[WIP] Janus - Document de conception

Auteur Envole

Date de génération 23/09/2020

Table des matières

Introduction

Contexte et visée de ce document

Ce document décrit les principes de fonctionnement de Janus, un service web de gestion de profils dynamiques.

L'objectif de ce document est d'apporter au lecteur une vision globale des fonctionnalités proposées par ce service web et les cas d'usages envisagés pour celui-ci.

Présentation de Janus

Janus est un service (voir un "microservice") web doté d'une API REST (avec format d'échange JSON) permettant de centraliser la gestion de profils utilisateurs.

À partir de fournisseurs de données d'identité (annuaire LDAP, base de données, etc) il dérive une fiche utilisateur avec des attributs multiples.

De manière équivalente, un ensemble de groupes peut être collecté/consolidé depuis différentes sources et associé à l'utilisateur.

Cet utilisateur et ces groupes sont ensuite utilisables pour identifier les ressources accessibles à partir de règles dynamiques définies par les administrateurs et évaluées à la demande par le service.

En résumé, Janus essaye de répondre à la question "Quelles ressources sont à disposition d'un utilisateur donné ?".

Terminologie

- **UID** Un UID est un identifiant unique associé à une identité. Il est noté `userId` dans la documentation de l'API.
- **Utilisateur** Un utilisateur est un ensemble d'attributs nommés associés à une identité.
- **Groupe** Un groupe est une étiquette à laquelle se rattache un ou plusieurs utilisateurs, ou un ou plusieurs autres groupes (formant ainsi une hiérarchie).
- **Règle** Une règle est une condition (exprimée sous la forme d'un DSL) permettant d'associer/dissocier un utilisateur avec des ressources ou des groupes.
- **Ressource** Une ressource est la représentation d'un contenu (Widget, URL, autre…) auquel un utilisateur pourrait avoir accès.
- **Fournisseur de données d'identité** Un fournisseur de données d'identité est un service externe de stockage des données utilisateurs (annuaire LDAP, base de données). Janus peut utiliser ces fournisseurs pour consolider les fiches utilisateurs stockées en interne.

Règles dynamiques

Afin de consolider les informations récoltées à partir des différents fournisseurs d'identité, Janus utilise un moteur de règles métiers qui sont définies par les consommateur du service.

Ces règles peuvent être créées via l'API de Janus. Elles prennent la forme d'un DSL (Domain Specific Language).

Dans le cadre de Janus, le moteur de règles est mobilisé pour 2 procédures:

1. La consolidation des groupes associés à un utilisateur
2. La sélection des ressources disponibles à un utilisateur

> **N.B.** La sélection des ressources (2) utilise les données produites par la consolidation des groupes (1).

Exemple d'un flux de travail

Fournisseurs d'identité

Dans ce cas d'exemple, 2 fournisseurs d'identité sont configurés (voir section Annexes):

- `my-database` - Base de données SQL
- `my-ldap` - Annuaire LDAP

Données initiales

Voici les données extraites et fusionnées à partir d'un UID (ici `jdoe`) fourni par l'utilisateur de l'API de Janus.

> **N.B.** Les données sont représentées ici au format JSON pour faciliter leur compréhension.

Utilisateur

```
{
    "uid":          { "value": "jdoe",              "source": "consumer" },
    "firstName":    { "value": "John",              "source": "my-database" },
    "lastName":     { "value": "Doe",               "source": "my-database" },
    "email":        { "value": "jdoe@myorg.com",    "source": "my-ldap" },
}
```

Groupes

```
[
    { "value": "admin",         "source": "my-ldap" },
    { "value": "dsi",           "source": "my-ldap" },
]
```

> **N.B.** Seul le fournisseur d'identité `my-ldap` est configuré pour collecter des groupes ici.

Consolidation des groupes

Imaginons que les règles de groupes suivantes soient renseignées dans Janus:

- **Règle 1** Suppression par défaut du groupe `admin`

  ```
  remove_group('admin')
  ```

- **Règle 2** Ajout d'un groupe par défaut correspondant à l'UID de l'utilisateur

  ```
  add_group(user.uid)
  ```

- **Règle 3** Ajout d'un groupe `equipe-tech` si l'utilisateur est dans le groupe `dsi`

  ```
  if has_group('dsi'):
      add_group('equipe-tech')
  ```

> **N.B.** Les règles ci-dessus sont écrites en pseudo-code.

> Des méthodes métiers (ex: `add_group()` et `remove_group()`) sont implémentées afin de permettre aux règles de modifier la liste des groupes de l'utilisateur.

Le moteur de règles exécute **séquentiellement** les règles.

À la sortie d'exécution du moteur de règles, les groupes associés à l'utilisateur sont les suivants:

```
[
    { "value": "dsi",           "source": "my-ldap" },
    { "value": "equipe-tech",   "source": "rule" },
    { "value": "jdoe",          "source": "rule" },
]
```

Filtrage des ressources

Imaginons que les ressources soient référencées dans Janus:

```
[
    {
        "type": "widget",
        "owner": "consumer-1",
        "attributes": {
            "widgetType": "iframe",
            "name": "calendar",
            "url": "http://myorg.com/calendar-widget.html"
        }
    },
    {
        "type": "widget",
        "owner": "consumer-1",
        "attributes": {
            "widgetType": "iframe",
            "name": "supervision",
            "url": "http://myorg.com/supervision-widget.html",
        }
    },
    {
        "type": "file",
        "owner": "consumer-2",
        "attributes": {
            "fileOwner": "emacgregor",
            "name": "Activity-Report.pdf",
            "url": "http://myorg.com/files/098f6bcd4621d373cade4e832627b4f6",
        }
    }
]
```

> **N.B.** Chaque type de ressource définit un ensemble d'attributs qui lui est propre. Voir section API.

Imaginons que les règles de filtrage de ressources suivantes soient définies:

- **Règle 1** Ajout du widget `supervision` si l'utilisateur est dans le groupe `equipe-tech`

```
if resource.type == "widget" and resource.attributes.name == "supervision" and has_group(groups,
        "equipe-tech"):
    add_resource(resource)
```

- **Règle 2** Ajout du widget `calendar` par défaut

```
if resource.type == "widget" and name == "calendar":
    add_resource(resource)
```

À la sortie d'exécution du moteur de règles, les ressources disponibles pour l'utilisateur sont les suivantes:

```
[
    {
        "type": "widget",
        "owner": "consumer-1",
        "attributes": {
            "widgetType": "iframe",
            "name": "calendar",
            "url": "http://myorg.com/calendar-widget.html"
        }
    },
    {
        "type": "widget",
        "owner": "consumer-1",
        "attributes": {
            "widgetType": "iframe",
            "name": "supervision",
            "url": "http://myorg.com/supervision-widget.html",
        }
    }
]
```

Cas d’usage

Application utilisant un annuaire LDAP et/ou base de données pour l’authentification

> TODO

Application utilisant un service d’authentification déporté (ex: CAS/OpenID Connect)

> TODO

API

L'ensemble des chemins des API sont préfixés par `/api/{version}` où `{version}` correspond à la version actuelle de l'API.

Authentification

L'authentification des consommateurs de l'API du service Janus s'effectue via l'utilisation d'une clé d'accès générée par l'administrateur.

La clé d'accès du consommateur doit être transmise via l'entête HTTP `Authorization` avec chaque requête de la manière suivante :

```
Authorization: Bearer <key>
```

Autorisation

Chaque consommateur de l'API peut générer une ou plusieurs clés avec des niveaux d'autorisation et des durées de validité distincts.

Deux niveaux d'autorisation sont possibles pour une clé :

- `read` - Autorisation d'utiliser l'ensemble des points d'entrée de lecture des données
- `write` - Autorisation d'utiliser l'ensemble des points d'entrée d'écriture ET de lecture des données

Erreurs

Les erreurs retournées par l'API suivent la RFC 7807.

Gestion des consommateurs

`GET /consumers`

> TODO

`POST /consumers`

> TODO

`GET /consumers/{consumerId}/keys`

> TODO

`POST /consumers/{consumerId}/keys`

> TODO

`DELETE /consumers/keys/{kId}`

> TODO

Gestion des utilisateurs

`GET /users/{userId}`

Retourne la fiche utilisateur consolidé correspondant à l'identité `{userId}`.

Corps de requête

Aucun

Paramètres

- `userId` Identifiant unique de l'utilisateur

Retours

Code HTTP	Type	Corps de réponse
`200` - OK	Succès	`User` (voir modèles de données)

`GET /users/{userId}/resources?refresh=yes|no`

Retourne la liste des ressources disponibles pour le l'utilisateur correspondant à l'identité `{userId}`.

Corps de requête

Aucun

Paramètres

- `userId` Identifiant unique de l'utilisateur
- `refresh` Force le rafraichissement (calcul) des ressources
  - Paramètre optionnel
  - Valeurs possibles: `yes` | `no`
  - Valeur par défaut: `no`

Retours

Code HTTP	Type	Corps de réponse
`200` - OK	Succès	`[]Resource` (voir modèles de données)

`GET /users/{userId}/favorites?refresh=yes|no`

Corps de requête

Aucun

Paramètres

- `userId` Identifiant unique de l'utilisateur
- `refresh` Force le rafraichissement (calcul) des favoris
  - Paramètre optionnel
  - Valeurs possibles: `yes` | `no`
  - Valeur par défaut: `no`

Retours

Code HTTP	Type	Corps de réponse
`200` - OK	Succès	`[]Favorite` (voir modèles de données)

`GET /users/{userId}/groups?refresh=yes|no`

Retourne la liste des groupes contenant l'utilisateur correspondant à l'identité `{userId}`.

Corps de requête

Aucun

Paramètres

- `userId` Identifiant unique de l'utilisateur
- `refresh` Force le rafraichissement (calcul) des ressources
  - Paramètre optionnel
  - Valeurs possibles: `yes` | `no`
  - Valeur par défaut: `no`

Retours

Code HTTP	Type	Corps de réponse
`200` - OK	Succès	`[]Group` (voir modèles de données)

`POST /users/resources`

Corps de requête

`UserResourcesRequest` (voir modèles de données)

Paramètres

Aucun

Retours

Code HTTP	Type	Corps de réponse
`200` - OK	Succès	`[]Resource` (voir modèles de données)

`POST /users/{userId}/groups`

Corps de requête

`NewLocalGroupRequest` (voir modèles de données)

Paramètres

- `userId` Identifiant unique de l'utilisateur

Retours

Code HTTP	Type	Corps de réponse
`200` - OK	Succès	`Group` (voir modèles de données)

`DELETE /users/{userId}/groups/{groupId}`

Corps de requête

`NewLocalGroupRequest` (voir modèles de données)

Paramètres

- `userId` Identifiant unique de l'utilisateur
- `groupId` Identifiant unique du groupe local

Retours

Code HTTP	Type	Corps de réponse
`204` - No Content	Succès	*Aucun*

Gestion des ressources

`GET /resources`

> TODO

`POST /resources`

> TODO

`GET /resources/{resourceType}`

> TODO

`GET /resources/{resourceType}/{resourceId}`

> TODO

Gestion des règles

`GET /rules/resources/{resourceType}`

> TODO

`POST /rules/resources/{resourceType}`

> TODO

`GET /rules/resources/{resourceType}/{ruleId}`

> TODO

`PUT /rules/resources/{resourceType}/{ruleId}`

> TODO

`DELETE /rules/resources/{resourceType}/{ruleId}`

> TODO

`GET /rules/groups`

> TODO

`POST /rules/groups`

> TODO

`PUT /rules/groups/{ruleId}`

> TODO

`DELETE /rules/groups/{ruleId}`

> TODO

Gestion des groupes locaux

`GET /groups`

> TODO

`POST /groups`

> TODO

`DELETE /users/{userId}/groups/{groupId}`

> TODO

Modèles de données

`UserResourcesRequest`

> TODO

`NewLocalGroupRequest`

> TODO

`User`

> TODO

`Key`

> TODO

`Group`

> TODO

`ResourceDefinition`

> TODO

`Resource`

> TODO

`Favorite`

> TODO

Annexes

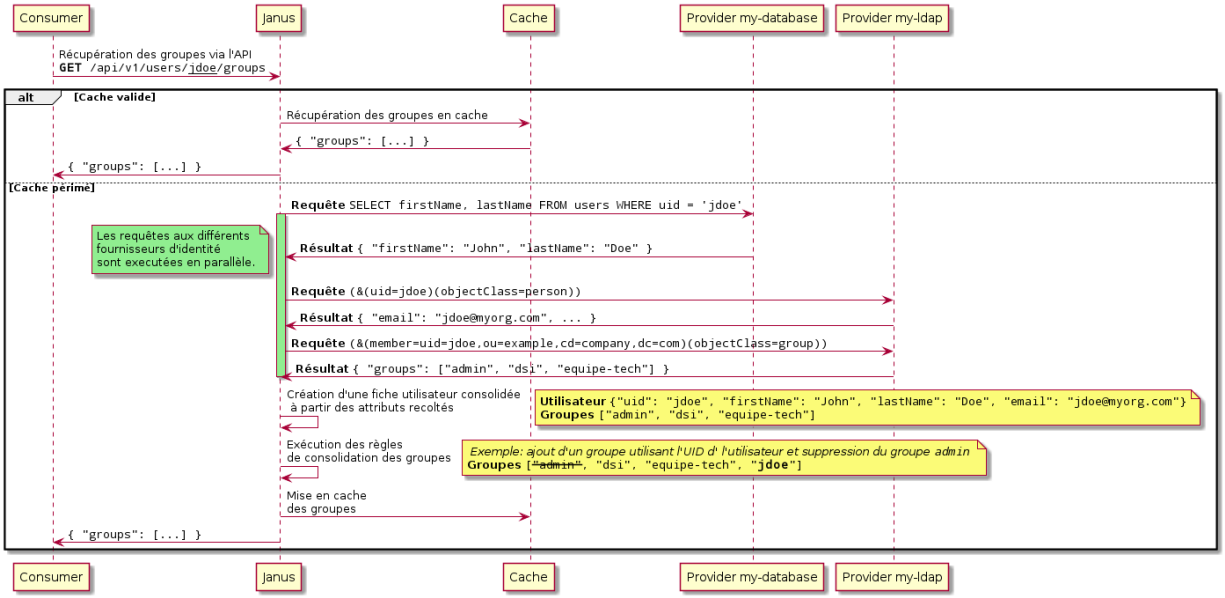
Exemple de fichier de configuration

```yaml
logLevel: INFO

# Configuration du serveur HTTP
http:
    address: '0.0.0.0:8443'

    # Configuration TLS (non exhaustive)
    # Pour désactiver TLS, passer le paramètre à 'null':
    # tls: ~
    tls:
        cert: my-server.crt
        key: my-server.key

# Il est possible de compléter/surcharger la configuration
# du serveur HTTP via l'inclusion de fichiers externes
http_include:
    - additional-http.yml

# Configuration du stockage des données locales de Janus
storage:
    type: local
    options:
        dataDir: /var/lib/janus/data

# Il est possible de compléter/surcharger la configuration
# du stockage via l'inclusion de fichiers externes
storage_include:
    - additional-storage.yml

# Configuration des fournisseurs d'identité
providers:

    # Configuration d'un fournisseur d'identité de type OpenLDAP
    my-ldap:
        # Définition du type de fournisseur d'identité
        type: ldap

        # Options de connexion
        options:
            host: my-ldap.lan
            port: 636
            user: "cn=John Doe, dc=example, dc=org"
            password: NotSoSecret

        # Définition de la requête de récupération des attributs
        # liés à l'identité
        user_query: (&(cn=:uid)(objectClass=person))

        # Définition de la requête de récupération des groupes
        # liés à l'identité
        groups_query: (&(member=uid=:uid,ou=example,cd=company,dc=com)(objectClass=group))

    # Configuration d'un fournisseur d'identité de type base de données SQL (MySQL/PostgreSQL/etc)
    my-database:
        type: sql
        options:
            driver: mysql
            dsn: johndoe:NotSoSecret@tcp(my-database:3306)/my-db
        user_query: SELECT uid, firstName, lastName FROM users WHERE uid = :uid
        # Ce fournisseur d'identité de produit pas de groupes
        groups_query: ~


# Il est possible de compléter/surcharger la configuration
# des fournisseurs via l'inclusion de fichiers externes
providers_include:
    - additional-provider.yml

# Configuration des attributs des utilisateurs
user:

    # Définition du temps de cache global
    # Cette valeur peut être surchargée individuellement
    # sur chaque attribut
    cache_ttl: 1h

    # Définition des attributs des utilisateurs
    attributes:
        # On définit un attribut "email" de type chaine de caractères
        # extrait des données produites par le fournisseur d'identité "my-ldap"
        email:
            provider: my-ldap
            from: email
```

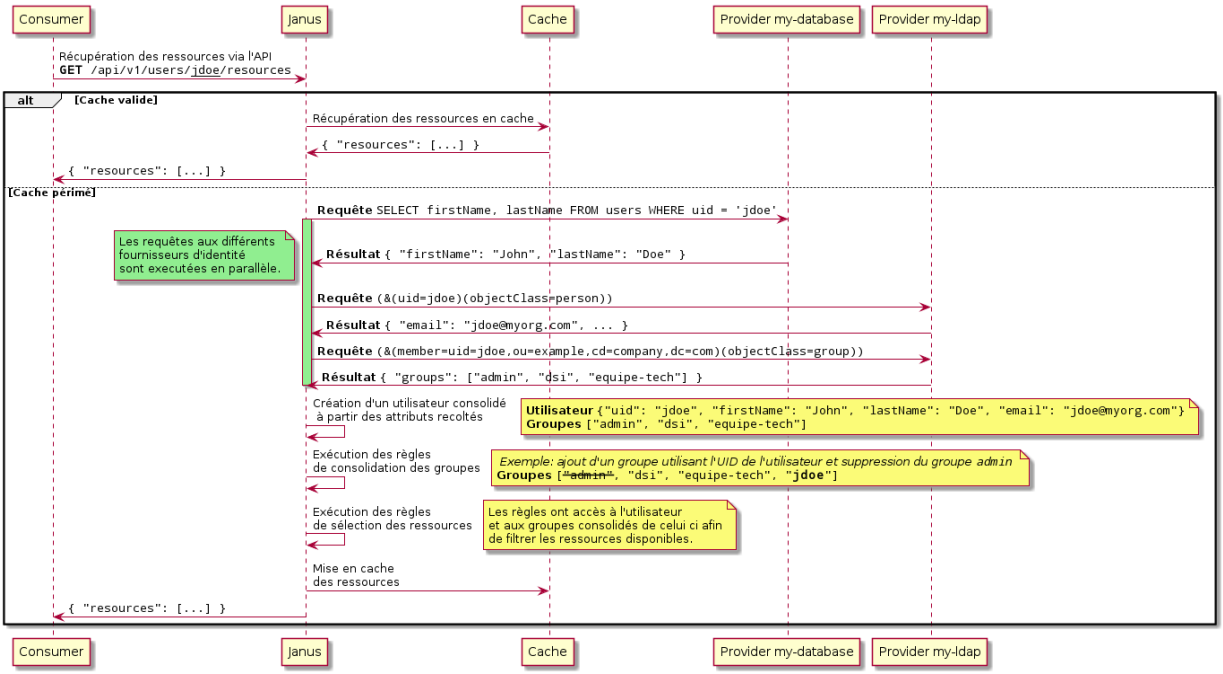
```yaml
            type: string
            # Définition de la valeur par défaut
            defaultValue: ""
            # On indique que la valeur de cet attribut peut être mise en cache
            # pendant 30 minutes avant de devoir être rafraichie.
            cache_ttl: 30m
            # Cet attribut ne peut pas être modifié via Janus
            readOnly: yes

        # Définition équivalente en ciblant le fournisseur d'identité "my-database"
        firstName:
            provider: my-database
            from: firstName
            type: string
            cache_ttl: 30m
            defaultValue: ""
            # Cet attribut peut être modifié via Janus
            # Une modification invalide automatiquement le cache local
            write_query: UPDATE users SET firstName = :firstName where uid = :uid

        lastName:
            provider: my-database
            from: lastName
            type: string
            defaultValue: ""
            cache_ttl: 30m
            readOnly: yes

# Il est possible de compléter/surcharger la configuration de définition des fiches utilisateurs via
        l'inclusion de fichiers externes
user_include:
    - additional-user-config.yml
```

Exemple de flux de travail

Récupération des groupes d'un utilisateur



Récupération des ressources d'un utilisateur


Consumer
Janus
Cache
Provider my-database
Provider my-ldap
Récupération des ressources via l'API
GET /api/v1/users/jdoe/resources
alt
[Cache valide]
Récupération des ressources en cache
{ "resources": [...] }
{ "resources": [...] }
[Cache périmé]
Requête SELECT firstName, lastName FROM users WHERE uid = 'jdoe'
Les requêtes aux différents fournisseurs d'identité sont executées en parallèle.
Résultat { "firstName": "John", "lastName": "Doe" }
Requête (&(uid=jdoe)(objectClass=person))
Résultat { "email": "jdoe@myorg.com", ... }
Requête (&(member=uid=jdoe,ou=example,cd=company,dc=com)(objectClass=group))
Résultat { "groups": ["admin", "dsi", "equipe-tech"] }
Création d'un utilisateur consolidé à partir des attributs recoltés
Utilisateur {"uid": "jdoe", "firstName": "John", "lastName": "Doe", "email": "jdoe@myorg.com"}
Groupes ["admin", "dsi", "equipe-tech"]
Exécution des règles de consolidation des groupes
Exemple: ajout d'un groupe utilisant l'UID de l'utilisateur et suppression du groupe admin
Groupes ["admin", "dsi", "equipe-tech", "jdoe"]
Exécution des règles de sélection des ressources
Les règles ont accès à l'utilisateur et aux groupes consolidés de celui ci afin de filtrer les ressources disponibles.
Mise en cache des ressources
{ "resources": [...] }
Consumer
Janus
Cache
Provider my-database
Provider my-ldap

/