Dossier d’architecture technique VITAMUI

Vitam

Mars 2021

Contents

3. Architecture

La solution VITAMUI popose des applications web accessibles depuis un portail. La solution VITAMUI est constitué de différents modules :

- Socle IAM CAS pour la gestion des identités et des accès
- Socle VITAM pour la gestion des archives
- Services d'infrastructure
  - Annuaire de service
  - Gestion des logs centralisée
- Applications web pour les utilisateurs
  - portail
  - gestion des organisations, utilisateurs, profils, etc.
  - Gestion des référentiels de la solution VITAM
  - Gestion d'entrée, accès et recherche d'archives
  - Consultation des journaux d'opérations d'archives etc.
- Services API externes
  - service de gestion des organisations, des utilisateurs, etc.
  - service CAS
  - service de référentiel externe
  - service d'ingest externe
  - service d'archive externe
- Services API internes
  - service des gestion des organisations, des utilisateurs, etc.

- service de gestion de la sécurité
- service de référentiel interne
- service d'ingest interne
- service d'archive interne

- Les services API internes communiquent avec les API externes VITAM.

  Schéma de l'architecture fonctionnelle VITAMUI:

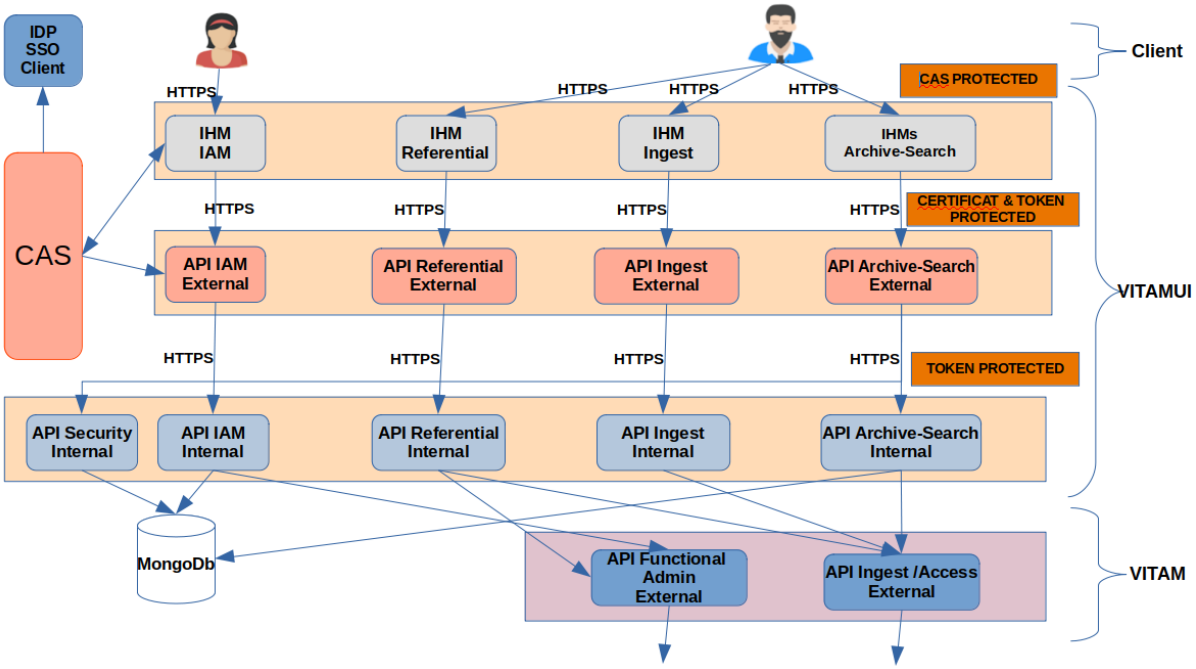

Figure 1: Architecture fonctionnelle

3.1. Applications Web

Les applications Web constituent les IHM de la solution. Elles sont accessibles depuis le portail de la solution. L'authentification d'un utilisateur dans une application cliente se fait par l'intermédiaire de l'IAM CAS. Une application cliente est constituée de 2 parties.

- Interface utilisateur Front (IHM WEB) qui donne accès aux fonctionnalités via un navigateur
- Interface utilisateur Back (Service BackOffice) qui gère la communication avec CAS et les accès aux API externes

Une double authentification est nécessaire pour qu'un utilisateur puissent accéder aux API externes :

- Le service UI Back de l'application cliente doit posséder un certificat reconnu par la solution
- L'utilisateur de l'application cliente doit être authentifié dans la solution (par CAS) et posséder un token valide

Les applications de base :

- portal : application portail donnant accès aux applications
- identity : application pour gérer les organisations, utilisateurs, profils, etc.

3.2. Services externes

Les services externes exposent des API REST publiques accessibles en HTTPS. Ces services constituent une porte d'accès aux services internes et assurent principalement un rôle de sécurisation des ressources internes.

La connexion d'une application cliente à un service externe nécessite le partage de certificats X509 client et serveur dans le cadre d'un processus d'authentification mutuel (Machine To Machine/M2M). Dans la solution VITAMUI, les certificats des clients sont associés à un contexte de sécurité stocké dans une collection MongoDb gérée par le service security_internal. D'autre part, les utilisateurs clients sont identifiés et authentifiés dans les services externes par le token fourni par CAS et transmis dans les headers des requêtes REST en HTTPS.

Le service externe a pour responsabilité de sécuriser les accès en effectuant les différentes étapes de vérifications des droits (générale, tenant, rôles, groupes, etc.) et de déterminer les droits résultants du client à l'origine de la requête, en réalisant l'intersection des droits applicatifs, définis dans le contexte de sécurité, avec les droits issus des profils de l'utilisateur. Le service externe s'assure ensuite que le client possède bien les droits pour accéder à la ressource demandée.

Les services externes s'auto-déclarent au démarrage dans l'annuaire de service Consul.

Les services disposent d'API REST pour suivre leur état et leur activité. Ces API ne sont pas accessibles publiquement.

- API Status pour connaitre la disponibilité du service (utilisé par Consul)
- API Health (basée sur SpringBoot) pour suivre l'activité

Les services génèrent les logs techniques dans la solution de log centralisée basée sur ELK.

3.2.1. Service iam-external

- Description : service externe pour la gestion des organisations, utilisateurs, profils, etc.
- Contraintes
- API swagger

3.2.2. Service cas-server

- Description : service d'authentification nécessaire et accessible uniquement par l'IAM CAS
- Contraintes
- API swagger

3.2.3. Service referential-external

- Description : service externe pour la gestion des référentiels de la solution logicielle VITAM.

Le service de référentiel externe a pour responsabilité la réception, la sécurisation des resources internes de gestion des référentiels, et la communication sécurisée avec les couches internes. Le service de référentiel externe est composé de plusieurs

points d'APIs: - API des contrats d'accès (/referential/accesscontract) - API des contrats d'entrées (/referential/ingestcontract) - API des contrats de gestion (/referential/managementcontract) - API des services agents (/referential/agency) - API des formats (/referential/fileformat) - API des ontologies (/referential/ontology) - API des profils d'archivages (/referential/profile) - API des règles de gestion (/referential/profile) - API des profils de sécurité (/referential/security-profile) - API des contexts applicatifs (/referential/context) - API des opérations permettant le lancement différents audits (cohérence, valeur probante . . .).

3.2.4. Service ingest-external

- Description : service externe pour la gestion des opérations d'entrées d'archives de la solution logicielle VITAM.

Le service d'ingest externe a pour responsabilité la réception, la sécurisation des resources internes de versement, et la communication sécurisée avec les couches internes. Le service d'ingest externe est composé de plusieurs points d'APIs: - API de versement des archives permettant la consommation des flux d'archives (/v1/ingest/upload) - API de visualisation des journaux d'opération des opérations d'entrées (API /v1/ingest) - API de visualisation détaillé d'un journal d'une opération d'entrées (/v1/ingest/{id}) - API permettant le téléchargement d'un rapport sous forme ODT d'une opération d'entrée (/v1/ingest/odtreport/{id}) - API commune est utilisé pour le téléchargement du Manifest et de l'ATR (Archival Transfer Reply) d'une opération d'entrée. (Manifest: /logbooks/operations/{id}/download/manifest, ATR: /logbooks/operations/{id}/download/atr)

3.2.5. Service archive-search-external

- Description : service externe pour la gestion d'accès et la recherche d'archives de la solution logicielle VITAM.

Le service d'archive externe a pour responsabilité la réception, la sécurisation des resources internes, et la communication sécurisée avec les couches internes d'accès aux archives. Le service d'archive externe est composé de plusieurs points d'APIs: - API de recherche des archive par requetes (/archive-search/search) - API de recherche des unités archivistiques (/archive-search/archiveunit/{id}) - API de recherche des arbres de positionnement et plans de classement (/archive-search/filingholdingscheme) - API de téléchargement des objets (/archive-search/downloadobjectfromunit/{id}) - API d'export des résultats sous format csv (/export-csv-search)

3.3. Services internes

Les services internes offrent des API REST accessibles en HTTPS uniquement depuis les services externes ou internes. Les API de ces services ne sont donc pas exposées publiquement. Les services internes implémentent les fonctionnalités de base de la solution ainsi que les fonctionnalités métiers. En fonction des besoins, les services internes peuvent être amenés à journaliser des évènements dans le logbook des opérations du socle VITAM.

Les utilisateurs sont identifiés dans les services internes grâce au token transmis dans les headers des requêtes HTTPS. L'utililisation du protocole HTTPS permet de chiffrer les tokens et les informations sensibles qui sont transportées dans les

requêtes. Les services internes peuvent éventuellement vérifier les droits d'accès de l'utilisateur avant d'accéder aux ressources.

Les services internes s'auto-déclarent au démarrage dans l'annuaire de service Consul.

Les services disposent d'API REST pour suivre leur état et leur activité.

- API Status pour connaitre la disponibilité du service (utilisé par Consul)
- API Health (basée sur SpringBoot) pour suivre l'activité du service

Les services génèrent les logs techniques dans la solution de log centralisée basée sur ELK.

3.3.1. Service iam-internal

- Description : service d'administration des clients, des utilisateurs et des profils, portail
- Contraintes
- API swagger
- Modèle de données

3.3.2. Service security-internal

- Description : service de gestion de la sécurité applicative
- Contraintes
- API swagger
- Modèle de données

3.3.3. Service referential-internal

- Description : service interne pour la gestion des référentiels de la solution logicielle VITAM.

Le service de référentiel interne reçoit les requtes du client référentiel externe, et communique avec VITAM via les clients Admin/Access pour la récupération des données. Le service de référentiel interne est composé de plusieurs points d'APIs: - API des contrats d'accès (/referential/accesscontract) - API des contrats d'entrées (/referential/ingestcontract) - API des contrats de gestion (/referential/managementcontract) - API des services agents (/referential/agency) - API des formats (/referential/fileformat) - API des ontologies (/referential/ontology) - API des profils d'archivages (/referential/profile) - API des règles de gestion (/referential/profile) - API des profils de sécurité (/referential/security-profile) - API des contexts applicatifs (/referential/context) - API des opérations permettant le lancement différents audits (cohérence, valeur probante . . .).

pour plus d'information: voir la documentation des référentiels

3.3.4. Service ingest-internal

- Description : service interne pour la gestion des opérations d'entrées d'archives de la solution logicielle VITAM.

Le service d'ingest interne a pour responsabilité la réception, et la communication sécurisée avec les couches externes de VITAM. Le service d'ingest interne est composé de plusieurs points d'APIs: - API de versement des archives permettant la consommation des flux d'archives (/v1/ingest/upload) - API de visualisation des journaux d'opération des opérations d'entrées (API /v1/ingest) - API de visualisation détaillé d'un journal d'une opération d'entrées (/v1/ingest/{id}) - API permettant le téléchargement d'un rapport sous forme ODT d'une opération d'entrée (/v1/ingest/odtreport/{id}) - API commune est utilisé pour le téléchargement du Manifest et de l'ATR (Archival Transfer Reply) d'une opération d'entrée. (Manifest: /logbooks/operations/{id}/download/manifest, ATR: /logbooks/operations/{id}/download/atr)

Ce service est configuré pour qu'il puisse communiquer avec la zone d'accès de la solution logicielle VITAM. Pour aller plus loin: 1, 2

3.3.5. Service archive-search-internal

- Description : service interne pour la gestion d'accès et la recherche d'archives de la solution logicielle VITAM.

Le service d'archive interne a pour responsabilité la réception, et la communication sécurisée avec les couches externes VITAM. Le service d'archive interne est composé de plusieurs points d'APIs: - API de recherche des archive par requetes (/archive-search/search) - API de recherche des unités archivistiques (/archive-search/archiveunit/{id}) - API de recherche des arbres de positionnement et plans de classement (/archive-search/filingholdingscheme) - API de téléchargement des objets (/archive-search/downloadobjectfromunit/{id}) - API d'export des résultats sous format csv (/export-csv-search)

3.4. Services d'infrastructure

La solution utilise plusieurs services d'infrastructures :

- l'annuaire de service. basé sur l'outil Consul, il permet de localiser les services actifs dans l'infrastructure
- le service de gestion des logs rsyslog. Il permet de collecter, gérer et de transporter les logs
- l'outil de centralisation et de recherche des logs ELK (Elasticsearch / Logstash / Kibana)

Les services d'infrastructures sont basés et mutualisés avec VITAM. Vous pouvez donc vous référer aux documentations VITAM pour avoir un détail précis du fonctionnement de ces services :

- Doc VITAM : Chaîne de log - rsyslog / ELK
- Doc VITAM : Annuaire de service consul

3.5. Service d'archivage VITAM

Le service d'archivage se base sur le socle logiciel VITAM a pour fonction de gérer l'archivage des documents. Il apporte une forte garantie de sécurité et de disponibilité pour les archives.

Ses principales caractéristiques sont :

- Fonctions d'archivage : versement, recherches, consultation, administration , structurations arborescentes, référentiels. . .
- Accès aux unités d'archives via un service de requêtage performant
- Garantie de la valeur probante par le respect des normes en vigueur, par la traçabilité des opérations et du cycle de vie des objets et leur journalisation sécurisée
- Sécurité et la robustesse : la gestion applicative du stockage permet une réplication des données, métadonnées, index et journaux sur plusieurs sites et plusieurs offres contrôlées. L'architecture interne du stockage assure la capacité de reconstruire le système à partir d'une seule offre, en une fois ou au fil de l'eau
- La possibilité d'une utilisation mutualisée grâce à la gestion multi-tenant des archives
- Offres de stockage multiple
- Capacité à absorber de fortes volumétries de données

La documentation de la solution VITAM est disponible ici.

3.6. Service d'authentification

3.6.1. Authentification des applications externes

A l'initialisation de la connexion HTTPS d'une application cliente à un service API VITAMUI, un processus d'authentification mutuelle entre le client et le serveur basé sur des certificats x509 est mis en oeuvre. Le service VITAMUI contrôle le certificat applicatif x509 transmis par le client pour s'assurer de sa validité. En cas de certificat invalide, expiré ou absent du truststore du service VITAMUI, la connexion échoue.

Il est fortement recommandé d'utiliser des certificats officiels pour toutes les authentifications publiques.

3.6.2. Authentification des utilisateurs externes

Lorsque la connexion applicative a été réalisée avec succès, la solution VITAMUI récupère dans la base MongoDB le contexte de sécurité applicatif associé au certificat client. Le contexte de sécurité applicatif définit les autorisations d'accès aux différents services (rôles) et le périmètre d'accès (tenant) de l'application. Un même contexte peut être associé à plusieurs certificats. L'utilisateur se voit alors attribuer l'intersection des rôles et tenants du contexte de sécurité applicatif et de ses profils.

La cinématique est la suivante :

1. Initialisation de la connexion par l'application cliente
2. Vérification du certificat client transmis par l'application
3. Vérification du contexte de sécurité associé au certificat
4. Récupération des profils (rôles & tenants) de l'utilisateur
5. Intersection des rôles et tenants entre le conexte de sécurité et les profils

6. L'utilisateur peut accéder aux ressources autorisées

Il est ainsi possible de limiter les risques d'élévations de privilèges en dissociant les contextes applicatifs de 2 instances d'une même application.

Par exemple, dans une première instance de l'application exposée sur un réseau public et associé à un contexte applicatif possédant des droits limités, un administrateur ne pourra pas accéder à des fonctions d'admnistration. En revanche, une deuxième instance, bénéficiant d'un contexte applicatif adéquat, sur un réseau protégé et accessible à ce même administrateur permettra d'effectuer des opérations à haut privilège.

3.6.3. Service d'authentification centralisé CAS

Dans VITAMUI, l'authentification des utilisateurs est réalisée au moyen du service CAS. CAS est un service d'authentification centralisé (SSO et fédération d'identité), développé depuis 2004 par une communauté open source, et destiné aux applications Web .

CAS propose les fonctionnalités suivantes :

- un protocole (CAS protocol) ouvert et documenté
- la prise en charge de moteur de stockage variés (LDAP, base de données, X.509, 2-facteur)
- la prise en charge de plusieurs protocoles (CAS, SAML, OAuth, OpenID)
- des bibliothèques de clients pour Java, .Net, PHP, Perl, Apache, uPortal, etc.
- l'intégration native avec uPortal, BlueSocket, TikiWiki, Mule, Liferay, Moodle, etc.

Architecture CAS

Dans la solution VITAMUI, CAS porte uniquement le processus d'authentification (délégué ou non) avec les informations (tickets, cookies, etc.) nécessaires au bon fonctionnement de l'authentification. En revanche, toutes les données des utilisateurs (compte, profils, rôles, etc.) sont stockés dans une base MongoDB gérée par les services VITAMUI. Lors du processus d'authentification, CAS récupère les données des utilisateurs via des services REST dédiés et sécurisés dans VITAMUI. Il est important de noter que les crédentials d'accès à la solution, les données des utilisateurs ou des applications ne sont donc jamais stockés dans CAS.

Ce choix simplifie l'exploitation de la solution car il n'est pas nécessaire de migrer les données lors de la mise à jour de CAS.

Protocole CAS

La documentation de CAS est disponible sur internet. CAS est livré sous licence Apache 2.0.

3.6.4. Intégration CAS dans VITAMUI

Les principes généraux de l'implémentation de CAS dans VITAMUI sont les suivants :

- l'email de l'utilisateur assure l'indification de l'utilisateur dans le système
- les applications VITAMUI (ie. Service Provider) raccordées au serveur CAS utilisent le protocole CAS. (Dans VITAMUI, la bibliothèque Spring Security fournit cette fonctionnalité)

- les applications VITAMUI faisant office de services providers sont déclarées dans CAS
- la délégation d'authentification du serveur CAS aux IDP des clients se fait en SAML 2.0
- les IDP SAML utilisés sont déclarés dans VITAMUI et sont stockés dans MongoDB
- la fonction de révocation périodique de mot de passe est assurée par CAS
- l'anti force brute est assurée par le serveur CAS (→ throttling)
- la fonction de récupération de mot de passe et le contrôle de robustesse du mot de passe sont assurés par le module password management de CAS
- l'authentification multi-facteur est assurée par SMS (Le fonctionnement du MFA : page de login CAS, étape supplémentaire est portée par le provider du deuxième facteur) est assurée par CAS
- le service CAS dispose d'un certificat client pour être authentifié par VITAMUI
- dans un environnement web clusterisé, le reverse proxy est configuré pour assurer l'affinité de session nécessaire à la conservation du cookie de session (JSESSIONID) dans l'application WEB

Dans le cas d'un utilisateur n'utilisant pas le SSO :

- le contrôle de robustesse du mot de passe est assuré par le service Identity de VITAMUI
- le chiffrement des mots de passe est assuré par le service Identity de VITAMUI
- le mot de passe est conservé chiffré dans le base MongoDB de VITAMUI

Intégration CAS

3.6.5. Authentification d'un utilisateur non authentifié

Pour un utilisateur, non préalablement authentifiés, l'authentification dans CAS se fait en plusieurs étapes :

1. L'utilisateur tente d'accéder à une page sécurisée et est alors redirigé var CAS
2. une première page est affichée dans CAS pour saisir l'identifiant (unique) et le mot de passe de l'utilisateur
3. selon le domaine email de l'utilisateur et les règles particulières à la délégation d'authentification, CAS délègue l'authentification ou authentifie lui-même l'utilisateur.
    - pas de délégation : une seconde page est affichée pour saisir le mot de passe et le serveur CAS vérifie les credentials auprès du service Identity de VITAMUI
    - délégation : l'utilisateur est redirigé pour authentification sur l'IDP de son organisation en SAML v2
4. CAS demande la création d'un token utilisateur via le service Identity de VITAMUI. Ce token assure l'identification de l'utilisateur dans les API external et internal de VITAMUI.

5. le serveur CAS récupère les informations de l'utilisateur via le service Identity/CAS de VITAMUI
6. l'application récupère le profil de l'utilisateur et son token API
7. lors d'un apple à l'API VITAM, le token est transmis dans le header de la requête.

Protocole détaillé CAS

3.6.6. Authentification d'un utilisateur préalablement authentifié

Si l'utilisateur est déjà authentifié auprès du CAS, aucune page de login ne s'affiche et l'utilisateur est redirigé vers l'application souhaitée, en étant authentifié dans cette application. Suivant les utilisateurs / applications demandées, une authentification multi-facteurs peut être jouée.

3.6.7. Délégation d'authentification

La délégation d'authentification est prise en charge par CAS. Actuellement seul le protocol SAML v2 est supporté.

Les étapes suivantes expliquent comment fonctionne la délégation d'authentification selon le protocole SAML v2 dans le cadre de VITAMUI.

En amont de ce processus, l' IDP (SSO) doit fournir à VITAMUI l'URL associée à son service d'authentification unique (SSO), ainsi que la clé publique qui lui sera nécessaire pour valider les réponses SAML.

Le schéma ci-dessous illustre les étapes et le mécanisme de connexion d'un utilisateur à une application VITAMUI, via un service d'authentification unique basé sur le protocole SAML. La liste numérotée qui suit le diagramme revient en détail sur chacune des étapes.

Connexion à VITAMUI via une délégation d'authentification en SAML v2

Délégation SAML CAS

L'utilisateur tente d'accéder à une application VITAMUI hébergée

1. VITAMUI génère une demande d'authentification SAML, qui est encodée et intégrée dans l'URL associée au service d'authentification unique (SSO) de l'IDP de l'organisation cliente. Le paramètre RelayState, qui contient l'URL encodée de l'application VITAMUI à laquelle tente d'accéder l'utilisateur, est également incorporé dans l'URL d'authentification unique. Il constitue un identificateur opaque qui sera par la suite renvoyé au navigateur de l'utilisateur sans modification ni vérification.
2. VITAMUI envoie une URL de redirection au navigateur de l'utilisateur. Cette URL inclut la demande d'authentification SAML encodée qui doit être envoyée au service d'authentification unique de l'organisation cliente.
3. L'IDP de l'organisation cliente décode la demande SAML et en extrait l'URL du service ACS (Assertion Consumer Service) de VITAMUI et de la destination de l'utilisateur (paramètre RelayState). Il authentifie ensuite l'utilisateur, soit en l'invitant à saisir ses identifiants de connexion, soit en vérifiant ses cookies de session.

4. L'IDP de l'organisation cliente génère une réponse SAML contenant le nom de l'utilisateur authentifié. Conformément aux spécifications SAML 2.0, cette réponse contient les signatures numériques des clés DSA/RSA publiques et privées du de l'organisation cliente.
5. L'IDP de l'organisation cliente encode la réponse SAML et le paramètre RelayState avant de les renvoyer au navigateur de l'utilisateur. Il fournit le mécanisme permettant au navigateur de transmettre ces informations au service ACS de VITAMUI. Par exemple, il peut incorporer la réponse SAML et l'URL de destination dans un formulaire, qui inclut un script JavaScript sur la page qui se charge alors d'envoyer automatiquement le formulaire à VITAMUI.
6. Le service ACS de VITAMUI vérifie la réponse SAML à l'aide de la clé publique du de l'organisation cliente. Si la réponse est validée, l'utilisateur est redirigé vers l'URL de destination.

L'utilisateur est redirigé vers l'URL de destination. Il est désormais connecté à VITAMUI.

3.6.8. Sécurisation de CAS

En production, le serveur CAS sera composé de plusieurs noeuds. Il est nécessaire d'acyiver la sécurité et de configurer : * une définition de services (dans MongoDB) propres aux URLs de production * une configuration Hazelcast adéquate (stockage des sessions SSO).

3.6.9. Activation de la sécurité

Configuration des propriétés de sécurité

La configuration de CAS se trouve dans le fichier YAML applicatif (en développement : cas-server-application-dev.yml). Elle concerne d'abord les trois propriétés suivantes :

```
cas.tgc.secure : cookie de session SSO en HTTPS
cas.tgc.crypto.enabled : cryptage / signature du cookie SSO
cas.webflow.crypto.enabled : cryptage / signature du webflow
```

En production, il est absolument nécessaire que ces trois propriétés soient à true.

Pour la propriété cas.tgc.crypto.enabled à true, il faut définir la clé de cryptage et de signature via les propriétés suivantes :

```
cas.tgc.crypto.encryption.key : clé de cryptage (ex. Jq-ZSJXTtrQ...)
cas.tgc.crypto.signing.key : clé de signature (ex. Qoc3V8oyK98a2Dr6...)
```

Pour la propriété cas.webflow.crypto.enabled à true, il faut définir la clé de cryptage et de signature via les propriétés suivantes :

```
cas.webflow.crypto.encryption.key : clé de cryptage
cas.webflow.crypto.signing.key : clé de signature
```

Si aucune clé n'est définie, le serveur CAS va les créer lui-même, ce qui ne fonctionnera pas car les clés générées seront différentes sur chaque noeud.

En outre, pour la délégation d'authentification et la gestion du du mot de passe, il existe deux propriétés qui sont déjà à true, mais pour lesquelles aucune clé n'a été définie :

```
cas.authn.pac4j.cookie.crypto.enabled : chiffrement & signature pour
                                        la délégation d'authentification

cas.authn.pm.reset.crypto.enabled : chiffrement & signature pour
                                    la gestion du mot de passe.
```

Pour la délégation d'authentification, il faut définir la clé de cryptage et de signature via les propriétés suivantes :

```
cas.authn.pac4j.cookie.crypto.encryption.key : clé de cryptage
cas.authn.pac4j.cookie.crypto.signing.key : clé de signature
```

Pour la gestion du mot de passe, il faut définir la clé de cryptage et de signature via les propriétés suivantes :

```
cas.authn.pm.reset.crypto.encryption.key : clé de cryptage
cas.authn.pm.reset.crypto.signing.key : clé de signature
```

Suppression des accès aux URLs d'auto-administration

Les URLs d'auto-administration du CAS doivent être désactivées. La configuration suivante doit être appliquée :

```
cas.adminPagesSecurity.ip: a
cas.monitor.endpoints.sensitive: true
cas.monitor.endpoints.enabled: false
endpoints.sensitive: true
endpoints.enabled: false
management.security.enabled: false
```

Cette dernière configuration est sans importance du moment que l'URL /status du serveur CAS n'est pas mappée en externe.

3.6.10. Définition des services supportés

Il est nécessaire de fournir lors du déploiement de la solution VITAMUI, la liste des services autorisés à interagir avec CAS en tant que Service Provider. Cette liste permet à CAS de s'assuer que le service est connu avant d'effectuer le callback. La liste des services est stockée lors du déploiement dans la base MongoDB de VITAM UI est accessible uniqument par CAS.

3.6.11. Configuration Hazelcast

Par défaut, les noeuds Hazelcast s'auto-découvrent et les tickets sont partitionnés entre tous les noeuds et chaque ticket a un backup. Il est néanmoins possible de configurer dans CAS des propriétés permettant d'affiner le réglage d'Hazelcast :

```
cas.ticket.registry.hazelcast.cluster.members: 123.456.789.000,123.456.789.001
cas.ticket.registry.hazelcast.cluster.instanceName: localhost
cas.ticket.registry.hazelcast.cluster.port: 5701
```

Ci-dessous sont listées des propriétés permettant une gestion avancée d'Hazelcast :

```
cas.ticket.registry.hazelcast.cluster.tcpipEnabled: true
cas.ticket.registry.hazelcast.cluster.partitionMemberGroupType:
    HOST_AWARE|CUSTOM|PER_MEMBER|ZONE_AWARE|SPI
cas.ticket.registry.hazelcast.cluster.evictionPolicy: LRU
cas.ticket.registry.hazelcast.cluster.maxNoHeartbeatSeconds: 300
cas.ticket.registry.hazelcast.cluster.loggingType: slf4j
cas.ticket.registry.hazelcast.cluster.portAutoIncrement: true
cas.ticket.registry.hazelcast.cluster.maxHeapSizePercentage: 85
cas.ticket.registry.hazelcast.cluster.backupCount: 1
cas.ticket.registry.hazelcast.cluster.asyncBackupCount: 0
cas.ticket.registry.hazelcast.cluster.maxSizePolicy: USED_HEAP_PERCENTAGE
cas.ticket.registry.hazelcast.cluster.timeout: 5
```

Multicast Discovery :

```
cas.ticket.registry.hazelcast.cluster.multicastTrustedInterfaces:
cas.ticket.registry.hazelcast.cluster.multicastEnabled: false
cas.ticket.registry.hazelcast.cluster.multicastPort:
cas.ticket.registry.hazelcast.cluster.multicastGroup:
cas.ticket.registry.hazelcast.cluster.multicastTimeout: 2
cas.ticket.registry.hazelcast.cluster.multicastTimeToLive: 32
```

La documentation pour la génération des clés pour le cluster CAS est disponible ici.

3.6.12. Fonctionnalités

Le serveur CAS VITAMUI est construit sur le serveur CAS Open Source v6.1.x via un mécanisme d'overlay Maven.

Les beans Spring sont chargés via les classes `AppConfig` et `WebflowConfig` déclarées par le fichier `src/main/resources/META-INF/spring.factories`.

Les propriétés spécifiques au client IAM sont mappées en Java via le bean `IamClientConfigurationProperties`.

La configuration est située dans le répertoire `src/main/config` et dans les fichiers `src/main/resources/application.properties` et `src/main/resources/bootstrap.propert`

Une bannière custom est affichée au lancement (`CasEmbeddedContainerUtils`).

Le serveur CAS VITAMUI contient les fonctionnalités suivantes :

Utilisation de MongoDB

Les applications autorisées à s'authentifier sur le serveur CAS sont définies dans une base de données MongoDB.

Cela est géré par la dépendance `cas-server-support-mongo-service-registry`.

Utilisation d'Hazelcast

Les informations nécessaires durant les sessions SSO sont stockées dans Hazelcast.

Cela est géré par la dépendance `cas-server-support-hazelcast-ticket-registry`.

Authentification login/mot de passe

Le `UserAuthenticationHandler` vérifie les credentials de l'utilisateur auprès de l'API IAM et le `UserPrincipalResolver` crée le profil utilisateur authentifié à partir des informations récupérées via l'API IAM.

Après avoir saisi son identifiant (classe `DispatcherAction`): - si l'utilisateur ou son subrogateur est inactif, il est envoyé vers une page dédiée (`casAccountDisabledView`) - si aucun fournisseur d'identité n'est trouvé pour l'utilisateur, il est envoyé vers une page dédiée (`casAccountBadConfigurationView`).

3.6.7. Délégation d'authentification

L'authentification peut être déléguée à un serveur SAML externe.

Cela est géré par la dépendance `cas-server-support-pac4j-webflow`.

Le flow d'authentification a été modifié (classe `CustomLoginWebflowConfigurer`) pour se dérouler en deux étapes : - saisie de l'identifiant - saisie du mot de passe (`src/main/resources/templates/casPwdView.html`) ou redirection vers le serveur SAML externe pour authentification. Cela est géré par l'action `DispatcherAction`.

Cette délégation d'authentification peut être faite de manière transparente si le paramètre `idp` est présent (il est sauvé dans un cookie de session pour mémorisation). Cela est gérée par la classe `CustomDelegatedClientAuthenticationAction`.

Subrogation

Un utilisateur peut subroger un utilisateur authentifié ("il se fait passer pour lui").

Cela est géré par la dépendance `cas-server-support-surrogate-webflow`.

La subrogation est gérée par CAS avec un identifiant contenant à la fois l'identifiant de l'utilisateur et le subrogateur séparé par une virgule.

Pour permettre un affichage séparé des deux informations, elles sont découpées en avance dans les classes `CustomDelegatedClientAuthenticationAction` et `DispatcherAction`.

Pour gérer correctement la subrogation lors d'une délégation d'authentification, le subrogé est sauvegardé en fin d'authentification (`DelegatedSurrogateAuthenticationPostProc`

Le droit de subroger est vérifié auprès de l'API IAM (`IamSurrogateAuthenticationService`).

Le temps de session SSO est allongée dans le cas d'une subrogation générique (`DynamicTicketGrantingTicketFactory`).

Interface graphique customisée

L'interface graphique du serveur CAS est adapté au look and feel VITAMUI.

Les pages HTML modifiées sont dans le répertoire `src/main/resources/templates` et les ressources statiques (JS, CSS, images) sont dans le répertoire `src/main/resources/static`.

Les messages customisés sont dans les fichiers `overriden_messages*.properties` (dans `src/main/resources`).

Les bons logos à afficher sont calculés via les actions `CustomInitialFlowSetupAction` (login) et `GeneralTerminateSessionAction` (logout).

Après une authentification réussie, une page “connexion sécurisée” est affichée avant de rediriger sur l’application demandée. Cela est gérée par l’action : `SelectRedirectAction`.

Double facteur SMS

Dans certains cas, l’authentification nécessite un second facteur sous forme de token reçu par SMS à re-saisir dans l’IHM.

Cela est géré par la dépendance `cas-server-support-simple-mfa`.

Un webflow spécifique est géré dans `src/main/resources/webflow/mfa-simple/mfa-simple-c` pour gérer le cas où l’utilisateur n’a pas de téléphone (`casSmsMissingPhoneView`, classe `CustomSendTokenAction`), le cas du code expiré (`casSmsCodeExpiredView`, classe `CheckMfaTokenAction`) et le fait que le token n’a pas de format CAS spécifique (“CASMFA-”).

Gestion du mot de passe

Le serveur CAS permet également de réinitialiser ou de changer son mot de passe.

Cela est géré par la dépendance `cas-server-support-pm-webflow`.

Le changement de mot de passe est effectué auprès de l’API IAM grâce à la classe `IamPasswordManagementService`.

Les emails envoyés lors de la réinitialisation du mot de passe sont internationalisés grâce aux classes `PmMessageToSend` et `I18NSendPasswordResetInstructionsAction`. Ils sont aussi différents suivant le type d’évènement : réinitialisation standard ou création de compte. Tout comme le temps d’expiration (classe `PmTransientSessionTicketExpirationPolicyBuilder`).

Une API REST dans CAS permet de déclencher la réinitialisation du mot de passe : `ResetPasswordController`.

La classe `TriggerChangePasswordAction` permet de provoquer le changement de mot de l’utilisateur même s’il est déjà authentifié.

La classe `CustomVerifyPasswordResetRequestAction` gère proprement les demandes de réinit de mot de passe expirées.

La durée de vie des tickets transient est réglée à 1 jour (classe `HazelcastTicketRegistryTicket` pour gérer les demandes de réinit de mot de passe lors de la création d’un compte.

Support serveur OAuth

Le serveur CAS se comporte comme un serveur OAuth pour permettre la cinématique “Resource Owner Password flow”.

Cela est géré par la dépendance `cas-server-support-oauth-webflow`.

L’utilisateur est authentifié via ses credentials et des credentials applicatifs et les access tokens OAuth générés sont le token d’authentification VITAM de l’utilisateur (classe `CustomOAuth20DefaultAccessTokenFactory`).

Déconnexion

Pour éviter tout problème avec des sessions applicatives persistantes, la déconnexion détruit toutes les sessions applicatives dans le cas où aucune session SSO n'est trouvée (classe `GeneralTerminateSessionAction`).

Throttling

Le nombre de requêtes d'authentification accepté par le serveur CAS est limité.

Cela est géré par la dépendance `cas-server-support-throttle`.

3.7. Sessions applicatives

3.7.1. Liste des sessions

Il existe 4 sessions définies dans la solution VITAMUI :

- la session applicative Web (cookie JSESSIONID)
- la session des services API (token X-AUTH-TOKEN)
- la session applicative CAS (cookie JSESSIONID / Domaine CAS)
- la session de l'IDP SAML utilisé pour la délégation d'authentification

3.7.2. Séquence de création des sessions

La séquence de création des sessions est liée à l'utisation du protocole CAS et à l'intégration des services API. Dans le processus de connexion, la création des sessions s'effectue dans l'ordre suivant :

1. création par l'application Web du cookie JSESSIONID
2. création de la session SAML (dans le cas d'une délégation d'authentification)
3. création dans CAS du cookie TGC
4. création par CAS dans l'API VITAMUI du token API

   Schéma des sessions applicatives

Sessions Applicatives

3.7.3. Session applicative Web

La session applicative est portée par le cookie JSESSIONID créée dans l'application Web. Le cookie expire à l'issue du délai d'inactivité et sa durée de vie est réinitialisée à chaque utilisation. [A vérifier]

Lorsque la session expire, le cookie est automatiquement recréé par l'application WEB et le client redirigé par un code HTTP 302 vers le service CAS.

Si la session CAS (cookie TGC) a expiré, l'utilisateur doit se reloguer et les sessions CAS (TGC), services API (Token), et si nécessaire SAML, sont recréées. En revanche, si la session CAS est valide, l'utilisateur n'a pas besoin de se reloguer et est directement redirigé sur l'application Web. Dans ce dernier cas, la session des services est conservée et le token n'est pas recréé.

3.7.4. Session des services API

La session des services API est porté par un token. Le token permet l'identification des utilisateurs dans les services API (external et internal) de VITAMUI. Le token expire à l'issue du délai d'inactivité et sa durée de vie est réinitialisée à chaque utilisation.

Lors du processus d'authentification, le resolver de CAS extrait l'identité de l'utilisateur (de la réponse SAML en cas de délégation d'authentification) et appelle le service Identity de VITAMUI pour créer un token conservé dans la base mongoDB.

Le token est fourni aux applications web, mais n'est pas visible dans le navigateur web du client car il est conservé dans la session applicative (JSESSIONID) de l'utilisateur. Dans chaque requête vers les services, le header X-Auth-Token est positionné avec la valeur du token. Avant d'accpter la requête, le service contrôle l'existence du header précédent et vérifie que le token est toujours valide.

Lorsque le token a expiré, les services API génèrent une erreur 401 transmis aux applications web. Lors de la réception d'une erreur 401, l'application web invalide la session applicative (JSESSIONID) concernée, puis effectue une redirection vers le logout CAS (afin de détruire le TGC et la session SAML). L'utilisateur doit obligatoirement se reconnecter pour utiliser à nouveau l'application.

3.7.5. Session CAS

La session CAS est portée par un cookie Ticket-Granting Cookie ou TGC. Le TGC est le cookie de session transmis par le serveur CAS au navigateur du client lors de la phase de login. Ce cookie ne peut être lu ou écrit que par le serveur CAS, sur canal sécurisé (HTTPS). Lors du processus d'authentification, le resolver de CAS extrait l'identité de l'utilisateur (de la réponse SAML en cas de délégation), crée le cookie TGC et un ticket dans l'URL puis stocke ces informations dans le cache HazelCast.

[A vérifier] En cas de délégation d'authentification, si la session CAS a expiré (TGC invalide)

- l'utilisateur doit se reconnecter si la session SAML a expiré
- sinon CAS recrée automatiquement le TGC et le token

Sans délégation d'authentification, l'utilisateur doit se reconnecter systématiquement pour que CAS puisse recréer le TGC et le token.

3.7.6. Session des IDP

La session de l'IDP (Identiy Provider) est propre à chaque IDP SAML. Il existe néanmoins un délai maximum dans CAS pour accepter la délégation d'authentification d'un IDP SAML.

L'utilisateur doit obligatoirement se reconnecter si la session SAML a expiré.

3.7.7. Expiration et cloture des sessions

Il existe deux politiques d'expiration possibles :

- expiration de session par délai d'inactivité : la session expire si aucune action n'est faite (par l'utilisateur) au bout du délai d'inactivité (session Token)

- expiration de session par délai maximum : la session expire au bout du délai maximum depuis la date de création, quelque soit les actions faites par l'utilisateur (Sessions Applicatives & CAS)

A l'expiration de la session CAS, toutes les sessions applicatives sont supprimées. [Quid du token ?] Les sessions applicatives sont détruites via une redirection dans le navigateur. [A Préciser le fonctionnement via le navigateur vs certificats]

Le logout d'une application web invalide la session applicative concernée, puis effectue une redirection vers le logout CAS afin de détruire la session CAS (destruction du TGC), la session API (destruction du token) et la session SAML. [à confirmer]

A près un logout ou l'utilisateur doit obligatoirement se reconnecter pour utiliser à nouveau l'application.

3.7.8. Paramétrages des sessions

Toutes ces valeurs sont paramétrables dans l'instance de la solution.

Compte principal : [à confirmer]

- la session applicative JSESSIONID : 15 minutes (délai d'inactivité) :
- la session du token : 165 minutes (délai maximum) :
- la session CAS TGC : 170 minutes (délai maximum) :
- délai maximum dans CAS pour accepter la délégation d'authentification : 14 jours (délai maximum)

Dans le cas de la subrogation, on a : [à confirmer]

- la session applicative JSESSIONID : 15 minutes (délai d'inactivité) :
- la session du token : 165 minutes (délai maximum) :
- la session CAS TGC : 170 minutes (délai maximum) :
- délai maximum dans CAS pour accepter la délégation d'authentification : 14 jours (délai maximum)

3.8. Profils et rôles

3.8.1. Groupe de profils

Un groupe de profils contient (entre autres) les informations suivantes :

- liste de profils
- niveau

Un groupe de profils est rattaché à un utilisateur, lui-même rattaché à une organisation. Un groupe de profil peut contenir des profils avec des tenants différents. Pour un tenant donné, un groupe de profil ne peut contenir qu'un seul profil d'une même APP.

3.8.2. Profils

Le profil contient (entre autres) les informations suivantes :

- tenant
- liste de rôles
- niveau

- APP

Un profil contient un seul et unique tenant.

L'APP permet d'autoriser l'affichage d'une application dans le portail. Le fait de pouvoir afficher l'application dans le portail ne préjuge pas des droits qui sont nécessaires au bon fonctionnement de l'application.

Un profil est modifiable uniquement par un utilisateur possédant un rôle autorisant la modification de profil et qui possède un niveau supérieur à celui du niveau du profil concerné.

Un profil ne peut être rattaché qu'à un groupe de profils de même niveau.

Dans une instance VITAMUI partagée, il convient de limiter les droits des administrateurs d'une organisation afin qu'ils ne puissent pas réaliser certains actions sur des ressources sensibles. (ie. customer, idp, tenant, etc.). Les profils créés à l'initialisation d'une nouvelle organisation ne doivent donc jamais comporter certains rôles (gestion des organisations, idp, tenants, etc.) afin d'interdire à l'administrateur d'une organisation d'utiliser ou de créer de nouveaux profils avec ces rôles pour réaliser des opérations multi-tenants.

Généralement l'adminitrateur de l'instance possède tous les droits (et donc tous les rôles).

3.8.3. Rôles

Le crôle constitue la granularité la plus fine dans le système de gestion des droits. Un rôle donne des droits d'accès à des endpoints (API) correspondants à des services. Un rôle peut être affecté à un ou plusieurs profils. Dans l'implémentation VITAMUI, l'accès à un endpoint est contrôlé par l'annotation @Secured. Il existe des rôles (dénommés sous-rôles) qui donnent accès à des fonctions protégées du service. Ces "sous-rôles" sont généralement contrôlés dans le corps de la méthode par le contexte de sécurité.

```
@Secured(ROLE_CREATE_XXX)
public MyDto create(final @Valid @RequestBody MyDto dto) {
    if ( SecurityContext.hasRole(ROLE_CREATE_XXX_YYY) {
        setProperty(...)
    }
    else {
        return HTTP.403 ;.
    }
}
```

Dans l'exemple ci-dessus :

- ROLE_CREATE_XXX est un rôle qui donne accès au service create
- ROLE_CREATE_XXX_YYY est un sous-rôle, utilisé dans le corps de la méthode, qui donne accès à une fonctionnalité spécifique de la méthode.

3.8.4. Niveaux

Dans une organisation, la gestion des utilisateurs, des profils et groupe de profils repose sur le principe de la filière unidirectionnelle d'autorité étendue. Elle donne

autorité au manageur sur les ressources d'une entité. Plusieurs manageurs peuvent avoir autorité sur une même entité. Un manageur n'a jamais autorité sur l'entité à laquelle il appartient. Il existe cependant un manageur administrateur qui a autorité sur toutes les ressources de toutes les entités.

Schéma de l'arbre de niveaux :

Arbre des niveaux

- Une entité dispose d'un niveau représenté par une chaine de caractère
- Une ressource est un objet (user, group, profile, etc.) appartenant à une entité
- Le manageur est un utilisateur qui a autorité sur des entités et leurs ressources associées

Ex. niveau : "World.France.DSI.Infra"

- World : entité racine - le niveau est vide (ou zéro). Le manageur World a autorité sur toutes les entités de l'arbre (dont lui-même)
- France : entité enfant de World - La manageur France a autorité sur les entités DSI et Infra
- DSI : entité enfant de France - La manageur DSI a autorité sur l'entité Infra
- Infra : entité enfant de DSI - La manageur Infra n'a autorité sur rien

Un utilisateur :

- manageur d'une ressource possède un niveau supérieur à celui de la ressource
- peut lister, modifier & supprimer une ressource dont il est le manageur
- peut créer une ressource dans une entité dont il est le manageur
- ne peut pas effectuer une action sur une ressource dont il n'est pas manageur
- ne peut pas effectuer des actions s'il ne dispose pas des rôles associés à ces actions
- ne peut pas affecter à un profil des rôles dont il ne dispose pas (cf. gestion des profils)

Un utilisateur avec un niveau vide (administrateur) :

- peut uniquement effectuer les actions associées aux rôles qu'il possède
- peut créer un profil ou un groupe de profils de niveau vide (admin)
- peut modifier ses ressources
- ne peut pas ajouter à un profil un rôle dont il ne dispose pas

Un administrateur d'une organisation possède donc des droits limités aux rôles qui ont été affectés à l'initialisation du système. Il ne peut pas par exemple créer une nouvelle organisation, si ce rôle ne lui pas été donné à l'origine. D'autre part, les droits de l'administrateur restent également limités par les droits associés à ceux du contexte de sécurité de l'application qu'il utilise.

- un profil ou un groupe de profils ne peuvent être supprimés que s'ils ne sont plus utilisés
- un profil avec un niveau ne peut être rattaché qu'à un groupe de un même niveau.

3.8.5. Matrice des droits

Les tableaux ci-dessous indiquent les droits d'un utilisateur en fonction du niveau de la ressource cible.

- Matrice des droits d'un utilisateur de niveau N pour réaliser des actions sur un utilisateur de niveau cible N+1, N, N-1 :

Niveau cible	N+1	N	N-1
Créer	Non	Non	Oui
Modifier	Non	Non	Oui
Lire	Non	Oui (1)	Oui
Supprimer	Non	Non	Oui (2)

Oui(1) : oui mais uniquement s'il s'agit de lui-même
Oui(2) : en théorie, car il est n'est pas possible de supprimer un utilisateur

- Matrice des droits d'un utilisateur de niveau N pour réaliser des actions sur un profil de niveau cible N+1, N, N-1 :

Niveau cible	N+1	N	N-1
Créer	Non	Non	Oui
Modifier	Non	Non	Oui
Lire	Non	Oui (1)	Oui
Attribuer	Non	Non	Oui
Supprimer	Non	Non	Oui

Oui(1) : oui mais uniquement si le profil est présent dans son groupe de profils
Lors de la modification du niveau du profil. Il faut vérifier qu'il n'est associé à aucun groupe. L'utilisateur ne peut affecter à un profil que les rôles et un tenant qu'il possède

- Matrice des droits d'un utilisateur de niveau N pour réaliser des actions sur un groupe de profils de niveau cible N+1, N, N-1 :

Niveau cible	N+1	N	N-1
Créer	Non	Non	Oui
Modifier	Non	Non	Oui
Lire	Non	Oui (1)	Oui
Attribuer	Non	Non	Oui
Supprimer	Non	Non	Oui

Oui(1) : oui mais uniquement s'il s'agit de son groupe
Lors de la modification du niveau d'un groupe. Il faut vérifier qu'il n'a pas de profils

- Matrice des droits d'un administrateur de niveau racine (niveau vide) pour réaliser des actions sur une ressource de niveau cible N+1, N, N-1 :

Niveau cible	N+1	N	N-1
Créer	-	Oui	Oui
Modifier	-	Oui	Oui
Lire	-	Oui	Oui
Attribuer	-	Oui	Oui

Niveau cible	N+1	N	N-1
Supprimer	-	Oui	Oui

Un administrateur ne peut pas affecter à un profil des rôles qui ne sont pas autorisés dans son organisation.

Matrice des profiles

La liste de profils crées par défaut pour chaque tenant :

```
- Nom: Profil pour la gestion des contrats d'accès
  Description: Gestion des contrats d'accès dans Vitam
  Application: ACCESS_APP
  Rôles:
    - ROLE_GET_ACCESS_CONTRACTS
    - ROLE_CREATE_ACCESS_CONTRACTS
    - ROLE_UPDATE_ACCESS_CONTRACTS
    - ROLE_GET_FILLING_PLAN_ACCESS

- Nom: Profil pour la gestion des contrats d'entrée
  Description: Gestion des contrats d'entrée dans Vitam
  Application: INGEST_APP
  Rôles:
    - ROLE_GET_INGEST_CONTRACTS
    - ROLE_CREATE_INGEST_CONTRACTS
    - ROLE_UPDATE_INGEST_CONTRACTS
    - ROLE_GET_FILLING_PLAN_ACCESS
    - ROLE_GET_MANAGEMENT_CONTRACTS
    - ROLE_GET_ARCHIVE_PROFILES

- Nom: Profil consultation des contrats d'entrée
  Description: Profil pour la consultation des contrats d'entrée dans Vitam sans
  Application: INGEST_APP
  Rôles:
    - ROLE_GET_INGEST_CONTRACTS

- Nom:  Profil gestion des services agents
  Description: Profil de gestion du référentiel des services agent avec possibil
  Application: AGENCIES_APP
  Rôles:
    - ROLE_GET_AGENCIES
    - ROLE_CREATE_AGENCIES
    - ROLE_UPDATE_AGENCIES
    - ROLE_DELETE_AGENCIES
    - ROLE_EXPORT_AGENCIES
    - ROLE_IMPORT_AGENCIES

- Nom: Profil consultation des services agents
  Description: Profil de consultation du référentiel des services agent sans pos
  Application: AGENCIES_APP
```

```
  Rôles:
    - ROLE_GET_AGENCIES

- Nom: Profil pour la gestion des Audits
  Description: Gestion des audits dans Vitam
  Application: AUDIT_APP
  Rôles:
    - ROLE_GET_AUDITS
    - ROLE_RUN_AUDITS

- Nom: Profil pour la gestion des opérations de sécurisation
  Description: Gestion des opérations de sécurisation dans Vitam
  Application: SECURE_APP
  Rôles:
    - ROLE_GET_OPERATIONS

- Nom: Profil de gestion des valeurs probantes
  Description: Gestion des valeurs probantes dans Vitam
  Application: PROBATIVE_VALUE_APP
  Rôles:
    - ROLE_GET_OPERATIONS
    - ROLE_RUN_PROBATIVE_VALUE

- Nom: Profil pour la lecture des formats de fichiers
  Description: Lecture des formats de fichiers dans Vitam
  Application: FILE_FORMATS_APP
  Rôles:
    - ROLE_GET_FILE_FORMATS

- Nom: Profil Journal des Opérations
  Description: Gestion des applications des Journaux des Opérations
  Application: LOGBOOK_OPERATION_APP
  Rôles:
    - ROLE_LOGBOOKS

- Nom: Profil pour le dépôt et suivi des versements
  Description: Gestion des applications de dépôt et suivi des versements
  Application: INGEST_MANAGEMENT_APP
  Rôles:
    - ROLE_GET_INGEST
    - ROLE_CREATE_INGEST
    - ROLE_GET_ALL_INGEST
    - ROLE_LOGBOOKS

- Nom: Profil Arbres et Plans
  Description: Gestion des applications d'import d'arbres de positionnement et p
  Application: HOLDING_FILLING_SCHEME_APP
  Rôles:
    - ROLE_CREATE_HOLDING_FILLING_SCHEME
    - ROLE_GET_HOLDING_FILLING_SCHEME
    - ROLE_GET_ALL_HOLDING_FILLING_SCHEME
```

- Nom: Profil pour la gestion des règles de gestion
  Description: Gestion des règles de gestion
  Application: RULES_APP
  Rôles:
    - ROLE_GET_RULES
    - ROLE_CREATE_RULES
    - ROLE_UPDATE_RULES
    - ROLE_DELETE_RULES
    - ROLE_IMPORT_RULES
    - ROLE_EXPORT_RULES

- Nom: Profil consultation des règles de gestion
  Description: Profil pour la consultation des règles de gestion dans Vitam sans
  Application: RULES_APP
  Rôles:
    - ROLE_GET_RULES

- Nom: Lancement de recherches par DSL
  Description: Lancement de recherches par DSL dans Vitam
  Application: DSL_APP
  Rôles:
    - ROLE_GET_UNITS

- Nom: Profil pour la gestion des opérations
  Description: Gérer et consulter l'ensemble des opérations d'entrées qui sont e
  Application: LOGBOOK_MANAGEMENT_OPERATION_APP
  Rôles:
    - ROLE_GET_LOGBOOK_OPERATION
    - ROLE_GET_ALL_LOGBOOK_OPERATION
    - ROLE_UPDATE_LOGBOOK_OPERATION

- Nom: Profil pour la création des profils paramétrage externe
  Description: Gérer et consulter l'ensemble des profils paramétrage externe
  Application: EXTERNAL_PARAM_PROFILE_APP
  Rôles:
    - ROLE_CREATE_EXTERNAL_PARAM_PROFILE
    - ROLE_EDIT_EXTERNAL_PARAM_PROFILE
    - ROLE_SEARCH_EXTERNAL_PARAM_PROFILE
    - ROLE_GET_PROFILES
    - ROLE_UPDATE_PROFILES
    - ROLE_LOGBOOKS

- Nom: Consultation
  Description: Profil pour la recherche et consultation des archives dans Vitam
  Application: ARCHIVE_SEARCH_MANAGEMENT_APP
  Rôles:
    - ROLE_CREATE_ARCHIVE_SEARCH
    - ROLE_GET_ARCHIVE_SEARCH
    - ROLE_GET_ALL_ARCHIVE_SEARCH
    - ROLE_SEARCH_WITH_RULES

```
    - ROLE_GET_ACCESS_CONTRACTS
    - ROLE_GET_RULES

- Nom: Archiviste
  Description: Profil pour la recherche et consultation des archives dans Vitam
  Application: ARCHIVE_SEARCH_MANAGEMENT_APP
  Rôles:
    - ROLE_CREATE_ARCHIVE_SEARCH
    - ROLE_GET_ARCHIVE_SEARCH
    - ROLE_GET_ALL_ARCHIVE_SEARCH
    - ROLE_EXPORT_DIP
    - ROLE_SEARCH_WITH_RULES
    - ROLE_GET_ACCESS_CONTRACTS
    - ROLE_GET_RULES

- Nom: Archiviste administrateur
  Description: Profil pour la recherche et consultation des archives dans Vitam
  Application: ARCHIVE_SEARCH_MANAGEMENT_APP
  Rôles:
    - ROLE_CREATE_ARCHIVE_SEARCH
    - ROLE_GET_ARCHIVE_SEARCH
    - ROLE_GET_ALL_ARCHIVE_SEARCH
    - ROLE_SEARCH_WITH_RULES
    - ROLE_EXPORT_DIP
    - ROLE_ELIMINATION
    - ROLE_UPDATE_MANAGEMENT_RULES
    - ROLE_COMPUTED_INHERITED_RULES
    - ROLE_GET_ACCESS_CONTRACTS
    - ROLE_RECLASSIFICATION
    - ROLE_UPDATE_UNIT_DESC_METADATA
    - ROLE_TRANSFER_REQUEST
    - ROLE_TRANSFER_ACKNOWLEDGMENT
    - ROLE_GET_RULES

- Nom: Registre des fonds
  Description: Visualisation de l'ensemble des données du registre des fonds
  Application: ACCESSION_REGISTER_APP
  Rôles:
    - ROLE_GET_ACCESSION_REGISTER_DETAIL

- Nom: Pastis-Gestion des profils documentaires
  Description: Pastis-Gestion des profils documentaires
  Application: PASTIS_APP
  Rôles:
    - ROL_GET_PASTIS
    - ROLE_GET_ARCHIVE_PROFILES_UNIT
    - ROLE_UPDATE_ARCHIVE_PROFILES_UNIT
    - ROLE_CREATE_ARCHIVE_PROFILES_UNIT
    - ROLE_IMPORT_ARCHIVE_PROFILES_UNIT
    - ROLE_DELETE_ARCHIVE_PROFILES_UNIT
    - ROLE_GET_ARCHIVE_PROFILES
```

```
    - ROLE_UPDATE_ARCHIVE_PROFILES
    - ROLE_CREATE_ARCHIVE_PROFILES
    - ROLE_IMPORT_ARCHIVE_PROFILES
    - ROLE_DELETE_ARCHIVE_PROFILES

- Nom: Collecte
  Description: Collecte de données, Application de préparation de versements
  Application: COLLECT_APP
  Rôles:
    - ROLE_GET_PROJECTS
    - ROLE_CREATE_PROJECTS
    - ROLE_UPDATE_PROJECTS
    - ROLE_GET_FILLING_PLAN_ACCESS
    - ROLE_GET_ACCESS_CONTRACTS
    - ROLE_GET_RULES
    - ROLE_SEND_TRANSACTIONS
    - ROLE_CLOSE_TRANSACTIONS
    - ROLE_REOPEN_TRANSACTIONS
    - ROLE_ABORT_TRANSACTIONS
    - ROLE_GET_TRANSACTIONS
    - ROLE_CREATE_TRANSACTIONS
    - ROLE_UPDATE_TRANSACTIONS
    - ROLE_DELETE_TRANSACTIONS
    - ROLE_UPDATE_UNITS_METADATA
```

3.8.6. Sécurisation des ressources

Vérification générale

Le processus de sécurisation des ressources est systématique et identique quelque soit l'utilisateur appelant la ressource. Ce processus, implémenté dans Spring Security, est essentiel car il permet de s'assurer qu'un utilisateur ne sorte jamais de son tenant. Ce processus de sécurisation est réalisé sur les accès aux ressources des services externals.

Les étapes du processus de sécurisation sont les suivantes :

1. récupérer l'utilisateur associé au token utilisateur fourni dans le header
2. vérifier que l'organisation de l'utilisateur possède le tenant fourni dans le header
3. vérifier que l'utilisateur possède un profil avec le tenant fourni dans le header
4. trouver le contexte applicatif par rapport au certificat x509 fourni dans la requête
5. vérifier que le contexte applicatif autorise le tenant fourni dans le header
6. créer un contexte de sécurité utilisateur qui correspond au tenant fourni dans le header et à l'intersection des rôles des profils de l'utilisateur et ceux du contexte applicatif
7. vérifier que les rôles du contexte de sécurité de l'utilisateur autorisent l'utilisateur authentifié à appeler la ressource

Si la ressource n'est pas accessible, une erreur 403 est retournée

Vérification des sous-rôles

Cette étape correspond à la vérification des sous-rôles dans le service appelé. Un sous-rôle donne accès à une fonction ou à un périmètre spécifique du service.

Exemple : Un utilisateur RH a le droit de modifier un autre utilisateur sauf son email (qui est sécurisé).

- L'utilisateur RH possède donc un rôle UPDATE_USER qui lui donne accès à l'API et au service de mise à jour globale des utilisateurs
- L'utilisateur RH ne possède pas le rôle UPDATE_USER_EMAIL qui permettrait de modifier l'email

La vérification du rôle UPDATE_USER_EMAIL est réalisée dans le service de mise à jour de l'utilisateur.

Vérification du tenant

En règle générale, le tenant concerné par la requête est vérifié par le processus de vérification générale. Il existe néanmoins des cas où le tenant est fourni en paramètre ou dans le corps de la requête.

Dans ce cas, les étapes de sécurisation sont les suivantes :

- vérifier la validité du tenant dans le contexte de sécurité
- Si le tenant n'est pas valide, il faut éventuellement vérifier si l'utilisateur a le droit de réaliser une opération multi-tenant. Cette dernière vérification est implémentée grâce aux rôle et sous-rôles (cf. gestion des customer, des idp, des tenants, des profils, etc).
- Si le tenant n'est pas valide, une erreur 403 est retournée

Cette implémentation permet ainsi de réaliser simplement des opérations multi-tenant en définissant des rôles appropriés. La solution VITAMUI fournit des services multi-tenant pour gérer les organisations, les fournisseurs d'identité, etc. Il est fondamental de limiter autant que possible l'utilisation de rôles muli-tenants. Il est en outre recommandé de borner l'usage des rôles multi-tenant à une zone protégée de l'infrastructure.

L'ensemble des rôles autorisés dans une organisation sont définis à la création de cette organisation.

3.9. Profils de paramétrage externes

3.9.1. External Parameter Profile

Un profil de paramétrage externe est une entité fictive, contient les informations suivantes :

- le nom du profil (nom)
- la description du profil (description)
- le contrat d'accès associé (accessContract: voir external parameters)
- le statut du profil (enabled)

Un profil de paramétrage externe permet d'associer un et unique profil à un contrat d'accès qui est lui meme lié à un paramétrage externe (ExternalParameters).

3.9.2. Profil

voir section 3.8

3.9.3. External Parameters

TODO voir section concernée.

3.9.4. Illustration

Donnée du profil


```
{
    "_id": "60d06c74663b6f71e8459eb0168d408ea49743f8bc4f80f21f3eeb266ec90cca",
    "identifier": "216",
    "name": "test profile",
    "enabled": true,
    "description": "test description profile",
    "tenantIdentifier": 1,
    "applicationName": "EXTERNAL_PARAM_PROFILE_APP",
    "roles": [
        {
            "name": "ROLE_CREATE_EXTERNAL_PARAM_PROFILE"
        },
        {
            "name": "ROLE_EDIT_EXTERNAL_PARAM_PROFILE"
        },
        {
            "name": "ROLE_SEARCH_EXTERNAL_PARAM_PROFILE"
        }
    ],
    "level": "",
    "readonly": false,
    "externalParamId": "reference_identifier",
    "customerId": "system_customer",
    "_class": "profiles"
}
```


Donnée de l’external parameter


```
{
    "_id": "60d06c73663b6f71e8459eae3ce591e616a1428bb086acd40c5c517eb8ccfda7",
    "identifier": "reference_identifier",
    "name": "test profile",
    "parameters": [
        {
            "key": "PARAM_ACCESS_CONTRACT",
            "value": "ContratTNR"
        }
    ],
    "_class": "externalParameters"
}
```


Le profil de paramétrage externe provenant des deux données ci-dessus

```
{
  "id": "60d06c74663b6f71e8459eb0168d408ea49743f8bc4f80f21f3eeb266ec90cca",
  "name": "test profile",
  "description": "test description profile",
  "accessContract": "ContratTNR",
  "profileIdentifier": "216",
  "idProfile": "60d06c74663b6f71e8459eb0168d408ea49743f8bc4f80f21f3eeb266ec90cca
  "externalParamIdentifier": "reference_identifier",
  "idExternalParam": "60d06c73663b6f71e8459eae3ce591e616a1428bb086acd40c5c517eb8
  "enabled": true,
  "dateTime": "2021-06-21T12:52:34.430803Z"
}
```


3.9.5. Événement lors de la mise à jour

La mise à jour du profil de paramétrage externe peut générer jusqu'à trois événements de journalisations.

Premier cas:

- Modification des données liés aux données du profil
  - Dans ce cas de figure, on émet un événement de journal externe de type EXT_VITAMUI_UPDATE_PROFILE.
  - et un événement de modification du profil de paramétrage externe EXT_VITAMUI_UPDATE_EXTERNAL_PARAM_PROFILE

Deuxième cas:

- Modification des données liés à la donnée du paramétrage externe
  - Dans ce cas de figure, on émet un événement de journal externe de type EXT_VITAMUI_UPDATE_EXTERNAL_PARAM.
  - et un événement de modification du profil de paramétrage externe EXT_VITAMUI_UPDATE_EXTERNAL_PARAM_PROFILE.

Troisième cas:

- Modification des données liés aux données du profil et du paramétrage externe, dans ce cas de figure, on émet 3 événements de journalisation :
  - événement de type EXT_VITAMUI_UPDATE_PROFILE.
  - événement de type EXT_VITAMUI_UPDATE_EXTERNAL_PARAM.
  - et un événement de modification du profil de paramétrage externe EXT_VITAMUI_UPDATE_EXTERNAL_PARAM_PROFILE.

Exemple de mise à jour de la description du profil:


```
{
    "_id": "aecaaaaaaghohlrwaan3ial2fyt7xnaaaaaq",
    "tenantIdentifier": 1,
    "accessContractLogbookIdentifier": "AC-000002",
    "evType": "EXT_VITAMUI_UPDATE_EXTERNAL_PARAM_PROFILE",
    "evTypeProc": "EXTERNAL_LOGBOOK",
    "outcome": "OK",
    "outMessg": "Le profil paramètrage externe a été modifié",
    "outDetail": "EXT_VITAMUI_UPDATE_EXTERNAL_PARAM_PROFILE.OK",
    "evIdReq": "5043309c-c8d7-4bc9-bfcc-bd20852ce90e",
```

```
    "evDateTime": "2021-06-21T10:40:10.164438Z",
    "obId": "216",
    "obIdReq": "externalparamprofile",
    "evDetData": "{\"diff\":{\"-Description\":\"test description profile\",\"+De
    "evIdAppSession": "EXTERNAL_PARAM_PROFILE_APP63597049221:5043309c-c8d7-4bc9-
    "creationDate": 3960212756529822,
    "status": "SUCCESS",
    "_class": "events",
    "synchronizedVitamDate": "2021-06-21T10:40:36.370328Z",
    "vitamResponse": "{\"httpCode\":201,\"code\":\"\"}"
}
```


3.10. Journalisation

3.10.1. Objectifs

La journalisation des événements VITAMUI a pour objectifs :

- Conservation de la valeur probante : être en capacité de prouver toute opération effectuée sur toute unité archivistique ou tout objet qui lui est associé.
- La sécurité d'un SAE doit être systémique, c'est-à-dire reposer sur un faisceau d'éléments redondants dont la modification simultanée et cohérente est impossible, ou plus exactement non réalisable en pratique.
- Les journaux constituent un élément central de cette sécurité systémique Utilisation des journaux vitam NF Z42-013

Journalisation

3.10.2. Événement

Vitam

- Un événement = Un événement Primaire (Primary) et ensemble de sous-événements secondaires (Secondary)
  - Primary : événement initial
    - les champs sont contrôlés par VITAM
    - Marque le début de la transaction au sens VITAM
    - L'heure de l'événement et mise par VITAM (cohérence des journeaux)
  - Secondary : note un sous événement réalisé suite à l'action principale
    - possède les mêmes champs que l'événement Master mais VITAM ne procède à aucun contrôle
    - l'heure de l'êvénement est à l'appréciation du client
  - Fin de la transaction : le dernier sous événement doit posséder le même champs "eventType" que l'événement Master pour finir la transaction.

VITAMUI

- Primaire et Secondaire => Un event VITAMUI cf : fr.gouv.vitamui.commons.logbook.domain.ev
- Un appel REST => Une ou plusieurs opération métier => ensemble d'events => le premier sera l'evénement primaire (Primary) et les suivants secondaires (Secondary)
- Stocker dans le tenant des éléments de preuves du client

Journalisation

3.10.3. Application dans VITAMUI

Modèle

Propriétés	valeurs
EventTypeProc	EXTERNAL_LOGBOOK
EventType	Nom du type d'événement (EXT_VITAMUI_CREATE_USER)
obIdReq	Nom de la collection Mongo (USERS)
obId	Identifiant métier de l'objet
evDetData	Contient les informations importantes (modification avant/après contenu du nouvelle objet) outcome : OK, KO (Pour le master -> OK, pour les sous-events le champ est libre)
evIdAppSession	applicationIdExt:requestId:applicationName:userIdentifier:superUserIdenti
evIdReq	X-Request-Id

3.10.4. Création

- L'ensemble des modifications de la base de données se font dans une unique transaction.
- Centralisation de la création des traces dans chaque module (IamLogbookService, ArchiveLogbookService, FlowLogbookService) (Responsable de la cohérence de la génération d'un event à partir d'un objet métier
- Chaque objet de notre modèle de données possède un converter associé (Capable de convertir un objet en json et qui sera mis dans le evDetData de l'event)

3.10.5. Sauvegarde

- Réalisation par les tâches asynchrones (Cf : SendEventToVitamTasks.java et DeleteSynchronizedEventsTasks.java)
- Les événements sont regroupés par rapport à leur X-Request-Id et triés par ordre chronologique croissant.
- Le premier événements du groupe devient le Primary et les autres des sous-events.
- Le premier est recopier a la fin des sous-events afin de fermer la "transaction au sens VITAM"
- Envoit vers vitam (La reponse vitam et la date d'envoi sont toujours stocké) :
  - Succès -> Les events sont conservés X jours et sont marqué au status "SUCCESS"
  - Erreur -> Les events sont marqués au statut "ERROR" et un retry sera effectué dans X heure.

3.11. Modèle de données

3.11.1. Liste des bases

```
iam
security
cas
archivesearch
```

3.11.2. Base IAM

Collections

```
applications
customers
events
groups
externalParameters
owners
profiles
providers
subrogations
tenants
tokens
users
```

- *Collection applications*

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
identifier	String	minimum = 1, maximum = 100	L’identifiant (unique) de l’application
url	String	minimum = 1, maximum = 100	
serviceId	String	minimum = 1, maximum = 100	Le meme serviceId que nous avons au niveau de la collection services
icon	String	minimum = 1, maximum = 50	Logo de l’application
name	String	minimum = 1, maximum = 50	Nom de l’application
category	String	minimum = 1, maximum = 12	La catégorie de l’application
position	int	Non null	L’ordre d’affichage dans la liste des applications
hasCustomerList	boolean	default=false	
hasTenantList	boolean	default=false	Pour pouvoir changer le tenant au niveau de l’application
hasHighlight	boolean	default=false	
tooltip	String	minimum = 1, maximum = 100	Un texte pour décrire l’application
target	String	maximum = 25	

- *Collection customers*

Nom	Type	Contrainte(s)
_id	String	Clé Primaire
identifier	String	minimum = 1, maximum = 12
code	String	minimum = 6, maximum = 20
companyName	String	maximum = 250
language	String	Non null, valeurs = [FRENCH,ENGLISH]

Nom	Type	Contrainte(s)
passwordRevocationDelay	Integer	Non null
otp	Enum	Non null, valeurs = [OPTIONAL,DISABLED,MANDAT
emailDomains	List<*String*>	Non null, Non vide
defaultEmailDomain	String	Non null
address	Address	Non null
name	String	maximum = 100
subrogeable	boolean	default=false
readonly	boolean	default=false
graphicIdentity	GraphicIdentity	
gdprAlert	boolean	default=false
gdprAlertDelay	int	minimum=1

- GraphicIdentity (Embarqué)

Nom	Type	Contrainte(s)	Remarque(s
hasCustomGraphicIdentity	boolean		
logoDataBase64	String		
logoHeaderBase64	String		Base64 enc
portalTitle	String		
portalMessage	String	maximum length = 500 chars)	
themeColors	Map<String, String>		

- themeColors

Nom	Type	Contrainte(s)	Remarque(s)
vitamui-primary	String	hexadeciaml color like	
vitamui-secondary	String	hexadeciaml color like	
vitamui-tertiary	String	hexadeciaml color like	
vitamui-header-footer	String	hexadeciaml color like	
vitamui-background	String	hexadeciaml color like	

- *Collection tenants*

Le tenant correspond à un container (ie. espace de travail) logique. Chaque tenant est unique dans le système et appartient à un seul et unique client. Un client peut posséder plusieurs tenants. Un client ne doit jamais pouvoir accéder au tenant d'un autre client. Les tenants VITAMUI correspondent aux tenants VITAM. Toutes les requêtes HTTP dans VITAMUI doivent renseigner le tenant dans le header. Dans VITAMUI, le tenant permet de vérifier les autorisations applicatives (certificat et contexte) et utilisateurs (profils).

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
customerId	String	Non null, Clé Étrangère	
identifier	Integer	Non null	correspond au tenant vitam

Nom	Type	Contrainte(s)	Remarque(s)
ownerId	String	Non null, Clé Étrangère	
name	String	maximum = 100	exprimé en jour
proof	Boolean		identifie le tenant de preuve
readonly	Boolean		
ingestContractHoldingIdentifier	String	Non null	contrat d’entrée pour l’arbre
accessContractHoldingIdentifier	String	Non null	contrat d’accès pour l’arbre
itemIngestContractIdentifier	String	Non null	contrat d’entrée pour les bordereaux
accessContractLogbookIdentifier	String	Non null	contrat d’accès pour le logbook
enabled	Boolean	Non null	

- *Collection owners*

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
identifier	String	minimum = 1, maximum = 12	
customerId	String	Clé Étrangère	Clé Étrangère
name	String	maximum = 100	
code	String	minimum = 6, maximum = 20	
companyName	String	maximum = 250	
address	Address		embedded
readonly	Boolean		

- Address (Embarqué)

Nom	Type	Contrainte(s)	Remarque(s)
street	String	maximum = 250	
zipCode	String	maximum = 10	
city	String	maximum = 100	
country	String	maximum = 50	

- *Collection Identity Provider*

L’identity provider L’IDP est soit externe (Clients/Organisations externes) soit interne. L’IDP interne est CAS lui même et les utilisateurs sont alors gérés uniquement dans l’annuaire CAS de VITAMUI.

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
customerId	String	Clé Étrangère	
identifier	String	minimum = 1, maximum = 12	
name	String	maximum = 100	
technicalName	String		
internal	Boolean	default=true	
patterns	List<*String*>	minimum = 1	
enabled	Boolean	default=true	
keystoreBase64	String		
keystorePassword	String		Mot de passe
privateKeyPassword	String		Mot de passe
idpMetadata	String		XML
spMetadata	String		XML
maximumAuthenticationLifetime	Integer		
readonly	Boolean		

- *Collection users*

Nom	Type	Contrainte(s)
_id	String	Clé Primaire
customerId	String	Clé Étrangère
enabled	boolean	default = true
status	Enum	default = ENABLED, BLOCKED, ANONYM, DISABLED
type	Enum	NOMINATIVE, GENERIC
password	String	maximum = 100
oldPasswords	List<*String*>	
identifier	String	minimum = 1, maximum = 12
email	String	email, Unique
firstname	String	maximum = 50
lastname	String	maximum = 50
language	String	Non null, valeurs = [FRENCH,ENGLISH]
phone	String	phone number
mobile	String	mobile phone number
otp	Boolean	default = false
groupId	String	Not null
subrogeable	Boolean	
lastConnection	OffsetDateTime	
nbFailedAttempts	int	
readonly	boolean	default=false
level	String	Not null
passwordExpirationDate	OffsetDateTime	
address	Address	
analytics	AnalyticsDto	

- AnalyticsDto (Embarqué)

Nom	Type	Contrainte(s)	Remarque(s)
applications	ApplicationAnalyticsDto		
lastTenantIdentifier	Integer		

- ApplicationAnalyticsDto (Embarqué)

Nom	Type	Contrainte(s)	Remarque(s)
applicationId	String		
accessCounter	int		
lastAccess	OffsetDateTime		ex: YYYY-MM-ddTHH:mm:ss.ssssssZ

- *Collection externalParameters*

La collection qui définit un contrat d'accès par défaut

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
identifier	String	minimum = 1, maximum = 12	
name	String	minimum = 2, maximum = 100	
parameters	ParameterDto	Not Null	

- ParameterDto (Embarqué)

Nom	Type	Contrainte(s)	Remarque(s)
key	String		exemple: PARAM_ACCESS_CONTRACT
value	String		exemple: AC-000001

- *Collection groups*

Le groupe de profil définit un ensemble de profils. Un groupe de profil ne peut contenir qu'un seul profil par "app:tenant". Par exemple : "profil(app1:tenant1), profil(app1:tenant2), profil(app2:tenant1)" est autorisé.

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
identifier	String	minimum = 1, maximum = 12	
customerId	String	Non Null, Clé Étrangère	
name	String	maximum = 100	
description	String	maximum = 250	
profileIds	List<*String*>	clé étrangére	les profils
level	String	maximum = 250	
readonly	Boolean		
enabled	Boolean		

- *Collection profiles*

Le profil définit les permissions (rôles) données à un utilisateur et l'accès à une application (applicationName), généralement une IHM qui regroupe un ensemble de

fonctionnalités selon une logique métier et appelant des API backoffice. Un profil appartient à une groupe (de profils). Il ne peut y avoir qu'un seule et unique profile par tenant, applicationName dans un groupe.

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
identifier	String	minimum = 1, maximum = 12	
tenantIdentifier	Integer		
name	String	maximum = 100	
enabled	boolean	default=true	
description	String	maximum = 250	
applicationName	String	maximum = 250	
roles	List<*Role*>		rôle Spring
readonly	Boolean		
level	String	maximum = 250	
externalParamId	String		

- *Collection subrogations*

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
status	Enum	CREATED, ACCEPTED	
date	Date		
surrogate	String	email, minimum = 4, maximum = 100	celui qui est subrogé
superUser	String	email, minimum = 4, maximum = 100	celui qui subroge
surrogateCustomerId	String	not null	
superUserCustomerId	String	not null	

- *Collection tokens*

Nom	Type	Contrainte(s)	Remarque(s)
_id	String		
updatedDate	Date	not null	
refId	String	not null	
surrogation	Boolean		

- *Collection events*

Nom	Type	Contrainte(s)
_id	String	Clé Primaire
tenantIdentifier	Integer	not null
accessContractLogbookIdentifier	String	not null
evParentId	String	
evIdProc	String	
evType	String	not null
evTypeProc	Enum	EXTERNAL_LOGBOOK
outcome	Enum	UNKNOWN, STARTED, ALREADY_EXECUTED
outMessg	String	not null

Nom	Type	Contrainte(s)
outDetail	String	not null
evIdReq	String	not null
evDateTime	String	not null
obId	String	not null
obIdReq	String	not null
evDetData	String	not null
evIdAppSession	String	not null
creationDate	Long	not null
status	Enum	CREATED, SUCCESS, ERROR
vitamResponse	String	
synchronizedVitamDate	OffsetDateTime	

Pour aller plus loin, le modèle de données Vitam concernant les journaux d'archives est accessible par ici

3.11.3. Base security

- *Collection Context*

Le contexte applicatif permet d'attribuer à une application cliente selon son certificat X509 transmis lors de la connexion https les droits d'accès (rôles) à différents services.

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
fullAccess	Boolean	default = false	
name	String	not null	
tenants	List<*Integer*>	Not Null, List de Clé Étrangère	Liste des tenants autorisés
roleNames	List<*String*>	Not Null	Liste des rôles autorisés

- *Collection Certificate*

La collection certificat permet de stocker les certificats correspondant à un contexte. Le certificat est transmis par l'application client lors de la connexion SSL.

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
contextId	String	Not Null	
serialNumber	String	Not Null	Numéro de série du certificat
subjectDN	String	Not Null	Identifiant unique (Distinguished Name) du certificat

Nom	Type	Contrainte(s)	Remarque(s)
issuerDN	String	Not Null	Identifiant unique (Distinguished Name) de l'autorité de certification
data	String	Not Null	Certificat en base64

- *Collection CustomSequence*

La collection sequence permet de stocker les différentes séquences utilisés.

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
name	String	Not Null	Nom de la séquence
sequence	int		Valeur courante

La liste des noms de séquences :

- tenant_identifier
- user_identifier
- profile_identifier
- group_identifier

- provider_identifier
- customer_identifier
- owner_identifier

3.11.4. Base Cas

Cette base est initialisée à la création de l'environnement. Elle est uniquement utilisée par CAS en lecture seule.

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
serviceId	String	Not Null	url du service web
name	String		nom du service
logoutType	String		
logoutUrl	String		url de logout
attributeReleasePolicy			Stratégie des attributs

3.11.5. Base archivesearch

Cette base est utilisé pour stocker les critères de filtres de recherche des utilisateurs, Aujourd'hui, elle est utilisée uniquement par le service Archive-Search, en particulier l'application de consultation et de recherche d'archives.

Collections

`searchCriteriaHistories`

- *Collection searchCriteriaHistories*

Nom	Type	Contrainte(s)	Remarque(s)
_id	String	Clé Primaire	
name	String	Not Null, minimum = 1, maximum = 150	nom de la recherche sauvegardée
userId	String	Not Null	l'identifiant de l'utilisateur
date	Date		date de la sauvegarde des critères de recherche
searchCriteriaList	List<*SearchCriteriasDto*>		liste des critères de recherches sauvegardées incluant les critères d'arbres et plan

`* SearchCriteriasDto (Embarqué)`

Nom	Type	Contrainte(s)	Remarque(s)
nodes	List<*String*>		liste des identifiants des noeuds d'arbre de position-nement/ plans de classemnt
criteriaList	List<*SearchCriteriaElementsDto*>		liste des critères de recherches sauvegardées

`* SearchCriteriaElementsDto (Embarqué)`

Nom	Type	Contrainte(s)	Remarque(s)
criteria	String		le nom du critère de recherche (eg: Title, StartDate, #opi, #id . . .)

Nom	Type	Contrainte(s)	Remarque(s)
values	List<*String*>		liste des valeurs du critère de filtre

4. Implémentation

4.1. Technologies

4.1.1. Briques techniques

La solution est développée principalement avec les briques technologies suivantes :

- Java 1.8+ (Java 11)
- Angular 8 : framework front
- Spring Boot 2 : framework applicatif
- MongoDB : base de données NoSQL
- Swagger : documentation API

4.1.2. COTS

Les composants suivant sont utilisés dans la solution :

- CAS : gestionnaire d'authentification centralisé (IAM)
- VITAM : socle d'archivage développé par le programme VITAM
- MongoDB : base de données orientée documents
- Curator : maintenance des index d'elasticsearch
- ELK : agrégation et traitement des logs et dashboards et recherche des logs techniques
- Consul : annuaire de services

Les solutions CAS et VITAM sont également développées en Java dans des technologies proches ou similaires.

En fonction du choix de l'implémentation de la solution, il est possible de partager des dépendances logicielles avec la solution VITAM.

4.2. Services

La solution est bâtie selon une architecture de type micro-services. Ces services communiquent entre eux en HTTPS via des API REST.

- Les services externes exposés publiquement sont sécurisés par la mise en oeuvre d'un protocole M2M nécessitant l'utilisation de certificats X509 client et serveur reconnus mutuellement lors de la connexion.
- Les services internes, ne sont jamais exposés publiquement. Ils sont accessibles uniquement en HTTPS par les services externes ou par d'autres services internes.
- Les accès aux bases de données MongoDb ou aux socles techniques externes (ie. VITAM) se font uniquement via les services internes.

- Les utilisateurs sont authentifiés via CAS et disposent d'un token, validé à chaque appel, qui les identifient durant toute la chaîne de traitement des requêtes.

4.2.1. Identification des services

Il est primordial que chaque service de la solution puisse être identifié de manière unique sur le système. A cet effet, les services disposent des différents identifiants suivant :

- ID de service (ou service_id) : c'est une chaîne de caractères qui nomme de manière unique un service. Cette chaîne de caractère doit respecter l'expression régulière suivante : [a-z][a-z-]*. Chaque cluster de service possède un ID unique de service.
- ID d'instance (ou instance_id) : c'est l'ID d'un service instancié dans un environnement ; ainsi, pour un même service, il peut exister plusieurs instances de manière concurrente dans un environnement donné. Cet ID a la forme suivante : -, avec respectant l'expression régulière suivante : [0-9]{2}. Chaque instance dans ce cluster possède un id d'instance (instance_id).
- ID de package (ou package_id) : il est de la forme vitamui. C'est le nom du package à déployer.

4.2.2. Communications inter-services

Les services VITAMUI suivent les principes suivants lors d'un appel entre deux composants :

1. Le composant amont effectue un appel (de type DNS) à l'annuaire de service en indiquant le service_id du service qu'il souhaite appeler
2. L'annuaire de service lui retourne une liste ordonnée d'instance_id. C'est de la responsabilité de l'annuaire de service de trier cette liste dans l'ordre préférentiel d'appel (en fonction de l'état des différents services, et avec un algorithme d'équilibrage dont il a la charge)
3. Le composant amont appelle la première instance présente dans la liste. En cas d'échec de cet appel, il recommence depuis le point 1. La communication vers une instance cible de type Service API utilise nécessairement le protocole sécurisé HTTPS.

Ces principes ont pour but de garantir les trois points suivants : * Les clients des services doivent être agnostiques de la topologie de déploiement, et notamment du nombre d'instances de chaque service dans chaque cluster. La connaissance de cette topologie est déléguée à l'annuaire de service.

- Le choix de l'instance cible d'un appel doit être décorrélé de l'appel effectif afin d'optimiser les performances et la résilience.
- La garantie de la confidentialité des informations transmises entre les services (hors COTS)

Dans le cas des COTS, la gestion de l'équilibrage de charge et de la haute disponibilité doit être intégrée de manière native dans le COTS utilisé. D'autre part, la sécurisation de la transmission dépend du COTS. Dans le cas où le chiffrement des

données transmises n'est pas assuré, il est alors recommandé d'isoler le COTS dans une zone réseau spécifique.

4.2.3. Cloisonnement des services

Le cloisonnement applicatif permet de séparer les services de manière physique (subnet/port) et ainsi limiter la portée d'une attaque en cas d'intrusion dans une des zones. Ce cloisonnement applique le principe de défense en profondeur préconisé par l'ANSI.

Chaque zone héberge des clusters de services. Un cluster doit être présent en entier dans une zone, et ne peut par conséquent pas être réparti dans deux zones différentes. Chaque noeud d'un cluster applicatif doit être installé sur un hôte (OS) distinct (la colocalisation de deux instances d'un même service n'étant pas supporté). La mise en oeuvre d'une infrastructure virtualisée impose de placer deux noeuds d'un même cluster applicatif sur deux serveurs physiques différents.

Un exemple de découpage en zones applicative est fourni ci-dessous. Ce découpage repose sur une logique assez classique adapté à une infrastructure de type VmWare ESX. Pour une architecture reposant sur une technologie de type Docker, il serait envisageable de découper plus finement les zones jusqu'à envisager une zone pour chaque cluster de service.

Dans cet exemple, il est prévu pour respecter les contraintes de flux inter-zones suivants :

- les utilisateurs de la zone USERS communiquent avec les services de la zone IHM
- les administrateurs de la zone ADMIN communiquent avec les services de la zone IHM ADMIN
- les services de la zone IHM et IHM-ADMIN communiquent avec les services de la zone API-EXTERNAL
- les services de la zone API-EXTERNAL communiquent avec les services de la zone API-INTERNAL
- les services de la zone API-INTERNAL communiquent avec les services de la zone DATA
- les services de toutes les zones communiquent avec les services déployés dans la zone INFRA
- les exploitants techniques accédent aux services de la zone EXPLOITATION puis intervenir dans toutes les zones

Les différentes zones:

zone IHM: La zone IHM se compose de plusieurs services: - UI Identity - UI Portal - UI Referential - UI Ingest - UI Archive Search ##### zone API-EXTERNAL: La zone API-EXTERNAL se compose de plusieurs services: - IAM EXTERNAL - REFERENTIAL EXTERNAL - INGEST EXTERNAL - ARCHIVE SEARCH EXTERNAL ##### zone API-INTERNAL: La zone API-INTERNAL se compose de plusieurs services: - IAM INTERNAL - REFERENTIAL INTERNAL - INGEST INTERNAL - ARCHIVE SEARCH INTERNAL ##### zone DATA: La zone stockage: MongoDB ##### zone INFRA: Les services consul, kibana, elk etc..

Tous les serveurs cibles doivent avoir accès aux dépôts de binaires contenant les paquets des logiciels VITAMUI et des composants externes requis pour l'installation.

Les autres éléments d'installation (playbook ansible, . . .) doivent être disponibles sur la machine ansible orchestrant le déploiement de la solution dans la zone INFRA.

`Schéma de zoning :`

Architecture IAM CAS

4.3. Intégration système

4.3.1. Utilisateurs et groupes d'exécution

La segmentation des droits utilisateurs permet de respecter les contraintes suivantes :

- Assurer une séparation des utilisateurs humains du système et des utilisateurs système sous lesquels tournent les processus

- Séparer les droits des rôles d'exploitation différents suivants :
  - Les administrateurs système (OS) ;
  - Les administrateurs techniques des logiciels VITAMUI;
  - Les administrateurs des bases de données VITAMUI

Les utilisateurs et groupes décrits dans les paragraphes suivants doivent être ajoutés par les scripts d'installation de la solution VITAMUI. En outre, les règles de sudoer associées aux groupes vitamui*-admin doivent également être mis en place par les scripts d'installation.

Les sudoers sont paramétrés en mode NOPASSWD, c'est à dire qu'aucun mot de passe n'est demandé à l'utilisateur faisant partie du groupe vitamui*-admin pour lancer les commandes d'arrêt relance des applicatifs vitamui.

Les fichiers de règles sudoers des groupes vitamui-admin et vitamuidb-admin sont systématiquement écrasés à chaque installation des paquets (rpm) déclarant les utilisateurs VITAMUI. (Un backup de l'ancien fichier est cependant effectué).

4.3.1.1. Utilisateurs

Les utilisateurs suivant sont définis :

- vitamui(UID : 4000) : user pour les services ne stockant pas les données
- vitamuidb (UID : 4001) : user pour les services stockant des données (Ex : MongoDB)

Les processus VITAMUI tournent sous ces utilisateurs. Leurs logins sont désactivés.

4.3.1.2. Groupes

Les groupes suivant sont définis :

- vitamui(GID : 4000) : groupe primaire des utilisateurs de service
- vitamui-admin (GID : 5000) : groupe d'utilisateurs ayant les droits "sudo" permettant le lancement des services VITAMUI
- vitamuidb-admin (GID : 5001) : groupe d'utilisateurs ayant les droits "sudo" permettant le lancement des services VITAMUI stockant de la donnée.

4.3.2. Arborescence de fichiers

L'arborescence /vitamui héberge les fichiers propres aux différents services. Elle est normalisée selon le pattern suivant : /vitamui// où :

Pour un service d'id service_id, les fichiers et dossiers impactés par VITAMUI sont les suivants.

- service_id est l'id du service auquel appartient les fichiers
- folder-type est le type de fichiers contenu par le dossier :
  - app : fichiers de ressources (non-jar) requis pour l'application (ex: .war)
  - bin : binaires (le cas échéant)
  - script : Répertoire des scripts d'exploitation du module (start/stop/status/backup)
  - conf : Fichiers de configuration
  - lib : Fichiers binaires (ex: jar)
  - log : Logs du composant
  - data : Données sauvegardes du composant
  - tmp : Données temporaires produites par l'application

Les dossiers /vitamui et /vitamui/ ont les droits suivants :

- Owner : root
- Group owner : root
- Droits : 0555

A l'intérieur de ces dossiers, les droits par défaut sont les suivants :

- Fichiers standards :
  - Owner : vitamui (ou vitamuidb)
  - Group owner : vitamui
  - Droits : 0640
- Fichiers exécutables et répertoires :
  - Owner : vitamui (ou vitamuidb)
  - Group owner : vitamui
  - Droits : 0750

Cette arborescence ne doit pas contenir de caractère spécial. Les éléments du chemin (notamment le service_id) doivent respecter l'expression régulière suivante : [0-9A-Za-z-_]+

Le système de déploiement et de gestion de configuration de la solution est responsable de la bonne définition de cette arborescence (tant dans sa structure que dans les droits utilisateurs associés).

4.3.3. Intégration au service d'initialisation Systemd

L'intégration est réalisée par l'utilisation du système d'initialisation systemd. La configuration se fait de la manière suivante :

- /usr/lib/systemd/system/ : répertoire racine des définitions de units systemd de type "service"
- .service : fichier de définition du service systemd associé au service VITAMUI

Les COTS utilisent la même nomenclature de répertoires et utilisateurs que les services VITAMUI, à l'exception des fichiers binaires et bibliothèques qui utilisent les dossiers de l'installation du paquet natif.

4.4. Sécurisation

4.4.1. Sécurisation des accès aux services externes

Les services exposants publiquement des API REST implémentent les mesures de sécurité suivantes :

- mise en place de filtres dans les applications IHM pour contrer les attaques de type CSRF et XSS
- utilisation du protocole HTTPS. Par défaut, la configuration suivante est appliquée (Protocoles exclus : TLS 1.0, TLS 1.1, SSLv2, SSLv3 & Ciphers exclus : *.NULL.*, *.RC4.*, *.MD5.*, *.DES.*, *.DSS.*)
- authentification par certificat X509 requise des applications externes (authentification M2M) basée sur une liste blanche de certificats valides
- mise à jour des droits utilisateurs grâce aux contextes applicatifs, associés certificats clients, stockés dans la collections XXX de base MongoDb gérée par le service SECURITY INTERNAL.
- un service batch contrôle régulièrement l'expéritaion des certificats stockés dans le truststore des services et dans le référentiel de certificats clients (MongoDb) géré par le service SECURITY INTERNAL.

4.4.2. Sécurisation des communications internes

Les communications internes sont sécurisées par le protocole HTTPS. D'autre part, dans chaque requête, le header X-Auth-Token est positionné. Il contient le token initialisé par CAS à la connexion de l'utilisateur.

A chaque requête le service VITAMUI internal procède aux contrôles suivants :

- vérification de l'existence du header X-Auth-Token
- vérification de la validité (non expiré) du token extrait du header

En cas d'échec, la requête est refusée et la connexion est fermée.

4.4.3. Sécurisation des accès aux bases de données

Les bases de données de MongoDB sont sécurisées via un cloisonnement physique (réseau) et/ou logique (compte utilisateur) des différentes bases de données qui les constituent.

4.4.4. Sécurisation des secrets de déploiement

Les secrets de l'intégralité de la solution VITAM déployée sont tous présents sur le serveur de déploiement ; par conséquent, ils doivent y être stockés de manière sécurisée, avec les principes suivants :

- Les mot de passe et token utilisés par ansible doivent être stockés dans des fichiers d'inventaire chiffrés par ansible-vault ;
- Les clés privées des certificats doivent être protégées par des mot de passe complexes et doivent suivre la règle précédente.

4.4.5. Liste des secrets

Les secrets nécessaires au bon déploiement de VITAMUI sont les suivants :

- Certificat ou mot de passe de connexion SSH à un compte sudoers sur les serveurs cibles (pour le déploiement)
- Certificats x509 serveur (comprenant la clé privée) pour les modules de la zone d'accès (services *-external), ainsi que les CA (finales et intermédiaires) et CRL associées. Ces certificats seront déployés dans des keystores java en tant qu'élément de configuration de ces services
- Certificats x509 client pour les clients du SAE (ex: les applications métier, le service ihm-admin), ainsi que les CA (finales et intermédiaires) et CRL associées. Ces certificats seront déployés dans des keystores java en tant qu'élément de configuration de ces services

Les secrets définis lors de l'installation de VITAM sont les suivants :

- Mots de passe des keystores ;
- Mots de passe des administrateurs fonctionnels de l'application VITAMUI
- Mots de passe d'administration de base de données MongoDB ;
- Mots de passe des comptes d'accès aux bases de données MongoDB.

Note. Les secrets de VITAMUI sont différents de ceux VITAM

4.4.6. Authentification du compte SSH

Il existe plusieurs méthodes envisageables pour authentifier le compte utilisateur utilisé pour la connexion SSH : * par clé SSH avec passphrase * par login/mot de passe * par clé SSH sans passphrase

La méthode d'authentification retenue dépend de plusieurs paramètres : * criticité des serveurs (services) * zone de confiance * technologie de déploiement

Dans un contexte sensible, il est fortement recommandé d'utiliser un bastion logiciel (par ex. https://www.wallix.com/bastion-privileged-access-management/) pour authentifier et tracer les actions des administrateurs du système.

4.4.7. Authentification des hôtes

Pour éviter les attaques de type MitM, le client SSH cherche à authentifier le serveur sur lequel il se connecte. Ceci se base généralement sur le stockage des clés publiques des serveurs auxquels il faut faire confiance (~/.ssh/known_hosts).

Il existe différentes méthodes pour remplir ce fichier (vérification humaine à la première connexion, gestion centralisée, DNSSEC). La gestion du fichier known_hosts est un pré-requis pour le lancement d'ansible.

4.4.8. Elévation de privilèges

Plusieurs solutions sont envisageables :

- par sudo avec mot de passe
  - Au lancement de la commande ansible, le mot de passe sera demandé par sudo

- par su
  - Au lancement de la commande ansible, le mot de passe root sera demandé
  - par sudo sans mot de passe ## 4.5. Certificats et PKI

La PKI permet de gérer de manière robuste les certificats de la solution VITAMUI. Une PKI est une architecture de confiance constituée d'un ensemble de systèmes fournissant des services permettant la gestion des cycles de vie des certificats numériques :

- émission de certificats à des entités préalablement authentifiées
- déploiement des certificats
- révocation des certificats
- établir, publier et respecter des pratiques de certification de confiance pour établir un espace de confiance

4.5.1. Principes de fonctionnement PKI de VITAMUI

La PKI VITAMUI gère les certificats nécessaires à l'authentification des services VITAMUI et des entités extérieurs. La logique de fonctionnement de la PKI VITAMUI est similaire à celle utilisée par la solution VITAM.

Les principes de fonctionnement de la PKI sont les suivants :

- Emission des certificats VITAMUI (les dates de création et de fin de validité des CA sont générées dans cette phase).
- Gestion du cycle de vie (révocation) des certificats
- Publication des certificats et des clés (.crt et .key)
- Déploiement :
  - Génération des magasins de certificats VITAMUI (les certificats .crt et .key sont utilisés pour construire un magasin de certificats qui contient des certificats .p12 et .jks)
  - Déploiement dans VITAMUI des certificats .p12 et .jks par Ansible

  Schéma de la PKI :

PKI

4.5.2. Explication avancée du fonctionnement

Le fonctionnement de la PKI de la solution *VitamUI* est basée à celle de Vitam - la logique d'architecture reste identique.

Lien des documentations existantes : PKI VITAM : http://www.programmevitam.fr/ressources/DocCourante/html/installation/annexes/10-overview_certificats.html?highlight=pki

PKI VITAM suite : https://www.programmevitam.fr/ressources/DocCourante/html/installation/annexes/15-certificates.html#

La PKI voit ses fichiers répartis à deux emplacements:

- deployment/pki

  A cet emplacement se trouvent les scripts et fichiers de configuration associés à la génération des assets (certificats, clés privées . . .)

Fichier	Description
pki/ca	Répertoire dans lequel sont stockés les CA de chaque zone
pki/config	Répertoire dans lequel sont stockées les configurations pour la génération des CA/certificats
pki/config/scripts	Répertoire dans lequel sont stockées les scripts de génération de la PKI.

4.5.3. Génération des certificats

Revenons en détails sur les scripts de génération des différents éléments de la PKI:

- generate_ca*.sh:
  - Paramètre(s):
    - ERASE [Facultatif]: Booléen indiquant si les CA et fichiers associés existants doivent être supprimés avant génération - Valeur par défaut: **false**
  - Description: Permet de générer les certificats d'autorité mentionnées dans le script de génération. Attention, toute autorité existante n'est pas regénérée, l'utilisation du paramètre **ERASE** sera recommandée lors de la première génération de la PKI.
- generate_certs*.sh
  - Paramètre(s):
    - ENVIRONNEMENT_FILE [Obligatoire]: Chemin vers le fichier d'environnement pour lequel les certificats vont être générés
    - ERASE [Facultatif]: Booléen indiquant si les certificats et fichiers associés existants doivent être supprimés avant génération - Valeur par défaut: **false**
  - Description: Permet de générer les certificats (serveur, client) mentionnés dans le script de génération. Attention, tout certificat existant n'est pas regénéré, l'utilisation du paramètre **ERASE** sera recommandée lors de la première génération de la PKI. Deux types de fichiers seront modifiés lors de cette exécution:
    - les fichiers de configuration des CA (serial, index.txt . . .)
    - les fichiers générés (`deployment/environment/certs`)

Les scripts suffixés par **_dev** concernent le matériel SSL utilisé pour le lancement de l'application en local sur l' environnement de developpement. L'ensemble des fichiers générés se trouveront dans l'arborescence **dev-deployment** du projet. Il faudra par la suite copier les fichiers générés associés à chaque module dans le repertoire /resources/dev du projet associé.

- deployment/environment/certs

A cet emplacement figure l'ensemble de la PKI de la solution. Par défaut, on retrouvera trois zones (une par autorité):

- server: l'ensemble du certificats permettant la communication HTTPS entre les différentes applications de la solution
- client-vitam: certificats utilisés par l'application pour communiquer avec Vitam. Avec le script **generate_certs.sh** fournis par la PKI, un certificat sera généré

pour s'interfacer avec Vitam.
- client-external: certificats des clients autorisés à solliciter les API externes

4.5.4. Cas pratiques

- Instaurer la communication entre la solution VitamUI <-> Vitam

  Quelques rappels:

  - au sein de la solution VitamUI, vous avez:
    - un client Vitam Java (access-external, ingest-external) permettant de réaliser des requêtes aurpès de Vitam. Ce client se base sur un fichier de configuration dans lesquels sont référencés un **keystore** (concenant le certificat utilisé pour chiffrer la requête) et un **trustore** (contenant le(s) CA(s) utilisé(s) pour les échanges avec les applications à l'extérieur de VitamUI)
    - `deployment/environement/certs/client-vitam/ca`: certificat d'autorité intervenant dans la communication VitamUI <-> Vitam. /! L'ensemble des CA présents dans ce répertoire seront embarqués dans le trustore exploités par le client Vitam Java lors de l'exécution du script *generate_keystores.sh*.
    - `deployment/environement/certs/client-vitam/clients/vitamui`: certificat utilisé pour la communication VitamUI <-> Vitam. /! Le certificat sera embarqué dans le keystore utilisé par le client Vitam Java lors de l'exécution du script *generate_keystores.sh*.
  - au sein de la solution Vitam, vous avez:
    - au sein du modèle de données, un certificat est associé à un contexte de sécurité (restriction d'actions par tenant à travers des contrats), lui-même associé à un profile de sécurité (permission sur les API externes). Cette association s'effectue dans le fichier `environment/group_vars/all/postinstall_param.yml`
    - la structure de la PKI VitamUI étant identique à celle de Vitam, le comportement est le suivant:
      - tout CA utilisé par un client pour solliciter les API externes et nécessaire à la chaine de vérification de son certificat doit se trouver dans le répertoire `envionment/certs/client-external/ca` /! L'ensemble des CA présents dans ce répertoire seront embarqués dans le trustore exploités par les API externes lors de l'exécution du script *generate_keystores.sh*.
      - le certificat d'un client accédant aux API externes doit figurer à l'emplacement `envionment/certs/client-external/clients/external`

  De ce fait, vous devez synchroniser vos PKI et vos solutions pour assurer une bonne communication:

  - VitamUI -> Vitam
    - Copier le CA du certificat VitamUI `{vitamui_inventory_dir}/certs/client-vitam` dans `{vitam_inventory_dir}/certs/client-external/ca`

        * Copier le certificat VitamUI {vitamui_inventory_dir}/certs/client-vitam/clie
          dans {vitam_inventory_dir}/certs/client-external/clients/external
        * Mise à jour de la PKI Vitam:
          · ./generate_stores.sh
          · ansible-playbook ansible-vitam/vitam.yml ${ANSIBLE_OPTS}
            --tags update_vitam_certificates
        * Création du contexte VitamUI:
          · Population du fichier postinstall_param.yml:

```
vitam_additional_securityprofiles:
- name: vitamui-security-profile
  identifier: vitamui-security-profile
  hasFullAccess: true
  permissions: "null"
  contexts:
    - name: vitamui-context
      identifier: vitamui-context
      status: ACTIVE
      enable_control: false
      # No control, idc about permissions, VitamUI will do it :)
      permissions: "[ { \"tenant\": 0, \"AccessContracts\": [], \"I
      certificates: ['external/vitamui.crt']
```

          · Exécution du playbook de mise à jour: ansible-playbook
            ansible-vitam-exploitation/add_contexts.yml
    – Vitam -> VitamUI
        * Copier le(s) CA(s) de Vitam {vitam_inventory_dir}/certs/client-vitam/ca
          dans {vitamui_inventory_dir}/certs/client-vitam/ca/
        * Mise à jour de la PKI VitamUI:
          · ./generate_stores.sh
          · ansible-playbook vitamui_apps.yml --tags update_vitamui_certificat

- Instaurer la communication entre la solution VitamUI <-> *Le monde extérieur*

  TODO (le process n'est pas encore industrialisé)

4.5.5. PKI de test

VITAMUI propose de générer à partir d'une PKI de tests les autorités de certification root et intermédiaires pour les clients et les serveurs. Cette PKI de test permet de connaître facilement l'ensemble des certificats nécessaires au bon fonctionnement de la solution. Attention, la PKI de test ne doit être utilisée que pour faire des tests, et ne doit surtout pas être utilisée en environnement de production.

4.5.6. Liste des certificats utilisés

Le tableau ci-dessous détail l'ensemble du contenu des keystores et truststores par service.

Composants	Keystores	Truststores
ui-portal	ui-portal.crt, ui-portal.key	ca-root.crt, ca-intermediate.crt
ui-identity	ui-identity.crt, ui-identity.key	ca-root.crt, ca-intermediate.crt

Composants	Keystores	Truststores
ui-identity-admin	ui-identity-admin.crt, ui-identity-admin.key	ca-root.crt, ca-intermediate.crt
ui-referential	ui-referential.crt, ui-referential.key	ca-root.crt, ca-intermediate.crt
ui-ingest	ui-ingest.crt, ui-ingest.key	ca-root.crt, ca-intermediate.crt
ui-archive-search	ui-archive-search.crt, ui-archive-search.key	ca-root.crt, ca-intermediate.crt
cas-server	cas-server.crt, cas-server.key	ca-root.crt, ca-intermediate.crt
iam-external	iam-external.crt, iam-external.key	ca-root.crt
iam-internal	iam-internal.crt, iam-internal.key	ca-root.crt
referential-external	referential-external.crt, referential-external.key	ca-root.crt
referential-internal	referential-internal.crt, referential-internal.key	ca-root.crt
ingest-external	ingest-external.crt, ingest-external.key	ca-root.crt
ingest-internal	ingest-internal.crt, ingest-internal.key	ca-root.crt
archive-search-external	archive-search-external.crt, archive-external.key	ca-root.crt
archive-search-internal	archive-search-internal.crt, archive-internal.key	ca-root.crt
security-server	security-server.crt, security-server	ca-root.crt

La liste des certificats utilisées par VITAM est décrite à cette adresse : http://www.programmevitam.fr/ressources/DocCourante/html/archi/securite/20-certificates.html

4.5.7. Procédure d'ajout d'un certificat client externe

Le certificat ou l'autorité de certification doit présent dans les truststores des APIs external VITAMUI. La procédure d'ajout d'un certificat client externe aux truststores des services de VITAMUI est la suivante :

- Déposer le(s) CA(s) du client dans le répertoire deployment/environment/certs/client-external/ca
- Déposer le certificat du client dans le répertoire deployment/environment/certs/client-external/clients/external/
- Régénérer les keystores à l'aide du script deployment/generate_stores.sh
- Exécuter le playbook pour redéployer les keystores sur la solution VITAMUI :

```
ansible-playbook vitamui_apps.yml -i environments/hosts --vault-password-file va
```

L’utilisation d’un certificat client sur les environnements VITAMUI nécessite également de vérifier que le certificat soit présent dans la base de données VITAMUI et rattaché à un contexte de sécurité du client.

4.6. Clusterisation

TODO

4.7. Détail des services

Service 1

- Description
- Contraintes

Service 2

- Description
- Contraintes

4.8. Détail des COTS

4.8.1. CAS

- Description
- Contraintes

4.8.2. Annuaire de services Consul

- Description
- Contraintes

La découverte des services est réalisée avec Consul via l’utilisation du protocole DNS. Le service DNS configuré lors du déploiement doit pouvoir résoudre les noms DNS associés à la fois aux service_id et aux instance_id. Tout hôte portant un service VITAMUI doit utiliser ce service DNS par défaut. L’installation et la configuration du service DNS applicatif sont intégrées à VITAMUI.

La résilience est assurée par l’annuaire de service Consul. Il est partagé avec VITAM. * Les services sont enregistrés au démarrage dans Consul * Les clients utilisent Consul (mode DNS) pour localiser les services * Consul effectue régulièrement des health checks sur les services enregistrés. Ces informations sont utilisées pour router les demandes des clients sur les services actifs

La solution de DNS applicatif intégrée à VITAMUI et VITAM est présentée plus en détails dans la section dédiée à Consul dans la documentation VITAM.

4.8.3. Base NOSQL MongoDB

- Description
- Contraintes

4.9. Multi instanciation des micro services

4.9.1. Multi instanciation

Les services vitamui multi instanciable à ce jour sont: - Service IAM Internal - Service IAM External - Service UI Identity - Service Portal - Service Referential Internal - Service Referential External - Service UI Referential - Service Ingest Internal - Service Ingest External - Service UI Ingest - Service Archive Search Internal - Service Archive Search External - Service UI Archive Search - Service Mongod (en cours de mise à niveau/!\)

Un load balancer/reverse proxy (à défaut Consul) est installé et configuré pour la répartition de charge entre différentes instances (cette configurtion est en cours de réalisation).

La configuration de la mémoire des services est par défaut: Xms=512m et Xmx=512m. cette configuration est modifiable, pour plus d'informations (cf: DEX).

4.9.2. Mono instanciation

Le fameux service mono instanciable dans vitamui est le serveur CAS.

5. Gestion du système

5.1. Chaîne de déploiement

- Les composants de la solution VITAM UI sont installés et configurés par un outil de déploiement automatique (ansible) dans des systèmes d'exploitation cibles (VM, container..).
- La procédure d'installation et de configuration se fait à travers un ensemble de scripts dont le source code est stocké dans le repository GIT.
- L'outil de déploiement (yum) utilise exclusivement les dépôts de packages pour installer les softs, afin de se décharger des étapes d'installation des dépendances et la gestion de conflit de fichiers.

  Schéma du processus de déploiement

Pocessus de déploiment

Packaging

Les packages VITAMUI sont disponibles au format RPM (CentOS).

Chaque package respecte les principes suivants :

- Nom des packages : vitamui du package
- Version du package : Numéro de "release" du projet

- Les dossiers (ainsi que les droits associés) compris dans les packages respectent les principes dictés dans la section dédiée aux utilisateurs, dossiers et droits.
- Les fichiers de configuration sont gérés par l'outil de déploiement de manière externe aux packages et ne sont pas inclus dans les packages.

Les composants de la solution VITAMUI sont tous disponibles sous forme de packages natif aux distributions supportées (rpm pour CentOS 7). Ceci inclut notamment :

- L'usage des pré-requis (au sens Require ou Depends) nativement inclus dans la distribution concernée
- L'arborescence des répertoires OS de la distribution concernée
- L'usage du système de démarrage systemd.

Les packages ne contiennent pas de pré/post action d'arrêt/démarrage/redémarrage de services. La gestion de démarrage des services et leur démarrage (a minima initial) est de la responsabilité de l'outillage de déploiement.

Les fichiers de configuration ne sont pas gérés dans les packages RPM. Par conséquent, ils n'apparaissent pas dans le résultat de commandes telles que rpm -ql. Les fichiers de configuration sont instanciés par l'outil de déploiement. Pour éviter la génération de fichier .rpmnew ou .rpmsave, il n'est pas utilisé la directive %config.

Les limitations associés au format de packaging choisi sont :

- L'instanciation d'une seule instance d'un même moteur par machine (il n'est ainsi pas possible d'installer 2 moteurs d'exécution sur le même OS) ;
- La redondance de certains contenus dans les packages (ex: les librairies Java sont embarquées dans les packages, et non tirées dans les dépendances de package)

Dépôts

L'installation de VITAMUI s'appuie sur des dépôts Nexus et dans le Repository RPM; ou tout autre repository mis en oeuvr suite au build. Il est également possible de déployer VitamUI en générant des dépôts locaux sur les machines cibles.

Principes de déploiement

Les principes généraux de déploiement sont les suivants :

- Les packages d'installation (rpm) sont identiques pour tous les environnements. Seule leur configuration change.
- La configuration des services est externalisée et gérée par l'outillage de déploiement.
- Le déploiement est décrit intégralement dans un fichier de définition du déploiement. En dehors des pré-requis, le déploiement initial est automatisé en totalité (sauf exception).
- Les services sont configurés par défaut pour permettre leur colocalisation (dans le sens de la colocalisation de deux instances de deux moteurs différents) (ex: dossiers d'installation / de fonctionnement différents, ports d'écoute différents, . . .).

- Le déploiement s'effectue à partir d'un point central. Les commandes passées sur chaque serveur à partir de ce point central utilisent le protocole SSH.

Les service de déploiement fourni permet le déploiement de la solution VITAMUI.

- Gestion des binaires d'installations (version, intégrité)
- Gestion des éléments de configuration spécifiques à chaque plate-forme
- Pilotage de l'installation des services sur les éléments d'infrastructure (VM/containers) de manière cohérente

Données gérées :

- Configuration technique du système VITAMUI
- Certificats x509 : le moteur de déploiement et de configuration doit posséder la référence des certificats techniques déployés sur la plate-forme (car il doit entre autres assurer la cohérence de ces certificats entre les différentes instances des composants VITAMUI déployés)

5.2. Cloisonnement

TODO

5.3. Logs techniques

La gestion des logs techniques dans VITAMUI est similaire à celle de VITAM. Pour une description complète du fonctionnement des logs et d'ELK, il est possible de se référer à la documentation VITAM.

- Doc VITAM : Chaîne de log - rsyslog / ELK

5.4. Supervision

TODO

5.5. Métriques

TODO

5.6. PRA

TODO

5.7. Les ontologies dans VitamUI

On affiche actuellement une liste statique des ontologies dans VitamUI, au niveau des applications ***Collecte***, ***Consultation et Recherche***.

En plus des ontologies statiques, nous avons ajouté une nouvelle option qui va permettre à un exploitant d'ajouter d'autres ontologies qui seront utilisées ensuite dans les deux applications de collecte et de recherche.

Il est nécessaire de déposer un fichier JSON dans **deployment/environments/ontology/** avec le nom : **external_ontology_fields.json** (c'est obligatoire d'avoir un fichier avec ce nom sinon la nouvelle liste des ontologies ne sera pas récupérée dans

VitamUI). C'est recommandé d'ajouter aussi les ontologies dans la base de données de Vitam.

Après l'installation de VitamUI le fichier des ontologies sera placé dans les deux répertoires (au niveau de la machine) : - vitamui/conf/archive-search-internal/ - vitamui/conf/collect-internal/

Ensuite, s'il y a un besoin d'ajouter des nouvelles ontologies, il suffit juste de modifier le fichier directement au niveau des machines. Sinon changer le fichier **environments/ontology/external_ontology_fields.json** et relancer les deux tâches : - Copy ontologies file to the service conf repository dans **Archive-search**. - Copy ontologies file to the service conf repository dans **Collect**.

Le fichier doit un être un JSON qui contient une liste d'objets.

Chaque objet représente une ontologie, et les informations qu'il faut renseigner pour chaque ontologie :

- ***Identifier*** : Identifier de l'ontologie (chaine de caractère).
- ***ApiField*** :
- ***Description*** : Description de l'ontologie (chaine de caractère).
- ***Type*** : le type de l'ontologie, les valeurs possible : `KEYWORD`, `DATE`, `LONG`, `BOOLEAN`, `DOUBLE`, `TEXT`
- ***Origin*** :
- ***CreationDate*** : date de création de l'ontologie.
- ***LastUpdate*** : date de la dernière modification de l'ontologie.
- ***ShortName*** : (chaine de caractère)
- ***TenantIds*** : La liste des tenants dont on pourra utiliser l'ontologie (liste des entiers), si le tenant **1** est parmi la liste des entiers donc l'ontologie en question sera visible sur l'ensemble des tenants.

Exemple :

```
{
  "Identifier": "DeactivationDate",
  "ApiField": "DeactivationDate",
  "Description": "Mapping : accesscontract-es-mapping",
  "Type": "DATE",
  "Origin": "EXTERNAL",
  "CreationDate": "2022-09-30T12:11:56.902",
  "LastUpdate": "2022-09-30T12:52:56.308",
  "ShortName": "Date de désactivation",
  "TenantIds" : [3,4]
}
```