VITAM - Documentation d'installation

Version 1.10.0-1

VITAM

nov. 19, 2018

Table des matières

CHAPITRE 1

Introduction

1.1 Objectif de ce document

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.

Il s'adresse aux personnes suivantes :

- Les architectes techniques des projets désirant intégrer la solution logicielle VITAM ;
- Les exploitants devant installer la solution logicielle VITAM.

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 1 – Documents de référence VITAM

Nom	Lien
DAT	http://www.programmevitam.fr/ressources/DocCourante/html/archi
DIN	http://www.programmevitam.fr/ressources/DocCourante/html/installation
DEX	http://www.programmevitam.fr/ressources/DocCourante/html/exploitation
Release notes	

2.2.2 Référentiels externes

2.3 Glossaire

API Application Programming Interface

BDD Base De Données

CA Certificate Authority

[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

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

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 NoSQL [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]

PCA Plan de Continuité d'Activité

PRA Plan de Reprise d'Activité

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 REST [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

SIP Submission Information Package

TNR Tests de Non-Régression

VITAM Valeurs Immatérielles Transférées aux Archives pour Mémoire

---

https://fr.wikipedia.org/wiki/Domain_Name_System_Security_Extensions
https://fr.wikipedia.org/wiki/Attaque_de_l'homme_du_milieu
https://fr.wikipedia.org/wiki/NoSQL
https://fr.wikipedia.org/wiki/Infrastructure_%C3%A0_cl%C3%A9s_publiques
https://fr.wikipedia.org/wiki/Representational_state_transfer

CHAPITRE 3

Prérequis à l'installation

3.1 Expertises requises

Les équipes en charge du déploiement et de l'exploitation de la solution VITAM devront disposer en interne des compétences suivantes :

- connaissance d'ansible en tant qu'outil de déploiement automatisé ;
- connaissance de Consul en tant qu'outil de découverte de services ;
- maîtrise de MongoDB et ElasticSearch par les administrateurs de bases de données.

3.2 Pré-requis plate-forme

Les pré-requis suivants sont nécessaires :

3.2.1 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 basée 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 **2.7** minimale et conseillée ; se référer à la documentation ansible [8] pour la procédure d'installation)
  - openssh-clients (client SSH utilisé par ansible)

[8] http://docs.ansible.com/ansible/latest/intro_installation.html

  - 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 sur lesquels la solution logicielle *VITAM* doit être installée (fichier `~/.ssh/known_hosts` correctement renseigné)

Note : Se référer à la documentation d'usage [9] pour les procédures de connexion aux machines-cibles depuis le serveur ansible.

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`.

3.2.2 PKI

La solution VITAM nécessite des certificats pour son bon fonctionnement (cf. *DAT* pour la liste des secrets et *Vue d'ensemble de la gestion des certificats* (page 56) pour une vue d'ensemble de leur usage.) La gestion de ces certificats, par le biais d'une ou plusieurs PKI, est à charge de l'équipe d'exploitation. La mise à disposition des certificats et des chaînes de validation CA, placés dans les répertoires de déploiement adéquats, est un pré-requis à tout déploiement en production de la solution VITAM.

Voir aussi :

Veuillez vous référer à la section *Vue d'ensemble de la gestion des certificats* (page 56) pour la liste des certificats nécessaires au déploiement de la solution VITAM, ainsi que pour leurs répertoires de déploiement.

3.2.3 Systèmes d'exploitation

Seules deux distributions Linux suivantes sont supportées à ce jour :

- CentOS 7
- Debian 8 (jessie)

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

Note : En cas de changement de mode SELinux, redémarrer les machines pour la bonne prise en compte de la modification.

---

[9] http://docs.ansible.com/ansible/latest/intro_getting_started.html

Prudence : En cas d'installation initiale, les utilisateurs et groupes systèmes (noms et UID) utilisés par VITAM (et listés dans le *DAT*) ne doivent pas être présents sur les serveurs cible. Ces comptes sont créés lors de l'installation de VITAM et gérés par VITAM.

3.2.3.1 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)

3.2.3.2 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)

3.2.4 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>`

3.3 Récupération de la version

3.3.1 Utilisation des dépôts open-source

Les scripts de déploiement de VITAM sont disponibles dans le dépôt github VITAM [10] , dans le répertoire `deployment`.

Les binaires de VITAM sont disponibles sur un dépôt vitam public indiqué ci-dessous par type de package ; ces dépôts doivent être correctement configurés sur la plate-forme cible avant toute installation.

3.3.1.1 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

```
[programmevitam-vitam-rpm-release-product]
name=programmevitam-vitam-rpm-release-product
baseurl=http://download.programmevitam.fr/vitam_repository/<vitam_version>/rpm/vitam-
↪product/
gpgcheck=0
repo_gpgcheck=0
enabled=1

[programmevitam-vitam-rpm-release-external]
name=programmevitam-vitam-rpm-release-external
baseurl=http://download.programmevitam.fr/vitam_repository/<vitam_version>/rpm/vitam-
↪external/
gpgcheck=0
repo_gpgcheck=0
enabled=1
```

Note : remplacer <vitam_version> par la version à déployer.

3.3.1.2 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] http://download.programmevitam.fr/vitam_repository/<vitam_version>/
↪deb jessie vitam-product vitam-external
```

Note : remplacer <vitam_version> par la version à déployer.

3.3.2 Utilisation du package global d'installation

Note : Le package global d'installation n'est pas présent dans les dépôts publics.

https://github.com/ProgrammeVitam/vitam

Le package global d'installation contient :

- le package proprement dit
- 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 fichier d'extension tar.gz les binaires d'installation (rpm pour CentOS ; deb pour Debian) et les faire prendre en compte par le repository.

CHAPITRE 4

Procédures d'installation / mise à jour

4.1 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.

4.2 Procédures

4.2.1 Multi-sites

4.2.1.1 Procédure

Dans le cadre d'une installation multi-sites, il est nécessaire de déployer la solution logicielle *VITAM* sur le site secondaire dans un premier temps, puis déployer le site « production ».

Prudence : La variable `secret_plateforme` doit être commune sur les différents sites.

4.2.1.2 Flux entre Storage et offer

Dans le cas d'appel en https entre les composants Storage et Offer, il convient également de rajouter :

- **Sur le site primaire**
    - Dans le truststore de Storage : la CA ayant signé le certificat de l'Offer du site secondaire
- **Sur le site secondaire**
    - Dans le truststore de Offer : la CA ayant signé le certificat du Storage du site primaire
    - Dans le grantedstore de Offer : le certificat du storage du site primaire

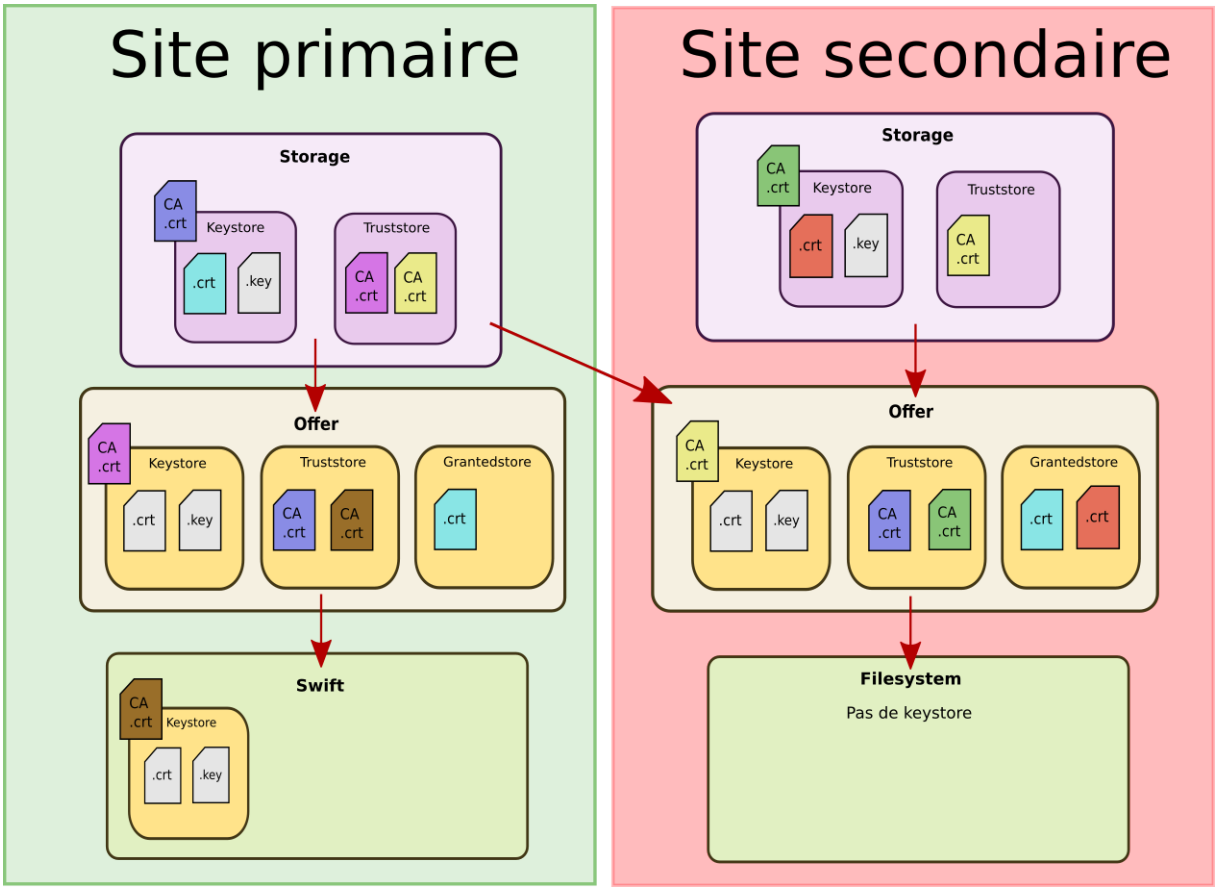

Fig. 1 – Vue détaillée des certificats entre le storage et l'offre en multi-site

Après la génération des keystore via le script deployment/generate_stores.sh, il convient donc de rajouter les CA et certificats indiqués ci-dessus.

Ajout d'un certificat : `keytool -import -keystore -file <certificat.crt> -alias <alias_certificat>`

Ajout d'une CA : `keytool -import -trustcacerts -keystore -file <ca.crt> -alias <alias_certificat>`

Il est également possible de placer les fichiers CA et certificats directement dans les bons répertoires avant de lancer generate_stores.sh.

4.2.2 Configuration du déploiement

Voir aussi :

L'architecture de la solution logicielle, les éléments de dimensionnement ainsi que les principes de déploiement sont définis dans le *DAT*.

4.2.2.1 Fichiers de déploiement

Les fichiers de déploiement sont disponibles dans la version VITAM livrée dans le sous-répertoire `deployment` . Concernant l'installation, ils consistent en 2 parties :

- les playbook ansible de déploiement, présents dans le sous-répertoire `ansible-vitam`, qui est indépendant de l'environnement à déployer ; ces fichiers ne sont normalement pas à modifier pour réaliser une installation.
- l'arborescence d'inventaire ; des fichiers d'exemple sont disponibles dans le sous-répertoire `environments`. Cette arborescence est valable pour le déploiement d'un environnement, et est à dupliquer lors de l'installation d'environnements ultérieurs. Les fichiers qui y sont contenus doivent être adaptés avant le déploiement, comme il est expliqué dans les paragraphes suivants.

4.2.2.2 Informations « plate-forme »

4.2.2.2.1 Inventaire

Pour configurer le déploiement, il est nécessaire de créer dans le répertoire `environments` un nouveau fichier d'inventaire (dans la suite, ce fichier sera communément appelé `hosts.<environnement>`). Ce fichier doit être basé sur la structure présente dans le fichier `hosts.example` (et notamment respecter scrupuleusement l'arborescence des groupes ansible) ; les commentaires dans ce fichier donnent les explications permettant l'adaptation à l'environnement cible :

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

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


########### 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 ###########

[ldap] # Extra : OpenLDAP server
# LDAP server !!! NOT FOR PRODUCTION !!! Test only

[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 ###########
```

(suite sur la page suivante)

(suite de la page précédente)

```
37
38  # Group definition ; DO NOT MODIFY
39  [vitam:children]
40  zone-external
41  zone-access
42  zone-applicative
43  zone-storage
44  zone-data
45  zone-admin
46
47
48  ##### Zone externe
49
50
51  [zone-external:children]
52  hosts-ihm-demo
53  hosts-cerebro
54  hosts-ihm-recette
55
56  # If you don't need consul for ihm-demo, you can set this var:
57  # consul_disabled=true
58  [hosts-ihm-demo]
59  # TODO: Put here servers where this service will be deployed : ihm-demo
60
61  [hosts-ihm-recette]
62  # TODO: Put here servers where this service will be deployed : ihm-recette (extra␣
    ↪feature)
63
64  [hosts-cerebro]
65  # TODO: Put here servers where this service will be deployed : vitam-elasticsearch-
    ↪cerebro
66
67
68  ##### Zone access
69
70  # Group definition ; DO NOT MODIFY
71  [zone-access:children]
72  hosts-ingest-external
73  hosts-access-external
74
75  [hosts-ingest-external]
76  # TODO: Put here servers where this service will be deployed : ingest-external
77
78
79  [hosts-access-external]
80  # TODO: Put here servers where this service will be deployed : access-external
81
82
83  ##### Zone applicative
84
85  # Group definition ; DO NOT MODIFY
86  [zone-applicative:children]
87  hosts-ingest-internal
88  hosts-processing
89  hosts-batch-report
90  hosts-worker
91  hosts-access-internal
```

(suite sur la page suivante)

(suite de la page précédente)

```
hosts-metadata
hosts-functional-administration
hosts-logbook
hosts-workspace
hosts-storage-engine
hosts-security-internal

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


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


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


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


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


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


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


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


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

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

[hosts-worker]
# TODO: Put here servers where this service will be deployed : worker
# Optional parameter after each host : vitam_worker_capacity=<integer> ; please refer␣
↪to your infrastructure for defining this number ; default is ansible_processor_
↪vcpus value (cpu number in /proc/cpuinfo file)


##### Zone storage

[zone-storage:children] # DO NOT MODIFY
hosts-storage-offer-default
```

(suite sur la page suivante)

(suite de la page précédente)

```
hosts-mongodb-offer

[hosts-storage-offer-default]
# TODO: Put here servers where this service will be deployed : storage-offer-default
# LIMIT : only 1 offer per machine and 1 machine per offer
# Mandatory param for each offer is offer_conf and points to offer_opts.yml & vault-vitam.yml (with same tree)
# hostname-offre-1.vitam offer_conf=offer-swift-1
# for filesystem
# hostname-offre-2.vitam offer_conf=offer-fs-1

[hosts-mongodb-offer:children]
hosts-mongos-offer
hosts-mongoc-offer
hosts-mongod-offer

[hosts-mongos-offer]
# WARNING : DO NOT COLLOCATE WITH [hosts-mongos-data]
# TODO: put here servers where this service will be deployed : mongos cluster for storage offers
# Mandatory param : mongo_cluster_name : name of the cluster (should exist in the offer_conf configuration)
# Example (for a more complete one, see the one in the group hosts-mongos-data) :
# vitam-mongo-swift-offer-01   mongo_cluster_name=offer-swift-1
# vitam-mongo-swift-offer-02   mongo_cluster_name=offer-swift-1
# vitam-mongo-fs-offer-01      mongo_cluster_name=offer-fs-1
# vitam-mongo-fs-offer-02      mongo_cluster_name=offer-fs-1

[hosts-mongoc-offer]
# WARNING : DO NOT COLLOCATE WITH [hosts-mongoc-data]
# TODO: put here servers where this service will be deployed : mongoc cluster for storage offers
# Mandatory param : mongo_cluster_name : name of the cluster (should exist in the offer_conf configuration)
# Optional param : mandatory for 1 node of the shard, some init commands will be executed on it
# Optional param : mongo_arbiter=true : the node will be only an arbiter ; do not add this paramter on a mongo_rs_bootstrap node
# Example :
# vitam-mongo-swift-offer-01   mongo_cluster_name=offer-swift-1                 mongo_rs_bootstrap=true
# vitam-mongo-swift-offer-02   mongo_cluster_name=offer-swift-1
# vitam-swift-offer            mongo_cluster_name=offer-swift-1                 mongo_arbiter=true
# vitam-mongo-fs-offer-01      mongo_cluster_name=offer-fs-1                    mongo_rs_bootstrap=true
# vitam-mongo-fs-offer-02      mongo_cluster_name=offer-fs-1
# vitam-fs-offer               mongo_cluster_name=offer-fs-1                    mongo_arbiter=true

[hosts-mongod-offer]
# WARNING : DO NOT COLLOCATE WITH [hosts-mongod-data]
# TODO: put here servers where this service will be deployed : mongod cluster for storage offers
# Mandatory param : mongo_cluster_name : name of the cluster (should exist in the offer_conf configuration)
# Mandatory param : id of the current shard, increment by 1 from 0 to n
```

(suite sur la page suivante)

(suite de la page précédente)

```
# Optional param : mandatory for 1 node of the shard, some init commands will be executed on it
# Optional param : mongo_arbiter=true : the node will be only an arbiter ; do not add this paramter on a mongo_rs_bootstrap node
# Example :
# vitam-mongo-swift-offer-01   mongo_cluster_name=offer-swift-1    mongo_shard_id=0    mongo_rs_bootstrap=true
# vitam-mongo-swift-offer-02   mongo_cluster_name=offer-swift-1    mongo_shard_id=0
# vitam-swift-offer            mongo_cluster_name=offer-swift-1    mongo_shard_id=0    mongo_arbiter=true
# vitam-mongo-fs-offer-01      mongo_cluster_name=offer-fs-1       mongo_shard_id=0    mongo_rs_bootstrap=true
# vitam-mongo-fs-offer-02      mongo_cluster_name=offer-fs-1       mongo_shard_id=0
# vitam-fs-offer               mongo_cluster_name=offer-fs-1       mongo_shard_id=0    mongo_arbiter=true

##### Zone data

# Group definition ; DO NOT MODIFY
[zone-data:children]
hosts-elasticsearch-data
hosts-mongodb-data

[hosts-elasticsearch-data]
# TODO: Put here servers where this service will be deployed : elasticsearch-data cluster
# 2 params available for huge environments (parameter to be declared after each server) :
#    is_data=true/false
#    is_master=true/false
#    other options are not handled yet
# defaults are set to true, if undefined. If defined, at least one server MUST be is_data=true
# Examples :
# server1 is_master=true is_data=false
# server2 is_master=false is_data=true
# More explanation here : https://www.elastic.co/guide/en/elasticsearch/reference/5.6/modules-node.html


# Group definition ; DO NOT MODIFY
[hosts-mongodb-data:children]
hosts-mongos-data
hosts-mongoc-data
hosts-mongod-data

[hosts-mongos-data]
# WARNING : DO NOT COLLOCATE WITH [hosts-mongos-offer]
# TODO: Put here servers where this service will be deployed : mongos cluster
# Mandatory param : mongo_cluster_name=mongo-data  ("mongo-data" is mandatory)
# Example :
# vitam-mdbs-01   mongo_cluster_name=mongo-data
# vitam-mdbs-01   mongo_cluster_name=mongo-data
# vitam-mdbs-01   mongo_cluster_name=mongo-data

[hosts-mongoc-data]
# WARNING : DO NOT COLLOCATE WITH [hosts-mongoc-offer]
```

(suite sur la page suivante)

(suite de la page précédente)

```
# TODO: Put here servers where this service will be deployed : mongoc cluster
# Mandatory param : mongo_cluster_name=mongo-data  ("mongo-data" is mandatory)
# Optional param : mandatory for 1 node of the shard, some init commands will be executed on it
# Example :
# vitam-mdbc-01   mongo_cluster_name=mongo-data                    mongo_rs_bootstrap=true
# vitam-mdbc-01   mongo_cluster_name=mongo-data
# vitam-mdbc-01   mongo_cluster_name=mongo-data

[hosts-mongod-data]
# WARNING : DO NOT COLLOCATE WITH [hosts-mongod-offer]
# TODO: Put here servers where this service will be deployed : mongod cluster
# Each replica_set should have an odd number of members (2n + 1)
# Reminder: For Vitam, one mongodb shard is using one replica_set
# Mandatory param : mongo_cluster_name=mongo-data ("mongo-data" is mandatory)
# Mandatory param : id of the current shard, increment by 1 from 0 to n
# Optional param : mandatory for 1 node of the shard, some init commands will be executed on it
# Example:
# vitam-mdbd-01  mongo_cluster_name=mongo-data   mongo_shard_id=0  mongo_rs_bootstrap=true
# vitam-mdbd-02  mongo_cluster_name=mongo-data   mongo_shard_id=0
# vitam-mdbd-03  mongo_cluster_name=mongo-data   mongo_shard_id=0
# vitam-mdbd-04  mongo_cluster_name=mongo-data   mongo_shard_id=1  mongo_rs_bootstrap=true
# vitam-mdbd-05  mongo_cluster_name=mongo-data   mongo_shard_id=1
# vitam-mdbd-06  mongo_cluster_name=mongo-data   mongo_shard_id=1

###### Zone admin

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

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

[hosts-kibana-data]
# TODO: Put here servers where this service will be deployed : kibana (for data cluster)

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


[hosts-kibana-log]
# TODO: Put here servers where this service will be deployed : kibana (for log cluster)

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

```

(suite sur la page suivante)

(suite de la page précédente)

```
287
288 [hosts-elasticsearch-log]
289 # TODO: Put here servers where this service will be deployed : elasticsearch-log␣
    ↪cluster
290
291 ########### Global vars ###########
292
293 [hosts:vars]
294
295 # ==============================
296 # VITAM
297 # ==============================
298
299 # Declare user for ansible on target machines
300 ansible_ssh_user=
301 # Can target user become as root ? ; true is required by VITAM (usage of a sudoer is␣
    ↪mandatory)
302 ansible_become=true
303
304 # Related to Consul ; apply in a table your DNS server(s)
305 # Example : dns_servers=["8.8.8.8","8.8.4.4"]
306 dns_servers=
307
308 # Vitam tenants to create
309 vitam_tenant_ids=[0,1,2]
310 vitam_tenant_admin=1
311
312 ### Logback configuration ###
313 # Days before deleting logback log files (java & access logs for vitam components)
314 days_to_delete_logback_logfiles=
315
316 # Configuration for Curator
317 #       Days before deletion on log management cluster; 365 for production␣
    ↪environment
318 days_to_delete_logstash_indexes=
319 #       Days before closing "old" indexes on log management cluster; 30 for␣
    ↪production environment
320 days_to_close_logstash_indexes=
321
322 # Define local Consul datacenter name
323 vitam_site_name=prod-dc1
324 # EXAMPLE : vitam_site_name = prod-dc1
325 # check whether on primary site (true) or secondary (false)
326 primary_site=true
327
328
329 # ==============================
330 # EXTRA
331 # ==============================
332 # Environment (defines title in extra on reverse homepage). Variable is DEPRECATED !
333 #environnement=
334
335 ### vitam-itest repository ###
336 vitam_tests_branch=master
337 vitam_tests_gitrepo_protocol=
338 vitam_tests_gitrepo_baseurl=
339 vitam_tests_gitrepo_url=
```

(suite sur la page suivante)

(suite de la page précédente)

```
340
341 # Curator configuration
342 #        Days before deletion for packetbeat index only on log management cluster
343 days_to_delete_packetbeat_indexes=5
344 #        Days before deletion for metricbeat index only on log management cluster; 30␣
    ↪for production environment
345 days_to_delete_metricbeat_indexes=30
346 # Days before closing metrics elasticsearch indexes
347 days_to_close_metrics_indexes=7
348 # Days before deleting metrics elasticsearch indexes
349 days_to_delete_metrics_indexes=30
350 days_to_delete_packetbeat_indexes=20
351 days_to_delete_metricbeat_indexes=20
352
353
354
355 # Used when VITAM is behind a reverse proxy (provides configuration for reverse proxy␣
    ↪&& displayed in header page)
356 vitam_reverse_external_dns=
357 # For reverse proxy use
358 reverse_proxy_port=80
359 # http_proxy env var to use ; has to be declared even if empty
360 http_proxy_environnement=
```

Pour chaque type de « host », indiquer le(s) serveur(s) défini(s) pour chaque fonction. Une colocalisation de composants est possible (Cf. le paragraphe idoine du *DAT*)

Note : Concernant le groupe « hosts-consul-server », il est nécessaire de déclarer un minimum de 3 machines.

Avertissement : Il n'est pas possible de colocaliser les clusters MongoDB « data » et « offer ».

Avertissement : Il n'est pas possible de colocaliser « kibana-data » et « kibana-log ».

4.2.2.2.2 Fichier `vitam_security.yml`

La configuration des droits d'accès à VITAM est réalisée dans le fichier `environments /group_vars/all/vitam_security.yml`, comme suit :

```
1  ---
2
3  # Business vars
4
5  ### Admin context name and tenants ###
6  admin_context_name: "admin-context"
7  admin_context_tenants: "{{vitam_tenant_ids}}"
8  # Indicate context certificates relative paths under {{inventory_dir}}/certs/client-
    ↪external/clients
9  # vitam-admin-int is mandatory for internal use (PRONOM upload)
10 admin_context_certs: [ "ihm-demo/ihm-demo.crt", "ihm-recette/ihm-recette.crt",
    ↪"reverse/reverse.crt", "vitam-admin-int/vitam-admin-int.crt" ]
```

(suite sur la page suivante)

(suite de la page précédente)

```
# Indicate here all the personal certificates relative paths under {{inventory_dir}}/certs/client-vitam-users/clients
admin_personal_certs: [ "userOK.crt" ]

# Admin security profile name
admin_security_profile: "admin-security-profile"

admin_basic_auth_user: "adminUser"
```

4.2.2.2.3 Fichier `offers_opts.yml`

Enfin, la déclaration des configuration des offres de stockage est réalisée dans le fichier `environments / group_vars/all/offers_opts.yml` :

```
# This list is ordered. It can and has to be completed if more offers are necessary
# Strategy order (1st has to be the prefered one)
vitam_strategy:
  - name: offer-fs-1
    referent: true
#    vitam_site_name: prod-dc2
#  - name: offer-swift-1
# Example :
#  - name: distant
#    referent: true
#    vitam_site_name: distant-dc2

# DON'T forget to add associated passwords in vault-vitam.yml with same tree when using provider openstack-swift*
# ATTENTION !!! Each offer has to have a distinct name, except for clusters binding a same physical storage
# WARNING : for offer names, please only use [a-z][a-z0-9-]* pattern
vitam_offers:
  offer-fs-1:
    # param can be filesystem-hash (recomended) or filesystem (not recomended)
    provider: filesystem-hash
  offer-swift-1:
    # provider : openstack-swift for v1 or openstack-swift-v3 for v3
    provider: openstack-swift-v3
    # swiftKeystoneAuthUrl : URL de connexion à keystone
    swiftKeystoneAuthUrl: https://openstack-hostname:port/auth/1.0
    # swiftDomain : domaine OpenStack dans lequel l'utilisateur est enregistré
    swiftDomain: domaine
    # swiftUser : identifiant de l'utilisateur
    swiftUser: utilisateur
    # swiftPassword: has to be set in vault-vitam.yml (encrypted) with same structure => DO NOT COMMENT OUT
    # swiftProjectName : nom du projet openstack
    swiftProjectName: monTenant
    # swiftUrl: optional variable to force the swift URL
    # swiftUrl: https://swift-hostname:port/swift/v1

  # example_swift_v1:
```

(suite sur la page suivante)

(suite de la page précédente)

```
  #    provider: openstack-swift
  #    swiftKeystoneAuthUrl: https://keystone/auth/1.0
  #    swiftDomain: domain
  #    swiftUser: user
  #    swiftPassword: has to be set in vault-vitam.yml (encrypted) with same␣
↪structure => DO NOT COMMENT OUT
  # example_swift_v3:
  #    provider: openstack-swift-v3
  #    swiftKeystoneAuthUrl: https://keystone/v3
  #    swiftDomain: domaine
  #    swiftUser: user
  #    swiftPassword: has to be set in vault-vitam.yml (encrypted) with same␣
↪structure => DO NOT COMMENT OUT
  #    swiftProjectName: monTenant
  #    projectName: monTenant
```

Se référer aux commentaires dans le fichier pour le renseigner correctement.

Note : Dans le cas d'un déploiement multi-sites, dans la section `vitam_strategy`, la directive `vitam_site_name` définit pour l'offre associée le nom du datacenter consul. Par défaut, si non définie, c'est la valeur de la variable `vitam_site_name` définie dans l'inventaire.

Avertissement : La cohérence entre l'inventaire et la section `vitam_strategy` est critique pour le bon déploiement et fonctionnement de la solution logicielle VITAM. En particulier, la liste d'offres de `vitam_strategy` doit correspondre *exactement* aux noms d'offre déclarés dans l'inventaire (ou les inventaires de chaque datacenter, en cas de fonctionnement multi-site).

Avertissement : Ne pas oublier, en cas de connexion à un keystone en https, de répercuter dans la *PKI* la clé publique de la CA du keystone.

4.2.2.2.4 Fichier `cots_vars.yml`

Dans le cas du choix du *COTS* d'envoi des messages syslog dans logastsh, il est possible de choisir entre `syslog-ng` et `rsyslog` dans le fichier `environments/group_vars/all/cots_vars.yml` :

```
---

consul:
    dns_port: 53

consul_remote_sites:
    # wan contains the wan addresses of the consul server instances of the␣
↪external vitam sites
    # Exemple, if our local dc is dc1, we will need to set dc2 & dc3 wan␣
↪conf:
    # - dc2:
    #   wan: ["10.10.10.10","1.1.1.1"]
    # - dc3:
    #   wan: ["10.10.10.11","1.1.1.1"]
```

(suite sur la page suivante)

(suite de la page précédente)

```

elasticsearch:
    log:
        host: "elasticsearch-log.service.{{ consul_domain }}"
        port_http: "9201"
        port_tcp: "9301"
        groupe: "log"
        baseuri: "elasticsearch-log"
        cluster_name: "elasticsearch-log"
        https_enabled: false
        # default index template
        index_templates:
            default:
                shards: 1
                replica: 1
    data:
        host: "elasticsearch-data.service.{{ consul_domain }}"
        # default is 0.1 (10%) and should be quite enough in most cases
        #index_buffer_size_ratio: "0.15"
        port_http: "9200"
        port_tcp: "9300"
        groupe: "data"
        baseuri: "elasticsearch-data"
        cluster_name: "elasticsearch-data"
        https_enabled: false
        # default index template
        index_templates:
            default:
                shards: 10
                replica: 2

mongodb:
    mongos_port: 27017
    mongoc_port: 27018
    mongod_port: 27019
    mongo_authentication: "true"
    host: "mongos.service.{{ consul_domain }}"

logstash:
    host: "logstash.service.{{ consul_domain }}"
    user: logstash
    port: 10514
    rest_port: 20514

# Curator units: days
curator:
    log:
        metrics:
            close: 5
            delete: 30
        logstash:
            close: 5
            delete: 30
        metricbeat:
            close: 5
            delete: 30
        packetbeat:
```

(suite sur la page suivante)

(suite de la page précédente)

```
            close: 5
            delete: 30

kibana:
    header_value: "reporting"
    log:
        baseuri: "kibana_log"
        api_call_timeout: 120
        groupe: "log"
        port: 5601
        default_index_pattern: "logstash-vitam*"
        # default shards & replica
        shards: 5
        replica: 1
        # pour index logstash-*
        metrics:
            shards: 5
            replica: 1
        # pour index metrics-vitam-*
        logs:
            shards: 5
            replica: 1
        # pour index metricbeat-*
        metricbeat:
            shards: 5 # must be a factor of 30
            replica: 1
    data:
        baseuri: "kibana_data"
        # OMA : bugdette : api_call_timeout is used for retries ; should ceate a separate variable rather than this one
        api_call_timeout: 120
        groupe: "data"
        port: 5601
        default_index_pattern: "logbookoperation_*"
        # index template for .kibana
        shards: 1
        replica: 1

syslog:
    # value can be syslog-ng or rsyslog
    name: "rsyslog"

cerebro:
    baseuri: "cerebro"
    port: 9000

siegfried:
    port: 19000

clamav:
    port: 3310
    # frequency freshclam for database update per day (from 0 to 24 - 24 meaning hourly check)
    db_update_periodicity: 1

mongo_express:
    baseuri: "mongo-express"
```

(suite sur la page suivante)

(suite de la page précédente)

```
125
126 ldap_authentification:
127     ldap_protocol: "ldap"
128     ldap_server: "{% if groups['ldap']|length > 0 %}{{ groups['ldap']|first }
    ↪}{% endif %}"
129     ldap_port: "389"
130     ldap_base: "dc=programmevitam,dc=fr"
131     ldap_login: "cn=Manager,dc=programmevitam,dc=fr"
132     uid_field: "uid"
133     ldap_userDn_Template: "uid={0},ou=people,dc=programmevitam,dc=fr"
134     ldap_group_request: "(&(objectClass=groupOfNames)(member={0}))"
135     ldap_admin_group: "cn=admin,ou=groups,dc=programmevitam, dc=fr"
136     ldap_user_group: "cn=user,ou=groups,dc=programmevitam, dc=fr"
137     ldap_guest_group: "cn=guest,ou=groups,dc=programmevitam, dc=fr"
```

Il faut alors modifier la valeur de la directive `syslog.name` ; la valur par défaut est `rsyslog`.

4.2.2.3 Déclaration des secrets

Avertissement : Cette section décrit des fichiers contenant des données sensibles ; il convient de sécuriser ces fichiers avec un mot de passe « fort ». En cas d'usage d'un fichier de mot de passe (« vault-password-file »), il faut renseigner ce mot de passe comme contenu du fichier et ne pas oublier de sécuriser ou supprimer ce fichier à l'issue de l'installation.

Les secrets utilisés par la solution logicielle (en-dehors des certificats qui sont abordés dans une section ultérieure) sont définis dans des fichiers chiffrés par ansible-vault.

Important : Tous les vault présents dans l'arborescence d'inventaire doivent être tous protégés par le même mot de passe !

La première étape consiste à changer les mots de passe de tous les vault présents dans l'arborescence de déploiement (le mot de passe par défaut est contenu dans le fichier `vault_pass.txt`) à l'aide de la commande `ansible-vault rekey <fichier vault>`.

Voici la liste des vaults pour lesquels il est nécessaire de modifier le mot de passe :

- environments/group_vars/all/vault-vitam.yml
- environments/group_vars/all/vault-keystores.yml
- environments/group_vars/all/vault-extra.yml
- environments/certs/vault-certs.yml

2 vaults sont principalement utilisés dans le déploiement d'une version ; leur contenu est donc à modifier avant tout déploiement :

- Le fichier `environments /group_vars/all/vault-vitam.yml` contient les secrets généraux :

```
1 ---
2 # Vitam platform secret key
3 plateforme_secret: vitamsecret
4
5 # The consul key must be 16-bytes, Base64 encoded: https://www.consul.io/docs/
  ↪agent/encryption.html
6 # You can generate it with the "consul keygen" command
```

(suite sur la page suivante)

(suite de la page précédente)

```
# Or you can use this script: deployment/pki/scripts/generate_consul_key.sh
consul_encrypt: Biz14ohqN4HtvZmrXp3N4A==

mongodb:
  mongo-data:
    passphrase: mongogo
    admin:
      user: vitamdb-admin
      password: azerty
    localadmin:
      user: vitamdb-localadmin
      password: qwerty
    metadata:
      user: metadata
      password: azerty1
    logbook:
      user: logbook
      password: azerty2
    report:
      user: report
      password: azerty5
    functionalAdmin:
      user: functional-admin
      password: azerty3
    securityInternal:
      user: security-internal
      password: azerty4
  offer-fs-1:
    passphrase: mongogo
    admin:
      user: vitamdb-admin
      password: azerty
    localadmin:
      user: vitamdb-localadmin
      password: qwerty
    offer:
      user: offer
      password: azerty5
  offer-fs-2:
    passphrase: mongogo
    admin:
      user: vitamdb-admin
      password: azerty
    localadmin:
      user: vitamdb-localadmin
      password: qwerty
    offer:
      user: offer
      password: azerty5
  offer-swift-1:
    passphrase: mongogo
    admin:
      user: vitamdb-admin
      password: azerty
    localadmin:
      user: vitamdb-localadmin
      password: qwerty
```

(suite sur la page suivante)

(suite de la page précédente)

```
    offer:
      user: offer
      password: azerty5

vitam_users:
  - vitam_aadmin:
    login: aadmin
    password: aadmin1234
    role: admin
  - vitam_uuser:
    login: uuser
    password: uuser1234
    role: user
  - vitam_gguest:
    login: gguest
    password: gguest1234
    role: guest
  - techadmin:
    login: techadmin
    password: techadmin1234
    role: admin

ldap_authentification:
    ldap_pwd: "admin"

admin_basic_auth_password: adminPassword

vitam_offers:
    offer-swift-1:
        swiftPassword: password
```

Note : Dans le cadre d'une installation avec au moins une offre « swift », il faut déclarer, dans la section `vitam_offers`, le nom de chaque offre et le mot de passe de connexion « swift » associé, défini dans le fichier `offers_opts.yml`. L'exemple ci-dessus présente la déclaration du mot de passe pour l'offre swift « offer-swift-1 ».

- Le fichier `environments/group_vars/all/vault-keystores.yml` contient les mots de passe des magasins de certificats utilisés dans VITAM :

```
keystores:
  server:
    offer: azerty1
    access_external: azerty2
    ingest_external: azerty3
    ihm_recette: azerty16
    ihm_demo: azerty17
  client_external:
    ihm_demo: azerty4
    gatling: azerty4
    ihm_recette: azerty5
    reverse: azerty6
  client_storage:
    storage: azerty7
  timestamping:
```

(suite sur la page suivante)

(suite de la page précédente)

```
    secure_logbook: azerty8
truststores:
  server: azerty9
  client_external: azerty10
  client_storage: azerty11
grantedstores:
  client_external: azerty12
  client_storage: azerty13
```

Avertissement : il convient de sécuriser votre environnement en définissant des mots de passe « forts ».

4.2.2.3.1 Cas des extra

- Le fichier `environments/group_vars/all/vault-extra.yml` contient les mot de passe des magasins de certificats utilisés dans VITAM :

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

Note : 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.

4.2.3 Gestion des certificats

Une vue d'ensemble de la gestion des certificats est présentée *dans l'annexe dédiée* (page 56).

4.2.3.1 Cas 1 : Configuration développement / tests

Pour des usages de développement ou de tests hors production, il est possible d'utiliser la *PKI* fournie avec la solution logicielle VITAM.

4.2.3.1.1 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 ! De plus il n'est pas prévu de l'utiliser pour générer les certificats d'une autre application qui serait cliente de Vitam.

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

4.2.3.1.2 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. Les mots de passe des clés privées des autorités de certification sont stockés dans le vault ansible environments/certs/vault-ca.yml

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.

4.2.3.1.3 Génération des certificats par les scripts Vitam

Le fichier d'inventaire de déploiement `environments/<fichier d'inventaire>` (cf. *Informations « plate-forme »* (page 11)) 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>
```

Ce script génère sous `environments/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 environments/certs/vault-certs.yml

Prudence : Les certificats générés à l'issue ont une durée de validité de 3 ans.

4.2.3.2 Cas 2 : Configuration production

4.2.3.2.1 Procédure générale

La procédure suivante s'applique lorsqu'une *PKI* est déjà disponible pour fournir les certificats nécessaires.

Les étapes d'intégration des certificats à la solution Vitam sont les suivantes :

- Générer les certificats avec les bons *key usage* par type de certificat
- Déposer les certificats et les autorités de certifications correspondantes dans les bons répertoires.
- Renseigner les mots de passe des clés privées des certificats dans le vault ansible `environments/certs/vault-certs.yml`
- Utiliser le script Vitam permettant de générer les différents keystores.

Note : Rappel pré-requis : vous devez disposer d'une ou plusieurs *PKI* pour tout déploiement en production de la solution VITAM.

4.2.3.2.2 Génération des certificats

En conformité avec le document RGSV2 de l'ANSSI, il est recommandé de générer des certificats avec les caractéristiques suivantes :

4.2.3.2.2.1 Certificats serveurs

- **Key Usage**
  - digitalSignature, keyEncipherment
- **Extended Key Usage**
  - TLS Web Server Authentication

Les certificats serveurs générés doivent prendre en compte des alias « web » (`subjectAltName`).

Le *subjectAltName* des certificats serveurs (`deployment/environments/certs/server/hosts/*`) doit contenir le nom dns du service sur consul associé.

Exemple avec un cas standard : <composant_vitam>.service.<consul_domain>. Ce qui donne pour le certificat serveur de access-external par exemple :

```
X509v3 Subject Alternative Name:
    DNS:access-external.service.consul, DNS:localhost
```

Il faudra alors mettre le même nom de domaine pour la configuration de consul (fichier `deployment/environments/group_vars/all/vitam_vars.yml`, variable `consul_domain`)

Cas particulier pour ihm-demo et ihm-recette : il faut rajouter le nom dns qui sera utilisé pour requêter ces deux applications si celles-ci sont appelées directement en frontal en https.

4.2.3.2.2.2 Certificat clients

- **Key Usage**
  - digitalSignature
- **Extended Key Usage**
  - TLS Web Client Authentication

4.2.3.2.2.3 Certificats d'horodatage

- **Key Usage**
  - digitalSignature, nonRepudiation
- **Extended Key Usage**
  - Time Stamping

4.2.3.2.3 Intégration de certificats existants

Une fois les certificats et CA mis à disposition par votre PKI, il convient de les positionner sous `environments/certs/....` en respectant la structure indiquée ci-dessous.

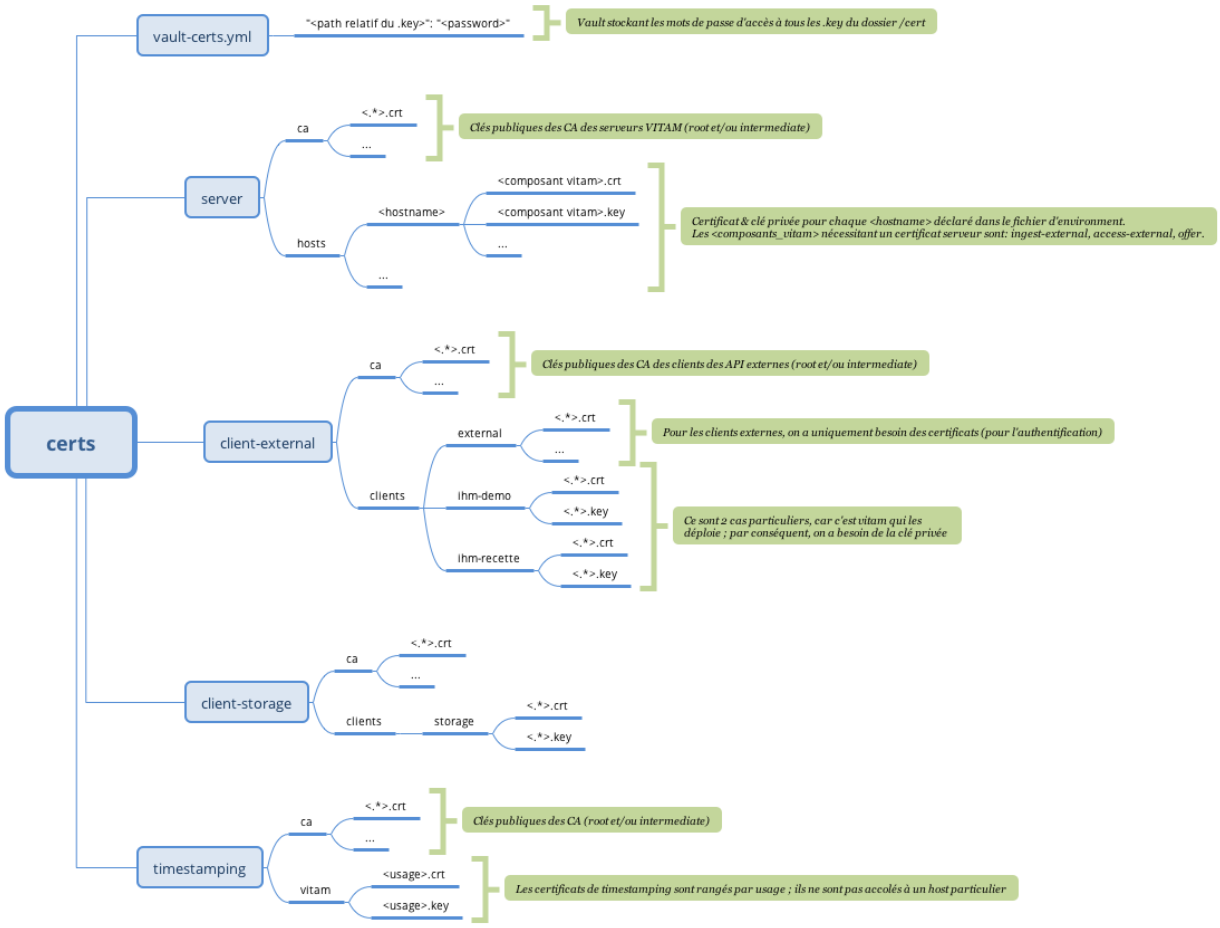

Fig. 2 – Vue détaillée de l'arborescence des certificats

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 lancer le déploiement.

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

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

4.2.3.2.4 Intégration d'une application externe (cliente)

Dans le cas d'ajout de certificats *SIA* externes :

- Déposer le certificat (`.crt`) de l'application client dans `environments/certs/client-external/clients/external/`
- Déposer les *CA* du certificat de l'application (`.crt`) dans `environments/certs/client-external/ca/`

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

- Editer le fichier `environments/group_vars/all/vitam_security.yml` et ajouter le(s) entrée(s) supplémentaire(s) (sous forme répertoire/fichier.crt, exemple : `external/mon_sia.crt`) dans la directive `admin_context_certs` pour que ceux-ci soient ajoutés aux profils de sécurité durant le déploiement de la solution logicielle *VITAM*.

4.2.3.3 Intégration de CA pour une offre swift

En cas d'utilisation d'une offre swift en https, il est nécessaire d'ajouter les *CA* du certificat de l'*API* swift.

Il faut les déposer dans `environments/certs/server/ca/`.

4.2.3.4 Génération des magasins de certificats

En prérequis, les certificats et les autorités de certification (*CA*) 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 : `environments/group_vars/all/vault-keystores.yml`, décrit dans la section *Déclaration des secrets* (page 23).

Lancer le script : `./generate_stores.sh`

Ce script génère sous `environments/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 : `environments/group_vars/all/vault-keystores.yml`

Note : Le mot de passe du fichier `vault-keystores.yml` est identique à celui des autres *vaults* ansible.

4.2.4 Paramétrages supplémentaires

4.2.4.1 Tuning JVM

Note : Cette section est en cours de développement.

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

Un tuning fin des paramètres JVM de chaque composant VITAM est possible. Pour cela, il faut modifier le fichier `group_vars/all/jvm_opts.yml`

Pour chaque composant, il est possible de modifier ces 3 variables :

- memory : paramètres Xms et Xmx
- gc : parmètres gc
- java : autres paramètres java

4.2.4.2 Paramétrage de l'antivirus (ingest-externe)

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

- Modifier le fichier `environments/group_vars/all/vitam_vars.yml` pour indiquer le nom de l'antivirus qui sera utilisé (norme : scan-<nom indiqué dans vitam-vars.yml>.sh)
- Créer un shell (dont l'extension doit être `.sh`) sous `environments/antivirus/` (norme : scan-<nom indiqué dans vitam-vars.yml>.sh) ; prendre comme modèle le fichier `scan-clamav.sh`. 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

Prudence : En cas de remplacement de clamAV par un autre antivirus, l'installation de celui-ci devient dès lors un prérequis de l'installation et le script doit être testé.

Avertissement : Sur plate-forme Debian, ClamAV est installé sans base de données. Pour que l'antivirus soit fonctionnel, il est nécessaire, durant l'installation, de la télécharger ; il est donc nécessaire de renseigner dans l'inventaire la directive `http_proxy_environnement`.

4.2.4.3 Paramétrage des certificats externes (*-externe)

Se reporter au chapitre dédié à la gestion des certificats : *Gestion des certificats* (page 26)

4.2.4.4 Placer « hors Vitam » le composant ihm-demo

Sous `deployment/environments/host_vars`, créer ou éditer un fichier nommé par le nom de machine hébergeant le composant ihm-demo et ajouter le contenu ci-dessous

```
consul_disabled: true
```

A l'issue, le déploiement n'installera pas l'agent Consul. Le composant ihm-demo appellera, alors, par l'adresse IP de services les composants « access-external » et « ingest-external ».

Il est également fortement recommandé de positionner la valeur de la directive `vitam.ihm_demo.metrics_enabled` à `false` dans le fichier `deployment/environments/group_vars/all/vitam_vars.yml`, afin que ce composant ne tente pas d'envoyer de données sur « elasticsearch-log ».

4.2.4.5 Paramétrage de la centralisation des logs Vitam

2 cas sont possibles :

- Utiliser le sous-système de gestion des logs fournis par la solution logicielle VITAM ;
- Utiliser un SIEM tiers.

4.2.4.5.1 Gestion par Vitam

Pour une gestion des logs par Vitam, il est nécessaire de déclarer les serveurs ad-hoc dans le fichier d'inventaire pour les 3 group

- hosts-logstash
- hosts-kibana-log
- hosts-elasticsearch-log

4.2.4.5.2 Redirection des logs sur un SIEM tiers

En configuration par défaut, les logs Vitam sont tout d'abord routés vers un serveur rsyslog installé sur chaque machine. Il est possible d'en modifier le routage, qui par défaut redirige vers le serveur logstash via le protocole syslog en TCP.

Pour cela, il est nécessaire de placer un fichier de configuration dédié dans le dossier `/etc/rsyslog.d/` ; ce fichier sera automatiquement pris en compte par rsyslog. Pour la syntaxe de ce fichier de configuration rsyslog, se référer à la documentation rsyslog [12].

Astuce : Pour cela, il peut être utile de s'inspirer du fichier de référence VITAM `deployment/ansible-vitam/roles/rsyslog/templates/vitam_transport.conf.j2` (attention, il s'agit d'un fichier template ansible, non directement convertible en fichier de configuration sans en ôter les directives jinja2).

4.2.4.6 Fichiers complémentaires

A titre informatif, le positionnement des variables ainsi que des dérivations des déclarations de variables sont effectuées dans les fichiers suivants :

- `environments/group_vars/all/vitam_vars.yml`, comme suit :

```
---

### global ###

# TODO MAYBE : permettre la surcharge avec une syntax du genre vitamopts.folder_root | default(vitam_default.folder_root) dans les templates ?

vitam_defaults:
    folder:
        root_path: /vitam
        folder_permission: "0750"
        conf_permission: "0640"
        folder_upload_permission: "0770"
        script_permission: "0750"
    users:
        vitam: "vitam"
        vitamdb: "vitamdb"
        group: "vitam"
    services:
        # Default log level for vitam components: logback values (TRACE, DEBUG, INFO, WARN, ERROR, OFF)
        log_level: WARN
        start_timeout: 300
```

(suite sur la page suivante)

http://www.rsyslog.com/doc/v7-stable/

(suite de la page précédente)

```
        stop_timeout: 3600
        port_service_timeout: 86400
        api_call_timeout: 120
    # Filter for the vitam package version to install
    # FIXME : commented as has to be removed becuase doesn't work under Debain
    #package_version: "*"
    ### Trust X-SSL-CLIENT-CERT header for external api auth ? (true | false) ###
    vitam_ssl_user_header: true
    ### Force chunk mode : set true if chunk header should be checked
    vitam_force_chunk_mode: false
    # syslog_facility
    syslog_facility: local0

# Used in ingest, unitary update, mass-update
classificationList: ["Secret Défense", "Confidentiel Défense"]
# Used in ingest, unitary update, mass-update
classificationLevelOptional: true


### consul ###
# FIXME: Consul à la racine pour le moment à cause de problèmes de récursivité␣
↪dans le parsing yaml
# WARNING: consul_domain should be a supported domain name for your organization
#          You will have to generate server certificates with the same domain␣
↪name and the service subdomain name
#          Example: consul_domain=vitam means you will have to generate some␣
↪certificates with .service.vitam domain
#                   access-external.service.vitam, ingest-external.service.vitam,␣
↪...
consul_domain: consul
consul_component: consul
consul_folder_conf: "{{ vitam_defaults.folder.root_path }}/conf/{{ consul_
↪component }}"

# Workspace should be useless but storage have a dependency to it...
# elastic-kibana-interceptor is present as kibana is present, if kibana-data &␣
↪interceptor are not needed in the secondary site, just do not add them in the␣
↪hosts file
vitam_secondary_site_components: [ "logbook" , "metadata" , "functional-
↪administration" , "storage" , "storageofferdefault" , "offer" , "elasticsearch-
↪log" , "elasticsearch-data" , "logstash" , "kibana" , "mongoc" , "mongod" ,
↪"mongos", "elastic-kibana-interceptor" ]


### Composants Vitam ###

vitam:
    accessexternal:
        # Component name: do not modify
        vitam_component: access-external
        # DNS record for the service:
        # Modify if ihm-demo is not using consul (typical production deployment)
        host: "access-external.service.{{ consul_domain }}"
        port_admin: 28102
        port_service: 8444
        baseuri: "access-external"
        https_enabled: true
```

(suite sur la page suivante)

(suite de la page précédente)

```
        # Use platform secret for this component ? : do not modify
        secret_platform: "false"
        # Force the log level for this component: this are logback values (TRACE, DEBUG, INFO, WARN, ERROR, OFF)
        # If this var is not set, the default one will be used (vitam_defaults.services.log_level)
        # log_level: "DEBUG"
        metrics_enabled: true
    accessinternal:
        vitam_component: access-internal
        host: "access-internal.service.{{ consul_domain }}"
        port_service: 8101
        port_admin: 28101
        baseuri: "access-internal"
        https_enabled: false
        secret_platform: "true"
        # log_level: "DEBUG"
        metrics_enabled: true
    functional_administration:
        vitam_component: functional-administration
        host: "functional-administration.service.{{ consul_domain }}"
        port_service: 8004
        port_admin: 18004
        baseuri: "functional-administration"
        https_enabled: false
        secret_platform: "true"
        cluster_name: "{{ elasticsearch.data.cluster_name }}"
        # log_level: "DEBUG"
        metrics_enabled: true
    elastickibanainterceptor:
        vitam_component: elastic-kibana-interceptor
        host: "elastic-kibana-interceptor.service.{{ consul_domain }}"
        port_service: 8014
        port_admin: 18014
        baseuri: ""
        https_enabled: false
        secret_platform: "false"
        cluster_name: "{{ elasticsearch.data.cluster_name }}"
        # log_level: "DEBUG"
        metrics_enabled: true
    batchreport:
        vitam_component: batch-report
        host: "batch-report.service.{{ consul_domain }}"
        port_service: 8015
        port_admin: 18015
        baseuri: "batch-report"
        https_enabled: false
        secret_platform: "false"
        # log_level: "DEBUG"
        metrics_enabled: true
    ingestexternal:
        vitam_component: ingest-external
        # DNS record for the service:
        # Modify if ihm-demo is not using consul (typical production deployment)
        host: "ingest-external.service.{{ consul_domain }}"
        port_admin: 28001
        port_service: 8443
```

(suite sur la page suivante)

(suite de la page précédente)

```
        baseuri: "ingest-external"
        https_enabled: true
        secret_platform: "false"
        antivirus: "clamav"
        # Directory where files should be placed for local ingest
        upload_dir: "/vitam/data/ingest-external/upload"
        # Directory where successful ingested files will be moved to
        success_dir: "/vitam/data/ingest-external/upload/success"
        # Directory where failed ingested files will be moved to
        fail_dir: "/vitam/data/ingest-external/upload/failure"
        # Action done to file after local ingest (see below for further information)
        upload_final_action: "MOVE"
        # log_level: "DEBUG"
        # upload_final_action can be set to three different values (lower or upper case does not matter)
        #   MOVE : After upload, the local file will be moved to either success_dir or fail_dir depending on the status of the ingest towards ingest-internal
        #   DELETE : After upload, the local file will be deleted if the upload succeeded
        #   NONE : After upload, nothing will be done to the local file (default option set if the value entered for upload_final_action does not exist)
        metrics_enabled: true
    ingestinternal:
        vitam_component: ingest-internal
        host: "ingest-internal.service.{{ consul_domain }}"
        port_service: 8100
        port_admin: 28100
        baseuri: "ingest-internal"
        https_enabled: false
        secret_platform: "true"
        # log_level: "DEBUG"
        metrics_enabled: true
    ihm_demo:
        vitam_component: "ihm-demo"
        host: "ihm-demo.service.{{ consul_domain }}"
        port_service: 8002
        port_admin: 28002
        baseurl: "/ihm-demo"
        static_content: "{{ vitam_defaults.folder.root_path }}/app/ihm-demo/v2"
        baseuri: "ihm-demo"
        https_enabled: false
        secret_platform: "false"
        # User session timeout in milliseconds (for shiro)
        session_timeout: 1800000
        secure_cookie: false
        # Specify here the realms you want to use for authentication in ihm-demo
        # You can set multiple realms, one per line
        # With multiple realms, the user will be able to choose between the allowed realms
        # Example: authentication_realms:
        #               - x509Realm
        #               - ldapRealm
        # Authorized values:
        # x509Realm: certificate
        # iniRealm: ini file
        # ldapRealm: ldap
```

(suite sur la page suivante)

(suite de la page précédente)

```
        authentication_realms:
            # - x509Realm
            - iniRealm
            # - ldapRealm
        # log_level: "DEBUG"
        metrics_enabled: true
    logbook:
        vitam_component: logbook
        host: "logbook.service.{{ consul_domain }}"
        port_service: 9002
        port_admin: 29002
        baseuri: "logbook"
        https_enabled: false
        secret_platform: "true"
        cluster_name: "{{ elasticsearch.data.cluster_name }}"
        # Temporization delay (in seconds) for recent logbook operation events.
        # Set it to a reasonable delay to cover max clock difference across servers + VM/GC pauses
        operationTraceabilityTemporizationDelay: 300
        # Temporization delay (in seconds) for recent logbook lifecycle events.
        # Set it to a reasonable delay to cover max clock difference across servers + VM/GC pauses
        lifecycleTraceabilityTemporizationDelay: 300
        # Max entries selected per (Unit or Object Group) LFC traceability operation
        lifecycleTraceabilityMaxEntries: 100000
        # log_level: "DEBUG"
        metrics_enabled: true
    metadata:
        vitam_component: metadata
        host: "metadata.service.{{ consul_domain }}"
        port_service: 8200
        port_admin: 28200
        baseuri: "metadata"
        https_enabled: false
        secret_platform: "true"
        cluster_name: "{{ elasticsearch.data.cluster_name }}"
        # log_level: "DEBUG"
        metrics_enabled: true
    processing:
        vitam_component: processing
        host: "processing.service.{{ consul_domain }}"
        port_service: 8203
        port_admin: 28203
        baseuri: "processing"
        https_enabled: false
        secret_platform: "true"
        # log_level: "DEBUG"
        metrics_enabled: true
    security_internal:
        vitam_component: security-internal
        host: "security-internal.service.{{ consul_domain }}"
        port_service: 8005
        port_admin: 28005
        baseuri: "security-internal"
        https_enabled: false
        secret_platform: "true"
```

(suite sur la page suivante)

(suite de la page précédente)

```
        # log_level: "DEBUG"
        metrics_enabled: true
    storageengine:
        vitam_component: storage
        host: "storage.service.{{ consul_domain }}"
        port_service: 9102
        port_admin: 29102
        baseuri: "storage-engine"
        https_enabled: false
        secret_platform: "true"
        storageTraceabilityOverlapDelay: 300
        restoreBulkSize: 1000
        # log_level: "DEBUG"
        metrics_enabled: true
    storageofferdefault:
        vitam_component: "offer"
        port_service: 9900
        port_admin: 29900
        baseuri: "storage-offer-default"
        https_enabled: false
        secret_platform: "true"
        # log_level: "DEBUG"
        metrics_enabled: true
    worker:
        vitam_component: worker
        port_service: 9104
        port_admin: 29104
        baseuri: "worker"
        https_enabled: false
        secret_platform: "true"
        # log_level: "DEBUG"
        metrics_enabled: true
    workspace:
        vitam_component: workspace
        host: "workspace.service.{{ consul_domain }}"
        port_service: 8201
        port_admin: 28201
        baseuri: "workspace"
        https_enabled: false
        secret_platform: "true"
        # log_level: "DEBUG"
        metrics_enabled: true
```

Note : Cas du composant ingest-external. Les directives `upload_dir`, `success_dir`, `fail_dir` et `upload_final_action` permettent de prendre en charge (ingest) des fichiers déposés dans `upload_dir` et appliquer une règle `upload_final_action` à l'issue du traitement (NONE, DELETE ou MOVE dans `success_dir` ou `fail_dir` selon le cas). Se référer au *DEX* pour de plus amples détails. Se référer au manuel de développement pour plus de détails sur l'appel à ce cas.

Avertissement : Selon les informations apportées par le métier, redéfinir les valeurs associées dans les directives `classificationList` et `classificationLevelOptional`. Cela permet de définir quels niveaux de protection du secret de la défense nationale supporte l'instance. Attention : une instance de niveau supérieur doit toujours supporter les niveaux inférieurs

- `environments/group_vars/all/cots_vars.yml`, comme suit :

```
---

consul:
    dns_port: 53

consul_remote_sites:
    # wan contains the wan addresses of the consul server instances of the external vitam sites
    # Exemple, if our local dc is dc1, we will need to set dc2 & dc3 wan conf:
    # - dc2:
    #   wan: ["10.10.10.10","1.1.1.1"]
    # - dc3:
    #   wan: ["10.10.10.11","1.1.1.1"]

elasticsearch:
    log:
        host: "elasticsearch-log.service.{{ consul_domain }}"
        port_http: "9201"
        port_tcp: "9301"
        groupe: "log"
        baseuri: "elasticsearch-log"
        cluster_name: "elasticsearch-log"
        https_enabled: false
        # default index template
        index_templates:
            default:
                shards: 1
                replica: 1
    data:
        host: "elasticsearch-data.service.{{ consul_domain }}"
        # default is 0.1 (10%) and should be quite enough in most cases
        #index_buffer_size_ratio: "0.15"
        port_http: "9200"
        port_tcp: "9300"
        groupe: "data"
        baseuri: "elasticsearch-data"
        cluster_name: "elasticsearch-data"
        https_enabled: false
        # default index template
        index_templates:
            default:
                shards: 10
                replica: 2

mongodb:
    mongos_port: 27017
    mongoc_port: 27018
    mongod_port: 27019
    mongo_authentication: "true"
    host: "mongos.service.{{ consul_domain }}"

logstash:
    host: "logstash.service.{{ consul_domain }}"
    user: logstash
    port: 10514
    rest_port: 20514
```

(suite sur la page suivante)

(suite de la page précédente)

```
56
57  # Curator units: days
58  curator:
59      log:
60          metrics:
61              close: 5
62              delete: 30
63          logstash:
64              close: 5
65              delete: 30
66          metricbeat:
67              close: 5
68              delete: 30
69          packetbeat:
70              close: 5
71              delete: 30
72
73  kibana:
74      header_value: "reporting"
75      log:
76          baseuri: "kibana_log"
77          api_call_timeout: 120
78          groupe: "log"
79          port: 5601
80          default_index_pattern: "logstash-vitam*"
81          # default shards & replica
82          shards: 5
83          replica: 1
84          # pour index logstash-*
85          metrics:
86              shards: 5
87              replica: 1
88          # pour index metrics-vitam-*
89          logs:
90              shards: 5
91              replica: 1
92          # pour index metricbeat-*
93          metricbeat:
94              shards: 5 # must be a factor of 30
95              replica: 1
96      data:
97          baseuri: "kibana_data"
98          # OMA : bugdette : api_call_timeout is used for retries ; should ceate a␣
    ↪separate variable rather than this one
99          api_call_timeout: 120
100         groupe: "data"
101         port: 5601
102         default_index_pattern: "logbookoperation_*"
103         # index template for .kibana
104         shards: 1
105         replica: 1
106
107 syslog:
108     # value can be syslog-ng or rsyslog
109     name: "rsyslog"
110
111 cerebro:
```

(suite sur la page suivante)

(suite de la page précédente)

```
    baseuri: "cerebro"
    port: 9000

siegfried:
    port: 19000

clamav:
    port: 3310
    # frequency freshclam for database update per day (from 0 to 24 - 24 meaning hourly check)
    db_update_periodicity: 1

mongo_express:
    baseuri: "mongo-express"

ldap_authentification:
    ldap_protocol: "ldap"
    ldap_server: "{% if groups['ldap']|length > 0 %}{{ groups['ldap']|first }}{% endif %}"
    ldap_port: "389"
    ldap_base: "dc=programmevitam,dc=fr"
    ldap_login: "cn=Manager,dc=programmevitam,dc=fr"
    uid_field: "uid"
    ldap_userDn_Template: "uid={0},ou=people,dc=programmevitam,dc=fr"
    ldap_group_request: "(&(objectClass=groupOfNames)(member={0}))"
    ldap_admin_group: "cn=admin,ou=groups,dc=programmevitam, dc=fr"
    ldap_user_group: "cn=user,ou=groups,dc=programmevitam, dc=fr"
    ldap_guest_group: "cn=guest,ou=groups,dc=programmevitam, dc=fr"
```

Note : installation multi-sites. Déclarer dans `consul_remote_sites` les datacenters Consul des autres site ; se référer à l'exemple fourni pour renseigner les informations.

- `environments/group_vars/all/jvm_vars.yml`, comme suit :

```
---

vitam:
    accessinternal:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    accessexternal:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    elastickibanainterceptor:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    batchreport:
          jvm_opts:
```

(suite sur la page suivante)

(suite de la page précédente)

```
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    ingestinternal:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    ingestexternal:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    metadata:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    ihm_demo:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    ihm_recette:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    logbook:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    workspace:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    processing:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    worker:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    storageengine:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    storageofferdefault:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
```

(suite sur la page suivante)

(suite de la page précédente)

```
            # java: ""
    functional_administration:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    security_internal:
        jvm_opts:
            # memory: "-Xms512m -Xmx512m"
            # gc: ""
            # java: ""
    library:
        jvm_opts:
            memory: "-Xms32m -Xmx128m"
            # gc: ""
            # java: ""
```

Note : Cette configuration est appliquée à la solution logicielle *VITAM* ; il est possible de créer un tuning par « groupe » défini dans ansible.

4.2.5 Procédure de première installation

4.2.5.1 Déploiement

4.2.5.1.1 Cas particulier : utilisation de ClamAv en environnement Debian

Dans le cas de l'installation en environnement Debian, la base de donnée n'est pas intégrée avec l'installation de ClamAv, C'est la commande `freshclam` qui en assure la charge. Si vous n'êtes pas connecté à internet, la base de données doit s'installer manuellement. Les liens suivants indiquent la procédure à suivre : Installation ClamAv [13] et Section Virus Database [14]

4.2.5.1.2 Fichier de mot de passe

Par défaut, le mot de passe des « vault » sera demandé à chaque exécution d'ansible. 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 (dans ce cas, changez l'option `--ask-vault-pass` des invocations ansible par l'option `--vault-password-file=VAULT_PASSWORD_FILES`.

4.2.5.1.3 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 :

[13] https://www.clamav.net/documents/installing-clamav
[14] https://www.clamav.net/downloads

```
#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 :

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

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

4.2.5.1.4 Génération des hostvars

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

4.2.5.1.4.1 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.

4.2.5.1.4.2 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.yml`

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.

4.2.5.1.4.3 Vérification de la génération des hostvars

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

Prudence : Cas d'une installation multi-sites. Sur site secondaire, s'assurer que, pour les machines hébergeant les offres, la directive `ip_wan` a bien été déclarée (l'ajouter manuellement, le cas échéant), pour que site le site « primaire » sache les contacter via une IP particulière. Par défaut, c'est l'IP de service qui sera utilisée.

4.2.5.1.5 Passage des identifiants des référentiels en mode esclave

La génération des identifiants des référentiels est géré par Vitam quand il fonctionne en mode maître.

Par exemple :

- Préfixé par PR- pour les profils
- Préfixé par IC- pour les contrats d'entrée
- Préfixé par AC- pour les contrats d'accès

Si vous souhaitez gérer vous-même les identifiants sur un service référentiel, il faut qu'il soit en mode esclave. Par défaut tous les services référentiels de Vitam fonctionnent en mode maître. Pour désactiver le mode maître de Vitam, il faut modifier le fichier ansible `deployment/ansible-vitam/roles/vitam/templates/functional-administration/functional-administration.conf.j2`. Un exemple de ce fichier se trouve dans la Documentation d'exploitation au chapitre « Exploitation des composants de la solution logicielle VITAM ».

```
# ExternalId configuration

listEnableExternalIdentifiers:
 0:
   - INGEST_CONTRACT
   - ACCESS_CONTRACT
 1:
   - INGEST_CONTRACT
```

(suite sur la page suivante)

(suite de la page précédente)

```
  - ACCESS_CONTRACT
  - PROFILE
  - SECURITY_PROFILE
  - CONTEXT
```

Depuis la version 1.0.4, la configuration par défaut de Vitam autorise des identifiants externes (ceux qui sont dans le fichier json importé).

- pour le tenant 0 pour les référentiels : contrat d'entrée et contrat d'accès.
- pour le tenant 1 pour les référentiels : contrat d'entrée, contrat d'accès, profil, profil de sécurité et contexte.

La liste des choix possibles, pour chaque tenant, est :

- INGEST_CONTRACT : contrats d'entrée
- ACCESS_CONTRACT : contrats d'accès
- PROFILE : profils SEDA
- SECURITY_PROFILE : profils de sécurité (utile seulement sur le tenant d'administration)
- CONTEXT : contextes applicatifs (utile seulement sur le tenant d'administration)
- ARCHIVEUNITPROFILE : profils d'unités archivistiques

4.2.5.1.6 Durées minimales permettant de contrôler les valeurs saisies

Afin de se prémunir contre une alimentation du référentiel des règles de gestion avec des durées trop courtes susceptibles de déclencher des actions indésirables sur la plate-forme (ex. éliminations) – que cette tentative soit intentionnelle ou non –, la solution logicielle *VITAM* vérifie que l'association de la durée et de l'unité de mesure saisies pour chaque champ est supérieure ou égale à une durée minimale définie lors du paramétrage de la plate-forme, dans un fichier de configuration.

Pour mettre en place le comportement attendu par le métier, il faut modifier le contenu de la directive `listMinimumRuleDuration` dans le fichier ansible `deployment/ansible-vitam/roles/vitam/templates/functional-administration/functional-administration.conf.j2`.

Exemple :

```
listMinimumRuleDuration:
  2:
    AppraisalRule : 1 year
    DisseminationRule : 10 year

  3:
    AppraisaleRule : 5 year
    StorageRule : 5 year
    ReuseRule : 2 year
```

Par tenant, les directives possibles sont :

- AppraisalRule
- DisseminationRule
- StorageRule
- ReuseRule
- AccessRule (valeur par défaut : 0 year)
- ClassificationRule

Les valeurs associées sont une durée au format <nombre> <unité en angais, au singulier>

Exemples :

```
6 month
1 year
5 year
```

Pour plus de détails, se rapporter à la documentation métier « Règles de gestion ».

4.2.5.1.7 Déploiement

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

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
```

Note : Une confirmation est demandée pour lancer ce script. Il est possible de rajouter le paramètre `-e confirmation=yes` pour bypasser cette demande de confirmation (cas d'un déploiement automatisé).

Prudence : Dans le cas où l'installateur souhaite utiliser un *repository* de binaires qu'il gère par lui-même, il est fortement recommandé de rajouter `--skip-tags "enable_vitam_repo"` à la commande `ansible-playbook` ; dans ce cas, le comportement de `yum` n'est pas impacté par la solution de déploiement.

4.2.6 Elements extras de l'installation

Prudence : Les élements décrits dans cette section sont des élements « extras » ; il ne sont pas officiellement supportés, et ne sont par conséquence pas inclus dans l'installation de base. Cependant, ils peuvent s'avérer utile, notamment pour les installation sur des environnements hors production.

Prudence : Dans le cas où l'installateur souhaite utiliser un *repository* de binaires qu'il gère par lui-même, il est fortement recommandé de rajouter `--skip-tags "enable_vitam_repo"` à la commande `ansible-playbook` ; dans ce cas, le comportement de `yum` n'est pas impacté par la solution de déploiement.

4.2.6.1 Configuration des extra

Le fichier `environments /group_vars/all/extra_vars.yml` contient la configuration des extra :

```
1 ---
2
3 vitam:
4     ihm_recette:
5         vitam_component: ihm-recette
6         host: "ihm-recette.service.{{consul_domain}}"
7         port_service: 8445
```

(suite sur la page suivante)

(suite de la page précédente)

```
        port_admin: 28204
        baseurl: /ihm-recette
        static_content: "{{ vitam_defaults.folder.root_path }}/app/ihm-recette"
        baseuri: "ihm-recette"
        secure_mode:
            - authc
        https_enabled: true
        secret_platform: "false"
        cluster_name: "{{ elasticsearch.data.cluster_name }}"
        session_timeout: 1800000
        secure_cookie: true
        use_proxy_to_clone_tests: "yes"
        metrics_enabled: true
    library:
        vitam_component: library
        host: "library.service.{{consul_domain}}"
        port_service: 8090
        port_admin: 28090
        baseuri: "doc"
        https_enabled: false
        secret_platform: "false"
        metrics_enabled: false

# Period units in seconds
metricbeat:
    system:
        period: 10
    mongodb:
        period: 10
    elasticsearch:
        period: 10

docker_opts:
    registry_httponly: yes
    vitam_docker_tag: latest
```

Avertissement : A modifier selon le besoin avant de lancer le playbook ! Le composant ihm-recette, s'il est déployé en « https », doit avoir un paramétrage différent dans `environments/group_vars/all/extra_vars.yml` sur le paramètre `secure_cookie` selon qu'il est attaqué en direct ou derrière un proxy https (`secure_cookie: true`) ou un proxy http (`secure_cookie: false`). Par défaut, la variable est à `true`. Le symptôme observé, en cas de problème, est une bonne authentification, mais l'impossibilité de sélectionner un tenant.

Note : La section `metricbeat` permet de configurer la périodicité d'envoi des informations collectées. Selon l'espace disponible sur le *cluster* Elasticsearch de log et la taille de l'environnement *VITAM* (en particulier, le nombre de machines), il peut être nécessaire d'allonger cette périodicité (en secondes).

Le fichier `environments /group_vars/all/all/vault-extra.yml` contient les secrets supplémentaires des extra ; 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`.

4.2.6.2 Déploiement des extra

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

4.2.6.2.1 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
```

Prudence : avant de jouer le *playbook*, ne pas oublier, selon le contexte d'usage, de positionner correctement la variable `secure_cookie` décrite plus haut.

4.2.6.2.2 extra complet

Ce playbook permet d'installer :

- des éléments de monitoring système
- 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 la documentation du projet
- le composant *VITAM* ihm-recette (utilise si configuré des dépôts de jeux de tests)
- un reverse proxy, afin de fournir une page d'accueil pour les environnements de test
- l'outillage de tests de performance

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

CHAPITRE 5

Procédures de mise à jour

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.

Prudence : Seule la mise à jour depuis la version « 1.0.3 » est supportée dans cette version de la solution logicielle VITAM. Se référer à upgrade_r6_r7 pour plus de détails.

5.1 Reconfiguration

5.1.1 Cas d'une modification du nombre de tenants

Modifier dans le fichier d'inventaire la directive `vitam_tenant_ids`

Exemple :

```
vitam_tenant_ids=[0,1,2]
```

A l'issue, il faut lancer le playbook de déploiement de VITAM (et, si déployé, les extra) avec l'option supplémentaire `--tags update_vitam_configuration`.

Exemple :

```
ansible-playbook -i environments/hosts.deployment ansible-vitam/vitam.yml --ask-vault-
↪pass --tags update_vitam_configuration
ansible-playbook -i environments/hosts.deployment ansible-vitam-extra/extra.yml --ask-
↪vault-pass --tags update_vitam_configuration
```

5.1.2 Cas d'une modification des paramètres JVM

Se référer à *Tuning JVM* (page 30)

Pour les partitions sur lesquelles une modification des paramètres JVM est nécessaire, il faut modifier les « hostvars » associées.

A l'issue, il faut lancer le playbook de déploiement de VITAM (et, si déployé, les extra) avec l'option supplémentaire `--tags update_jvmoptions_vitam`.

Exemple :

```
ansible-playbook -i environments/hosts.deployment ansible-vitam/vitam.yml --ask-vault-
↪pass --tags update_jvmoptions_vitam
ansible-playbook -i environments/hosts.deployment ansible-vitam-extra/extra.yml --ask-
↪vault-pass --tags update_jvmoptions_vitam
```

Prudence : Limitation technique à ce jour ; il n'est pas possible de définir des variables JVM différentes pour des composants colocalisés sur une même partition.

CHAPITRE 6

Post installation

6.1 Validation du déploiement

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

6.1.1 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`).

6.1.2 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`

6.1.3 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.

6.1.4 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 [15]) 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.

6.2 Sauvegarde des éléments d'installation

Après installation, il est fortement recommandé de sauvegarder les élements de configuration de l'installation (i.e. le contenu du répertoire `déploiement/environnements`) ; ces éléments seront à réutiliser pour les mises à jour futures.

Astuce : Une bonne pratique consiste à gérer ces fichiers dans un gestionnaire de version (ex : git)

Prudence : Si vous avez modifié des fichiers internes aux rôles, ils devront également être sauvegardés.

6.3 Troubleshooting

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

6.3.1 Erreur au chargement des tableaux de bord Kibana

Dans le cas de machines petitement taillées, il peut arriver que, durant le déploiement, la tâche `Wait for the kibana port port to be opened` prenne plus de temps que le *timeout* défini (`vitam_defaults.services.start_timeout`). Pour fixer cela, il suffit de relancer le déploiement.

[15] http://www.nationalarchives.gov.uk/aboutapps/pronom/droid-signature-files.htm

6.4 Retour d'expérience / cas rencontrés

6.4.1 Crash rsyslog, code killed, signal : BUS

Il a été remarqué chez un partenaire du projet Vitam, que rsyslog se faisait killer peu après son démarrage par le signal SIGBUS. Il s'agit très probablement d'un bug rsyslog <= 8.24 https://github.com/rsyslog/rsyslog/issues/1404

Pour fixer ce problème, il est possible d'upgrader rsyslog sur une version plus à jour en suivant cette documentation :

- Centos [16]
- Debian [17]

6.4.2 Mongo-express ne se connecte pas à la base de données associée

Si mongoDB a été redémarré, il faut également redémarrer mongo-express.

6.4.3 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"
```

(suite sur la page suivante)

https://www.rsyslog.com/rhelcentos-rpms/
https://www.rsyslog.com/debian-repository/

(suite de la page précédente)

```
                                }
                        }
                ]
}
```

Sur tous ces sujets, Cf. la documentation officielle [18].

6.4.4 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 [19]. 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.

6.4.5 MongoDB semble lent

Pour analyser la performance d'un cluster MongoDB, ce dernier fournit quelques outils permettant de faire une première analyse du comportement : mongostat [20] et mongotop [21] .

Dans le cas de VITAM, le cluster MongoDB comporte plusieurs shards. Dans ce cas, l'usage de ces deux commandes peut se faire :

- soit sur le cluster au global (en pointant sur les noeuds mongos) : cela permet d'analyser le comportement global du cluster au niveau de ses points d'entrées ;

```
mongostat --host <ip_service> --port 27017 --username vitamdb-admin --
↪password <password ; défaut : azerty> --authenticationDatabase admin
mongotop --host <ip_service> --port 27017 --username vitamdb-admin --
↪password <password ; défaut : azerty> --authenticationDatabase admin
```

- soit directement sur les noeuds de stockage (mongod) : cela donne des résultats plus fins, et permet notamment de séparer l'analyse sur les noeuds primaires & secondaires d'un même replicaset.

```
mongotop --host <ip_service> --port 27019 --username vitamdb-localadmin --
↪password <password ; défaut : qwerty> --authenticationDatabase admin
mongostat --host <ip_service> --port 27019 --username vitamdb-localadmin -
↪-password <password ; défaut : qwerty> --authenticationDatabase admin
```

D'autres outils sont disponibles directement dans le client mongo, notamment pour troubleshooter les problèmes dûs à la réplication [22] :

```
mongo --host <ip_service> --port 27019 --username vitamdb-localadmin --password
↪<password ; défaut : qwerty> --authenticationDatabase admin
> rs.printSlaveReplicationInfo()
> rs.printReplicationInfo()
> db.runCommand( { serverStatus: 1 } )
```

D'autres commandes plus complètes existent et permettent d'avoir plus d'informations, mais leur analyse est plus complexe :

https://www.elastic.co/guide/en/elasticsearch/reference/current/cluster-reroute.html
https://www.elastic.co/guide/en/elasticsearch/reference/1.4/states.html
https://docs.mongodb.com/manual/reference/program/mongostat/
https://docs.mongodb.com/manual/reference/program/mongotop/
https://docs.mongodb.com/manual/tutorial/troubleshoot-replica-sets

```
# returns a variety of storage statistics for a given collection
> use metadata
> db.stats()
> db.runCommand( { collStats: "Unit" } )
```

Enfin, un outil est disponible en standard afin de mesurer des performances des lecture/écritures avec des patterns proches de ceux utilisés par la base de données (mongoperf[23]) :

```
echo "{nThreads:16,fileSizeMB:10000,r:true,w:true}" | mongoperf
```

6.4.6 Les shards de MongoDB semblent mal équilibrés

Normalement, un processus interne à MongoDB (le `balancer`) s'occupe de déplacer les données entre les shards (par `chunk`) pour équilibrer la taille de ces derniers. Les commandes suivantes (à exécuter dans un shell mongo sur une instance mongos - attention, ces commandes ne fonctionnent pas directement sur les instances mongod) permettent de s'assurer du bon fonctionnement de ce processus :

- `sh.status()` : donne le status du sharding pour le cluster complet ; c'est un bon premier point d'entrée pour connaître l'état du balancer.
- `use <dbname>`, puis `db.<collection>.getShardDistribution()`, en indiquant le bon nom de base de données (ex : `metadata`) et de collection (ex : `Unit`) : donne les informations de répartition des chunks dans les différents shards pour cette collection.

6.4.7 L'importation initiale (profil de sécurité, certificats) retourne une erreur

Les playbooks d'initialisation importent des éléments d'administration du système (profils de sécurité, certificats) à travers des APIs de la solution VITAM. Cette importation peut être en échec, par exemple à l'étape `TASK [init_contexts_and_security_profiles : Import admin security profile to functionnal-admin]`, avec une erreur de type 400. Ce type d'erreur peut avoir plusieurs causes, et survient notamment lors de redéploiements après une première tentative non réussie de déploiement ; même si la cause de l'échec initial est résolue, le système peut se trouver dans un état instable. Dans ce cas, un déploiement complet sur environnement vide est nécessaire pour revenir à un état propre.

Une autre cause possible ici est une incohérence entre l'inventaire, qui décrit notamment les offres de stockage liées aux composants offer, et le paramétrage `vitam_strategy` porté par le fichier `offers_opts.yml`. Si une offre indiquée dans la stratégie n'existe nulle part dans l'inventaire, le déploiement sera en erreur. Dans ce cas, il faut remettre en cohérence ces paramètres et refaire un déploiement complet sur environnement vide.

6.4.8 Problème d'ingest et/ou d'access

Si vous repérez un message de ce type dans les log *VITAM* :

```
fr.gouv.vitam.common.security.filter.AuthorizationWrapper.
↪checkTimestamp(AuthorizationWrapper.java:117) : [vitam-env-int8-app-04.vitam-
↪env:storage:239079175] Timestamp check failed
```

Il faut vérifier / corriger l'heure des machines hébergeant la solution logicielle *VITAM* ; un *delta* de temps supérieur à 10s a été détecté entre les machines.

[23] https://docs.mongodb.com/manual/reference/program/mongoperf/

CHAPITRE 7

Annexes

7.1 Vue d'ensemble de la gestion des certificats

7.1.1 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).

7.1.2 Vue d'ensemble de la gestion des certificats

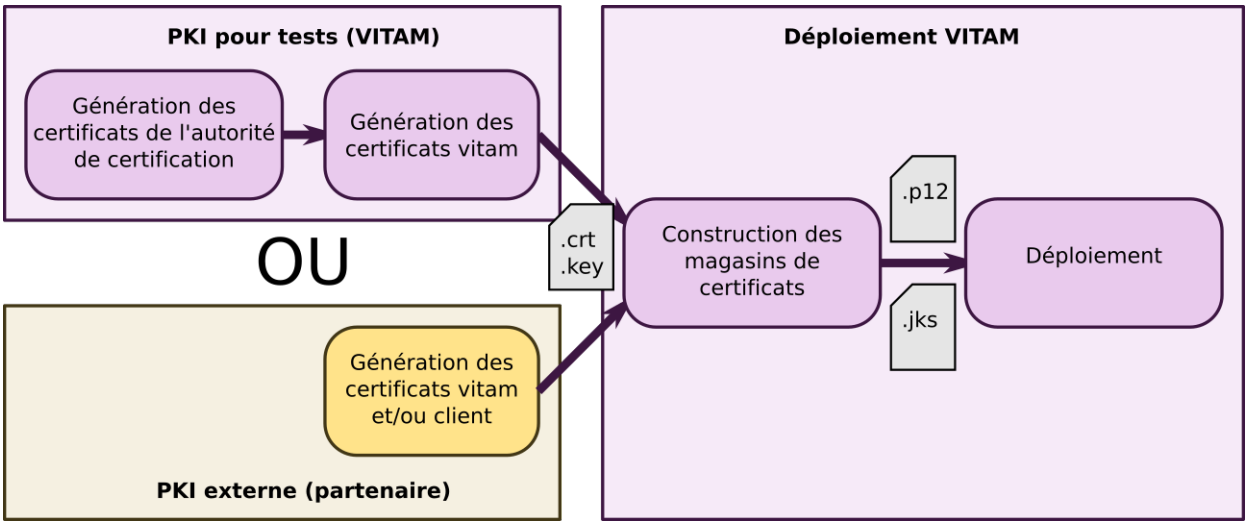

Fig. 1 – Vue d'ensemble de la gestion des certificats au déploiement

7.1.3 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 60)

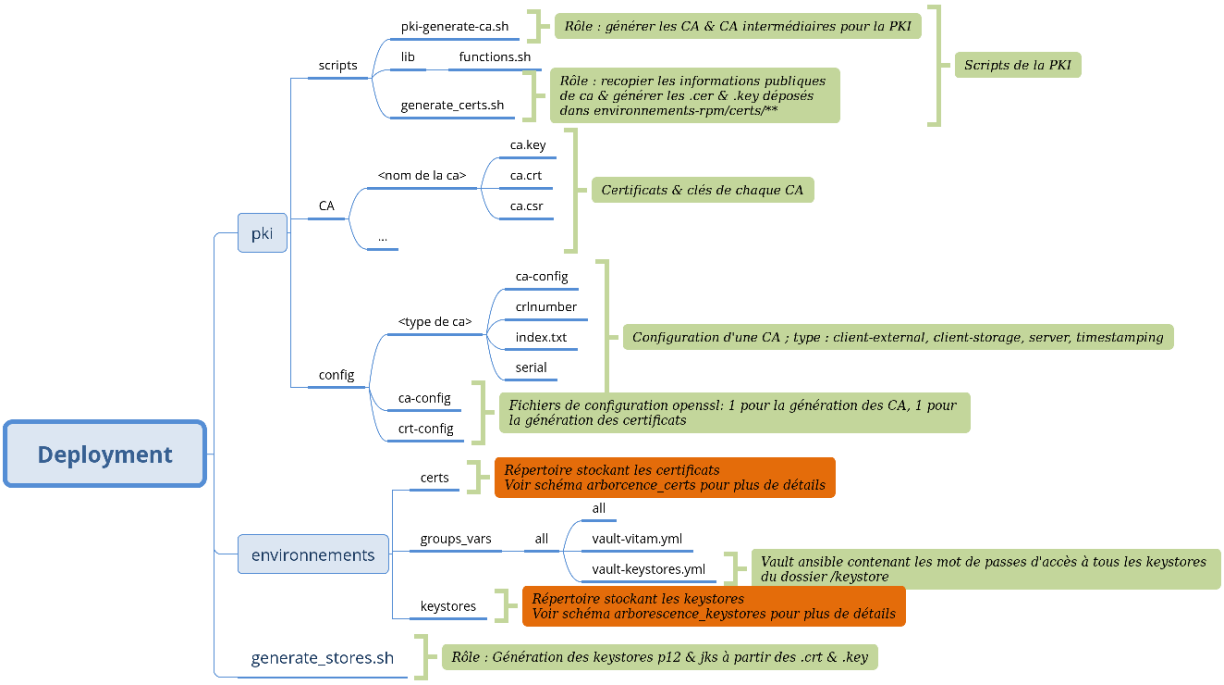

Fig. 2 – Vue l'arborescence de la PKI Vitam

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

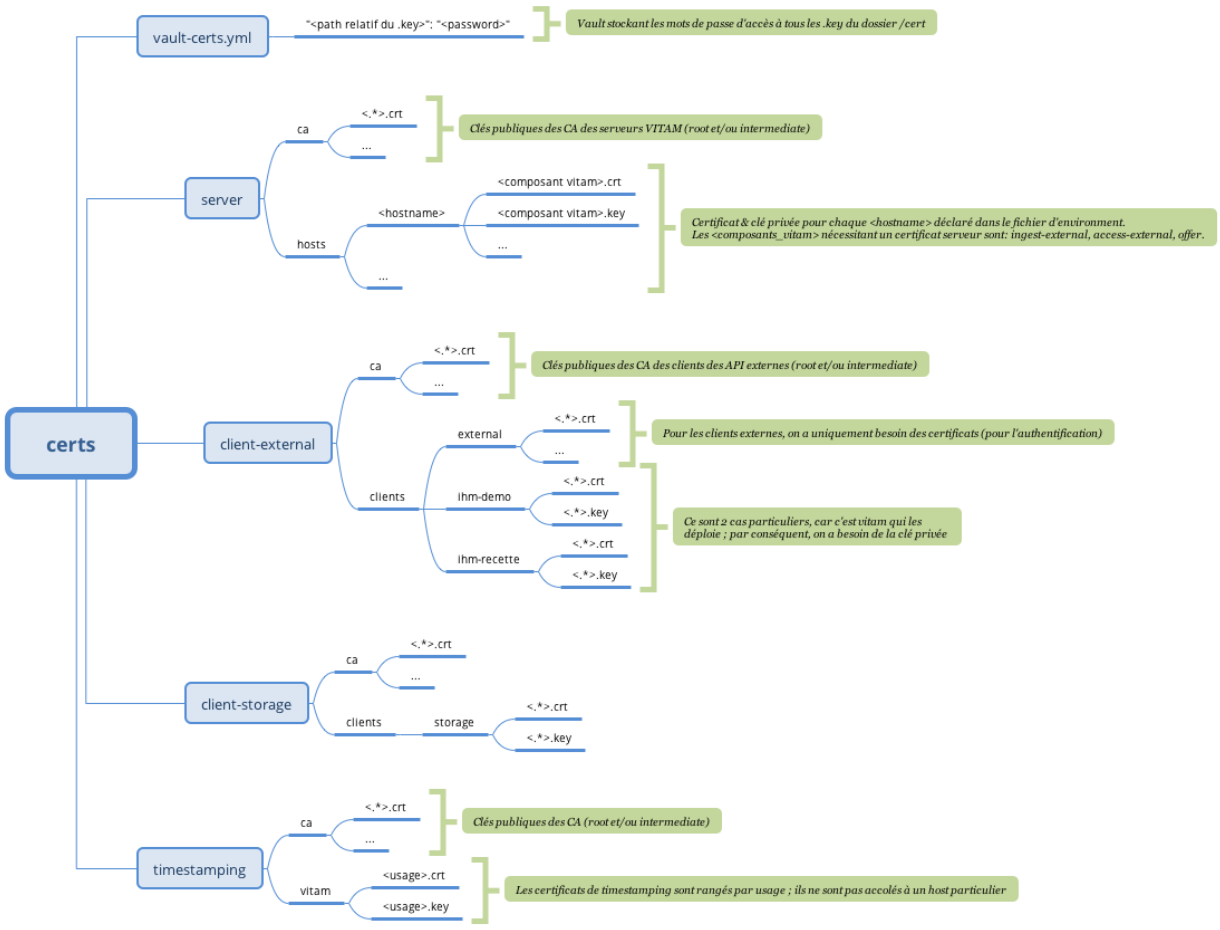

Fig. 3 – Vue détaillée de l'arborescence des certificats

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

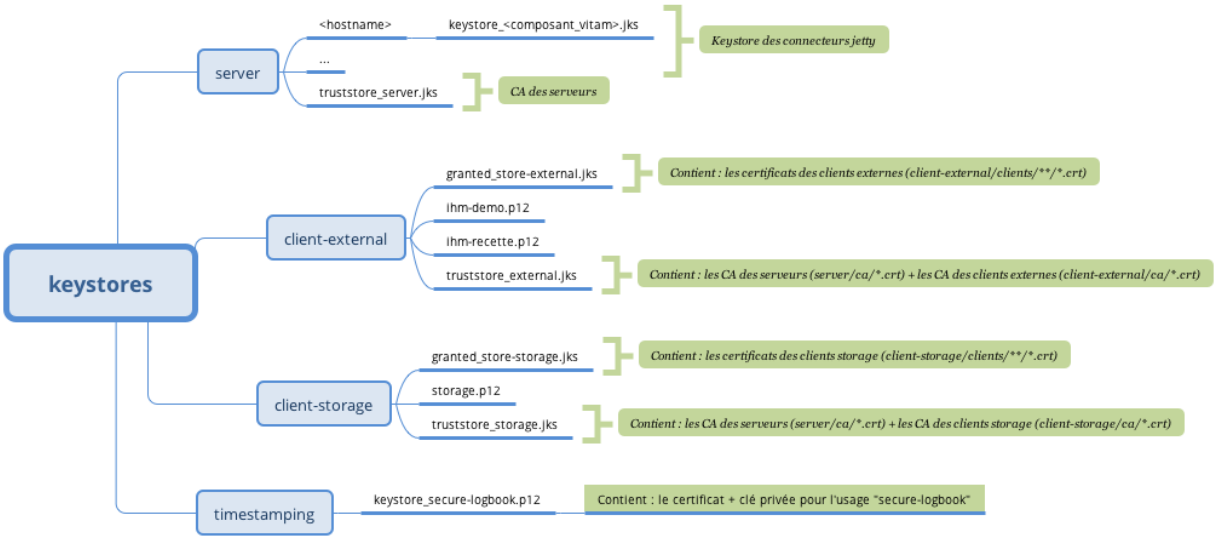

Fig. 4 – Vue détaillée de l'arborescence des keystores

7.1.6 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 *l'image ci-dessus* (page 59)).

7.2 Cycle de vie des certificats

Le tableau ci-dessous indique le mode de fonctionnement actuel pour les différents certificats et CA. Précisions :

- Les « procédures par défaut » liées au cycle de vie des certificats dans la présente version de la solution VITAM peuvent être résumées ainsi :
  - Création : génération par PKI partenaire + copie dans répertoires de déploiement + script `generate_stores.sh` + déploiement ansible

  - Suppression : suppression dans répertoires de déploiement + script `generate_stores.sh` + déploiement ansible
  - Renouvellement : regénération par PKI partenaire + suppression / remplacement dans répertoires de déploiement + script `generate_stores.sh` + redéploiement ansible
- Il n'y a pas de contrainte au niveau des CA utilisées (une CA unique pour tous les usages VITAM ou plusieurs CA séparées – cf. *DAT*). On appelle ici :
  - « PKI partenaire » : PKI / CA utilisées pour le déploiement et l'exploitation de la solution VITAM par le partenaire.
  - « PKI distante » : PKI / CA utilisées pour l'usage des frontaux en communication avec le back office VITAM.

Classe	Type	Usages	Origine	Création	Suppression	Renouvellement
Interne	CA	ingest & access	PKI partenaire	*proc. par défaut*	*proc. par défaut*	*proc. par défaut*
		offer	PKI partenaire	*proc. par défaut*	*proc. par défaut*	*proc. par défaut*
	Certif	Horodatage	PKI partenaire	*proc. par défaut*	*proc. par défaut*	*proc. par défaut*
		Storage (swift)	Offre de stockage	*proc. par défaut*	*proc. par défaut*	*proc. par défaut*
		ingest	PKI partenaire	*proc. par défaut*	*proc. par défaut*	*proc. par défaut*
		access	PKI partenaire	*proc. par défaut*	*proc. par défaut*	*proc. par défaut*
		offer	PKI partenaire	*proc. par défaut*	*proc. par défaut*	*proc. par défaut*
		Timestamp	PKI partenaire	*proc. par défaut*	*proc. par défaut*	*proc. par défaut*
IHM demo	CA	ihm-demo	PKI partenaire	*proc. par défaut*	*proc. par défaut*	*proc. par défaut*
	Certif	ihm-demo	PKI partenaire	*proc. par défaut*	*proc. par défaut*	*proc. par défaut*
SIA	CA	Appel API	PKI distante	proc. par défaut (PKI distante)	*proc. par défaut*	proc. par défaut (PKI distante) + recharger Certifs
	Certif	Appel API	PKI distante	Génération + copie répertoire + deploy (par la suite, appel API d'insertion)	Suppression Mongo	Suppression Mongo + API d'insertion
Personae	Certif	Appel API	PKI distante	API ajout	API suppression	API suppression + API ajout

Remarques :

- Lors d'un renouvellement de CA SIA, il faut s'assurer que les certificats qui y correspondaient sont retirés de MongoDB et que les nouveaux certificats sont ajoutés par le biais de l'API dédiée.
- Lors de toute suppression ou remplacement de certificats SIA, s'assurer que la suppression / remplacement des contextes associés soit également réalisée.
- L'expiration des certificats n'est pas automatiquement prise en charge par la solution VITAM (pas de notification en fin de vie, pas de renouvellement automatique). Pour la plupart des usages, un certificat expiré est proprement rejeté et la connexion ne se fera pas ; les seules exceptions sont les certificats Personae, pour lesquels la validation de l'arborescence CA et des dates est à charge du front office en interface avec VITAM.

7.3 Ansible & ssh

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

7.3.1 Authentification du compte utilisateur utilisé pour la connexion SSH

Pour le login du compte utilisateur, voir la section *Informations « plate-forme »* (page 11).

7.3.1.1 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.

7.3.1.2 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

7.3.1.3 Par clé SSH sans passphrase

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

7.3.2 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.

7.3.3 Elevation de privilèges

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

7.3.3.1 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`

7.3.3.2 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

7.3.3.3 Par sudo sans mot de passe

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

7.3.3.4 Déjà Root

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

Table des figures

Liste des tableaux

Index