaine pour le noeud.
node['recipes']	Une liste de recettes associées à un nœud (et une partie de la liste d'exécution de ce nœud).
node['roles']	Une liste de rôles associés à un nœud (et une partie de la liste d'exécution de ce nœud).
node['ohai_time']	L'heure à laquelle Ohai a été exécuté pour la dernière fois. Cet attribut n'est pas couramment utilisé dans les recettes, mais il est enregistré sur le serveur Chef et est accessible à l'aide de la sous-commande knife status .

La liste des attributs automatiques collectés par Ohai au début de chaque exécution chef-client varie d'une organisation à l'autre et varie souvent entre les différents types de serveur configurés et les plates-formes sur lesquelles ces serveurs sont exécutés. Tous les attributs collectés par Ohai ne sont pas modifiables par le chef-client. Pour voir quels attributs automatiques sont collectés par Ohai pour un nœud particulier, exécuter la commande suivante:

```
find  /opt/chefdk/embedded/lib/ruby/gems/*/gems/ohai-*/lib -name "*.rb" -
print | xargs grep -R "provides" -h |sed 's/^\s*//g'|sed
"s/\\\"/\'/g"|sort|uniq|grep "\sprovides"
```

Les fichiers d'attributs

Un fichier d'attributs se trouve dans le sous-répertoire d' attributes/ d'un livre de recettes. Lorsqu'un livre de recettes est exécuté sur un nœud, les attributs contenus dans tous les fichiers d'attributs sont évalués dans le contexte de l'objet nœud. Les méthodes de nœud (lorsqu'elles sont présentes) sont utilisées pour définir les valeurs d'attribut sur un nœud. Par exemple, le livre de recettes apache2

contient un fichier d'attribut appelé default.rb , qui contient les attributs suivants:

```
default['apache']['dir']          = '/etc/apache2'
default['apache']['listen_ports'] = [ '80','443' ]
```

L'utilisation de l'objet node (node) est implicite dans l'exemple précédent. L'exemple suivant définit l'objet noeud en tant que partie intégrante de l'attribut:

```
node.default['apache']['dir']          = '/etc/apache2'
node.default['apache']['listen_ports'] = [ '80','443' ]
```

Ordre d'évaluation des attributs

Chef-client évalue les attributs dans l'ordre défini par la liste d'exécution, y compris tous les attributs figurant dans la liste d'exécution en raison des dépendances du livre de recettes.

Utilisation des fichiers d'attributs

Un attribut est un détail spécifique concernant un nœud, tel qu'une adresse IP, un nom d'hôte, une liste des modules du noyau chargés, la ou les versions des langages de programmation disponibles disponibles, etc. Un attribut peut être unique pour un nœud spécifique ou identique pour tous les nœuds de l'organisation. Les attributs sont le plus souvent définis à partir d'un livre de recettes, à l'aide d'un couteau, ou sont récupérés par Ohai de chaque nœud avant chaque exécution chef-client. Tous les attributs sont indexés pour la recherche sur le serveur Chef. Les bons candidats pour les attributs incluent:

- toute abstraction multiplateforme pour une application, telle que le chemin d'accès à un fichier de configuration
- valeurs par défaut des paramètres ajustables, telles que la quantité de mémoire affectée à un processus ou le nombre de travailleurs à générer
- tout ce qui peut devoir être conservé dans les données de nœud entre les exécutions chef-client

En général, la priorité des attributs est définie pour permettre aux livres de recettes et aux rôles de définir les valeurs par défaut des attributs, aux attributs normaux de définir les valeurs devant être spécifiques à un nœud et aux attributs de substitution de forcer une certaine valeur, même lorsqu'un nœud possède déjà cette valeur. spécifié.

Une approche consiste à définir des attributs au même niveau de priorité en définissant des attributs dans les fichiers d'attributs d'un livre de recettes, puis en définissant également les mêmes attributs par défaut (mais avec des valeurs différentes) à l'aide d'un rôle. Les attributs définis dans le rôle seront profondément fusionnés par-dessus les attributs du fichier d'attributs, et les attributs définis par le rôle auront priorité sur les attributs spécifiés dans les fichiers d'attributs du livre de recettes.

Une autre approche (beaucoup moins courante) consiste à définir une valeur uniquement si un attribut n'a aucune valeur. Cela peut être fait en utilisant les variantes _unless des méthodes de priorité d'attribut:

- default_unless

- set_unless (normal_unless est un alias de set_unless ; utiliser l'un ou l'autre pour définir un attribut avec une priorité d'attribut normale.) Cette méthode était obsolète dans le client Chef 12.12 et supprimée dans Chef 14. Veuillez utiliser plutôt default_unless ou override_unless .
- override_unless

</WRAP info>Utiliser les variantes _unless avec précaution (et uniquement lorsque cela est nécessaire), car lorsqu'elles sont utilisées, les attributs appliqués aux nœuds peuvent ne pas être synchronisés avec les valeurs des livres de recettes au fur et à mesure de leur mise à jour. Cette approche peut créer des situations où deux nœuds par ailleurs identiques finissent par avoir des configurations légèrement différentes et peut également constituer un défi pour le débogage.</WRAP>

Méthodes de fichier

Utiliser les méthodes suivantes dans le fichier d'attributs pour un livre de recettes ou dans une recette. Ces méthodes correspondent au type d'attribut du même nom:

- override
- default
- normal (ou set , où set est un alias pour normal)
- _unless
- attribute?

attribute? vérifie l'existence d'un attribut afin que le traitement puisse être effectué dans un fichier d'attributs ou une recette, mais uniquement s'il existe un attribut spécifique.

Utiliser attribute?() Dans un fichier d'attributs:

```
if attribute?('ec2')
  # ... set stuff related to EC2
end
```

Utiliser attribute?() Dans une recette:

```
if node.attribute?('ec2')
  # ... do stuff on EC2 nodes
end
```

Les recettes

Une recette est l'élément de configuration le plus fondamental de l'organisation. Une recette:

- Est créé avec Ruby, un langage de programmation conçu pour lire et se comporter de manière prévisible
- Est principalement une collection de ressources, définie à l'aide de modèles (noms de

ressources, paires attribut-valeur et actions); du code d'assistance est ajouté autour de cela à l'aide de Ruby, si nécessaire
- Doit définir tout ce qui est nécessaire pour configurer une partie d'un système
- Doit être stocké dans un cookbook
- Peut être inclus dans une autre recette
- Peut utiliser les résultats d'une requête de recherche et lire le contenu d'un sac de données (y compris un sac de données crypté)
- Peut avoir une dépendance sur une (ou plusieurs) recettes
- Doit être ajouté à une liste de course avant de pouvoir être utilisé par le chef-client
- Est toujours exécuté dans le même ordre que celui indiqué dans une liste d'exécution

Un attribut peut être défini dans un cookbook (ou une recette), puis utilisé pour remplacer les paramètres par défaut sur un nœud. Lorsqu'un livre de recettes est chargé lors d'une exécution chef-client, ces attributs sont comparés aux attributs déjà présents sur le nœud. Les attributs définis dans les fichiers d'attributs sont d'abord chargés en fonction de l'ordre du livre de recettes. Pour chaque livre de recettes, les attributs du fichier default.rb sont chargés en premier, puis des fichiers d'attributs supplémentaires (le cas échéant) sont chargés dans l'ordre de tri lexical. Lorsque les attributs du livre de recettes ont priorité sur les attributs par défaut, chef-client appliquera ces nouveaux paramètres et valeurs lors de l'exécution chef-client sur le nœud.

Les rôles

Un rôle est un moyen de définir certains modèles et processus existant dans les nœuds d'une organisation comme appartenant à une fonction unique. Chaque rôle consiste en zéro (ou plus) attributs et une liste d’exécution. Chaque nœud peut se voir attribuer zéro (ou plus) rôles. Lorsqu'un rôle est exécuté sur un nœud, les détails de la configuration de ce nœud sont comparés aux attributs du rôle, puis le contenu de la liste d'exécution de ce rôle est appliqué aux détails de la configuration du nœud. Lorsqu'un chef-client est exécuté, il fusionne ses propres attributs et listes de fonctionnement avec ceux contenus dans chaque rôle attribué.

Un attribut peut être défini dans un rôle, puis utilisé pour remplacer les paramètres par défaut sur un nœud. Lorsqu'un rôle est appliqué lors d'une exécution chef-client, ces attributs sont comparés aux attributs déjà présents sur le nœud. Lorsque les attributs de rôle ont priorité sur les attributs par défaut, chef-client appliquera ces nouveaux paramètres et valeurs lors de l'exécution chef-client sur le nœud.

Un attribut de rôle ne peut être défini que comme attribut par défaut ou attribut de remplacement. Un attribut de rôle ne peut pas être défini pour être un attribut normal. Utilisez les méthodes default*attribute et override*attribute dans le fichier Ruby DSL ou les hachages default*attributes et override*attributes dans un fichier de données JSON.

Les environnements

Un environnement est un moyen de mapper le flux de travail réel d'une organisation à ce qui peut être configuré et géré lors de l'utilisation du serveur Chef. Chaque organisation commence par un environnement unique appelé environnement _default , qui ne peut pas être modifié (ou supprimé). Des environnements supplémentaires peuvent être créés pour refléter les modèles et le flux de travail de chaque organisation. Par exemple, créer development environnements de production , de

staging , de testing et de development . Généralement, un environnement est également associé à une (ou plusieurs) version (s) du livre de recettes.

Un attribut peut être défini dans un environnement, puis utilisé pour remplacer les paramètres par défaut sur un nœud. Lorsqu'un environnement est appliqué lors d'une exécution chef-client, ces attributs sont comparés aux attributs déjà présents sur le nœud. Lorsque les attributs d'environnement ont priorité sur les attributs par défaut, le chef-client appliquera ces nouveaux paramètres et valeurs lors de l'exécution du chef-client sur le nœud.

Un attribut d'environnement ne peut être défini que comme un attribut par défaut ou un attribut de remplacement. Un attribut d'environnement ne peut pas être défini pour être un attribut normal . Utilisez les méthodes default*attribute et override*attribute dans le fichier Ruby DSL ou les hachages default*attributes et override*attributes dans un fichier de données JSON.

Priorité des attributs

Les attributs sont toujours appliqués par le chef-client dans l'ordre suivant:

- Un attribut default situé dans un fichier d'attributs de livre de recettes
- Un attribut default situé dans une recette
- Un attribut default situé dans un environnement
- Un attribut default situé dans un rôle
- Un attribut force*default situé dans un fichier d'attributs de livre de recettes * Un attribut force*default situé dans une recette
- Un attribut normal situé dans un fichier d'attributs de livre de recettes
- Un attribut normal situé dans une recette
- Un attribut de override situé dans un fichier d'attributs de livre de recettes
- Un attribut de override situé dans une recette
- Un attribut de override situé dans un rôle
- Un attribut de override situé dans un environnement
- Un attribut force*override situé dans un fichier d'attributs de livre de recettes * Un attribut force*override situé dans une recette
- Un attribut automatic identifié par Ohai au début du cycle chef-client

où le dernier attribut de la liste est celui qui est appliqué au nœud.

L'ordre de priorité des attributs pour les rôles et les environnements est inversé pour les attributs par default et les attributs de override . L'ordre de priorité pour les attributs default est environnement, puis rôle. L'ordre de priorité pour les attributs de override est rôle, puis environnement. L'application d'attributs de override environnement après les attributs de override de rôle permet d'utiliser le même rôle dans plusieurs environnements, tout en garantissant la possibilité de définir des valeurs spécifiques à chaque environnement (le cas échéant). Par exemple, le rôle d'un serveur d'applications peut exister dans tous les environnements, mais un environnement peut utiliser un serveur de base de données différent des autres.

Attributs de la liste noire

Lorsque les paramètres d'attributs de liste noire sont utilisés, tout attribut défini dans une liste noire ne sera pas enregistré et tout attribut non défini dans une liste noire sera enregistré. Chaque type d'attribut est mis à l'index indépendamment des autres types d'attributs. Par exemple, si automatic*attribute*blacklist définit des attributs qui ne seront pas enregistrés, alors que normal*attribute*blacklist , default*attribute*blacklist et override*attribute*blacklist ne sont pas définis, tous les attributs normaux, les attributs par défaut et les attributs de remplacement seront enregistrés, ainsi que les attributs automatiques non exclus. via la liste noire.

Les attributs qui ne doivent pas être enregistrés par un nœud peuvent être mis sur liste noire dans le fichier client.rb. La liste noire est un hachage de clés qui spécifie chaque attribut à filtrer.

Les attributs sont mis en liste noire par type d'attribut, chaque type d'attribut étant mis en liste noire indépendamment. Chaque type d'attribut (automatic , default , normal et override peut définir des listes noires à l'aide des paramètres suivants dans le fichier client.rb:

Réglage	**Description**
automatic_attribute_blacklist	Un hachage qui liste en noir automatic attributs automatic , empêchant ainsi la sauvegarde des attributs en liste noire. Par exemple: ['network/interfaces/eth0'] . Valeur par défaut: nil , tous les attributs sont enregistrés. Si le tableau est vide, tous les attributs sont enregistrés.
default_attribute_blacklist	Un hachage qui liste en noir les attributs default , empêchant ainsi la sauvegarde des attributs de la liste noire. Par exemple: ['filesystem/dev/disk0s2/size'] . Valeur par défaut: nil , tous les attributs sont enregistrés. Si le tableau est vide, tous les attributs sont enregistrés.
normal_attribute_blacklist	Un hachage qui liste en noir normal attributs normal , empêchant ainsi la sauvegarde des attributs en liste noire. Par exemple: ['filesystem/dev/disk0s2/size'] . Valeur par défaut: nil , tous les attributs sont enregistrés. Si le tableau est vide, tous les attributs sont enregistrés.
override_attribute_blacklist	Un hachage dont les listes noires override attributs, empêchant ainsi la sauvegarde des attributs figurant sur la liste noire. Par exemple: ['map -autohome/size'] . Valeur par défaut: nil , tous les attributs sont enregistrés. Si le tableau est vide, tous les attributs sont enregistrés.

La pratique recommandée consiste à utiliser uniquement **automatic_attribute_blacklist** pour les attributs de liste noire. Cela est principalement dû au fait que les attributs automatiques génèrent le plus de données, mais également que les attributs normal, par défaut et de substitution sont généralement des attributs beaucoup plus importants et risquent davantage de causer des problèmes s'ils ne sont pas correctement mis en liste noire.

Par exemple, des données d'attribut automatiques similaires à:

```
{
"filesystem" => {
  "/dev/disk0s2" => {
    "size" => "10mb"
  },
  "map - autohome" => {
    "size" => "10mb"
  }
},
"network" => {
  "interfaces" => {
    "eth0" => {...},
    "eth1" => {...},
    }
  }
}
```

Pour mettre en liste noire les attributs du filesystem et autoriser la sauvegarde des autres attributs, mettre à jour le fichier client.rb:

```
automatic_attribute_blacklist ['filesystem']
```

Lorsqu'une liste noire est définie, tout attribut de ce type qui n'est pas spécifié dans cette liste noire d'attribut sera enregistré. Donc, sur la base de la liste noire précédente pour les attributs automatiques, les attributs de filesystem et map - autohome ne seront pas enregistrés, mais les attributs de network seront.

Pour les attributs contenant des barres obliques (/) dans la valeur de l'attribut, tels que l'attribut du filesystem '/dev/diskos2' , utilisez un tableau. Par exemple:

```
automatic_attribute_blacklist [['filesystem','/dev/diskos2']]
```

Attributs de la liste blanche

Lorsque des paramètres de liste blanche d'attributs sont utilisés, seuls les attributs définis dans une liste blanche sont enregistrés et tout attribut non défini dans une liste blanche ne sera pas enregistré. Chaque type d'attribut est ajouté à la liste blanche indépendamment des autres types d'attribut. Par exemple, si automatic*attribute*whitelist définit des attributs à enregistrer, alors que normal*attribute*whitelist , default*attribute*whitelist et override*attribute*whitelist ne sont pas définis, tous les attributs normaux, les attributs par défaut et les attributs de substitution sont également enregistrés, ainsi que les attributs automatiques spécifiquement inclus dans la mise en liste blanche.

Les attributs qui doivent être enregistrés par un nœud peuvent être ajoutés à la liste blanche dans le fichier client.rb. La liste blanche est un hachage de clés qui spécifie chaque attribut à enregistrer.

Les attributs sont ajoutés à la liste blanche par type d'attribut, chaque type d'attribut étant ajouté à la liste blanche indépendamment. Chaque type d'attribut (automatic , default , normal et override peut définir des listes blanches à l'aide des paramètres suivants dans le fichier client.rb:

Réglage	**Description**
automatic_attribute_whitelist	Un hachage qui ajoute aux attributs automatic liste blanche, empêchant ainsi la sauvegarde des attributs non inscrits sur la liste blanche. Par exemple: ['network/interfaces/eth0'] . Valeur par défaut: nil , tous les attributs sont enregistrés. Si le hachage est vide, aucun attribut n'est enregistré.
default_attribute_whitelist	Un hachage qui ajoute aux attributs default liste blanche, empêchant ainsi la sauvegarde des attributs non inscrits sur la liste blanche. Par exemple: ['filesystem/dev/disk0s2/size'] . Valeur par défaut: nil , tous les attributs sont enregistrés. Si le hachage est vide, aucun attribut n'est enregistré.
normal_attribute_whitelist	Un hachage qui ajoute normal attributs normal liste blanche, empêchant ainsi la sauvegarde des attributs non inscrits sur la liste blanche. Par exemple: ['filesystem/dev/disk0s2/size'] . Valeur par défaut: nil , tous les attributs sont enregistrés. Si le hachage est vide, aucun attribut n'est enregistré.
override_attribute_whitelist	Un hachage dont la liste blanche override attributs, empêchant ainsi la sauvegarde des attributs non inscrits sur la liste blanche. Par exemple: ['map - autohome/size'] . Valeur par défaut: nil , tous les attributs sont enregistrés. Si le hachage est vide, aucun attribut n'est enregistré.

La pratique recommandée consiste à n'utiliser que automatic_attribute_whitelist pour la liste blanche d'attributs. Cela est principalement dû au fait que les attributs automatiques génèrent le plus de données, mais également que les attributs normal, par défaut et de substitution sont généralement des attributs beaucoup plus importants et risquent davantage de causer des problèmes s'ils ne sont pas correctement ajoutés à la liste blanche.

Par exemple, des données d'attribut automatiques similaires à:

```
{
"filesystem" => {
  "/dev/disk0s2" => {
    "size" => "10mb"
  },
  "map - autohome" => {
    "size" => "10mb"
  }
},
"network" => {
  "interfaces" => {
    "eth0" => {...},
    "eth1" => {...},
    }
  }
```

```
}
```

Pour ajouter les attributs network la liste blanche et empêcher l'enregistrement des autres attributs, mettez à jour le fichier client.rb:

```
automatic_attribute_whitelist ['network/interfaces/']
```

Lorsqu'une liste blanche est définie, tout attribut de ce type qui n'est pas spécifié dans cette liste blanche d'attributs ne sera pas enregistré. Ainsi, en fonction de la liste blanche précédente pour les attributs automatiques, les attributs de filesystem et map - autohome ne seront pas enregistrés, mais les attributs de network seront.

Laisser la valeur vide pour empêcher l'enregistrement de tous les attributs de ce type d'attribut:

```
automatic_attribute_whitelist []
```

Pour les attributs contenant des barres obliques (/) dans la valeur de l'attribut, tels que l'attribut du filesystem '/dev/diskos2' , utilisez un tableau. Par exemple:

```
automatic_attribute_whitelist [['filesystem','/dev/diskos2']]
```

Exemples

Les exemples suivants sont énumérés de priorité faible à élevée.

Attribut par défaut dans /attributes/default.rb

```
default['apache']['dir'] = '/etc/apache2'
```

Attribut par défaut dans l'objet nœud de la recette

```
node.default['apache']['dir'] = '/etc/apache2'
```

Attribut par défaut dans /environments/environment_name.rb

```
default_attributes({ 'apache' => {'dir' => '/etc/apache2'}})
```

Attribut par défaut dans /roles/role_name.rb

```
default_attributes({ 'apache' => {'dir' => '/etc/apache2'}})
```

Attribut normal défini en tant qu'attribut cookbook

```
set['apache']['dir'] = '/etc/apache2'
normal['apache']['dir'] = '/etc/apache2'#set est un alias de normal.
```

Attribut normal défini dans une recette

```
node.normal['apache']['dir'] = '/etc/apache2'
```

Remplacer l'attribut dans /attributes/default.rb

```
override['apache']['dir'] = '/etc/apache2'
```

Remplacer l'attribut dans /roles/role_name.rb

```
override_attributes({ 'apache' => {'dir' => '/etc/apache2'}})
```

Remplacer l'attribut dans /environments/environment_name.rb

```
override_attributes({ 'apache' => {'dir' => '/etc/apache2'}})
```

Remplacer l'attribut dans un objet nœud (à partir d'une recette)

```
node.override['apache']['dir'] = '/etc/apache2'
```

S'assurer qu'un attribut par défaut a la priorité sur les autres attributs

Quand un attribut par défaut est défini comme ceci:

```
default['attribut'] = 'valeur'
```

toute valeur définie par un rôle ou un environnement la remplacera. Pour empêcher le remplacement de cette valeur, utiliser force_default :

```
force_default['attribut' ] = 'Je vais écraser, attribut de rôle ou
d'environnement'
```

ou:

```
 défaut![ 'attribut'] = "Le '!'  signifie que je gagne! "
```

S'assurer qu'un attribut de substitution a la priorité sur les autres attributs

Quand un attribut de substitution est défini comme ceci:

```
override['attribut'] = 'valeur'
```

toute valeur définie par un rôle ou un environnement la remplacera. Pour empêcher le remplacement de cette valeur, utiliser force_override :

```
force_override['attribut'] = 'Je vais écraser, attribut de rôle ou
d'environnement'
```

ou:

```
override!['attribut'] = "Le '!'  signifie que je gagne! "
```

Changer les attributs

À partir de chef-client 12.0, les niveaux de priorité des attributs peuvent être

- Supprimé pour un niveau de priorité d'attribut nommé spécifique
- Supprimé pour tous les niveaux de priorité d'attribut
- Attributs entièrement attribués

Supprimer un niveau de priorité

Un niveau de priorité d'attribut spécifique pour les attributs par défaut, normaux et de substitution peut être supprimé à l'aide de l'un des modèles de syntaxe suivants.

Pour les attributs par défaut:

```
node.rm_default('foo','bar')
```

Pour les attributs normaux:

```
node.rm_normal('foo','bar')
```

Pour les attributs de substitution:

```
node.rm_override('foo','bar')
```

Ces modèles renvoient la valeur calculée de la clé en cours de suppression pour le niveau de priorité spécifié.

Exemples

Les exemples suivants montrent comment supprimer un niveau de priorité d'attribut nommé spécifique.

Supprimer une valeur par défaut lorsqu'il n'existe que des valeurs par défaut

Étant donné la structure de code suivante sous 'foo' :

```
node.default['foo'] = {
  'bar' => {
    'baz' => 52,
    'thing' => 'stuff',
  },
  'bat' => {
    'things' => [5, 6],
  },
}
```

Et quelques attributs de rôle:

```
# S'il vous plaît ne faites jamais cela dans le code réel :)
node.role_default['foo']['bar']['thing'] = 'otherstuff'
```

Et un attribut force:

```
node.force_default['foo']['bar']['thing'] = 'allthestuff'
```

Lorsque la priorité d'attribut par défaut node['foo']['bar'] est supprimé:

```
node.rm_default('foo', 'bar') #=> {'baz' => 52, 'thing' => 'allthestuff'}
```

Ce qui reste sous 'foo' n'est que 'bat' :

```
node.attributes.combined_default['foo'] #=> {'bat' => { 'things' => [5,6] }
}
```

Supprimer les valeurs par défaut sans toucher aux attributs de priorité plus élevés

Étant donné la structure de code suivante:

```
node.default['foo'] = {
  'bar' => {
    'baz' => 52,
    'thing' => 'stuff',
  },
  'bat' => {
```

```
    'things' => [5, 6],
  },
}
```

Et quelques attributs de rôle:

```
# S'il vous plaît ne faites jamais cela dans le code réel :)
node.role_default['foo']['bar']['thing'] = 'otherstuff'
```

Et un attribut force:

```
node.force_default['foo']['bar']['thing'] = 'allthestuff'
```

Et aussi des substitutions d'attributs :

```
node.override['foo']['bar']['baz'] = 99
```

Même suppression que précédemment:

```
node.rm_default('foo', 'bar') #=> { 'baz' => 52, 'thing' => 'allthestuff' }
```

Les autres niveaux de priorité des attributs ne sont pas affectés:

```
node.attributes.combined_override['foo'] #=> { 'bar' => {'baz' => 99} }
node['foo'] #=> { 'bar' => {'baz' => 99}, 'bat' => { 'things' => [5,6] }
```

Supprimer le remplacement sans toucher aux attributs de priorité inférieurs

Étant donné la structure de code suivante, qui a un attribut override:

```
node.override['foo'] = {
  'bar' => {
    'baz' => 52,
    'thing' => 'stuff',
   },
  'bat' => {
     'things' => [5, 6],
   },
}
```

avec une seule valeur par défaut:

```
node.default['foo']['bar']['baz'] = 11
```

et une force à chaque préséance d'attribut:

```
node.force_default['foo']['bar']['baz'] = 55
node.force_override['foo']['bar']['baz'] = 99
```

Supprimer le override:

```
node.rm_override('foo', 'bar') #=> { 'baz' => 99, 'thing' => 'stuff' }
```

Les autres niveaux de priorité des attributs ne sont pas affectés:

```
node.attributes.combined_default['foo'] #=> { 'bar' => {'baz' => 55} }
```

Supprimer une clé inexistante retourne nil

```
noeud .  rm_default ( "non" , "tel" , "chose" ) # => nil
```

Supprimer tous les niveaux

Tous les niveaux de priorité d'attribut peuvent être supprimés à l'aide du modèle de syntaxe suivant:

```
node.rm('foo', 'bar')
```

</WRAP info>L'utilisation du node['foo'].delete('bar') lève une exception qui pointe vers la nouvelle API.</WRAP>

Exemple

Les exemples suivants montrent comment supprimer tous les niveaux de priorité d'attribut.

Supprimer tous les niveaux de priorité des attributs

Étant donné la structure de code suivante:

```
node.default['foo'] = {
  'bar' => {
    'baz' => 52,
  'thing' => 'stuff',
  },
  'bat' => {
    'things' => [5, 6],
  },
}
```

Avec substitution d'attributs:

```
node.override['foo']['bar']['baz'] = 999
```

Le retrait de laa clé 'bar' renvoie la valeur calculée:

```
node.rm('foo', 'bar') #=> {'baz' => 999, 'thing' => 'stuff'}
```

En regardant 'foo' , il ne reste que l'entrée 'bat' :

```
node['foo'] #=> {'bat' => { 'things' => [5,6] } }
```

La suppression d'une clé inexistante retourne nil

```
node.rm_default("no", "such", "thing") #=> nil
```

De manière globale

Utiliser ! pour effacer la clé pour le niveau de priorité de l'attribut nommé, puis terminer l'écriture en utilisant l'un des modèles de syntaxe suivants:

- node.default!['foo']['bar'] = {...}
- node.force_default!['foo']['bar'] = {...}
- node.normal!['foo']['bar'] = {...}
- node.override!['foo']['bar'] = {...}
- node.force_override!['foo']['bar'] = {...}

Exemples

Les exemples suivants montrent comment supprimer tous les niveaux de priorité d'attribut.

Juste un composant

Étant donné la structure de code suivante:

```
node.default['foo']['bar'] = {'a' => 'b'}
node.default!['foo']['bar'] = {'c' => 'd'}
```

Le '!' a provoqué l'écrasement de la clé 'bar' entière:

```
node[‘foo’] #=> {‘bar’ => {‘c’ => ‘d’}
```

Plusieurs composants; un “après”

Étant donné la structure de code suivante:

```
node.default['foo']['bar'] = {'a' => 'b'}
# Please don't ever do this in real code :)
node.role_default['foo']['bar'] = {'c' => 'd'}
```

```
node.default!['foo']['bar'] = {'d' => 'e'}
```

Le '!' a remplacé la valeur 'bar' de default mais comme les données de role_default figurent plus loin dans la liste, elles n'ont pas été affectées:

```
node['foo'] #=> {'bar' => {'c' => 'd', 'd' => 'e'}
```

Plusieurs composants; tous "avant"

Étant donné la structure de code suivante:

```
node.default['foo']['bar'] = {'a' => 'b'}
# Please don't ever do this in real code :)
node.role_default['foo']['bar'] = {'c' => 'd'}
node.force_default!['foo']['bar'] = {'d' => 'e'}
```

Avec force_default! il n'y a pas d'autres données sous 'bar' :

```
node['foo'] #=> {'bar' => {'d' => 'e'}
```

Plusieurs niveaux de priorité

Étant donné la structure de code suivante:

```
node.default['foo'] = {
  'bar' => {
    'baz' => 52,
    'thing' => 'stuff',
  },
  'bat' => {
    'things' => [5, 6],
  },
}
```

Et quelques attributs:

```
# Please don't ever do this in real code :)
node.role_default['foo']['bar']['baz'] = 55
node.force_default['foo']['bar']['baz'] = 66
```

Et d'autres niveaux de priorité:

```
node.normal['foo']['bar']['baz'] = 88
node.override['foo']['bar']['baz'] = 99
```

Affectation d'une valeur de manière globale:

```
node.default!['foo']['bar'] = {}
```

Le rôle par défaut et la valeur par défaut sont laissés par défaut, plus d'autres niveaux de priorité:

```
node.attributes.combined_default['foo'] #=> {'bar' => {'baz' => 66},
'bat'=>{'things'=>[5, 6]}}
node.attributes.normal['foo'] #=> {'bar' => {'baz' => 88}}
node.attributes.combined_override['foo'] #=> {'bar' => {'baz' => 99}}
node['foo']['bar'] #=> {'baz' => 99}
```

Si force_default! est écrit:

```
node.force_default!['foo']['bar'] = {}
```

la différence est:

```
node.attributes.combined_default['foo'] #=> {'bat'=>{'things'=>[5, 6]},
'bar' => {}}
node.attributes.normal['foo'] #=> {'bar' => {'baz' => 88}}
node.attributes.combined_override['foo'] #=> {'bar' => {'baz' => 99}}
node['foo']['bar'] #=> {'baz' => 99}
```

Fusion en profondeur

Les attributs sont généralement définis dans les livres de recettes, les recettes, les rôles et les environnements. Ces attributs sont cumulés au niveau du nœud lors d'une exécution chef-client. Une recette peut stocker des valeurs d'attribut en utilisant un hachage ou un tableau à plusieurs niveaux.

Par exemple, un groupe d'attributs pour les serveurs Web pourrait être:

```
override_attributes (
   : apache => {
     : listen_ports => [ 80 ] ,
     : prefork => {
       : startservers => 20 ,
       : minspareservers => 20 ,
       : maxspareservers => 40
     }
   }
 )
```

Mais que se passe-t-il si tous les serveurs Web ne sont pas identiques? Et si certains des serveurs Web nécessitaient un attribut unique pour avoir une valeur différente? Vous pouvez stocker ces paramètres à deux emplacements, une fois comme dans l'exemple précédent et une fois comme suit:

```
  override_attributes (
    : apache => {
      : listen_ports => [ 80 ] ,
```

```
      : prefork => {
        : startservers => 30 ,
        : minspareservers => 20 ,
        : maxspareservers => 40
      }
    }
  )
```

Mais ce n'est pas très efficace, surtout parce que la plupart d'entre elles sont identiques. Les capacités de fusion profonde du chef-client permettent de superposer des attributs à des livres de recettes, des recettes, des rôles et des environnements. Cela permet à un attribut d'être réutilisé sur plusieurs nœuds, en utilisant les attributs par défaut définis au niveau du livre de recettes, mais en fournissant également un moyen d'appliquer certains attributs (avec une priorité d'attribut plus élevée) uniquement lorsqu'ils sont supposés l'être.

Par exemple, un rôle nommé baseline.rb :

```
name "baseline"
 description "Le rôle le plus fondamental pour toutes les configurations"
 run_list "recipe [baseline]"

 override_attributes (
   : apache => {
     : listen_ports => [ 80 ] ,
     : prefork => {
       : startservers => 20 ,
       : minspareservers => 20 ,
       : maxspareservers => 40
     }
   }
 )
```

puis un rôle nommé web.rb :

```
name 'web'
description 'configuration serveur Web'
run_list 'role [baseline]'

override_attributes (
  : apache => {
    : prefork => {
     : startservers => 30
    }
  }
)
```

Ces deux fichiers sont similaires car ils partagent la même structure. Lorsqu'une valeur d'attribut est un hachage, ces données sont fusionnées. Lorsqu'une valeur d'attribut est un tableau, si les niveaux de priorité des attributs sont les mêmes, ces données sont fusionnées. Si les niveaux de priorité de la valeur d'attribut dans un tableau sont différents, ces données sont remplacées. Pour tous les autres types de valeur (tels que chaînes, entiers, etc.), ces données sont remplacées.

Par exemple, le web.rb référence au rôle baseline.rb . Le fichier web.rb fournit uniquement une valeur pour un attribut :startservers . Lorsque le chef-client compare ces attributs, la fonction de fusion approfondie garantit que :startservers (et sa valeur de 30) seront appliqués à tout nœud pour lequel la structure d'attribut web.rb doit être appliquée.

Cette approche permettra une recette comme celle-ci:

```
include_recipe 'apache2'
Chef::Log.info(node['apache']['prefork'].to_hash)
```

et un run_list comme ceci:

```
run_list/web.json
{
   "run_list": [ "role[web]" ]
}
```

pour produire des résultats comme celui-ci:

```
[ Mar , 16 août 2011 14 : 44 : 26 - 0700 ] INFO :
      {
        "startservers" => 30 ,
        "minspareservers" => 20 ,
        "maxspareservers" => 40 ,
        "serverlimit" => 400 ,
        "maxclients" => 400 ,
        "maxrequestsperchild" => 10000
      }
```

Même si le fichier web.rb ne contient pas d'attributs ni de valeurs pour minspareservers , maxspareservers , serverlimit , maxclients et maxrequestsperchild , les fonctionnalités de fusion profonde les ont maxrequestsperchild .

Les sections suivantes montrent comment la logique permet d'utiliser la fusion profonde pour effectuer des substitutions et des ajouts d'attributs.

Substitution

Les exemples suivants montrent comment la logique fonctionne :

pour remplacer une chaîne existante à l'aide d'un hachage:

```
role_or_environment 1 { :x => '1' ,:y => '2' }
+
role_or_environment 2 { :y => '3' }
=
{ :x => '1' ,:y => '3' }
```

Pour remplacer un booléen existant par un hachage:

```
role_or_environment 1 { :x => true ,:y => false }
+
role_or_environment 2 { : y => true }
=
{ :x => true ,:y => true }
```

Pour substituer un tableau avec un hachage:

```
role_or_environment 1 [ '1' , '2' , '3' ]
+
role_or_environment 2 { :x => '1' ,:y => '2' }
=
{ :x => '1' ,:y => '2' }
```

Lorsque les éléments ne peuvent pas être fusionnés par substitution, les données d'origine sont écrasées.

Ajout

Les exemples suivants montrent comment la logique fonctionne :

Pour ajouter une chaîne à l'aide d'un hachage

```
role_or_environment 1 { :x => '1' ,:y => '2' }
+
role_or_environment 2 { :z => '3' }
=
{ :x => '1' ,:y => '2' ,:z => '3' }
```

Pour ajouter une chaîne en utilisant un tableau:

```
role_or_environment 1 [ '1' , '2' ]
+
role_or_environment 2 [ '3' ]
=
[ '1' , '2' , '3' ]
```

Pour ajouter une chaîne en utilisant un hachage à plusieurs niveaux:

```
role_or_environment 1 { :x => { :y => '2' } }
+
```

```
role_or_environment 2 { :x => { :z => '3' } }
=
{ :x => { :y => '2' ,:z => '3' } }
```

Pour ajouter une chaîne en utilisant un tableau à plusieurs niveaux:

```
role_or_environment 1 [[ 1 , 2 ]]
+
role_or_environment 2 [[ 3 ]]
=
[[ 1 , 2 ], [ 3 ]]
```