VITAM - Documentation d'installation

Version 0.20.0

VITAM

juil. 21, 2017

Table des matières

CHAPITRE 1

Introduction

1.1 But de cette documentation

Ce document a pour but de permettre de fournir à une équipe d'exploitants de VITAM les procédures et informations utiles et nécessaires pour l'installation de la solution logicielle.

1.2 Destinataires de ce document

Ce document s'adresse à des exploitants du secteur informatique ayant de bonnes connaissances en environnement Linux.

CHAPITRE 2

Rappels

2.1 Information concernant les licences

La solution logicielle *VITAM* est publiée sous la license CeCILL 2.1 [1] ; la documentation associée (comprenant le présent document) est publiée sous Licence Ouverte V2.0 [2].

2.2 Documents de référence

2.2.1 Documents internes

Tableau 2.1 – Documents de référence VITAM

Nom	Lien
DAT	(à renseigner)
DIN	(à renseigner)
DEX	(à renseigner)
Release notes	(à renseigner)

2.2.2 Référentiels externes

2.3 Glossaire

API Application Programming Interface

BDD Base De Données

COTS Component Off The Shelves ; il s'agit d'un composant "sur étagère", non développé par le projet *VITAM*, mais intégré à partir d'un binaire externe. Par exemple : MongoDB, ElasticSearch.

DAT Dossier d'Architecture Technique

DEX Dossier d'EXploitation

DIN Dossier d'Installation

1. http://www.cecill.info/licences/Licence_CeCILL_V2.1-fr.html
2. https://www.etalab.gouv.fr/wp-content/uploads/2017/04/ETALAB-Licence-Ouverte-v2.0.pdf

DNSSEC *Domain Name System Security Extensions* est un protocole standardisé par l'IETF permettant de résoudre certains problèmes de sécurité liés au protocole DNS. Les spécifications sont publiées dans la RFC 4033 et les suivantes (une version antérieure de DNSSEC n'a eu aucun succès). Définition DNSSEC [3]

DUA Durée d'Utilité Administrative

IHM Interface Homme Machine

JRE Java Runtime Environment ; il s'agit de la machine virtuelle Java permettant d'y exécuter les programmes compilés pour.

JVM Java Virtual Machine ; Cf. *JRE*

MitM L'attaque de l'homme du milieu (HDM) ou *man-in-the-middle attack* (MITM) est une attaque qui a pour but d'intercepter les communications entre deux parties, sans que ni l'une ni l'autre ne puisse se douter que le canal de communication entre elles a été compromis. Le canal le plus courant est une connexion à Internet de l'internaute lambda. L'attaquant doit d'abord être capable d'observer et d'intercepter les messages d'une victime à l'autre. L'attaque « homme du milieu » est particulièrement applicable dans la méthode d'échange de clés Diffie-Hellman, quand cet échange est utilisé sans authentification. Avec authentification, Diffie-Hellman est en revanche invulnérable aux écoutes du canal, et est d'ailleurs conçu pour cela. Explication [4]

NoSQL Base de données non-basée sur un paradigme classique des bases relationnelles. Définition [5]

OAIS *Open Archival Information System*, acronyme anglais pour Systèmes de transfert des informations et données spatiales – Système ouvert d'archivage d'information (SOAI) - Modèle de référence.

PDMA Perte de Données Maximale Admissible ; il s'agit du pourcentage de données stockées dans le système qu'il est acceptable de perdre lors d'un incident de production.

PKI Une infrastructure à clés publiques (ICP) ou infrastructure de gestion de clés (IGC) ou encore Public Key Infrastructure (PKI), est un ensemble de composants physiques (des ordinateurs, des équipements cryptographiques logiciels ou matériel type HSM ou encore des cartes à puces), de procédures humaines (vérifications, validation) et de logiciels (système et application) en vue de gérer le cycle de vie des certificats numériques ou certificats électroniques. Définition PKI [6]

REST REpresentational State Transfer : type d'architecture d'échanges. Appliqué aux services web, en se basant sur les appels http standard, il permet de fournir des API dites "RESTful" qui présentent un certain nombre d'avantages en termes d'indépendance, d'universalité, de maintenabilité et de gestion de charge. Définition [7]

RPM Red Hat Package Manager ; il s'agit du format de packets logiciels nativement utilisé par les distributions CentOS (entre autres)

SAE Système d'Archivage Électronique

SEDA Standard d'Échange de Données pour l'Archivage

SIA Système d'Informations Archivistique

TNR Tests de Non-Régression

VITAM Valeurs Immatérielles Transférées aux Archives pour Mémoire

3. https://fr.wikipedia.org/wiki/Domain_Name_System_Security_Extensions
4. https://fr.wikipedia.org/wiki/Attaque_de_l'homme_du_milieu
5. https://fr.wikipedia.org/wiki/NoSQL
6. https://fr.wikipedia.org/wiki/Infrastructure_%C3%A0_cl%C3%A9s_publiques
7. https://fr.wikipedia.org/wiki/Representational_state_transfer

CHAPITRE 3

Architecture de la solution logicielle VITAM

Le schéma ci-dessous représente une solution *VITAM* :

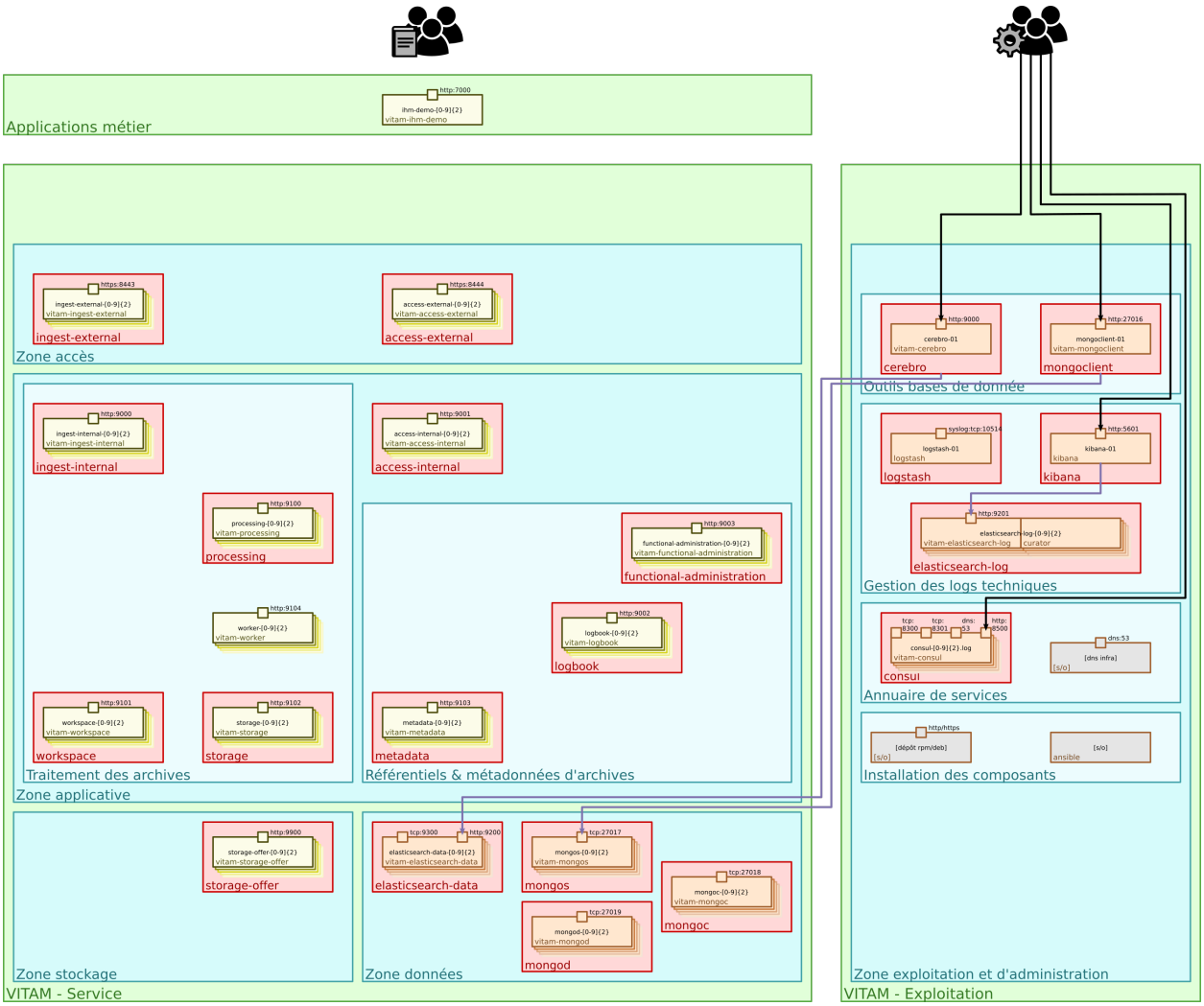

Fig. 3.1 – Vue d'ensemble d'un déploiement VITAM : zones, composants

Voir aussi :

Se référer au *DAT* (et notamment le chapitre dédié à l'architecture technique) pour plus de détails, en particulier concernant les flux entre les composants.

CHAPITRE 4

Pré-requis

4.1 Description

Les pré-requis logiciels suivants sont nécessaires :

4.1.1 Base commune

- Tous les serveurs hébergeant la solution *VITAM* doivent être synchronisés sur un serveur de temps (pas de stratum 10)
- Disposer de la solution de déploiement basé sur ansible

Le déploiement est orchestré depuis un poste ou serveur d'administration ; les pré-requis suivants doivent y être présents :

- packages nécessaires :
  - ansible (version 2.3 minimale et conseillée)
  - openssh-clients (client SSH utilisé par ansible)
  - java-1.8.0-openjdk & openssl (du fait de la génération de certificats / stores, l'utilitaire `keytool` est nécessaire)
- un accès ssh vers un utilisateur d'administration avec élévation de privilèges vers les droits root, vitam, vitamdb sur les serveurs cibles.
- Le compte utilisé sur le serveur d'administration doit avoir confiance dans les serveurs cibles (fichier ~/.ssh/known_hosts correctement renseigné)

Prudence : Les IP des machines sur lesquelles la solution Vitam sera installée ne doivent pas changer d'IP au cours du temps, en cas de changement d'IP, la plateforme ne pourra plus fonctionner.

Prudence : dans le cadre de l'installation des packages "extra", il est nécessaire, pour les partitions hébergeant des containeurs docker (mongo-express, head), qu'elles aient un accès internet.

Avertissement : dans le cas d'une installation du composant **vitam-offer** en *filesystem-hash*, il est fortement recommandé d'employer un système de fichiers **xfs** pour le stockage des données. Se référer au *DAT* pour connaître la structuration des filesystems dans *VITAM*. En cas d'utilisation d'un autre type, s'assurer que le filesystem possède/gère bien l'option **user_xattr**.

4.1.2 Déploiement sur environnement CentOS

- Disposer d'une plate-forme Linux CentOS 7 installée selon la répartition des services souhaitée. En particulier, ces serveurs doivent avoir :
  - une configuration de temps synchronisée (ex : en récupérant le temps à un serveur centralisé)
  - Des autorisations de flux conformément aux besoins décrits dans le *DAT*
  - une configuration des serveurs de noms correcte (cette configuration sera surchargée lors de l'installation)
  - un accès à un dépôt (ou son miroir) CentOS 7 (base et extras) et EPEL 7
- Disposer des binaires VITAM : paquets RPM de VITAM (vitam-product) ainsi que les paquets d'éditeurs tiers livrés avec Vitam (vitam-external)

4.1.3 Déploiement sur environnement Debian

- Disposer d'une plate-forme Linux Debian "jessie" installée selon la répartition des services souhaitée. En particulier, ces serveurs doivent avoir :
  - une configuration de temps synchronisée (ex : en récupérant le temps à un serveur centralisé)
  - Des autorisations de flux conformément aux besoins décrits dans le *DAT*
  - une configuration des serveurs de noms correcte (cette configuration sera surchargée lors de l'installation)
  - un accès à un dépôt (ou son miroir) Debian (base et extras) et jessie-backports
  - un accès internet, car le dépôt docker sera ajouté
- Disposer des binaires VITAM : paquets deb de VITAM (vitam-product) ainsi que les paquets d'éditeurs tiers livrés avec Vitam (vitam-external)

4.2 Matériel

Les prérequis matériel sont définis dans le *DAT* ; à l'heure actuelle, le minimum recommandé pour la solution Vitam est 2 CPUs. Il également est recommandé de prévoir (paramétrage par défaut à l'installation) 512Mo de RAM disponible par composant applicatif *VITAM* installé sur chaque machine (hors elasticsearch et mongo).

Concernant l'espace disque, à l'heure actuelle, aucun pré-requis n'a été défini ; cependant, sont à prévoir par la suite des espaces de stockage conséquents pour les composants suivants :

- storage-offer-default
- solution de centralisation des logs (elasticsearch)
- workspace
- worker (temporairement, lors du traitement de chaque fichier à traiter)
- elasticsearch des données Vitam

L'arborescence associée sur les partitions associées est : `/vitam/data/<composant>`

CHAPITRE 5

Dépendances aux services d'infrastructures

5.1 Ordonnanceurs techniques / batchs

5.1.1 Curator

Curator permet d'effectuer des opérations périodiques de maintenance sur les index elasticsearch. Les jobs Curator sont initiés automatiquement au déploiement de VITAM et sont lancés via un timer systemd [8] sur chaque serveur.

Voir aussi :

Plus de détails sont disponibles dans la présentation de curator

5.1.2 Sécurisation des journaux d'opérations

Job de sécurisation du logbook : lancé toutes les nuits peu après minuit sur une des machines (la dernière dans la liste de déploiement) hébergeant le composant vitam-logbook.

Prudence : Dans cette release, ce job est le seul à être lancé par le crontab système ; il sera migré en timer systemd prochainement.

5.1.3 Sécurisation des journaux d'écriture

La sécurisation des journaux d'écriture est un processus local à chaque serveur hébergeant une instance du moteur de stockage ;

5.1.4 Cas de la sauvegarde

Se référer au *DAT* (dans la section dédiée) et *DEX*.

8. https://www.freedesktop.org/software/systemd/man/systemd.timer.html

5.2 Socles d'exécution

5.2.1 OS

Seules deux distributions Linux sont supportées à ce jour :

- CentOS 7
- Debian 8 (jessie)

SELinux doit être configuré en mode `permissive` ou `disabled`.

5.2.2 Middlewares

- Java : JRE 8 ; les versions suivantes ont été testées :
  - OpenJDK 1.8.0, dans la version présente dans les dépôts officiels au moment de la parution cette release de Vitam (Centos et Debian en 1.8.0_131)

CHAPITRE 6

Fiche type de déploiement VITAM

6.1 Fiche-type VITAM

Prudence : cette liste a pour but d'évoluer et s'étoffer au fur et à mesure des mises à jour des composants et du contenu des fichiers de déploiement de VITAM.

Tableau 6.1 – Tableau récapitulatif des informations à renseigner pour VITAM

Nom du composant	Descriptif	Valeur d'exemple	Valeur choisie	Si HA ?
IHM-demo machine	interface web	vitam-prod-app-1.internet.agri		
ingest-external machine	interface web	vitam-prod-app-1.internet.agri		
ingest-internal machine	interface web	vitam-prod-app-1.internet.agri		
access-external machine	interface web	vitam-prod-app-1.internet.agri		
access-internal machine	interface web	vitam-prod-app-1.internet.agri		
logbook machine	interface web	vitam-prod-app-1.internet.agri		
metadata machine	interface web	vitam-prod-app-1.internet.agri		
processing machine(s)	base de données	vitam-prod-app-1.internet.agri		
worker machine(s)	Traitement de fichiers	vitam-prod-wrk-1.internet.agri		
storage-engine machine(s)	xxxx	vitam-prod-app-1.internet.agri		
storage-offer-default machine(s)	implémentation de pilote de stockage	vitam-prod-app-1.internet.agri		
Consul servers	implémentation de Consul pour un DNS applicatif (nécessite 3 serveurs minimum ; règle (2*n+1))	vitam-prod-app-1.internet.agri, vitam-prod-app-2.internet.agri, vitam-prod-app-3.internet.agri		
elasticsearch data machine(s)	Cluster ElasticSearch de données VITAM (3 machines)	vitam-prod-ela-1.internet.agri,vitam-prod-ela-2.internet.agri,vitam-prod-ela-3.internet.agri		
elasticsearch log machine(s)	Cluster ElasticSearch de log VITAM (3 machines)	vitam-prod-log-1.internet.agri,vitam-prod-log-2.internet.agri,vitam-prod-log-3.internet.agri		
mongo-s machine(s)	Cluster MongoDB de routage de data VITAM (3 machines)	vitam-prod-ms-1.internet.agri,vitam-prod-ms-2.internet.agri,vitam-prod-ms-3.internet.agri		
mongo-c machine(s)	Cluster MongoDB de configuration des données VITAM (3 machines)	vitam-prod-mc-1.internet.agri,vitam-prod-mc-2.internet.agri,vitam-prod-mc-3.internet.agri		
mongo-d machine(s)	Cluster Mongo de données VITAM (3 machines)	vitam-prod-md-1.internet.agri,vitam-prod-md-2.internet.agri,vitam-prod-md-3.internet.agri		
log central machine(s)	Centralisation des logs	vitam-prod-log-1.internet.agri		

CHAPITRE 7

Récupération de la version

7.1 Cas particulier des partenaires

Se connecter sur l'URL de support partenaires et récupérer :

- le package de livraison
- la release notes
- les empreintes de contrôle

Sur la machine "ansible" dévouée au déploiement de *VITAM*, décompacter le package (au format `tar.gz`).

Sur le repository "VITAM", récupérer également depuis le tar.gz les binaires d'installation (rpm pour CentOS ; deb pour Debian) et les faire prendre en compte par le repository.

7.2 Pour les autres

Les scripts de déploiement de VITAM sont disponibles dans le dépôt github VITAM [9] , dans le répertoire `deployment`.

Les binaires de VITAM sont disponibles sur les dépôts bintray [10] ; ces dépôts doivent être correctement configurés sur la plate-forme cible avant toute installation.

7.2.1 Repository pour environnement CentOS

Sur les partitions cibles, configurer le fichier `/etc/yum.repos.d/vitam-repositories.repo` (remplacer <branche_vitam> par le nom de la branche de support à installer) comme suit

```
[vitam-bintray--programmevitam-vitam-rpm-release-product]
name=vitam-bintray--programmevitam-vitam-rpm-release-product
baseurl=https://dl.bintray.com/programmevitam/vitam-rpm-release/centos/7/vitam-
↪product/<branche_vitam>/
gpgcheck=0
repo_gpgcheck=0
enabled=1

[vitam-bintray--programmevitam-vitam-rpm-release-external]
name=vitam-bintray--programmevitam-vitam-rpm-release-external
```

9. https://github.com/ProgrammeVitam/vitam
10. https://bintray.com/programmevitam

```
baseurl=https://dl.bintray.com/programmevitam/vitam-rpm-release/centos/7/vitam-
↪external/<branche_vitam>/
gpgcheck=0
repo_gpgcheck=0
enabled=1
```

7.2.2 Repository pour environnement Debian

Sur les partitions cibles, configurer le fichier `/etc/apt/sources.list.d/vitam-repositories.list` (remplacer <branche_vitam> par le nom de la branche de support à installer) comme suit

```
deb [trusted=yes] https://dl.bintray.com/programmevitam/vitam-deb-release jessie␣
↪vitam-product-<branche_vitam> vitam-external-<branche_vitam>
```

CHAPITRE 8

Explications relatives à la génération des certificats

Cette section a pour but de présenter les notions relatives à la sécurité liée aux certificats ; l'application de ces principes d'utilisation sera revue dans la section de procédure d'installation.

8.1 Introduction sur les certificats dans Vitam

8.1.1 Liste des suites cryptographiques & protocoles supportés par Vitam

Il est possible de consulter les ciphers supportés par Vitam dans deux fichiers disponibles sur ce chemin : *ansible-vitam/roles/vitam/templates/*

- **Le fichier jetty-config.xml.j2**
    - La balise contenant l'attribut name="IncludeCipherSuites" référence les ciphers supportés
    - La balise contenant l'attribut name="ExcludeCipherSuites" référence les ciphers non supportés
- **Le fichier java.security.j2**
    - La ligne jdk.tls.disabledAlgorithms renseigne les ciphers désactivés au niveau java

Avertissement : Les 2 balises concernant les ciphers sur le fichier jetty-config.xml.j2 sont complémentaires car elles comportent des wildcards (*) ; en cas de conflit, l'exclusion est prioritaire.

Voir aussi :

Ces fichiers correspondent à la configuration recommandée ; celle-ci est décrite plus en détail dans le DAT (chapitre sécurité).

8.1.2 Vue d'ensemble de la gestion des certificats

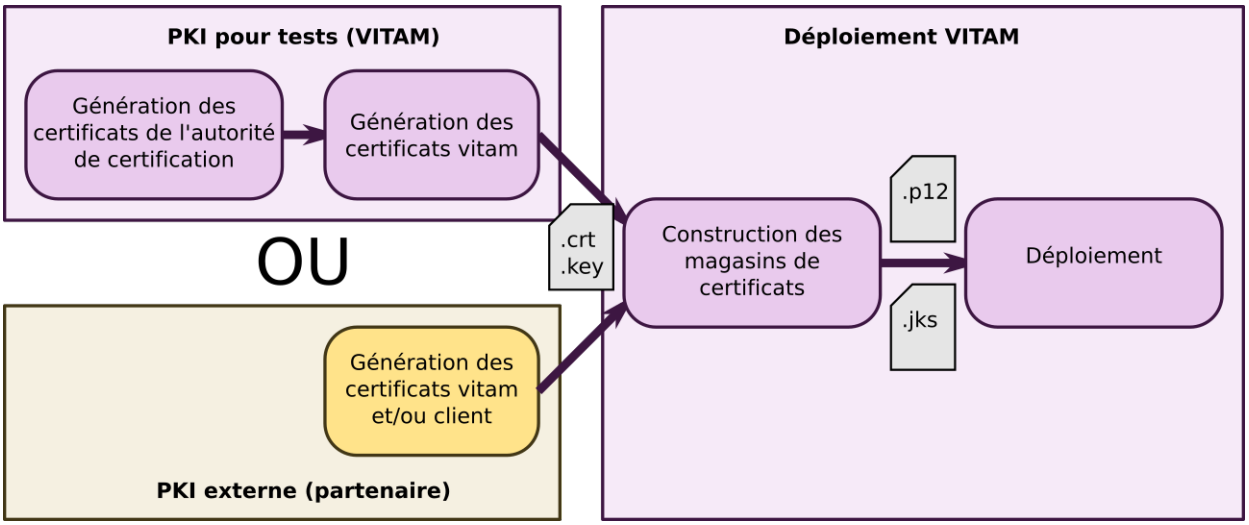

Fig. 8.1 – Vue d'ensemble de la gestion des certificats au déploiement

8.1.3 Description de l'arborescence de la PKI

Tous les fichiers de gestion de la PKI se trouvent dans le répertoire `deployment` de l'arborescence Vitam :

- Le sous répertoire `pki` contient les scripts de génération des CA & des certificats, les CA générées par les scripts, et les fichiers de configuration d'openssl
- Le sous répertoire `environments` contient tous les certificats nécessaires au bon déploiement de Vitam :
  - certificats publics des CA
  - Certificats clients, serveurs, de timestamping, et coffre fort contenant les mots de passe des clés privées des certificats (sous-répertoire `certs`)
  - Magasins de certificats (keystores / truststores / grantedstores), et coffre fort contenant les mots de passe des magasins de certificats (sous-répertoire `keystores`)
- Le script `generate_stores.sh` génère les magasins de certificats (keystores), cf la section *Fonctionnement des scripts de la PKI* (page 19)

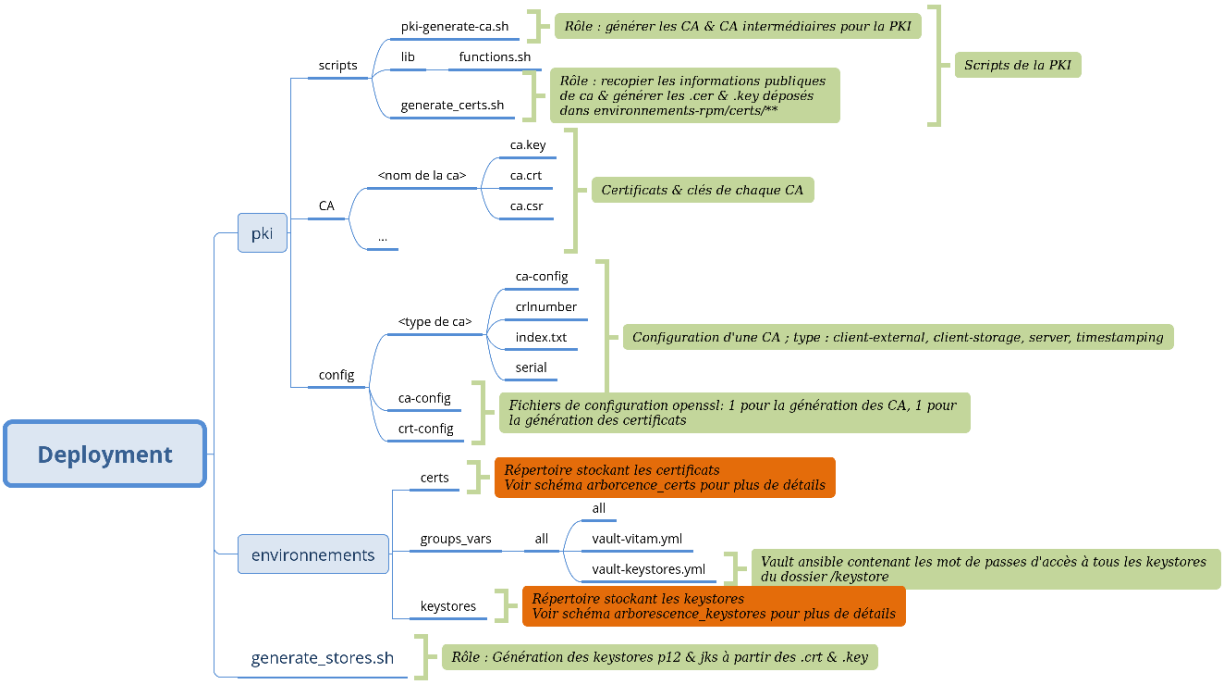

Fig. 8.2 – Vue l'arborescence de la PKI Vitam

8.1.4 Description de l'arborescence du répertoire deployment/environments/certs

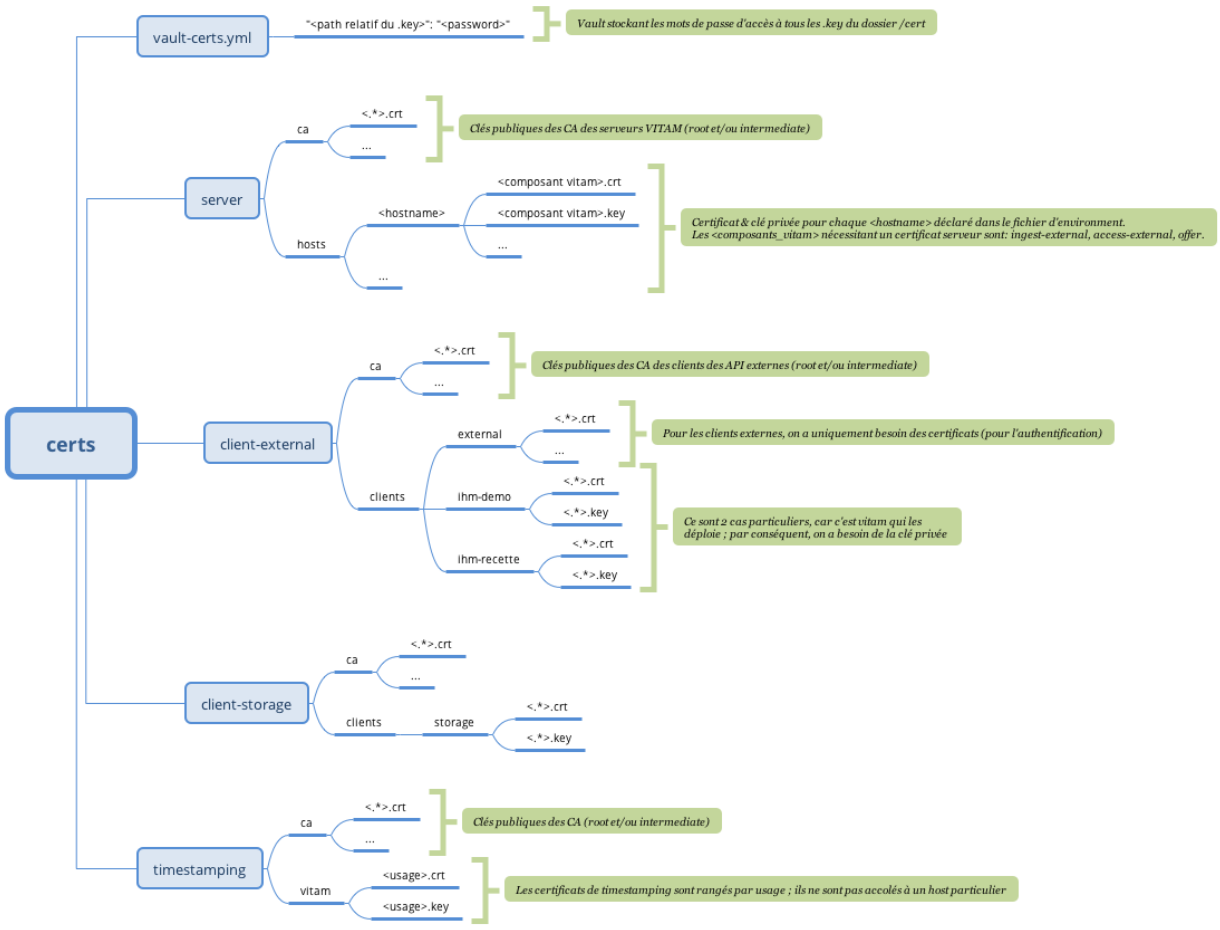

Fig. 8.3 – Vue détaillée de l'arborescence des certificats

8.1.5 Description de l'arborescence du répertoire deployment/environments/keystores

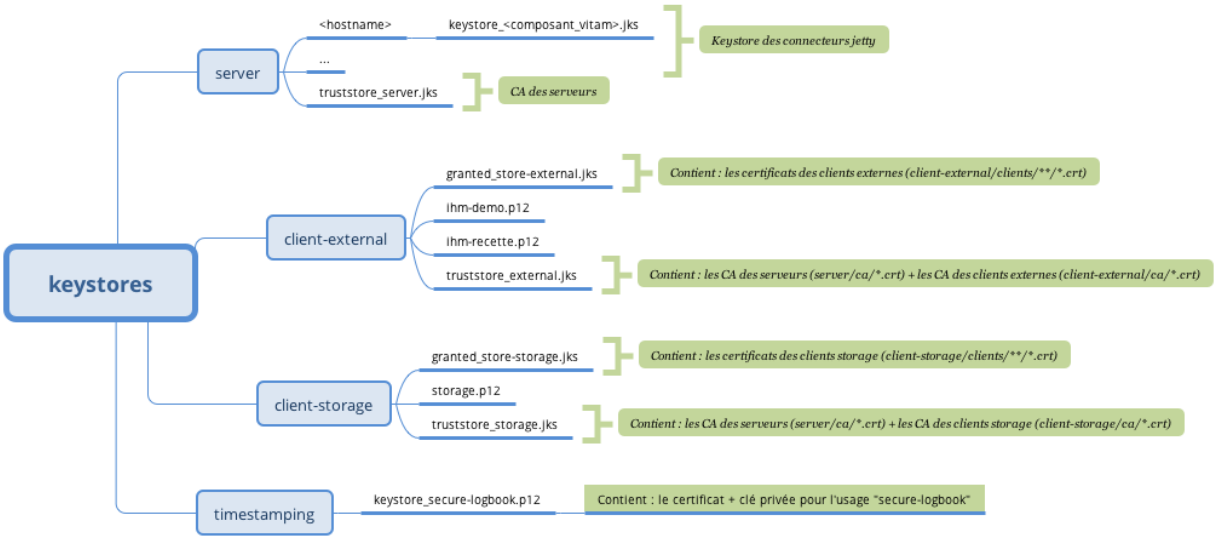

Fig. 8.4 – Vue détaillée de l'arborescence des keystores

8.1.6 Fonctionnement des scripts de la PKI

La gestion de la PKI se fait avec 3 scripts dans le répertoire deployment de l'arborescence Vitam :

- `pki/scripts/generate_ca.sh` : génère des autorités de certifications (si besoin)
- `pki/scripts/generate_certs.sh` : génère des certificats à partir des autorités de certifications présentes (si besoin)
  - Récupère le mot de passe des clés privées à générer dans le vault `environments/certs/vault-certs.yml`
  - Génère les certificats & les clés privées
- `generate_stores.sh` : génère les magasins de certificats nécessaires au bon fonctionnement de Vitam
  - Récupère le mot de passe du magasin indiqué dans `environments/group_vars/all/vault-keystore.yml`
  - Insère les bon certificats dans les magasins qui en ont besoin

Si les certificats sont créés par la PKI externe, il faut donc les positionner dans l'arborescence attendue avec le nom attendu pour certains (cf *Vue détaillée de l'arborescence des certificats* (page 26))

8.2 Cas 1 : Je ne dispose pas de PKI, je souhaite utiliser celle de Vitam

Dans ce cas, il est nécessaire d'utiliser la *PKI* fournie avec la solution logicielle VITAM.

8.2.1 Procédure générale

Danger : La *PKI* fournie avec la solution logicielle Vitam ne doit être utilisée que pour faire des tests, et ne doit par conséquent surtout pas être utilisée en environnement de production !

La *PKI* de la solution logicielle VITAM est une suite de scripts qui vont générer dans l'ordre ci-dessous :

- Les autorités de certifcation (CA)
- Les certificats (clients, serveurs, de timestamping) à partir des CA
- Les keystores, en important les certificats et CA nécessaires pour chacun des keystores

8.2.2 Génération des CA par les scripts Vitam

Il faut faire générer les autorités de certification par le script décrit ci-dessous.

Dans le répertoire de déploiement, lancer le script :

```
pki/scripts/generate_ca.sh
```

Ce script génère sous `pki/ca` les autorités de certification root et intermédiaires pour générer des certificats clients, serveurs, et de timestamping.

Avertissement : Bien noter les dates de création et de fin de validité des CA. En cas d'utilisation de la PKI fournie, la CA root a une durée de validité de 10 ans ; la CA intermédiaire a une durée de 3 ans.

Voici ci-dessous un exemple de rendu du script :

```
[INFO] [generate_ca.sh] Lancement de la procédure de création des CA
[INFO] [generate_ca.sh] ================================================
[INFO] [generate_ca.sh] Répertoire /home/utilisateur/git/vitam/deployment/pki/ca␣
↪absent ; création...
[INFO] [generate_ca.sh] Création du répertoire de travail temporaire tempcerts sous /
↪home/utilisateur/git/vitam/deployment/pki/tempcerts...
[INFO] [generate_ca.sh] Création de CA root pour server...
[INFO] [generate_ca.sh] Create CA request...
Generating a 2048 bit RSA private key
.........................+++
.....................+++
writing new private key to '/home/utilisateur/git/vitam/deployment/pki/ca/server/ca-
↪root.key'
-----
[INFO] [generate_ca.sh] Create CA certificate...
Using configuration from /home/utilisateur/git/vitam/deployment/pki/config/server/ca-
↪config
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'CA_server'
organizationName      :ASN.1 12:'Vitam.'
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :ASN.1 12:'idf'
localityName          :ASN.1 12:'paris'
Certificate is to be certified until Feb 26 16:29:14 2027 GMT (3650 days)
```

```
Write out database with 1 new entries
Data Base Updated
[INFO] [generate_ca.sh] Création de la CA intermediate pour server...
     [...]
Write out database with 1 new entries
Data Base Updated
[INFO] [generate_ca.sh] ----------------------------------------------
[INFO] [generate_ca.sh] ==============================================
[INFO] [generate_ca.sh] Fin de la procédure de création des CA
```

8.2.3 Génération des certificats par les scripts Vitam

Le fichier d'inventaire de déploiement `environments/<fichier d'inventaire>` (cf. *Informations "plate-forme"* (page 31)) doit être correctement renseigné pour indiquer les serveurs associés à chaque service. En prérequis les CA doivent être présentes.

Puis, dans le répertoire de déploiement, lancer le script :

```
pki/scripts/generate_certs.sh <fichier d'inventaire>
```

Ci-dessous un exemple de sortie du script :

```
[INFO] [generate_certs.sh] Suppression de l'ancien vault
[INFO] [generate_certs.sh] Recopie des clés publiques des CA
[INFO] [generate_certs.sh] Copie de la CA (root + intermediate) de client-external
[INFO] [generate_certs.sh] Copie de la CA (root + intermediate) de client-storage
[INFO] [generate_certs.sh] Copie de la CA (root + intermediate) de server
[INFO] [generate_certs.sh] Copie de la CA (root + intermediate) de timestamping
[INFO] [generate_certs.sh] Génération des certificats serveurs
[INFO] [generate_certs.sh] Création du certificat server pour ingest-external hébergé␣
↪sur localhost...
[INFO] [generate_certs.sh] Generation de la clé...
Generating a 4096 bit RSA private key
..............................................................................
↪.....................................................++
.....................................++
writing new private key to '/home/utilisateur/git/vitam/deployment/environments/certs/
↪server/hosts/localhost/ingest-external.key'
-----
[INFO] [generate_certs.sh] Generation du certificat signé avec CA server...
Using configuration from /home/utilisateur/git/vitam/deployment/pki/config/server/ca-
↪config
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'ingest-external.service.consul'
organizationName      :ASN.1 12:'Vitam.'
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :ASN.1 12:'idf'
localityName          :ASN.1 12:'paris'
Certificate is to be certified until Feb 29 09:37:00 2020 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
Encryption successful
```

```
[INFO] [generate_certs.sh] Création du certificat server pour access-external hébergé␣
↪sur localhost...
[INFO] [generate_certs.sh] Generation de la clé...
Generating a 4096 bit RSA private key
..........++
..................++
writing new private key to '/home/utilisateur/git/vitam/deployment/environments/certs/
↪server/hosts/localhost/access-external.key'
-----
[INFO] [generate_certs.sh] Generation du certificat signé avec CA server...
Using configuration from /home/utilisateur/git/vitam/deployment/pki/config/server/ca-
↪config
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'access-external.service.consul'
organizationName      :ASN.1 12:'Vitam.'
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :ASN.1 12:'idf'
localityName          :ASN.1 12:'paris'
Certificate is to be certified until Feb 29 09:37:01 2020 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
Decryption successful
Encryption successful
[INFO] [generate_certs.sh] Création du certificat server pour storage-offer-default␣
↪hébergé sur localhost...
[INFO] [generate_certs.sh] Generation de la clé...
Generating a 4096 bit RSA private key
.....................++
........++
writing new private key to '/home/utilisateur/git/vitam/deployment/environments/certs/
↪server/hosts/localhost/storage-offer-default.key'
-----
[INFO] [generate_certs.sh] Generation du certificat signé avec CA server...
Using configuration from /home/utilisateur/git/vitam/deployment/pki/config/server/ca-
↪config
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'storage-offer-default.service.consul'
organizationName      :ASN.1 12:'Vitam.'
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :ASN.1 12:'idf'
localityName          :ASN.1 12:'paris'
Certificate is to be certified until Feb 29 09:37:02 2020 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
Decryption successful
Encryption successful
[INFO] [generate_certs.sh] Génération des certificats timestamping
[INFO] [generate_certs.sh] Création du certificat timestamping pour logbook
[INFO] [generate_certs.sh] Generation de la clé...
Generating a 4096 bit RSA private key
..........................................................................................
↪...................++
```

```
...........++
writing new private key to '/home/utilisateur/git/vitam/deployment/environments/certs/
↪timestamping/vitam/logbook.key'
-----
[INFO] [generate_certs.sh] Generation du certificat signé avec CA timestamping...
Using configuration from /home/utilisateur/git/vitam/deployment/pki/config/
↪timestamping/ca-config
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'logbook.service.consul'
organizationName      :ASN.1 12:'Vitam.'
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :ASN.1 12:'idf'
localityName          :ASN.1 12:'paris'
Certificate is to be certified until Feb 29 09:37:04 2020 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
Decryption successful
Encryption successful
[INFO] [generate_certs.sh] Génération des certificats clients
[INFO] [generate_certs.sh] Création du certificat client pour ihm-demo
[INFO] [generate_certs.sh] Generation de la clé...
Generating a 4096 bit RSA private key
....++
........++
writing new private key to '/home/utilisateur/git/vitam/deployment/environments/certs/
↪client-external/clients/ihm-demo/ihm-demo.key'
-----
[INFO] [generate_certs.sh] Generation du certificat signé avec client-external...
Using configuration from /home/utilisateur/git/vitam/deployment/pki/config/client-
↪external/ca-config
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'ihm-demo'
organizationName      :ASN.1 12:'Vitam.'
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :ASN.1 12:'idf'
localityName          :ASN.1 12:'paris'
Certificate is to be certified until Feb 29 09:37:04 2020 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
Decryption successful
Encryption successful
[INFO] [generate_certs.sh] Création du certificat client pour ihm-recette
[INFO] [generate_certs.sh] Generation de la clé...
Generating a 4096 bit RSA private key
.....................................................................++
.....................................................................
↪.....................................................................++
writing new private key to '/home/utilisateur/git/vitam/deployment/environments/certs/
↪client-external/clients/ihm-recette/ihm-recette.key'
-----
[INFO] [generate_certs.sh] Generation du certificat signé avec client-external...
Using configuration from /home/utilisateur/git/vitam/deployment/pki/config/client-
↪external/ca-config
```

```
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'ihm-recette'
organizationName      :ASN.1 12:'Vitam.'
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :ASN.1 12:'idf'
localityName          :ASN.1 12:'paris'
Certificate is to be certified until Feb 29 09:37:06 2020 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
Decryption successful
Encryption successful
[INFO] [generate_certs.sh] Création du certificat client pour reverse
[INFO] [generate_certs.sh] Generation de la clé...
Generating a 4096 bit RSA private key
..............................++
.....................................................................................
 ↪...........++
writing new private key to '/home/utilisateur/git/vitam/deployment/environments/certs/
 ↪client-external/clients/reverse/reverse.key'
-----
[INFO] [generate_certs.sh] Generation du certificat signé avec client-external...
Using configuration from /home/utilisateur/git/vitam/deployment/pki/config/client-
 ↪external/ca-config
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'reverse'
organizationName      :ASN.1 12:'Vitam.'
countryName           :PRINTABLE:'FR'
stateOrProvinceName   :ASN.1 12:'idf'
localityName          :ASN.1 12:'paris'
Certificate is to be certified until Feb 29 09:37:07 2020 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
Decryption successful
Encryption successful
[INFO] [generate_certs.sh] Création du certificat client pour storage-engine
[INFO] [generate_certs.sh] Generation de la clé...
Generating a 4096 bit RSA private key
...........++
.....................................................................................
 ↪....++
writing new private key to '/home/utilisateur/git/vitam/deployment/environments/certs/
 ↪client-storage/clients/storage-engine/storage-engine.key'
-----
[INFO] [generate_certs.sh] Generation du certificat signé avec client-storage...
Using configuration from /home/utilisateur/git/vitam/deployment/pki/config/client-
 ↪storage/ca-config
Check that the request matches the signature
Signature ok
The Subject's Distinguished Name is as follows
commonName            :ASN.1 12:'storage-engine'
organizationName      :ASN.1 12:'Vitam.'
countryName           :PRINTABLE:'FR'
```

```
stateOrProvinceName   :ASN.1 12:'idf'
localityName          :ASN.1 12:'paris'
Certificate is to be certified until Feb 29 09:37:08 2020 GMT (1095 days)

Write out database with 1 new entries
Data Base Updated
Decryption successful
Encryption successful
[INFO] [generate_certs.sh] Fin de script
```

Ce script génère sous `environmements/certs` les certificats (format crt & key) nécessaires pour un bon fonctionnement dans VITAM. Les mots de passe des clés privées des certificats sont stockés dans le vault ansible environmements/certs/vault-certs.yml

Prudence : Les certificats générés à l'issue ont une durée de validité de (à vérifier).

8.2.4 Génération des magasins de certificats

En prérequis, les certificats et les autorités de certification doivent être présents dans les répertoires attendus.

Prudence : Avant de lancer le script de génération des stores, il est nécessaire de modifier le vault contenant les mots de passe des stores : `environmements/group_vars/all/vault-keystores.yml`, décrit dans la section *PKI* (page 43).

Lancer le script :

```
./generate_stores.sh
```

Ci-dessous un exemple de sortie du script :

```
[INFO] [generate_stores.sh] ------------------------------------------
[INFO] [generate_stores.sh] Creation du keystore de access-external pour le serveur␣
↪localhost
[INFO] [generate_stores.sh] Génération du p12
[INFO] [generate_stores.sh] Génération du jks
Entry for alias access-external successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or␣
↪cancelled
[INFO] [generate_stores.sh] Suppression du p12
[INFO] [generate_stores.sh] ------------------------------------------
[INFO] [generate_stores.sh] Creation du keystore de ingest-external pour le serveur␣
↪localhost
    [...]
[INFO] [generate_stores.sh] ------------------------------------------
[INFO] [generate_stores.sh] Fin de la génération des stores
```

Ce script génère sous `environmements/keystores` les stores (jks / p12) associés pour un bon fonctionnement dans VITAM.

Il est aussi possible de déposer directement les keystores au bon format en remplaçant ceux fournis par défaut, en indiquant les mots de passe d'accès dans le vault : `environmements/group_vars/all/vault-keystores.yml`

8.3 Cas 2 : Je dispose d'une PKI

8.3.1 Procédure générale

Si vous disposez d'une PKI, il n'est pas nécessaire d'utiliser celle de Vitam. Il va par contre être nécessaire de déposer les certificats et les autorités de certifications correspondantes dans les bon répertoires. Il sera aussi nécessaire de renseigner les mots de passe des clés privées des certificats dans le vault ansible environmements/certs/vault-certs.yml Il faudra alors ensuite utiliser le script Vitam permettant de générer les différents keystores.

8.3.2 Intégration de certificats existants

Si vous possédez déjà une *PKI*, il convient de positionner les certificats et CA sous `environnements/certs/....` en respectant la structure indiquée ci-dessous.

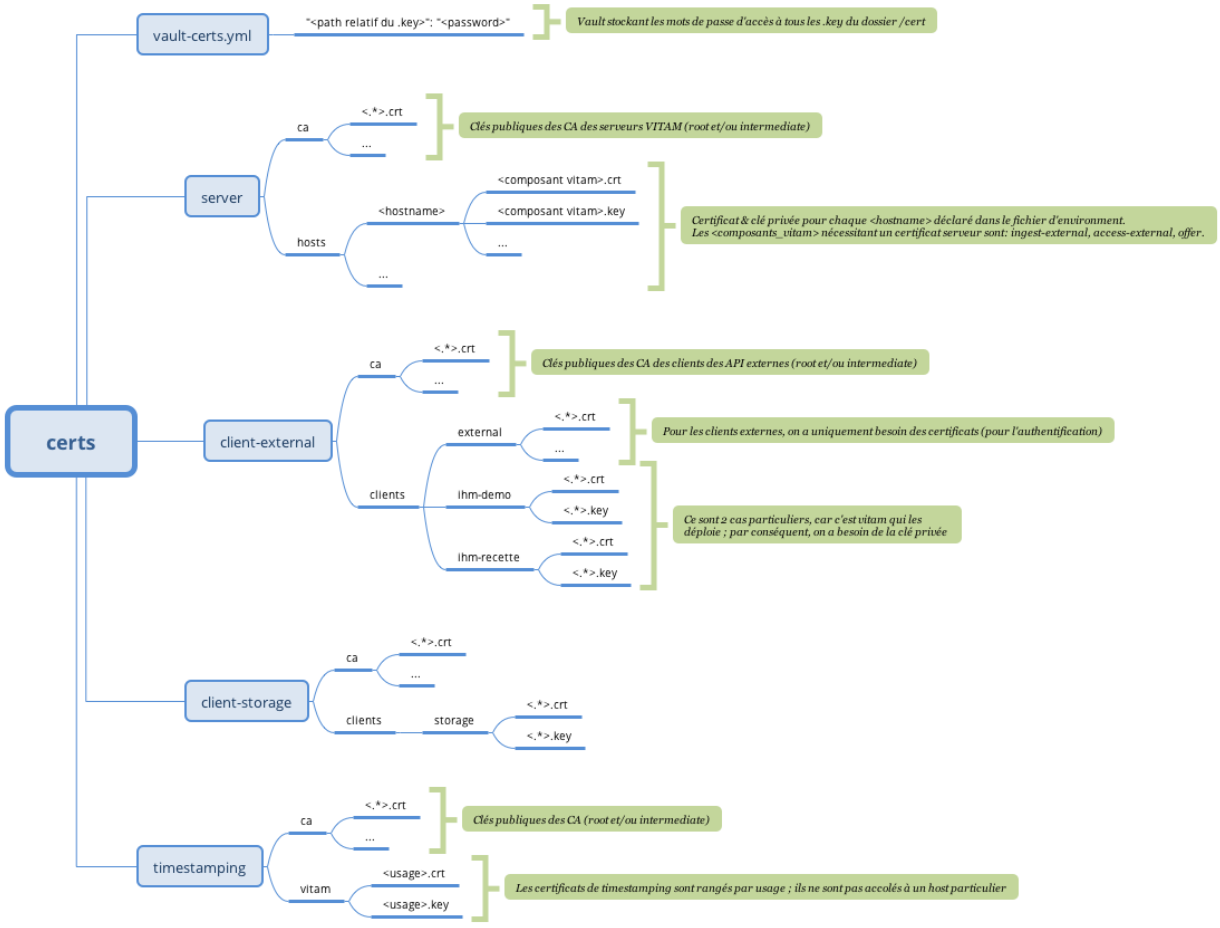

Fig. 8.5 – Vue détaillée de l'arborescence des certificats

Astuce : Dans le doute, n'hésitez pas à utiliser la PKI de test (étapes de génération de CA et de certificats) pour générer les fichiers requis au bon endroit et ainsi voir la structure exacte attendue ; il vous suffira ensuite de remplacer ces certificats "placeholders" par les certificats définitifs avant de lncer le déploiement.

Ne pas oublier de renseigner le vault contenant les passphrases des clés des certificats : `environmements/certs/vault-certs.yml`

Pour modifier/créer un vault ansible, se référer à la documentation sur cette url [11].

8.3.3 Génération des magasins de certificats

En prérequis, les certificats et les autorités de certification doivent être présents dans les répertoires attendus.

Prudence : Avant de lancer le script de génération des stores, il est nécessaire de modifier le vault contenant les mots de passe des stores : `environmements/group_vars/all/vault-keystores.yml`, décrit dans la section *PKI* (page 43).

Lancer le script :

```
./generate_stores.sh
```

Ci-dessous un exemple de sortie du script :

```
[INFO] [generate_stores.sh] ------------------------------------------
[INFO] [generate_stores.sh] Creation du keystore de access-external pour le serveur␣
↪localhost
[INFO] [generate_stores.sh] Génération du p12
[INFO] [generate_stores.sh] Génération du jks
Entry for alias access-external successfully imported.
Import command completed:  1 entries successfully imported, 0 entries failed or␣
↪cancelled
[INFO] [generate_stores.sh] Suppression du p12
[INFO] [generate_stores.sh] ------------------------------------------
[INFO] [generate_stores.sh] Creation du keystore de ingest-external pour le serveur␣
↪localhost
    [...]
[INFO] [generate_stores.sh] ------------------------------------------
[INFO] [generate_stores.sh] Fin de la génération des stores
```

Ce script génère sous `environmements/keystores` les stores (jks / p12) associés pour un bon fonctionnement dans VITAM.

Il est aussi possible de déposer directement les keystores au bon format en remplaçant ceux fournis par défaut, en indiquant les mots de passe d'accès dans le vault : `environmements/group_vars/all/vault-keystores.yml`

11. http://docs.ansible.com/ansible/playbooks_vault.html

CHAPITRE 9

Procédures d'installation / mise à jour

9.1 Vérifications préalables

Tous les serveurs cibles doivent avoir accès aux dépôts de binaires contenant les paquets des logiciels VITAM 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.

9.2 Procédures

9.2.1 Configuration de sécurité

En fonction de la méthode d'authentification sur les serveurs et d'élevation de privilège, il faut rajouter des options aux lignes de commande ansible. Ces options seront à rajouter pour toutes les commandes ansible du document .

Pour chacune des 3 sections suivantes, vous devez être dans l'un des cas décrits

9.2.1.1 Authentification du compte utilisateur utilisé pour la connexion SSH

Pour le login du compte utilisateur, voir la section *Informations "plate-forme"* (page 31).

9.2.1.1.1 Par clé SSH avec passphrase

Dans le cas d'une authentification par clé avec passphrase, il est nécessaire d'utiliser ssh-agent pour mémoriser la clé privée. Pour ce faire, il faut :

- exécuter la commande `ssh-agent <shell utilisé>` (exemple `ssh-agent /bin/bash`) pour lancer un shell avec un agent de mémorisation de la clé privée associé à ce shell
- exécuter la commande `ssh-add` et renseigner la passphrase de la clé privée

Vous pouvez maintenant lancer les commandes ansible comme décrites dans ce document.

A noter : ssh-agent est un démon qui va stocker les clés privées (déchiffrées) en mémoire et que le client ssh va interroger pour récupérer les informations privées pour initier la connexion. La liaison se fait par un socket UNIX présent dans /tmp (avec les droits 600 pour l'utilisateur qui a lancé le ssh-agent). Cet agent disparaît avec le shell qui l'a lancé.

9.2.1.1.2 Par login/mot de passe

Dans le cas d'une authentificatioon par login/mot de passe, il est nécessaire de spécifier l'option –ask-pass (ou -k en raccourci) aux commandes ansible ou ansible-playbook de ce document .

Au lancement de la commande `ansible` (ou `ansible-playbook`), il sera demandé le mot de passe

9.2.1.1.3 Par clé SSH sans passphrase

Dans ce cas, il n'y a pas de paramétrage particulier à effectuer.

9.2.1.2 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 de fichier est hors périmètre Vitam mais c'est un pré-requis pour le lancement d'ansible.

9.2.1.3 Elevation de privilèges

Une fois que l'on est connecté sur le serveur cible, il faut définir la méthode pour accéder aux droits root

9.2.1.3.1 Par sudo avec mot de passe

Dans ce cas, il faut rajouter les options `--ask-sudo-pass`

Au lancement de la commande `ansible` (ou `ansible-playbook`), il sera demandé le mot de passe demandé par `sudo`

9.2.1.3.2 Par su

Dans ce cas, il faut rajouter les options `--become-method=su --ask-su-pass`

Au lancement de la commande ansible (ou ansible-playbook), il sera demandé le mot de passe root

9.2.1.3.3 Par sudo sans mot de passe

Il n'y a pas d'option à rajouter (l'élévation par sudo est la configuration par défaut)

9.2.1.3.4 Déjà Root

Dans ce cas, il n'y a pas de paramétrages supplémentaire à effectuer.

9.2.2 Procédure de première installation

Les fichiers de déploiement sont disponibles dans la version VITAM livrée dans le sous-répertoire `deployment` . Ils consistent en 2 parties :

- le playbook ansible, présent dans le sous-répertoire `ansible-vitam`, qui est indépendant de l'environnement à déployer
- les fichiers d'inventaire (1 par environnement à déployer) ; des fichiers d'exemple sont disponibles dans le sous-répertoire `environments`

9.2.2.1 Configuration du déploiement

9.2.2.1.1 Informations "plate-forme"

Pour configurer le déploiement, il est nécessaire de créer dans le répertoire `environments` un nouveau fichier d'inventaire à nommer `hosts.<environnement>` (où <environnement> sera utilisé par la suite) comportant les informations suivantes :

```
# Group definition ; DO NOT MODIFY
[hosts]

# Group definition ; DO NOT MODIFY
[hosts:children]
vitam
reverse
library
hosts-dev-tools


########### Tests environments specifics ###########

# EXTRA : Front reverse-proxy (test environments ONLY) ; add machine name after
[reverse]
# optional : after machine, if this machine is different from VITAM machines, you can␣
↪specify another become user
# Example
# vitam-centos-01.vitam ansible_ssh_user=centos

########### Extra VITAM applications ###########

[library]
# TODO: Put here servers where this service will be deployed : library

[hosts-dev-tools]
# TODO: Put here servers where this service will be deployed : mongo-express,␣
↪elasticsearch-head

[elasticsearch:children] # EXTRA : elasticsearch
hosts-elasticsearch-data
hosts-elasticsearch-log

########### VITAM services ###########

# Group definition ; DO NOT MODIFY
[vitam:children]
zone-external
zone-access
```

```
zone-applicative
zone-storage
zone-data
zone-admin


##### Zone externe


[zone-external:children]
hosts-ihm-demo
hosts-cerebro
hosts-ihm-recette

[hosts-ihm-demo]
# TODO: Put here servers where this service will be deployed : ihm-demo

[hosts-ihm-recette]
# TODO: Put here servers where this service will be deployed : ihm-recette (extra␣
↪feature)

[hosts-cerebro]
# TODO: Put here servers where this service will be deployed : vitam-elasticsearch-
↪cerebro


##### Zone access

# Group definition ; DO NOT MODIFY
[zone-access:children]
hosts-ingest-external
hosts-access-external

[hosts-ingest-external]
# TODO: Put here servers where this service will be deployed : ingest-external


[hosts-access-external]
# TODO: Put here servers where this service will be deployed : access-external


##### Zone applicative

# Group definition ; DO NOT MODIFY
[zone-applicative:children]
hosts-ingest-internal
hosts-processing
hosts-worker
hosts-access-internal
hosts-metadata
hosts-functional-administration
hosts-logbook
hosts-workspace
hosts-storage-engine

[hosts-logbook]
# TODO: Put here servers where this service will be deployed : logbook

```

```
94
95  [hosts-workspace]
96  # TODO: Put here servers where this service will be deployed : workspace
97
98
99  [hosts-ingest-internal]
100 # TODO: Put here servers where this service will be deployed : ingest-internal
101
102
103 [hosts-access-internal]
104 # TODO: Put here servers where this service will be deployed : access-internal
105
106
107 [hosts-metadata]
108 # TODO: Put here servers where this service will be deployed : metadata
109
110
111 [hosts-functional-administration]
112 # TODO: Put here servers where this service will be deployed : functional-
    ↪administration
113
114
115 [hosts-processing]
116 # TODO: Put here servers where this service will be deployed : processing
117
118
119 [hosts-storage-engine]
120 # TODO: Put here servers where this service will be deployed : storage-engine
121
122
123 [hosts-worker]
124 # TODO: Put here servers where this service will be deployed : worker
125 # Optional parameter after each host : vitam_worker_capacity=<integer> ; please refer␣
    ↪to your infrastructure for defining this number ; default is 1
126
127
128 ##### Zone storage
129
130 [zone-storage:children] # DO NOT MODIFY
131 hosts-storage-offer-default
132
133
134 [hosts-storage-offer-default]
135 # TODO: Put here servers where this service will be deployed : storage-offer-default
136 # LIMIT : only 1 offer per machine and 1 machine per offer
137 # Additional params for openstack-swift
138 # hostname-offre-1.vitam vitam_keystone_auth_url=http://hostname-rados-gw:port/auth/1.
    ↪0 vitam_swift_subuser=subuser vitam_swift_uid=tenant$user vitam_provider_
    ↪offer=openstack-swift
139 # for filesystem
140 # hostname-offre-2.vitam vitam_provider_offer=filesystem
141
142
143 ##### Zone data
144
145 # Group definition ; DO NOT MODIFY
146 [zone-data:children]
147 hosts-elasticsearch-data
```

```
mongo_common


[hosts-elasticsearch-data]
# TODO: Put here servers where this service will be deployed : elasticsearch-data cluster


# Group definition ; DO NOT MODIFY
[mongo_common:children]
mongos
mongoc
mongod

[mongos]
# TODO: Put here servers where this service will be deployed : mongos cluster ; add after name shard_id=0
# Example : vitam-iaas-mongos-01.int shard_id=0

[mongoc]
# TODO: Put here servers where this service will be deployed : mongoc cluster


[mongod] # mongod declaration ; add machines name after ; add after shard_id=0 & rs_member_id=<increasing number, starting from 0, for each line>
# TODO: Put here servers where this service will be deployed : mongod cluster ; add after name shard_id=0
# Example : vitam-iaas-db-01.int rs_member_id=0 shard_id=0
# Example : vitam-iaas-db-02.int rs_member_id=1 shard_id=0
# Example : vitam-iaas-db-03.int rs_member_id=2 shard_id=0

###### Zone admin

# Group definition ; DO NOT MODIFY
[zone-admin:children]
hosts-consul-server
log-servers
hosts-elasticsearch-log
hosts-mongoclient

[hosts-consul-server]
# TODO: Put here servers where this service will be deployed : consul


[log-servers:children]
hosts-kibana
hosts-logstash

[hosts-logstash]
# TODO: Put here servers where this service will be deployed : logstash

[hosts-kibana]
# TODO: Put here servers where this service will be deployed : kibana

[hosts-elasticsearch-log]
# TODO: Put here servers where this service will be deployed : elasticsearch-log cluster

```

```
[hosts-mongoclient]
# TODO: Put here servers where this service will be deployed : mongos cluster ; add after name shard_id=0
# Example : vitam-iaas-mongos-01.int shard_id=0

########### Global vars ###########

[hosts:vars]
# Declare user for ansible on target machines
ansible_ssh_user=

# Can target user become as root ? ; true is required by VITAM (usage of a sudoer is mandatory)
ansible_become=true

# Environment (defines consul environment name ; in extra on homepage)
environnement=

# EXTRA : FQDN of the front reverse-proxy ; used when VITAM is behind a reverse proxy (provides configuration for reverse proxy && displayed in header page)
vitam_reverse_external_dns=

# Version that has to be deployed (defined in the release note)
# Example: package_version=0.9.0-RC1*
package_version=

# Configuration for Curator
#        Days before deletion on log management cluster; 365 for production environment
days_to_delete=

#        Days before closing "old" indexes on log management cluster; 30 for production environment
days_to_close=

#        Days before deletion for topbeat index only on log management cluster; 365 for production environment
days_to_delete_topbeat=

# Related to Consul ; apply in a table your DNS server(s)
# Example : dns_servers=["8.8.8.8","8.8.4.4"]
dns_servers=

#        LOG level defined in logback files ; can be a value in "ERROR","WARN","INFO","DEBUG","TRACE". Recommended value is "WARN"
log_level=

# For reverse proxy use
reverse_proxy_port=80

# For metrics
# curator job : days before closing
days_to_close_metrics=7
# curator job : days before deleting
days_to_delete_metrics=30
# Installation ClamAV ? true/false
installation_clamav=true

```

```
# cas de l'appel au webDAV pour récupérer les jeux de tests
http_proxy_environnement=

vitam_tenant_ids=[0,1,2]

# ces paramètres peuvent être soit globaux, ici, soit ajoutés après chaque partition hébergeant une offre de stockage

# useless now as it is declared in [hosts-storage-offer-default]
# vitam_provider_offer=openstack-swift
# URL d'authentification si openstack-swift
# vitam_keystone_auth_url=http://xxxxx.xxxx.xxx:8080/auth/1.0
# subUser pour swift
# vitam_swift_subuser=
# Nom du tenant associé (concaténation tenant$user ; le mot de passe est renseigné dans vault.yml sous la directive vitam_keystone_passwd )
# vitam_swift_uid=

# Pour les Tests de Non-Regression (en git LFS), si URL et branche définies, récupération des jeux de tests
vitam_swift_uid=

```

Pour chaque type de "host", indiquer le(s) serveur(s) défini(s) pour chaque fonction. Une colocalisation de composants est possible.

Prudence : en cas de colocalisation, bien prendre en compte la taille JVM de chaque composant (VITAM : -Xmx512m par défaut) pour éviter de swapper.

Note : pour les "hosts-worker", il est possible d'ajouter, à la suite de chaque "host", 2 paramètres optionnels : capacity et workerFamily. Se référer au *DEX* pour plus de précisions.

Ensuite, dans la section `hosts:vars`, renseigner les valeurs comme décrit :

Tableau 9.1 – Définition des variables

Clé	Description	Valeur d'exemple
ansible_ssh_user	Utilisateurs ansible sur les machines sur lesquelles VITAM sera déployé	
ansible_become	Propriété interne à ansible pour passer root	
local_user	En cas de déploiement en local	
environnement	Suffixe	
vitam_reverse_domain	Cas de la gestion d'un reverse proxy	
consul_domain	nom de domaine consul	
vitam_ihm_demo_external_dns	Déprécié ; ne pas utiliser	
package_version	Version à installer	
days_to_delete	Période de grâce des données sous Elastricsearch avant destruction (valeur en jours)	
days_to_close	Période de grâce des données sous Elastricsearch avant fermeture des index (valeur en jours)	
days_to_delete_topbeat	Période de grâce des données sous Elastricsearch - index Topbeat - avant destruction (valeur en jours)	
days_to_delete_local	Période de grâce des log VITAM - logback (valeur en jours)	
dns_server	Serveur DNS que Consul peut appeler s'il n'arrive pas à faire de résolution	172.16.1.21
log_level	Niveau de log de logback	WARN
web_dir_soapui_tests	URL pour récupérer data.json et les tests pour SoapUI	http://vitam-prod-ldap-1.internet.agri:8083/webdav
reverse_proxy_port	port du reverse proxy pour configuration du vhost	8080
days_to_close_metrics	Période de grâce avant fermeture des index des métriques JVM	7
days_to_delete_metrics	Période de grâce avant destruction des index fermés des métriques JVM	30
installation_clamav	Choix d'installation de ClamAV (true/false)	true
http_proxy_environnement	Cas particulier de la récupération des jeux de tests ; URL de squid	
mongoclientPort	Port par lequel mongoclient est acessible	27016
mongoclientDbName	Nom de la Base de donnée stockant la configuration mongoclient	mongoclient
vitam_tenant_ids	Liste des tenants de plateforme	[0,1,2] ; [0] par défaut
vitam_tests_gitrepo_protocol	Protocole d'attaque du git lfs des TNR	
vitam_tests_gitrepo_baseurl	domaine du git lfs des TNR	
vitam_tests_gitrepo_url	Création de l'URL à partir des lignes précédentes	
vitam_tests_branch	Branche à récupérer sur le git lfs	master

A titre informatif, le positionnement des variables ainsi que des dérivations des déclarations de variables sont effectuées sous `environments /group_vars/all/all`, comme suit :

```
---

vitam_folder_root: /vitam
docker_registry_httponly: yes
vitam_docker_tag: latest
port_http_timeout: 86400
syslog_facility: local0

# Composants colocalisés

vitam_accessinternal_host: "access-internal.service.{{consul_domain}}"
vitam_accessinternal_port: 8101
vitam_accessinternal_port_admin: 28101
vitam_accessinternal_baseurl: "http://{{vitam_accessinternal_host}}:{{vitam_
↪accessinternal_port}}"
vitam_accessinternal_baseuri: "/access-internal"

vitam_accessexternal_host: "access-external.service.{{consul_domain}}"
vitam_accessexternal_port: 8102
vitam_accessexternal_port_admin: 28102
vitam_accessexternal_port_https: 8444
vitam_accessexternal_baseurl: "http://{{vitam_accessexternal_host}}:{{vitam_
↪accessexternal_port}}"
vitam_accessexternal_baseuri: "/access-external"

vitam_ingestinternal_host: "ingest-internal.service.{{consul_domain}}"
vitam_ingestinternal_port: 8100
vitam_ingestinternal_port_admin: 28100
vitam_ingestinternal_baseurl: "http://{{vitam_ingestinternal_host}}:{{vitam_
↪ingestinternal_port}}"
vitam_ingestinternal_baseuri: "/ingest-internal"

vitam_ingestexternal_host: "ingest-external.service.{{consul_domain}}"
vitam_ingestexternal_port: 8001
vitam_ingestexternal_port_admin: 28001
vitam_ingestexternal_port_https: 8443
vitam_ingestexternal_baseurl: "http://{{vitam_ingestexternal_host}}:{{vitam_
↪ingestexternal_port}}"
vitam_ingestexternal_baseuri: "/ingest-external"

vitam_metadata_host: "metadata.service.{{consul_domain}}"
vitam_metadata_port: 8200
vitam_metadata_port_admin: 28200
vitam_metadata_baseurl: "http://{{vitam_metadata_host}}:{{vitam_metadata_port}}"
vitam_metadata_baseuri: "/metadata"

vitam_ihm_demo_host: "{{groups['hosts-ihm-demo'][0]}}"
vitam_ihm_demo_port: 8002
vitam_ihm_demo_port_admin: 28002
vitam_ihm_demo_baseurl: /ihm-demo
vitam_ihm_demo_static_content: "{{vitam_folder_root}}/app/ihm-demo"
vitam_ihm_demo_baseuri: "/ihm-demo"

vitam_ihm_recette_host: "{{groups['hosts-ihm-recette'][0]}}"
vitam_ihm_recette_port: 8204
vitam_ihm_recette_port_admin: 28204
vitam_ihm_recette_baseurl: /ihm-recette
vitam_ihm_recette_static_content: "{{vitam_folder_root}}/app/ihm-recette"
```

```
vitam_ihm_recette_baseuri: "/ihm-recette"

# Internal components communication configuration
vitam_logbook_host: "logbook.service.{{consul_domain}}"
vitam_logbook_port: 9002
vitam_logbook_port_admin: 29002
vitam_logbook_baseurl: "http://{{vitam_logbook_host}}:{{vitam_logbook_port}}"
vitam_logbook_baseuri: "/logbook"

vitam_workspace_host: "workspace.service.{{consul_domain}}"
vitam_workspace_port: 8201
vitam_workspace_port_admin: 28201
vitam_workspace_baseurl: "http://{{vitam_workspace_host}}:{{vitam_workspace_port}}"
vitam_workspace_baseuri: "/workspace"

vitam_processing_host: "processing.service.{{consul_domain}}"
vitam_processing_port: 8203
vitam_processing_port_admin: 28203
vitam_processing_baseurl: "http://{{vitam_processing_host}}:{{vitam_processing_port}}"
vitam_processing_baseuri: "/processing"

vitam_worker_port: 9104
vitam_worker_port_admin: 29104
vitam_worker_baseuri: "/worker"

vitam_storageengine_host: "storage.service.{{consul_domain}}"
vitam_storageengine_port: 9102
vitam_storageengine_port_admin: 29102
vitam_storageengine_baseurl: "http://{{vitam_storageengine_host}}:{{vitam_
↪storageengine_port}}"
vitam_storageengine_baseuri: "/storage-engine"
test_tls_offer_enabled: true

vitam_storageofferdefault_port: 9900
vitam_storageofferdefault_port_admin: 29900
vitam_storageofferdefault_port_https: 9901
vitam_storageofferdefault_baseuri: "/storage-offer-default"

vitam_functional_administration_host: "functional-administration.service.{{consul_
↪domain}}"
vitam_functional_administration_port: 8004
vitam_functional_administration_port_admin: 18004
vitam_functional_administration_baseurl: "http://{{vitam_functional_administration_
↪host}}:{{vitam_functional_administration_port}}"
vitam_functional_administration_baseuri: "/functional-administration"

# Normally no need for the host ? Maybe use the same strategy as data ?
elasticsearch_log_host: "elasticsearch-log.service.{{consul_domain}}"
elasticsearch_log_http_port: "9201"
elasticsearch_log_tcp_port: "9301"

elasticsearch_data_http_port: "9200"
elasticsearch_data_tcp_port: "9300"

mongo_base_path: "{{vitam_folder_root}}"
mongos_port: 27017
mongoc_port: 27018
mongod_port: 27019
```

```
mongo_authentication: "true"
mongoclientDbName: "mongoclient"
mongoclientPort: 27016
mongoclientbaseUrl: "/mongoclient"

vitam_mongodb_host: "mongos.service.{{consul_domain}}"
vitam_mongodb_port: "{{mongos_port}}"

vitam_logstash_host: "{{ groups['hosts-logstash'][0] }}"
vitam_logstash_port: 10514

# Normally no need for the host ?
vitam_kibana_host: "kibana.service.{{consul_domain}}"
vitam_kibana_port: 5601

vitam_curator_host: "{{ (groups['hosts-log-server'] | length > 0) | ternary(groups[
 ↪'hosts-log-server'][0], '') }}"

vitam_library_port: 8090
vitam_library_port_admin: 28090

vitam_siegfried_port: 19000

vitam_user: vitam
vitamdb_user: "vitamdb"
vitam_group: vitam

consul_domain: consul

vitam_folder_permission: 0750

vitam_conf_permission: 0640

consul_component: consul
consul_folder_conf: "{{vitam_folder_root}}/conf/{{consul_component}}"

mongod_folder_database: "{{vitam_folder_root}}/data/mongod/db"
mongoc_folder_database: "{{vitam_folder_root}}/data/mongoc/db"

service_restart_timeout: 150
service_stop_timeout: 3600
clamav_port: 3310

cerebro_port: 9000
```

Le fichier `vault-vitam.yml` est également présent sous `environments/group_vars/all/all` et contient les secrets ; ce fichier est encrypté par `ansible-vault` et doit être paramétré avant le lancement de l'orchestration de déploiement.

```
plateforme_secret: vitamsecret
cerebro_secret_key: changemeornot
mongoAdminUser: vitamdb-admin
mongoAdminPassword: qwertz
mongoMetadataUser: metadata
mongoMetadataPassword: qwertz
mongoLogbookUser: logbook
mongoLogbookPassword: qwertz
mongoFunctionalAdminUser: functional-admin
```

```
mongoFunctionalAdminPassword: qwertz
mongoClientUser: mongoclient
mongoClientPassword: qwertz
mongoPassPhrase: mongogogo
vitam_keystone_passwd: 2kwRWUKXyjR64VtUCma1vd5TS8DFZjQnpeJ0sLbN
consul_encrypt: Biz14ohqN4HtvZmsXp3N4A==
vitam_users:
  - vitam_uuser:
    login: uuser
    password: monuuser
    role: user
  - vitam_aadmin:
    login: aadmin
    password: monaadmin
    role: admin
  - vitam_gguest:
    login: gguest
    password: mongguest
    role: guest
  - techadmin:
    login: techadmin
    password: montechadmin
    role: admin
```

Note : Si le mot de passe du fichier `vault-vitam.yml` est changé, ne pas oublier de le répercuter dans le fichier `vault_pass.txt` (et le sécuriser à l'issue de l'installation).

Le fichier `vault-extra.yml` peut être également présent sous `environments /group_vars/all/all` et contient des secrets supplémentaires ; ce fichier est encrypté par `ansible-vault` et doit être paramétré avant le lancement de l'orchestration de déploiement, si le composant ihm-recette est déployé avec récupération des TNR.

```
# Example for git lfs ; uncomment & use if needed
#vitam_gitlab_itest_login: "account"
#vitam_gitlab_itest_password: "password"
```

Note : Pour ce fichier, l'encrypter avec le même mot de passe que `vault-vitam.yml`.

Le déploiement s'effectue depuis la machine "ansible" et va distribuer la solution VITAM selon l'inventaire correctement renseigné.

Avertissement : le playbook `vitam.yml` comprend des étapes avec la mention `no_log` afin de ne pas afficher en clair des étapes comme les mots de passe des certificats. En cas d'erreur, il est possible de retirer la ligne dans le fichier pour une analyse plus fine d'un éventuel problème sur une de ces étapes.

9.2.2.2 Paramétrage de mongoclient (administration mongoclient)

Le package vitam-mongoclient nécessite une bases de données mongoDB (mongoclient) pour stocker sa configuration. Cette base de données est créée dans *VITAM* durant la première installation. La configuration est également générée en fonction des paramètres de l'inventaire.

Mongoclient permet de se connecter aux différentes bases de données mongoDB utilisées par VITAM.

9.2.2.3 Première utilisation de mongoclient

Par défault, mongoclient est accessible par l'url : http://hostname:27016/mongoclient suivant les hôtes configurés dans le groupes hosts-mongoclients de l'inventaire Vitam.

> **Avertissement :** les versions de mongoclient inférieures à la version 1.5.0 présentent un message d'erreur "route not found" à l'apparition de l'interface. les fonctionnalités de l'application sont indisponibles dans cet état. Ce problème est aisemment contournable en cliquant sur le bouton "Go to Dashboard" pour revenir à un état normal de l'application.

Lors de la première utilisation de mongoclient, il convient de configurer les connexions aux bases de données à superviser. (Cette procédure devrait disparaître à l'issue de la phase Beta)

Procédure pour configurer la connexion aux bases vitam : :

1. Cliquer sur le bouton "Connect" situé en haut de la page (l'emplacement dépend de la taille de la fenêtre)
2. Dans la fenêtre "Connections", cliquer sur le bouton "Create New". => la fenêtre Add connection apparait contenant 4 sections : Connection, Authentication, URL, SSH
3. Dans la section "Connection", saisir un nom à donner à la connexion dans "name", le nom ou l'ip du server mongos à cibler dans "hostname", changer éventuellement le "port", définir la base de donnée sur laquelle le client doit se connecter
4. Dans la section "Authentication", saisir les paramètres d'autentification du compte à utiliser pour se connecter à la base configurée en section "connection"
5. Dans la section URL, en fonction du la configuration des services, choisir cette méthode de connexion en lieu et place des autres méthodes.
6. Dans la section "SSH", si le service mongoDB n'est accessible qu'au travers d'une connexion SSH, renseigner les paramètres de cette connexion pour accéder au serveur.
7. Sauvegarder les paramètres avec le boutton "save changes"
8. La nouvelle connexion doit apparaître avec un résumé de ses paramètres dans la fenêtre "Connections"
9. CLiquer sur la ligne de la connexion puis cliquer sur le boutton "Connect Now" pour utiliser se connecter.

Si les identifiants utilisés disposent de droit suffisants, Mongoclient vas afficher les métriques du service mongoDB.

Mongoclient ne permet de gérer qu'une seule base à la fois, il est toutefois possible de changer de base de donnée rapidement en ouvrant le menu "More" => "Switch Database" qui affichera la liste des bases de données accessibles (suivant les identifiants renseignés).

9.2.2.3.1 Paramétrage de l'antivirus (ingest-externe)

L'antivirus utilisé par ingest-externe est modifiable (par défaut, ClamAV) ; pour cela :

- Créer un autre shell (dont l'extension doit être `.sh.j2`) sous `ansible-vitam/roles/vitam/templates/ingest-ext` prendre comme modèle le fichier `scan-clamav.sh.j2`. Ce fichier est un template Jinja2, et peut donc contenir des variables qui seront interprétées lors de l'installation.
- Modifier le fichier `ansible-vitam/roles/vitam/templates/ingest-external/ingest-external.conf.j` en pointant sur le nouveau fichier.

Ce script shell doit respecter le contrat suivant :

- Argument : chemin absolu du fichier à analyser
- Sémantique des codes de retour
  - 0 : Analyse OK - pas de virus
  - 1 : Analyse OK - virus trouvé et corrigé

    - 2 : Analyse OK - virus trouvé mais non corrigé
    - 3 : Analyse NOK
  - Contenu à écrire dans stdout / stderr
    - stdout : Nom des virus trouvés, un par ligne ; Si échec (code 3) : raison de l'échec
    - stderr : Log « brut » de l'antivirus

9.2.2.3.2 Paramétrage des certificats (*-externe)

Se reporter au chapitre dédié à la gestion des certificats : *Introduction sur les certificats dans Vitam* (page 15)

9.2.2.4 Déploiement

9.2.2.4.1 Fichier de mot de passe

Si le fichier `deployment/vault_pass.txt` est renseigné avec le mot de passe du fichier `environments/group_vars/all/vault-vitam.yml`, le mot de passe ne sera pas demandé. Si le fichier est absent, le mot de passe du "vault" sera demandé.

9.2.2.4.2 PKI

Se positionner dans le répertoire `deployment`.

1. paramétrer les fichiers `environments/group_vars/all/vault-vitam.yml` et `environments/group_vars/all/vault-keystores.yml` (définition des mots de passe des différents stores java - à adapter aux exigences de sécurité de l'exploitant), ainsi que le fichier d'inventaire de la plate-forme sous `environments` (se baser sur le fichier hosts.example)

Exemple de fichier `vault-keystores.yml` :

```
keystores:
  server:
    storage_offer_default: azerty1
    access_external: azerty2
    ingest_external: azerty3
  client_external:
    ihm_demo: azerty4
    ihm_recette: azerty5
    reverse: azerty6
  client_storage:
    storage: azerty7
  timestamping:
    secure_logbook: azerty8
truststores:
  server: azerty
  client_external: azerty9
  client_storage: azerty10
grantedstores:
  client_external: azerty11
  client_storage: azerty12
```

2. En absence d'une PKI, exécuter le script

```
./pki/scripts/generate_ca.sh
```

Note : En cas d'absence de *PKI*, il permet de générer une *PKI*, ainsi que des certificats pour les échanges https entre composants. Autrement, passer à l'étape suivante.

3. Génération des certificats, si aucun n'est fourni par le client

```
pki/scripts/generate_certs.sh <environnement>
```

Note : Ce script génère des certificats nécessaires au bon fonctionnement de VITAM ainsi qu'un fichier `(deployment)/environments/certs/vault-certs.yml` contenant les mots de passe correspondants.

4. Génération des stores Java, s'ile ne sont pas fournis par le client

```
./generate_stores.sh <environnement>
```

Note : Basé sur le contenu du fichier `vault.yml`, ce script génère des stores nécessaires au bon fonctionnement de VITAM et les positionne au bon endroit pour le déploiement.

9.2.2.4.3 Mise en place des repositories VITAM (optionnel)

VITAM fournit un playbook permettant de définir sur les partitions cible la configuration d'appel aux repositories spécifiques à VITAM :

Editer le fichier `environments/group_vars/all/repositories.yml` à partir des modèles suivants (décommenter également les lignes) :

Pour une cible de déploiement CentOS :

```
#vitam_repositories:
#- key: repo 1
#  value: "file:///code"
#  proxy: http://proxy
#- key: repo 2
#  value: "http://www.programmevitam.fr"
#  proxy: _none_
#- key: repo 3
#  value: "ftp://centos.org"
#  proxy:
```

Pour une cible de déploiement Debian :

```
#vitam_repositories:
#- key: repo 1
#  value: "file:///code"
#  subtree: "./"
#  trusted: "[trusted=yes]"
#- key: repo 2
#  value: "http://www.programmevitam.fr"
```

```
#  subtree: "./"
#  trusted: "[trusted=yes]"
#- key: repo 3
#  value: "ftp://centos.org"
#  subtree: "binary"
#  trusted: "[trusted=yes]"
```

Ce fichier permet de définir une liste de repositories. Décommenter et adapter à votre cas.

Pour mettre en place ces repositories sur les machines cibles, lancer la commande :

Note : En environnement CentOS, il est recommandé de créer des noms de repository commençant par "vitam-".

9.2.2.4.4 Réseaux

Une fois l'étape de PKI effectuée avec succès, il convient de procéder à la génération des hostvars, qui permettent de définir quelles interfaces réseau utiliser. Actuellement la solution logicielle Vitam est capable de gérer 2 interfaces réseau :

- Une d'administration
- Une de service

9.2.2.4.4.1 Cas 1 : Machines avec une seule interface réseau

Si les machines sur lesquelles Vitam sera déployé ne disposent que d'une interface réseau, ou si vous ne souhaitez en utiliser qu'une seule, il convient d'utiliser le playbook `ansible-vitam/generate_hostvars_for_1_network_interface.yml`

Cette définition des host_vars se base sur la directive ansible `ansible_default_ipv4.address`, qui se base sur l'adresse IP associée à la route réseau définie par défaut.

Avertissement : Les communication d'administration et de service transiteront donc toutes les deux via l'unique interface réseau disponible.

9.2.2.4.4.2 Cas 2 : Machines avec plusieurs interfaces réseau

Si les machines sur lesquelles Vitam sera déployé disposent de plusieurs interfaces, si celles-ci respectent cette règle :

- Interface nommée eth0 = ip_service
- Interface nommée eth1 = ip_admin

Alors il est possible d'utiliser le playbook `ansible-vitam-extra/generate_hostvars_for_2_network_interfaces.`

Note : Pour les autres cas de figure, il sera nécessaire de générer ces hostvars à la main ou de créer un script pour automatiser cela.

9.2.2.4.4.3 Vérification de la génération des hostvars

A l'issue, vérifier le contenu des fichiers générés sous `environments/host_vars/` et les adapter au besoin.

9.2.2.4.5 Déploiement

Une fois l'étape de la génération des hosts a été effectuée avec succès, le déploiement est à réaliser avec la commande suivante :

```
ansible-playbook ansible-vitam/vitam.yml -i environments/<ficher d'inventaire> --ask-
↪vault-pass
```

9.2.2.4.6 Extra

Deux playbook d'extra sont fournis pour usage "tel quel".

1. ihm-recette

Ce playbook permet d'installer également le composant *VITAM* ihm-recette.

```
ansible-playbook ansible-vitam-extra/ihm-recette.yml -i environments/<ficher d
↪'inventaire> --ask-vault-pass
```

2. extra complet

Ce playbook permet d'installer :

- topbeat
- packetbeat
- un serveur Apache pour naviguer sur le `/vitam` des différentes machines hébergeant *VITAM*
- mongo-express (en docker ; une connexion internet est alors nécessaire)
- le composant *VITAM* library, hébergeant les documentations du projet
- le composant *VITAM* ihm-recette (nécessite un accès à un répertoire "partagé" pour récupérer les jeux de tests)
- un reverse proxy, afin de simplifier les appels aux composants

```
ansible-playbook ansible-vitam-extra/extra.yml -i environments/<ficher d'inventaire> -
↪-ask-vault-pass
```

9.2.2.5 Import automatique d'objets dans Kibana

Il peut être utile de vouloir automatiquement importer dans l'outil de visualisation Kibana des dashboards préalablement crées. Cela ce fait simplement avec le système d'import automatique mis en place. Il suffit de suivre les différentes étapes :

1. Ouvrir l'outil Kibana dans son navigateur.
2. Créer ses dashboards puis sauvegarder.
3. Aller dans l'onglets **Settings** puis **Objects**.
4. Sélectionner les composants à exporter puis cliquer sur le bouton **Export**. (ou bien cliquer sur **Export Everything** pour tout exporter).
5. Copier le/les fichier(s) *.json* téléchargés à l'emplacement `deployment\ansible-vitam\roles\log-server\files\k`
6. Les composants sont prêts à être importés automatique lors du prochain déploiement.

Pour éviter d'avoir à recréer les "index-pattern" définis dans l'onglet **Settings** de Kibana, ceux-ci aussi sont pris en charge par le système de déploiement automatique. En revanche ils ne sont pas exportables, il est donc nécessaire de créer à la main le fichier *.json* correspondant. Pour ce faire :

1. Faire une requête GET sur l'url suivante `http://<ip-elasticsearch-log>/.kibana/index-pattern/_search`.
2. Récupérer le contenu au format JSON et extraire le contenu de la clé **hits.hits** (qui doit être un tableau).
3. Copier ce tableau dans un fichier.
4. Copier le fichier créé à l'étape 3 dans l'emplacement `deployment\ansible-vitam\roles\log-server\files\kiba`
5. Les index-pattern sont prêts à être importés.

NB : Il ne faut pas oublier de selectionner l'index pattern par defaut avant toutes recherches (se referer à la documentation officielle de Kibana pour plus d'informations)

9.2.3 Procédure de mise à niveau

Cette section décrit globalement le processus de mise à niveau d'une solution VITAM déjà en place et ne peut se substituer aux recommandations effectuées dans la "release note" associée à la fourniture des composants mis à niveau.

La mise à jour peut actuellement être effectuée comme une "première installation".

CHAPITRE 10

Validation de la procédure

La procédure de validation est commune aux différentes méthodes d'installation.

10.1 Sécurisation du fichier `vault_pass.txt`

Le fichier `vault_pass.txt` est très sensible ; il contient le mot de passe du fichier `environments/group_vars/all/vault.yml` qui contient les divers mots de passe de la plate-forme. Il est fortement déconseillé de ne pas l'utiliser en production. A l'issue de l'installation, il est nécessaire de le sécuriser (suppression du fichier ou application d'un `chmod 400`).

10.2 Validation manuelle

Chaque service VITAM (en dehors de bases de données) expose des URL de statut présente à l'adresse suivante : `<protocole web http ou https>://<host>:<port>/admin/v1/status` Cette URL doit retourner une réponse HTTP 204 sur une requête HTTP GET, si OK.

Un playbook d'appel de l'intégralité des autotests est également inclus (`deployment/ansible-vitam-exploitation/status_vitam.yml`). Il est à lancer de la même manière que pour l'installation de vitam (en changeant juste le nom du playbook à exécuter).

Avertissement : les composants VITAM "ihm" n'intègrent pas /admin/v1/status".

Il est également possible de vérifier la version installée de chaque composant par l'URL :

`<protocole web http ou https>://<host>:<port>/admin/v1/version`

10.3 Validation via Consul

Consul possède une *IHM* pour afficher l'état des services VITAM et supervise le "/admin/v1/status" de chaque composant VITAM, ainsi que des check TCP sur les bases de données.

Pour se connecter à Consul : `http//<Nom du 1er host dans le groupe ansible hosts-consul-server>:8500/ui`

Pour chaque service, la couleur à gauche du composant doit être verte (correspondant à un statut OK).

Si une autre couleur apparaît, cliquer sur le service "KO" et vérifier le test qui ne fonctionne pas.

Avertissement : les composants *VITAM* "ihm" (ihm-demo, ihm-recette) n'intègrent pas /admin/v1/status" et donc sont indiqués "KO" sous Consul ; il ne faut pas en tenir compte, sachant que si l'IHM s'affiche en appel "classique", le composant fonctionne.

10.4 Post-installation : administration fonctionnelle

A l'issue de l'installation, puis la validation, un **administrateur fonctionnel** doit s'assurer que :

- le référentiel PRONOM (lien vers pronom [12]) est correctement importé depuis "Import du référentiel des formats" et correspond à celui employé dans Siegfried
- le fichier "rules" a été correctement importé via le menu "Import du référentiel des règles de gestion"
- à terme, le registre des fonds a été correctement importé

Les chargements sont effectués depuis l'*IHM* demo.

10.4.1 Cas du référentiel PRONOM

Un playbook a été créé pour charger le référentiel PRONOM dans une version compatible avec celui intégré dans le composant Siegfried.

Ce playbook n'est à passer que si aucun référentiel PRONOM n'a été chargé, permettant d'accélérer l'utilisation de VITAM.

```
ansible-playbook ansible-vitam-extra/init_pronom.yml -i environments/<fichier
d'inventaire> --ask-vault-pass
```

Prudence : le playbook ne se termine pas correctement (code HTTP 403) si un référentiel PRONOM a déjà été chargé.

12. http://www.nationalarchives.gov.uk/aboutapps/pronom/droid-signature-files.htm

CHAPITRE 11

Troubleshooting

Cette section a pour but de recenser les problèmes déjà rencontrés et apporter une solution associée.

1. Le service ihm-demo est toujours dans l'état "critical" dans Consul ; cela correspond à une limitation connue. Cependant, cet état ne nuit en rien au bon fonctionnement du système.

CHAPITRE 12

Retour d'expérience / cas rencontrés

Mongo-express ne se connecte pas à la base de données associée Si mongoDB a été redémarré, il faut également redémarrer mongo-express.

Elasticsearch possède des shard non alloués (état "UNASSIGNED") Lors de la perte d'un noeud d'un cluster elasticseach, puis du retour de ce noeud, certains shards d'elasticseach peuvent rester dans l'état `UNASSIGNED` ; dans ce cas, cerebro affiche les shards correspondant en gris (au-dessus des noeuds) dans la vue "cluster", et l'état du cluster passe en "yellow". Il est possible d'avoir plus d'informations sur la cause du problème via une requête `POST` sur l'API elasticsearch `_cluster/reroute?explain`. Si la cause de l'échec de l'assignation automatique a été résolue, il est possible de relancer les assignations automatiques en échec via une requête `POST` sur l'API `_cluster/reroute?retry_failed`. Dans le cas où l'assignation automatique ne fonctionne pas, il est nécessaire de faire l'assignation à la main pour chaque shard incriminé (requête `POST` sur `_cluster/reroute`):

```
{
  "commands": [
    {
      "allocate": {
        "index": "topbeat-2016.11.22",
        "shard": 3,
        "node": "vitam-iaas-dblog-01.int"
      }
    }
  ]
}
```

Cependant, un shard primaire ne peut être réalloué de cette manière (il y a risque de perte de données). Si le défaut d'allocation provient effectivement de la perte puis de la récupération d'un noeud, et que TOUS les noeuds du cluster sont de nouveaux opérationnels et dans le cluster, alors il est possible de forcer la réallocation sans perte.

```
{
  "commands": [
    {
      "allocate": {
        "index": "topbeat-2016.11.22",
        "shard": 3,
        "node": "vitam-iaas-dblog-01.int",
        "allow_primary": "true"
      }
    }
  ]
}
```

Sur tous ces sujets, Cf. la documentation officielle [13].

Elasticsearch possède des shards non initialisés (état "INITIALIZING") Tout d'abord, il peut être difficile d'identifier les shards en questions dans cerebro ; une requête HTTP `GET` sur l'API `_cat/shards` permet d'avoir une liste plus compréhensible. Un shard non initialisé correspond à un shard en cours de démarrage (Cf. une ancienne page de documentation [14]. Si les shards non initialisés sont présents sur un seul noeud, il peut être utile de redémarrer le noeud en cause. Sinon, une investigation plus poussée doit être menée.

13. https://www.elastic.co/guide/en/elasticsearch/reference/current/cluster-reroute.html
14. https://www.elastic.co/guide/en/elasticsearch/reference/1.4/states.html

CHAPITRE 13

Elements extras de l'installation

Plusieurs playbook d'extra sont fournis pour usage "tel quel".

1. ihm-recette

Ce playbook permet d'installer également le composant *VITAM* ihm-recette.

```
ansible-playbook ansible-vitam-extra/ihm-recette.yml -i environments/<ficher d
↪'inventaire> --ask-vault-pass
```

2. extra complet

Ce playbook permet d'installer :

- topbeat
- un serveur Apache pour naviguer sur le `/vitam` des différentes machines hébergeant *VITAM*
- mongo-express (en docker ; une connexion internet est alors nécessaire)
- le composant *VITAM* library, hébergeant les documentations du projet
- le composant *VITAM* ihm-recette (nécessite un accès à un répertoire "partagé" pour récupérer les jeux de tests)
- un reverse proxy, afin de simplifier les appels aux composants

```
ansible-playbook ansible-vitam-extra/extra.yml -i environments/<ficher d'inventaire> -
↪-ask-vault-pass
```

CHAPITRE 14

Annexes

Table des figures

Liste des tableaux