```
#
# Le mapping.
#
# - C'est la définition des champs d'un type de document.
# - Elasticsearch contient des index qui contiennent des types auxquels
sont attachés les documents.
#
# Le mapping défini pour chaque champ :
# - son nom,
# - son type,
# - ses options d'indexation,
# - ses analyseurs (pour les champs de type texte).
#


#
# 1. Exemple de fonctionnement de l'analyseur
#
#
# L'analyseur est un composant qui transforme un texte en une séquence de
termes
# qui seront indexés ou utilisés comme argument de la recherche.
#
# Pour comprendre le fonctionnement de l'analyse, Elasticsearch offre l'API
# « _analyze »
#

# Analyse avec l'analyseur par défaut.

GET _analyze
{
  "text": "Devoxx, la conférence des développeurs passionnnés."
}

# Il est possible de choisir une autre analyseur comme l'analyseur « french
»

GET _analyze
{
  "text": "Devoxx, la conférence des développeurs passionnnés.",
  "analyzer": "french"
}

#
# Lors de la recherche, l'analyseur sera joué sur les arguments puis les
termes
# obtenus seront recherchés dans l'index. Pour qu'un document soit trouvé,
# il faut qu'il y ait une correspondance exacte entre un terme issu du
critère
# de recherche et un terme issu du document lors de l'indexation.
#

# Analyze lors de la recherche, il n'y a pas de correspondance avec les
termes
# issus de l'indexation avec l'analyseur par défaut.

GET _analyze
{
  "text": "passionné"
}
```

```
# Analyze lors de la recherche, cette fois, le terme obtenu correspond à un
# terme de l'analyse lors de l'indexation avec l'analyseur « french » : «
pasion ».

GET _analyze?text=passionn%C3%A9&analyzer=french
{
  "text": "passionné",
  "analyzer": "french"
}


#
# 2. Les risques de l'absence de schéma
#

# Suppression de l'index (risque de remonter un 404 lors de la première
lecture)
# DELETE devoxx

# Ajouter un premier élément dans l'index.
PUT devoxx/talk/OMZ-7895
{
  "speaker": ["David Pilato", "Colin Surprenant" ],
  "title": "Elles ressemblent à quoi mes données ?",
  "salle": 241
}

# Vérifier la présence de cet élément.
GET devoxx/talk/_search

# On constate que l'index « devoxx » et le type « talk » qui n'existait pas
on été créés automatiquement.
# Il est donc possible d'utiliser Elasticsearch sans définir de schéma au
préalable.


# Ajouter un deuxième document.
PUT devoxx/talk/VRH-4567
{
  "speaker": "Jérôme Mainaud",
  "title": "De l'importance du mapping",
  "salle": "Maillot"
}

# L'ajout a échoué !
# Le message indique que le champ « salle » ne peut pas être interprété car
# la valeur « Maillot » n'est pas un nombre valide.

# Pour mieux comprendre la cause de l'erreur, allons voir quel est le
mapping
# du type devoxx/talk.

# Pour cela on va utiliser l'API _mapping :

GET devoxx/_mapping/talk

# On voit que le champ « salle » est de type « long ».
# Le champ « salle » de notre deuxième document étant un texte, l'ajout du
# document échoue. En plus que de créer l'index et le type, ES a aussi créé
le
```

```
# mapping et a défini les champs présents dans le document en devinant leur
type.
# Si cet inférence n'est pas correcte, on se retrouvera avec des champs de
type
# invalides qui risque de provoquer le rejet de certains documents.

#
# Ajout du mapping
#

# Supprimer l'index existant

DELETE devoxx

# Créer un index avec un mapping.

PUT devoxx
{
  "mappings": {
    "talk": {
      "properties": {
        "speaker": { "type" : "text" },
        "title": { "type": "text" },
        "salle": { "type": "text" }
      }
    }
  }
}

# Ajouter le premier document.

PUT devoxx/talk/OMZ-7895
{
  "speaker": ["David Pilato", "Colin Surprenant" ],
  "title": "Elles ressemblent à quoi mes données ?",
  "salle": 241
}

# Ajouter le deuxième document. Cette fois ça marche !

PUT devoxx/talk/VRH-4567
{
  "speaker": "Jérôme Mainaud",
  "title": "De l'importance du mapping",
  "salle": "Maillot",
  "comment": "2015-04-08T18:40:00"
}

# Vérifier que les deux documents ont été enregistrés.

GET devoxx/talk/_search

# Ajouter un troisième document

PUT devoxx/talk/JMX-7798
{
  "speaker": "Cécilia Bossard",
  "title": "Gitflow in action",
  "salle": "Amphi Bleu",
  "comment": "Gitflow ou github flow ?"
}
```

```
# On a un problème avec le commentaire qui est interprété comme une date
# à cause du contenu du deuxième document.

GET devoxx/_mapping/talk

#
#
# 3. Gestion des objects imbriqués
#
#

# Supprimer l'index existant

DELETE devoxx

#
# Nous créons des documents avec une imbrication.
# Le champ « speaker » est un objet contenant deux champs :
#   - firstname
#   - lastname

# Créer l'index avec son mapping

PUT devoxx
{
  "mappings": {
    "talk": {
      "properties": {
        "speaker": {
            "type": "object",
            "properties": {
                "firstname": { "type": "text" },
                "lastname": { "type": "text" }
            }
        },
        "title": { "type": "text" }
      }
    }
  }
}

# Ajouter les données (2 documents)

PUT devoxx/talk/SFJ-5646
{
  "speaker": [
    {
      "firstname": "Pierre-Antoine",
      "lastname": "Grégoire"
    },
    {
      "firstname": "Dimitri",
      "lastname": "Baeli"
    },
    {
      "firstname": "Arnaud",
      "lastname": "Héritier"
    }
  ],
  "title": "Les Mercenaires de Devops"
```

```
}

PUT devoxx/talk/GHN-6102
{
  "speaker": [
    {
      "firstname": "Mathieu",
      "lastname": "Poumeyrol"
    },
    {
      "firstname": "Pierre",
      "lastname": "Baillet"
    }
  ],
  "title": "Présentation de Rust, le langage le plus excitant depuis
l'arrivée de Scala"
}


#
# On veux chercher la présentation de Pierre Baillet mais comme on n'a pas
bien
# entendu son nom, on demande "Pierre Baeli".
# La recherche ne devrait donc rien retourner.
#
GET devoxx/talk/_search
{
  "query" : {
    "bool": {
      "must": [
{ "match": { "speaker.firstname": "Pierre" }},
{ "match": { "speaker.lastname": "Baeli" }}
      ]
    }
  }
}

# Et pourtant si ! Elle retourne l'autre document.
# Lors de l'indexation ES a mis à plat de document imbriqué.
# Au niveau de l'index Lucene soujacent, l'indexation est identique à celle
# du document ci dessous.

# Indexer le document suivant.

PUT devoxx/talk/SFJ-5646-2
{
  "speaker.firstname": [
    "Pierre-Antoine",
    "Dimitri",
    "Arnaud"
  ],
  "speaker.lastname": [
    "Grégoire",
    "Baeli",
    "Héritier"
  ],
  "title": "Les Mercenaires de Devops"
}


# Si on refait la même recherche, on voit maintenant que les deux documents
```

```
# sont trouvés équitablement.

GET devoxx/talk/_search
{
  "query" : {
    "bool": {
      "must": [
{ "match": { "speaker.firstname": "Pierre" }},
{ "match": { "speaker.lastname": "Baeli" }}
      ]
    }
  }
}

# Ce comportement simple fonctionne parfaitement lorsque le champs de type
# objet est monovalué ou qu'on ne cherche jamais que sur un champ à la fois
# de l'objet imbriqué. Dans le cas contraire, on obtient du du bruit
# liés à une combinaison entre les champs de deux sous-objets différents.
#
# Deux cas se présentent alors :
# - si le bruit est faible ou attendu, on peut conserver ce fonctionnement.
# - si le bruit est génant, il faut utiliser un type « nested » à la place.

# Supprimer l'index existant

DELETE devoxx

# Créer le nouvel index avec son mapping.
# Le type « nested » est utilisé à la place de « object ».

PUT devoxx
{
  "mappings": {
    "talk": {
      "properties": {
        "speaker": {
            "type": "nested",
            "properties": {
                "firstname": { "type": "text" },
                "lastname": { "type": "text" }
            }
        },
        "title": { "type": "text" }
      }
    }
  }
}

# Ajouter les données.

PUT devoxx/talk/SFJ-5646
{
  "speaker": [
    {
      "firstname": "Pierre-Antoine",
      "lastname": "Grégoire"
    },
    {
      "firstname": "Dimitri",
      "lastname": "Baeli"
    },
```

```
    {
      "firstname": "Arnaud",
      "lastname": "Héritier"
    }
  ],
  "title": "Les Mercenaires de Devops"
}

PUT devoxx/talk/GHN-6102
{
  "speaker": [
    {
      "firstname": "Mathieu",
      "lastname": "Poumeyrol"
    },
    {
      "firstname": "Pierre",
      "lastname": "Baillet"
    }
  ],
  "title": "Présentation de Rust, le langage le plus excitant depuis
l'arrivée de Scala"
}


# Nested oblige à utiliser une requête spéciale « nested ».
# Nous adaptons donc la requête antérieure.

GET devoxx/talk/_search
{
  "query" : {
    "nested": {
      "path": "speaker",
      "query": {
        "bool": {
          "must": [
{ "match": { "speaker.firstname": "Pierre" }},
{ "match": { "speaker.lastname": "Baeli" }}
          ]
        }
      }
    }
  }
}

# Cette fois-ci, nous ne trouvons plus rien.
# C'est le résultat attendu car il n'y a pas de speaker s'appelant Pierre
Baeli.

# Avec une recherche correspondant à un vrai cas, on trouve le bon
document.

GET devoxx/talk/_search
{
  "query" : {
    "nested": {
      "path": "speaker",
      "query": {
        "bool": {
          "must": [
{ "match": { "speaker.firstname": "Pierre" }},
```

```
{ "match": { "speaker.lastname": "Baillet" }}
          ]
        }
      }
    }
  }
}


#
#
# Comment retrouver un fonctionnement dynamique ?
#
#

#
# Pour retrouver un fonctionnement dynamique au niveau des champs d'un
type,
# il est possible de choisir l'analyseur par défaut.
# Tout nouveau champ de type texte utilise cet analyseur.

# Supprimer l'index (peut échouer [404] la première fois)

# DELETE citations

# Ajouter une citation

PUT citations/t/devoxx
{
  "citation": "Devoxx, la conférence des développeurs passionnnés."
}

# Rechercher une citation contenant « passionné »
# La requête ne retourne rien.

GET citations/t/_search
{
  "query": {
    "match": {
      "citation": "passionné"
    }
  }
}

#
# Mapping avec analyser par défaut
#

# Supprimer l'index

DELETE citations

# Créer l'index avec des settings qui définissent l'analyseur par défaut.
# Il s'agit de l'analyseur dont le nom est « default ».

PUT citations
{
  "settings": {
    "analysis": {
      "analyzer": {
        "default": {
```

```
          "type": "french"
        }
      }
    }
  }
}

# Ajouter une citation

PUT citations/t/devoxx
{
  "citation": "Devoxx, la conférence des développeurs passionnnés."
}


# Rechercher une citation contenant « passionné ».
# Cette fois, la recherche retourne un résultat.

GET citations/t/_search
{
  "query": {
    "match": {
      "citation": "passionné"
    }
  }
}

# Regardons le mapping créé

GET citations/_mapping/t

#
# Dynamic templates.
#

#
# Les « dynamic_templates » permettent de définir des modèles de champ.
# Ils s'appliquent aux nouveaux champs dont le nom et le type répondent
# à un modèle.

# Supprimer l'index existant

DELETE devoxx

# Créer l'index avec un type qui contient deux dynamic_templates.
# L'un pour les champs texte dont le nom se termine par "-id" dont
# l'analyseur conserve l'entrée à l'identique.
# L'autre pour tous les autres champs qui utilise l'analyseur français.

PUT devoxx
{
  "mappings": {
    "tools-in-action": {
      "dynamic_templates": [
        {
          "id_field": {
            "match": "*-id",
            "match_mapping_type": "string",
            "mapping": {
              "type": "text",
              "analyzer": "keyword"
```

```
            }
          }
        },
        {
          "text_field": {
            "match": "*",
            "match_mapping_type": "string",
            "mapping": {
              "type": "text",
              "analyzer": "french"
            }
          }
        }
      ]
    }
  }
}

# Ajouter des données

PUT devoxx/tools-in-action/_bulk
{"index":{"_id":"TIA-006"}}
{"talk-id":"TIA-006","talk-code":"TIA-006","speaker":"Nicolas
Muller","title":"InfluxDB : la base de données chronologique OpenSource
autonome"}
{"index":{"_id":"TIA-007"}}
{"talk-id":"TIA-007","talk-code":"TIA-007","speaker":"Jérôme
Mainaud","title":"De l'importance du mapping"}

# Rechercher sur le champ talk-id.
# Le résultat est une liste exacte. Car le contenu est interprété
strictement.

GET devoxx/tools-in-action/_search
{
  "query": {
    "match": {
      "talk-id": "TIA-007"
    }
  }
}

# Rechercher sur le champ talk-code.
# Les deux documents sont remontés car le terme TIA est présent dans les
deux cas.

GET devoxx/tools-in-action/_search
{
  "query": {
    "match": {
      "talk-code": "TIA-007"
    }
  }
}

# Rechercher sur le champ title.
# On obtient un fonctionnement de recherche classique.
# L'analyseur français est bien en place.

GET devoxx/tools-in-action/_search
{
```

```
  "query": {
    "match": {
      "title": "mapping"
    }
  }
}

# Visualiser le mapping créé dynamiquement.
# Les champs ont peut vérifier que les champs introduits dynamiquement ont
# l'analyseur qui correspond à leur nom.

GET devoxx/_mapping/tools-in-action


#
# Index Template.
#


# Pour allez plus loin, nous allons introduire deux notion de templating
# supplémentaires.
#
# 1. Les templates
#       définissent des modèles d'index et leurs mappings.
#       sont utilisés lors de la création dynamique d'un index.
# 2. Le type « _default_ »
#       définit la définition par défaut d'un type créé dynamiquement.
#

# Supprimer tous les index.

DELETE devoxx*

# Créer un template d'index pour tous les index dont le nom commence par «
devoxx ».
# Il contient un type par défaut avec des champs dynamiques.

PUT _template/devoxx
{
  "template": "devoxx*",
  "mappings": {
    "_default_": {
      "dynamic_templates": [
        {
          "id_field": {
            "match": "*-id",
            "match_mapping_type": "string",
            "mapping": {
              "type": "text",
              "analyzer": "keyword"
            }
          }
        },
        {
          "text_field": {
            "match": "*",
            "match_mapping_type": "string",
            "mapping": {
              "type": "text",
              "analyzer": "french"
            }
```

```
          }
        }
      ]
    }
  }
}

# Ajouter un document dans l'index 2014
PUT devoxx2014/conference/CNF-007
{
  "talk-id": "CNF-007",
  "talk-code": "CNF-007",
  "speaker": [ "Thibault Chassagnette", "Ghislain Seguy"],
  "title": "Pas à PaaS : CloudFoundry in action"
}

# Ajouter un document dans l'index 2015
PUT devoxx2015/tools-in-action/TIA-007
{
  "talk-id": "TIA-007",
  "talk-code": "TIA-007",
  "speaker": "Jérôme Mainaud",
  "title": "De l'importance du mapping"
}


# Rechercher sur tous les index sur le champ talk-id.
# Le résultat de recherche est précis.

GET devoxx*/_search
{
  "query": {
    "match": {
      "talk-id": "TIA-007"
    }
  }
}

# Rechercher sur tous les index sur le champ talk-id.
# Le résultat de recherche est plus large.
# Conformément à l'utilisation de l'analyseur français. (sur le terme 007)

GET devoxx*/_search
{
  "query": {
    "match": {
      "talk-code": "TIA-007"
    }
  }
}

# Vérifier l'existance des deux index et de leur mapping

GET devoxx2015/_mapping
GET devoxx2014/_mapping


#
#
#
# 5. Gestion d'une arborescence
```

```
#
#
#

# Nous avons vu jusqu'à présent que le choix de l'analyseur permet
# d'améliorer la qualité de la recherche, soit en réduissant le nombre de
document
# non pertinant (bruit), soit en ellargissant la recherche.
#
# Nous allons maintenant voir que le choix des analyseurs utilisés à bon
essient,
# Permettent de changer les fonctionnalités apportées par la recherche.
#
#
# Pour cela, nous prendrons l'exemple d'un index d'entreprise qui recense
des
# documents normatifs pour. Les services sont organisés hiérarchiquement :
#
# DG
#    DSI
#       ARCHI
#       ETUDES
#       SECURITE
#    DRH
#       RECRUTEMENT
#       CARRIERES
#    OPERATIONS
#       VENTES
#       RISQUES
#
# Le fichier load-normes est un script permettant le chargement des
données.
# Lisez le pour voir la forme de ces données. On s'intéressera
principalement
# au champ service.

# Supprimer l'index (peut échouer [404] la première fois)

# DELETE normes

# Créer l'index avec un mapping simple.

PUT normes
{
  "mappings": {
    "normes": {
      "properties": {
        "titre": { "type": "text" },
        "service": { "type": "keyword" }
      }
    }
  }
}

# Charger les données.

PUT _bulk
{"index":{"_index":"normes","_type":"normes","_id":1}}
{"titre":"Procuration des fournitures","service":"DG"}
{"index":{"_index":"normes","_type":"normes","_id":2}}
{"titre":"Ouverture des accès réseau","service":"DG/DSI"}
```

```
{"index":{"_index":"normes","_type":"normes","_id":3}}
{"titre":"Couleur des crayons pour
schemas","service":"DG/DSI/ARCHITECTURE"}
{"index":{"_index":"normes","_type":"normes","_id":4}}
{"titre":"Organiser une réunion avec le métier","service":"DG/DSI/ETUDES"}
{"index":{"_index":"normes","_type":"normes","_id":5}}
{"titre":"Interdire d'abord, autoriser jamais","service":"DG/DSI/SECURITE"}
{"index":{"_index":"normes","_type":"normes","_id":6}}
{"titre":"Définir les métiers entreprise","service":"DG/DRH"}
{"index":{"_index":"normes","_type":"normes","_id":7}}
{"titre":"Passe ton bac d'abord","service":"DG/DRH/RECRUTEMENT"}
{"index":{"_index":"normes","_type":"normes","_id":8}}
{"titre":"Devenir chef de projet","service":"DG/DRH/CARRIERES"}
{"index":{"_index":"normes","_type":"normes","_id":9}}
{"titre":"Attitude à tenir face au
client","service":"DG/OPERATIONS/VENTES"}
{"index":{"_index":"normes","_type":"normes","_id":10}}
{"titre":"Organiser ses ventes","service":"DG/OPERATIONS/VENTES"}

# Puis vérifier que les données sont chargées.
# La recherche doit trouver 10 éléments. (hits.total)

GET normes/normes/_search


# Une recherche sur le champ « service » permet de trouver les normes
# définies par un service donné.

# Rechercher les documents définis par la DSI

GET normes/normes/_search
{
  "query": {
    "match": { "service": "DG/DSI" }
  }
}


#
# Souvent, les utilisateurs considéreront que les normes définies par la
DSI
# incluent celles définies par les sous-services de la DSI.
#
# Pour répondre à cette question, on utilise deux analyseurs.
# 1. path_hierarchy, lors de l'indexation, qui décompose les structures de
#     type chemin dans un arbre.
# 2. keyword, lors de la recherche, qui garde le texte d'origine.
#
# Celà donne:
#
# Indexation:
#     DG            --- path_hierarchy ---> DG
#     DG/DSI        --- path_hierarchy ---> DG, DG/DSI
#     DG/DSI/ETUDE  --- path_hierarchy ---> DG, DG/DSI, DG/DSI/ETUDE
#
# Recherche:
#     DG            ---     keyword    ---> DG
#     DG/DSI        ---     keyword    ---> DG/DSI
#     DG/DSI/ETUDE  ---     keyword    ---> DG/DSI/ETUDE
#
```

```
# Supprimer l'ancien index

DELETE normes

# Créer l'index avec les différents analyseurs.
# On utilise un champ multifield pour que service soit indexé de
différentes façons.
# 1. « service » sans analyse
# 2. « service.children »
#    analyzer: path_hierarchy
#    search_analyzer: keyword

PUT normes
{
  "settings": {
    "analysis": {
      "analyzer": {
        "pathAnalyzer": {
          "type": "custom",
          "tokenizer": "pathTokenizer"
        }
      },
      "tokenizer": {
        "pathTokenizer": {
          "type": "path_hierarchy"
        }
      }
    }
  },
  "mappings": {
    "normes": {
      "properties": {
        "titre": {
          "type": "text"
        },
        "service": {
          "type": "keyword",
          "fields" : {
            "children": {
              "type": "text",
              "analyzer": "pathAnalyzer",
              "search_analyzer": "keyword"
            }
          }
        }
      }
    }
  }
}

# Importer de nouveau les données.
PUT _bulk
{"index":{"_index":"normes","_type":"normes","_id":1}}
{"titre":"Procuration des fournitures","service":"DG"}
{"index":{"_index":"normes","_type":"normes","_id":2}}
{"titre":"Ouverture des accès réseau","service":"DG/DSI"}
{"index":{"_index":"normes","_type":"normes","_id":3}}
{"titre":"Couleur des crayons pour
schemas","service":"DG/DSI/ARCHITECTURE"}
{"index":{"_index":"normes","_type":"normes","_id":4}}
{"titre":"Organiser une réunion avec le métier","service":"DG/DSI/ETUDES"}
```

```
{"index":{"_index":"normes","_type":"normes","_id":5}}
{"titre":"Interdire d'abord, autoriser jamais","service":"DG/DSI/SECURITE"}
{"index":{"_index":"normes","_type":"normes","_id":6}}
{"titre":"Définir les métiers entreprise","service":"DG/DRH"}
{"index":{"_index":"normes","_type":"normes","_id":7}}
{"titre":"Passe ton bac d'abord","service":"DG/DRH/RECRUTEMENT"}
{"index":{"_index":"normes","_type":"normes","_id":8}}
{"titre":"Devenir chef de projet","service":"DG/DRH/CARRIERES"}
{"index":{"_index":"normes","_type":"normes","_id":9}}
{"titre":"Attitude à tenir face au
client","service":"DG/OPERATIONS/VENTES"}
{"index":{"_index":"normes","_type":"normes","_id":10}}
{"titre":"Organiser ses ventes","service":"DG/OPERATIONS/VENTES"}

# La recherche sur le champ service nous donne le même résultat qu'avant.

GET normes/normes/_search
{
  "query": {
    "match": { "service": "DG/DSI" }
  }
}

# La recherche sur le champ service.children nous donne tous les documents
# définis par la DSI et ses sous-services.

GET normes/normes/_search
{
  "query": {
    "match": { "service.children": "DG/DSI" }
  }
}


#
# Une fois toutes ses normes établies, on peut vouloir chercher
# celles qui s'appliquent à un service donnée.
# Traditionnellement, il s'agit des normes définies par le service
# et ses parents.
#
# Pour cela, on va utiliser la même astuce que précédemment en
# inversant les analyseurs. Le keyword sera utilisé lors de l'indexation,
# le path_hierarchie lors de la recherche.
#
# Indexation:
#     DG            ---     keyword    ---> DG
#     DG/DSI        ---     keyword    ---> DG/DSI
#     DG/DSI/ETUDE  ---     keyword    ---> DG/DSI/ETUDE
#
# Recherche:
#     DG            --- path_hierarchy ---> DG
#     DG/DSI        --- path_hierarchy ---> DG, DG/DSI
#     DG/DSI/ETUDE  --- path_hierarchy ---> DG, DG/DSI, DG/DSI/ETUDE
#

# Supprimer l'ancien index

DELETE normes

# Créer l'index avec les différents analyseurs.
```

```
# On utilise un champ multifield pour que service soit indexé de
différentes façons.
# 1. « service » sans analyse
# 2. « service.children »
#    analyzer: path_hierarchy
#    search_analyzer: keyword
# 3. « service.inherited »
#    analyzer: keyword
#    search_analyzer: path_hierarchy

PUT normes
{
  "settings": {
    "analysis": {
      "analyzer": {
        "pathAnalyzer": {
          "type": "custom",
          "tokenizer": "pathTokenizer"
        }
      },
      "tokenizer": {
        "pathTokenizer": {
          "type": "path_hierarchy"
        }
      }
    }
  },
  "mappings": {
    "normes": {
      "properties": {
        "titre": {
          "type": "text"
        },
        "service": {
          "type": "keyword",
          "fields" : {
            "children": {
              "type": "text",
              "analyzer": "pathAnalyzer",
              "search_analyzer": "keyword"
            },
            "inherited": {
              "type": "text",
              "analyzer": "keyword",
              "search_analyzer": "pathAnalyzer"
            }
          }
        }
      }
    }
  }
}


# Importer de nouveau les données.

PUT _bulk
{"index":{"_index":"normes","_type":"normes","_id":1}}
{"titre":"Procuration des fournitures","service":"DG"}
{"index":{"_index":"normes","_type":"normes","_id":2}}
{"titre":"Ouverture des accès réseau","service":"DG/DSI"}
```

```
{"index":{"_index":"normes","_type":"normes","_id":3}}
{"titre":"Couleur des crayons pour
schemas","service":"DG/DSI/ARCHITECTURE"}
{"index":{"_index":"normes","_type":"normes","_id":4}}
{"titre":"Organiser une réunion avec le métier","service":"DG/DSI/ETUDES"}
{"index":{"_index":"normes","_type":"normes","_id":5}}
{"titre":"Interdire d'abord, autoriser jamais","service":"DG/DSI/SECURITE"}
{"index":{"_index":"normes","_type":"normes","_id":6}}
{"titre":"Définir les métiers entreprise","service":"DG/DRH"}
{"index":{"_index":"normes","_type":"normes","_id":7}}
{"titre":"Passe ton bac d'abord","service":"DG/DRH/RECRUTEMENT"}
{"index":{"_index":"normes","_type":"normes","_id":8}}
{"titre":"Devenir chef de projet","service":"DG/DRH/CARRIERES"}
{"index":{"_index":"normes","_type":"normes","_id":9}}
{"titre":"Attitude à tenir face au
client","service":"DG/OPERATIONS/VENTES"}
{"index":{"_index":"normes","_type":"normes","_id":10}}
{"titre":"Organiser ses ventes","service":"DG/OPERATIONS/VENTES"}



# La recherche sur le champ service.inherited nous donne tous les documents
# définis par la DSI et ses parents.

GET normes/normes/_search
{
  "query": {
    "match": { "service.inherited": "DG/DSI" }
  }
}

# idem pour DG/DSI/SERVICE.

GET normes/normes/_search
{
  "query": {
    "match": { "service.inherited": "DG/DSI/ETUDES" }
  }
}

#
# Cet exercice est terminé.
# Merci d'avoir suivi jusqu'au bout.
#
#

DELETE devoxx*
DELETE _template/devoxx
DELETE normes
DELETE citations
```