|  |  |
| --- | --- |
|  | Direction des Systèmes d’information  Domaine Ingénierie / Pôle Architecture & Expertise Système |
|  |
| 95 rue de MAUBEUGE  75010 PARIS |
|  |
|  |
|  | |
| RegLes d’ingenierie & EXPLOITATION Ansible Tower | |
| barre_jaune | |

Regles d’ingenieries & EXPLOITATION Ansible Tower

* date : 06/05/2022
* version du document : 0.5
* auteurs : Philippe NAKACHE, ANTOINE VILLETTE, SOFIANE MEDJKOUNE

Historique des versions

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Date | Description | Auteur |
| 0.1 | 23/09/2021 | Version initiale | Erwin Poul |
| 0.2 | 21/10/2021 | Présentation à équipe AES | Erwin Poul |
| 0.3 | 26/10/2021 | Modifications après relecture AES | Erwin Poul |
| 0.4 | 12/01/2021 | Modification des infos liés à PAM et au workflow | Philippe Nakache |
| 0.5 | 06/05/2022 | Relecture et validation | Ludwig Mainguenaud |

Table des matières

* [1 Règle d’ingénierie 5](#_Toc86170839)

[1.1 Responsabilité 5](#_Toc86170840)

[1.2 Contextualisation des règles d’ingénierie 6](#_Toc86170841)

[1.2.1 Contexte offres du cadre normatif 6](#_Toc86170842)

[1.2.2 Contexte Area 6](#_Toc86170843)

[1.3 Dérogation 6](#_Toc86170844)

[1.4 Exceptions 7](#_Toc86170845)

* [2 Disponibilité de la brique technique 8](#_Toc86170846)

[2.1 Hébergement 8](#_Toc86170847)

[2.2 Area 8](#_Toc86170848)

[2.3 Offre 8](#_Toc86170849)

* [3 Version de la brique technique 9](#_Toc86170850)
* [4 Fonctionnement de la brique technique 10](#_Toc86170851)

[4.1 Livraison applicative 10](#_Toc86170852)

[4.2 Déploiement d’un middleware 11](#_Toc86170853)

* [5 Caractéristiques de la brique technique 12](#_Toc86170854)

[5.1 Ansible 12](#_Toc86170855)

[5.2 Ansible Tower 12](#_Toc86170856)

[5.3 Glossaire 13](#_Toc86170857)

[5.4 Outillage tiers 14](#_Toc86170858)

[5.4.1 GitLab 14](#_Toc86170859)

[5.4.2 VMware 14](#_Toc86170860)

[5.4.3 Artifactory 15](#_Toc86170861)

[5.4.4 Ansible Vault 15](#_Toc86170862)

[5.4.5 PAM (Cyberark) 15](#_Toc86170863)

* [6 Prise en main de la brique technique 16](#_Toc86170864)

[6.1 Prérequis 16](#_Toc86170865)

[6.2 Ecriture d’un Playbook Ansible 16](#_Toc86170866)

[6.2.1 Accès à GitLab 16](#_Toc86170867)

[6.2.2 Comment se connecte Tower aux serveurs pour exécuter le code Ansible ? 17](#_Toc86170868)

[6.2.3 Description de l’application (role des serveurs) 19](#_Toc86170869)

[6.2.4 Gestion des variables 19](#_Toc86170870)

[6.2.5 Liste des Roles Ansible mis à disposition 20](#_Toc86170871)

[6.2.6 Consommation d’un Role Ansible existant 20](#_Toc86170872)

[6.2.7 Modules interdits dans les Playbook 21](#_Toc86170873)

[6.2.8 Description du git flow 22](#_Toc86170874)

[6.3 Ecriture d’un Role Ansible 23](#_Toc86170875)

[6.3.1 Standardisation et réutilisation des Roles 23](#_Toc86170876)

[6.3.2 Variables internes VS externes 23](#_Toc86170877)

[6.3.3 Proscrire tout type de *Hardcoding* 24](#_Toc86170878)

[6.3.4 Nommage du Role et de ces variables 24](#_Toc86170879)

[6.3.5 Règles d'écriture 25](#_Toc86170880)

[6.3.6 Prise en main d'un nouveau Role Ansible 28](#_Toc86170881)

[6.4 Conformité 29](#_Toc86170882)

[6.5 Exploitation 29](#_Toc86170883)

* [7 Utilisation de la brique technique 30](#_Toc86170884)

[7.1 Exemple de projet 30](#_Toc86170885)

[7.2 Connexion à Ansible Tower 30](#_Toc86170886)

[7.3 Projet 31](#_Toc86170887)

[7.4 Inventaires 32](#_Toc86170888)

[7.5 Job Templates 33](#_Toc86170889)

|  |  |  |
| --- | --- | --- |
|  |  |  |
| GRDF – Société Anonyme au capital de 1 800 745 000 euros – Siège social : 6 rue Condorcet – 75009 Paris – RCS : PARIS 444 786 511 |
|  |
| pieddepage_grdf_cahier.png | | |

# Règle d’ingénierie

Les règles d’ingénierie sont l’ensemble des règles qu’impose la DSI de GRDF à une brique technique. Ces règles ont pour but d’homogénéiser et d’industrialiser la brique technique afin de :

* Améliorer l’exploitabilité et donc la qualité de service fournie par la brique.
* Automatiser les gestes liés à la brique afin de diminuer les coûts et les délais.
* Diminuer la surface d’attaque des risques Cyber.

|  |  |
| --- | --- |
|  | Les règles d’ingénierie de la brique technique sont appliquées. |

 Toute règle ne figurant pas dans ce document et n’étant pas un comportement normal de la brique technique tel qu’on pourrait la trouver chez l’éditeur sans modification de GRDF est considérée comme une spécificité.

## Responsabilité

Les règles d’ingénierie sont écrites par l’expertise de la brique et validées par l’exploitant de la brique. Elles sont aussi validées par les architectes infrastructures, ceux-ci sont garants de la cohérence transverse des briques.

Les responsabilités pour la brique technique sont :

|  |  |
| --- | --- |
| Rôle | Entité responsable |
| Expertise | DSI/ING/AES |
| Exploitant | DSI/EXP/POI |

|  |  |
| --- | --- |
|  | La brique technique a une organisation décrite et validée par la DSI de GRDF. |

## Contextualisation des règles d’ingénierie

Les règles d’ingénierie peuvent varier en fonction de plusieurs critères. Dans ce cas, le document décrit, pour chaque contexte, les règles applicables. Les règles d’ingénierie pour la brique Ansible Tower sont contextualisées en fonction des critères suivants :

* Offre du cadre normatif
* Area
* Statut

### Contexte offres du cadre normatif

Le cadre normatif impose à la brique technique d’être contextualisée en fonction des offres. Celles-ci permettent de mieux répondre aux besoins de la DSI GRDF en donnant des niveaux / fonctionnalités ainsi que des coûts différenciés en fonction des offres.

Les 4 offres du cadre normatif sont :

* PRODUCTION
* HOMOLOGATION
* CONCEPTION
* INNOVATION

### Contexte Area

L’area est la segmentation réseau de 1er niveau définie dans le cadre d’EDGAR. Elle a pour but principal de définir des périmètres de sécurité macro, notamment en fonction de leur exposition (Interne / Externe) ou de leur criticité (SIE / Administration).

Les 4 areas sont

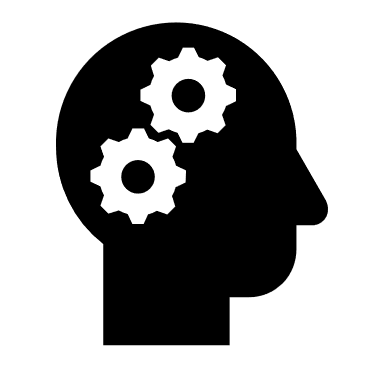
* ADMINISTRATION
* INTERNE
* EXTERNE
* SIE

## Dérogation

Des dérogations aux règles d’ingénierie peuvent être faites ; néanmoins toute dérogation entraine mécaniquement une situation non maitrisée et spécifique qui nécessite un arbitrage vis-à-vis :

* Des coûts induits pour traiter la spécificité (documentation, licence, charges humaines, etc.)
* D’une revalidation du niveau d’exploitation par rapport aux offres standard.

|  |  |
| --- | --- |
|  | L’ensemble des dérogations doivent être référencées dans le document de dérogations de la brique technique |

Plusieurs dérogations tracées peuvent amener à intégrer la spécificité aux règles d’ingénierie, dans ce cas les dérogations sont supprimées du fichier de suivi des dérogations.

La dérogation doit être validée par les responsables de domaine de l’expertise et de l’exploitant (voir Chapitre Annexe / Processus de dérogation).

## Exceptions

Néant

# Disponibilité de la brique technique

La disponibilité de la brique technique permet de présenter aux utilisateurs de la brique technique les zones validées pour le déploiement et l’exploitation de la brique technique.

|  |  |
| --- | --- |
|  | L’utilisation de la brique technique en dehors des zones de disponibilités validées est interdite. |

## Hébergement

Les hébergements autorisés pour l’utilisation et l’exploitation de la brique technique Ansible Tower sont :

|  |  |
| --- | --- |
| Hébergement | Disponibilité |
| Hébergement S4 | Coche (Avertissement) |
| Hébergement EDGAR | Coche |
| Hébergement hors DC | Fermer |
| Hébergement Cloud | Fermer |

La brique technique Ansible Tower est disponible sur l’hébergement S4 **uniquement** dans le cadre de la transformation des VMs (migration) vers EDGAR

|  |  |
| --- | --- |
|  | Il n’est pas possible d’utiliser la brique Ansible Tower en dehors des zones d’hébergements autorisées. |

## Area

Les areas autorisées pour l’utilisation et l’exploitation de la brique technique Ansible Tower sont :

|  |  |
| --- | --- |
| AREA | Disponibilité |
| ADMINISTRATION | Coche |
| INTERNE | Coche |
| EXTERNE | Coche |
| SIE | Fermer |

|  |  |
| --- | --- |
|  | Il n’est pas possible d’utiliser la brique Ansible Tower en dehors des AREA autorisées. |

## Offre

Les offres du cadre normatif autorisées pour l’utilisation et l’exploitation de la brique technique Ansible Tower sont :

| OFFRE | Disponibilité |
| --- | --- |
| PRODUCTION | Coche |
| HOMOLOGATION | Coche |
| CONCEPTION | Coche |
| INNOVATION | Fermer |

|  |  |
| --- | --- |
|  | Il n’est pas possible d’utiliser la brique Ansible Tower en dehors des Offres autorisées. |

# Version de la brique technique

Ansible Tower est un outil de CMS (*Configuration Management System*) développé par Red Hat.

Pour des raisons de stabilité, de compatibilité logicielle et de ressources documentaires, GRDF a retenu les versions fournies dans les référentiels Yum officiels de l’éditeur. La version du paquet ansible est mise à jour en même temps que le paquet ansible-tower

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Paquet | Version | OS | Mainstream support end date | Fin de vie |
| ansible-tower | 3.8 | RHEL8.X | 18/11/2021 | 18/11/2022 |
| ansible | 2.9.20 | RHEL8.X | NA | NA |

 Pour garantir un support complet tout au long de la vie de cette brique technique, les équipes d’expertise GRDF auront la charge de réaliser des mises à jour régulières. Au minimum une fois par an.

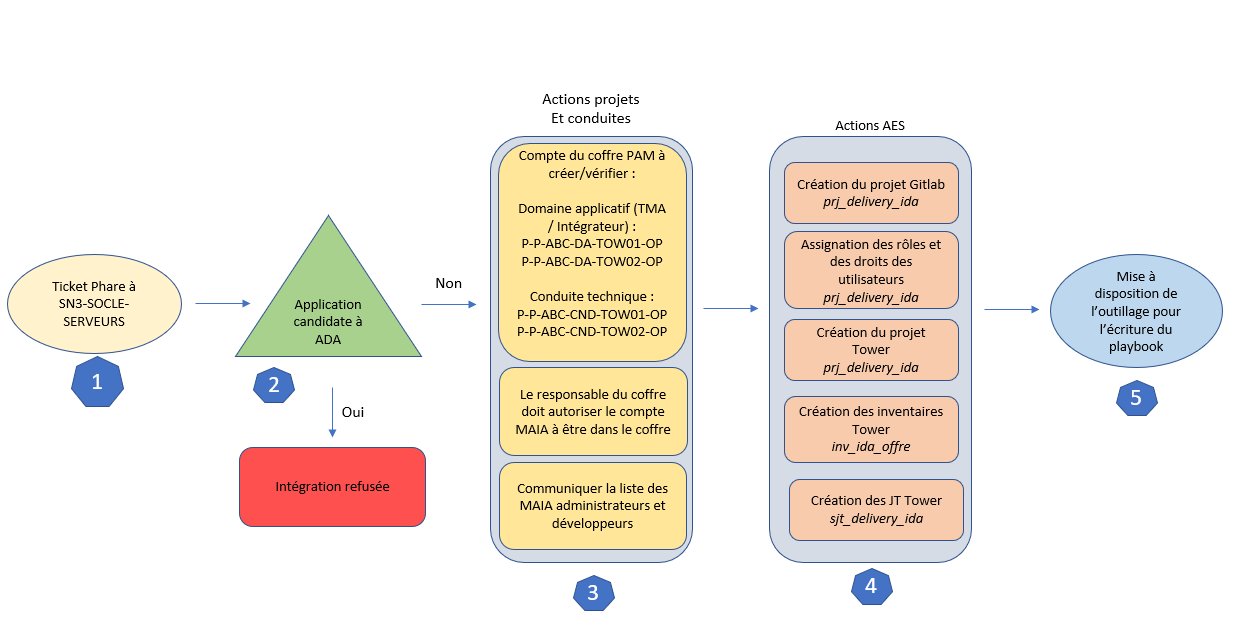
|  |  |
| --- | --- |
|  | La version Ansible Tower 3.8 évoluera tout au long de la vie de cette brique technique vers des versions plus récentes et supportées |

# Fonctionnement de la brique technique

Sont illustrés ici les processus de « Livraison applicative » et « Déploiement d’un middleware ». La suite de ce document explique comment développer des Playbook et des Roles

## Livraison applicative

Le diagramme ci-dessous représente le processus permettant à un domaine applicatif de s’intégrer dans la livraison applicative avec Ansible Tower.



Ce processus comprend 5 grandes étapes :

1. Pour s’inscrire dans la démarche de livraison applicative avec Ansible Tower, le domaine applicatif doit en faire la demande par phare :

*« Création de package Appli sous Tower »*

1. Suite à cette demande, une étude est alors faite entre les parties pour déterminer si l’application concernée peut être prise en compte par l’usine logicielle (ADA : Automatisation du Déploiement des Applications). Si oui, alors la demande est refusée et renvoyée vers les bons interlocuteurs.
2. Si l’application est candidate à la livraison applicative avec Ansible Tower, le domaine applicatif doit alors faire la demande du coffre PAM lié à l’IDA.

Le responsable du coffre autorisera tel ou tel utilisateur à se connecter à Ansible Tower

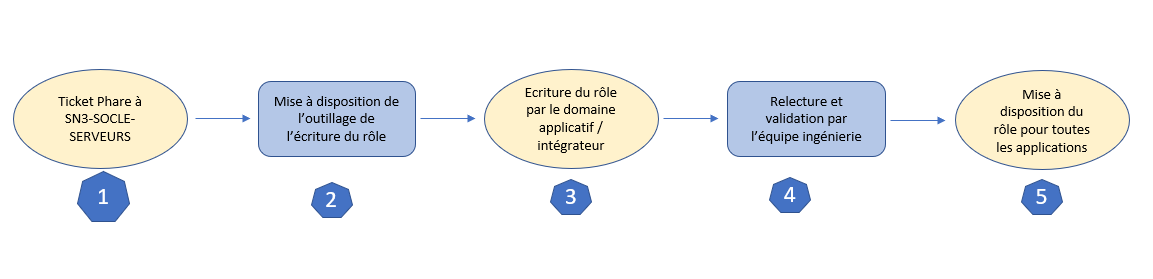
1. A partir de ces éléments, l’équipe Ingénierie peut prendre en charge la demande et créer les différentes ressources, à savoir :
   * Le projet GitLab avec les permissions correspondantes
   * Le projet Tower correspondant au projet GitLab accessible aux groupes AD (Lecture seule)
   * L’inventaire Tower accessible aux groupes AD (Lecture seule)
   * Le Job Template Tower accessible aux groupes AD (Exécution)
2. L’outillage pour l’écriture du Playbook est alors mis à disposition du domaine applicatif

## Déploiement d’un middleware

Un middleware peut appartenir ou non à la politique industrielle. A l’écriture de ce document, les middlewares faisant partie de la politique industrielle GRDF sont listés dans le tableau ci-après. Chacun de ces middlewares est déployé selon des règles d’ingénierie et d’exploitation écrites par les équipes en charge de ces middlewares.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Middleware | Version | Provenance | Editeur/Intégrateur | Support |
| Apache HTTPD | 2.4 | Repos RHEL8 | Red Hat | Oui (Red Hat) |
| Nginx+ | R23 | Repos NGinx | Nginx | Oui (Nginx) |
| PHP | 8.0 | Repos REMI | Communautaire | Non |
| Apache Tomcat | 10.0 | Repos REMI | Communautaire | Non |
| OpenJDK | 8 (puis 17) | Repos RHEL8 | Red Hat | Oui (Red Hat) |
| PostgreSQL | 13 | Repos EDB | EDB | Oui (EDB) |
| Postgres EnterpriseDB | 13 | Repos EDB | EDB | Oui (EDB) |
| Microsoft IIS | 10.0 v1809 | Microsoft | Microsoft | Oui (MS) |
| Microsoft .NET | 3.5 SP1 | Microsoft | Microsoft | Oui (MS) |
| Microsoft .NET | 4.6 (puis 4.8) | Microsoft | Microsoft | Oui (MS) |
| MS SQLServer | 2019 | Microsoft | Microsoft | Oui (MS) |
| MS SQLServer AO | 2019 | Microsoft | Microsoft | Oui (MS) |
| MS RDLS | 2019 | Microsoft | Microsoft | Oui (MS) |

Tout autre middleware, ne faisant donc pas partie de la politique industrielle GRDF, est géré différemment. Il peut, à la demande, être intégré dans un Role Ansible selon le processus suivant.



Ce processus comprend 5 grandes étapes :

1. Une demande est faite auprès de l’équipe Ingénierie en passant par un ticket Phare : « Création d’un package non politique Industrielle » pour développer ou faire évoluer un Rôle Ansible de déploiement de middleware hors politique industrielle

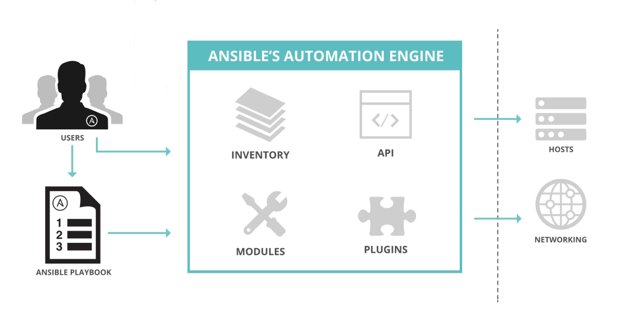
1. L’équipe Ingénierie met à disposition l’outillage pour l’écriture du Rôle Ansible auprès du domaine applicatif.
2. Le domaine applicatif développe le Rôle Ansible en suivant les règles décrites dans ce document
3. L’équipe Ingénierie participe à la relecture et la validation du Rôle Ansible
4. L’équipe Ingénierie rend ce nouveau Rôle Ansible disponible à l’ensemble des domaines applicatifs

# Caractéristiques de la brique technique

## Ansible

Ansible est à la fois un outil d’automatisation (*control node*) et un langage d’automatisation (*playbook*). Il permet de décrire les opérations (*tasks*) que l’on souhaite exécuter sur un ou plusieurs serveurs (Linux ou Windows, physiques ou virtuels, équipement réseau…) ou sur une API. Cette description représente l’état (*Desired State*) que l’utilisateur souhaite obtenir à la fin de son opération (déploiement, provisioning, orchestration, intégration continue…). Pour cela, le playbook est qualifié de langage *descriptif* – à l’inverse du langage *impératif*, comme le sont les scripts shell ou powershell. Néanmoins ceci est vrai seulement si le développeur a écrit le playbook de manière *idempotente*, signifiant alors qu’il a le même effet qu'on l'exécute une ou plusieurs fois.

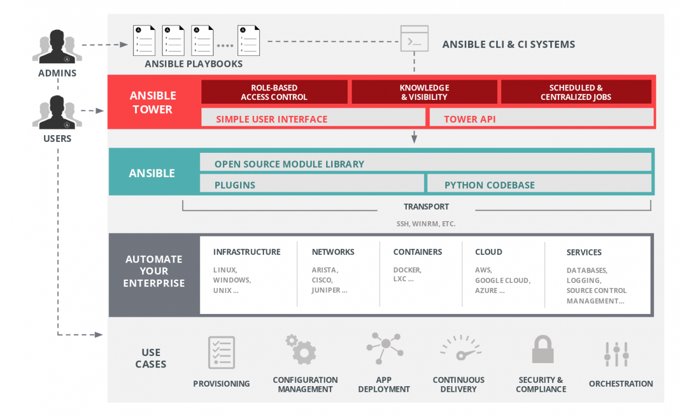
L’architecture d’Ansible est représentée dans le schéma suivant. Le control node (appelé ici *automation engine*) s’appuient sur une collection de *modules* et de *plugins* (code développé en Python). A travers une interface en ligne de commande (*CLI*) il offre aux utilisateurs la possibilité d’exécuter leurs playbooks sur des serveurs Linux (en SSH) ou Windows (en WinRM) contenus dans des fichiers d’inventaires (*inventory*). Cela signifie que ce serveur a un accès privilégié à ces serveurs.



L’interface (CLI) et la gestion des accès, mais aussi la scalabilité sont des points faibles à Ansible.

## Ansible Tower

Ansible Tower offre une interface Web (*Web UI*) simple d’utilisation avec une gestion des accès et des délégations parfaitement intégrés. Cette surcouche à Ansible est également *scalable* car elle se déploie en configuration HA avec répartition de charge (*Load Balancing*) et permet aussi de piloter des zones réseaux restreintes avec l’utilisation d’*Isolated Node*(s). Son API est conçue pour que Tower puisse être piloté par un orchestrateur tiers, type CMP (*Cloud Management Platform*) ou CI/CD (GitLab ou Jenkins par exemple).



## Glossaire spécifique Ansible

|  |  |
| --- | --- |
| Terme / composant | Description |
| Ansible Playbooks | Fichiers YAML dans lesquels sont mentionnés toutes les tâches qu'Ansible doit exécuter |
| Ansible CLI | L’outil en ligne de commande qui permet d’exécuter du code Ansible sans Tower |
| Web UI | Interface Web simple permettant de piloter l’ensemble de l’infrastructure gérée par Tower, d’exécuter des Job template et d’avoir une vue globale |
| Tower API | Inclut un service API RESTful qui permet aux administrateurs système et aux développeurs d'écrire des scripts et applications de tierce-partie personnalisés pouvant interagir avec Ansible Tower. |
| Inventory | Statique ou dynamique permettant de lister/classer les hosts sur lesquels ansible interviendra |
| Credential | Les credentials sont utilisés par Ansible Tower pour se connecter aux différentes sources extérieures. |
| Project | Les projects sont une collection logique de playbooks qu’utilisent Ansible Tower. Ils représentent généralement les repository SCM (GitLab, Subversion, etc…) contenant les playbooks attenant au job\_template |
| Job Template | Ils permettent aux playbooks d’être contrôler, déléguer et d’être scalable en termes d’organisation.  Ils nécessitent un inventaire, un credential pour se connecter aux VM et un project (SCM ou local) dans lequel seront stockés les playbooks |
| Workflows | Possibilité de lancer un séquencement de playbooks pour aboutir in fine aux actions désirées |
| RBAC | RBAC (Role Based Access Control) permet de gérer les permissions des utilisateurs sur les objets Tower (Inventory, Credential, Projects, Job Templates …) |
| Centralisation des logs | Les logs d’exécution de Ansible Tower peuvent être centralisés sur un puit de log (splunk, logstash, etc ...) |
| Modules | Bouts de codes écrits principalement en python pour modifier l’état de propriété d’un host. Les modules sont invoqués par l’exécution de taches soit directement en ligne de commande soit via des playbooks |
| Plugins | Les Plugins apportent des fonctionnalités complémentaires à Ansible (become ; action ; …) |

## Outillage tiers

### GitLab

Tower a une forte adhérence avec l’outil de *SCM* (*Source Code Management*) GitLab. C’est ce dernier qui centralise l’ensemble du code Ansible (*Playbooks* et *Roles*). Tower peut récupérer le contenu d’un ou plusieurs *project*(s) GitLab à tout moment, à la demande ou lors de l’exécution d’un *Job Template*. Cela signifie que les développeurs ont la responsabilité de s’assurer que leur code est bien en ligne et à jour sur leurs projets respectifs.

Il est accessible via l’adresse https://gitlab-cd.in.grdf.net (Sans Pulse) ou https://ws-gitlab-cd.in.grdf.net (Avec Pulse).

L'arborescence pour les projets est centrée autour des IDA, la voici :

ansible-playbooks

└── delivery

├── abc

├── cde

├── fgh

└── ijk

ansible-roles

└── delivery

├── abc

│   ├── role1

│   └── role2

├── fgh

│   └── role3

└── common

├── common1

└── common2

Pour chaque IDA, un projet sera mis à disposition dans le namespace "ansible-playbooks/delivery", ce projet sera automatiquement ajouté dans Tower et permettra de créer des Jobs template à partir de ses playbooks.

Il sera possible de faire la demande d'un namespace "ansible-roles/delivery/<ida>" pour y développer des rôles lorsque cela répond à un besoin.

Il sera également possible de consommer des rôles communs, mis à disposition de tout les utilisateurs afin de simplifier les déploiements ainsi que participer à l'effort d'uniformisation.

|  |  |
| --- | --- |
|  | La création des projets GitLab et Tower est de la responsabilité de DSI/ING/AES et DSI/EXP/POI |
|  | Un utilisateur GitLab n’a accès qu’à ses projets (avec modifications possibles) |
|  | Un utilisateur Tower n’a accès qu’à ses projets (sans modifications possible) |

### VMware

L’essentiel du parc serveur chez GRDF étant hébergé sur un socle virtuel VMware, Tower utilise celui-ci comme source d’inventaire. Cela lui permet d’avoir en mémoire l’ensemble des serveurs du parc GRDF (hors serveurs physiques) mais également d’exécuter un Playbook sur une ou plusieurs VM ciblée(s) en fonction de son environnement / offre, trigramme applicatif, role / tiers applicatif …

L’inventaire VMware est mis à jour chaque matin, cela permet de disposer des serveurs créés à J-1. Cette mise à jour d’inventaire comprend également le déploiement de *facts* personnalisés comme l’offre de la VM (Conception, Homologation ou Production) et son trigramme applicatif. Cela permet de définir des inventaires de type « Smart » qui cible les VMs d’un trigramme applicatif pour une offre donnée.

Le nommage des inventaires Tower est donné dans le tableau ci-dessous.

| Type d’objet | Norme de nommage | Exemple |
| --- | --- | --- |
| Inventory | *inv\_delivery*\_*<offre>\_<ida>*  Avec :   * *offre* : conception, homologation ou production * *ida* : Trigramme de l’application | inv\_delivery\_ccpt\_abc |

|  |  |
| --- | --- |
|  | La création des inventaires Tower est de la responsabilité de DSI/ING/AES et DSI/EXP/POI |
|  | Un utilisateur Tower n’a accès qu’à ses inventaires (sans modifications possible) |

### Artifactory

Un serveur Artifactory sur les offres d’homologation et de production est installé.

Ces serveurs gérés par la CARMA hébergeront les middlewares et les applications des domaines applicatifs.

Ils seront accessibles sur l’ensemble des VM via https

### Ansible Vault

Le vault est un outil d’Ansible destiné à protéger les secrets et documents sensibles présents dans un déploiement Ansible.

Tout les mot de passes, logins, clés, etc … qui pourrait être présent lors du déploiement doivent être chiffrés avant d’être commité sur Gitlab.

Afin de chiffrer les données, il faut posséder un mot de passe vault. Ce mot de passe doit respecter les prérequis suivants :

* Longueur de 25 caractères minimum
* Utilisation de 3 des variants (min, maj, chiffre, caractère spécial)
* Pas d'utilisation de suite triviale dans le mot de passe (une date, un nom , un prénom, une suite de chiffres ou de lettres, un nom connu comme GRDF ou le nom de l'applicatif / service ou autre, etc.)
* Recommandation de l’utilisation d’un outil pour générer le mot de passe de manière aléatoire (ex. : le générateur de mot de passe de KeePass disponible dans le centre logiciel)

On peut ensuite à partir de ce mot de passe réaliser des opérations sur les fichiers à chiffrer avec la commande « ansible-vault » :

$ ansible-vault --ask-vault-password <action> <file>

Les commandes disponibles sont : {create,decrypt,edit,view,encrypt,encrypt\_string,rekey}

Les fichiers pouvant être chiffrés dans Ansible sont les fichiers de variable, les templates, les fichiers texte à copier. Pour se faire, on utilisera la commande « encrypt » sur un fichier en clair.

Par convention, un fichier de variable vaulté sera nommé « vaulted.yml ».

 Il est possible de ne pas préciser le mot de passe à chaque fois en ligne de commande en ayant un vault-password-file contenant le mot de passe vault et en utilisant --vault-password-file ou l’option équivalente du fichier « ansible.cfg ». Dans ce cas, il faut alors s’assurer que ce fichier soit ignoré par un fichier « .gitignore » et non commité par mégarde dans votre repository.

### PAM (Cyberark)

L’accès à Ansible Tower se fera au travers de PAM via des comptes génériques.

Ces comptes génériques seront normés et nommés en fonction de l’IDA de l’application

Les utilisateurs MAIA associés à ces comptes génériques seront de la responsabilité du coffre PAM associé.

# Prise en main de la brique technique

Dans ce chapitre sont détaillées les règles d’écritures Ansible en vigueur chez GRDF.

## Prérequis

 Ce chapitre est à destination des développeurs et responsables d’application. Sa compréhension implique la maîtrise des outils git, GitLab et Ansible. Il ne se substitue pas à un support de formation sur ces outils considérés ici comme standards.

La lecture et la prise en compte de celui-ci et de l’intégralité du document peut, à la demande des équipes concernées, donner suite à un transfert de connaissances.

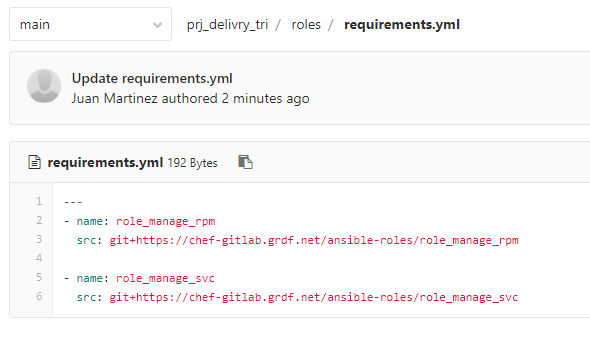
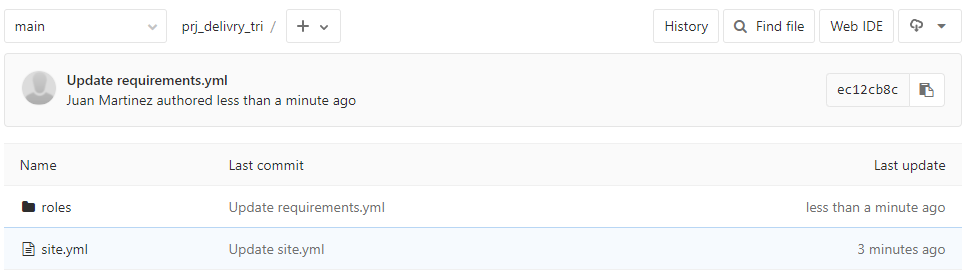
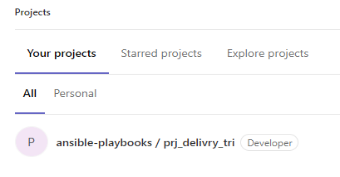
## Ecriture d’un Playbook Ansible

Dans le processus de livraison applicative, les développeurs écrivent leur code de déploiement au format Playbook. Les fichiers sont contenus sur un serveur GitLab sur lequel Ansible Tower se synchronise pour en récupérer la dernière version et le déployer sur les serveurs.

|  |  |
| --- | --- |
|  | Les développeurs ont la charge de définir le déploiement de leur application dans un ou plusieurs Playbook situés sur un serveur GitLab |

### Accès à GitLab

En se connectant au Web UI de GitLab, les développeurs visualisent les projets accessibles. Ces projets sont contenus dans le groupe GitLab *ansible-playbook*. Dans l’exemple ci-dessous, le seul projet visible s’intitule « prj\_delivery\_ida ».



L’utilisateur a cette visibilité dans GitLab en fonction du choix du responsable.

* les utilisateurs ayant le profil administrateur bénéficient de la permission *Maintainer* sur le projet GitLab
* les utilisateurs ayant un profil développeur bénéficient de la permission *Developer* sur le projet GitLab

La permission *Maintainer* permet d’approuver des Merge Requests sur une branche protégée, ce qui n’est pas le cas de la permission *Developer*.

|  |  |
| --- | --- |
|  | Seuls les utilisateurs ayant été désigné en tant qu’administrateur pourront approuver une Merge Request sur une branche protégée. |

### Comment se connecte Tower aux serveurs pour exécuter le code Ansible ?

Tower se connecte aux serveurs Linux par SSH en tant que *tower*, utilisateur pouvant élever ses privilèges en tant que *root*. Pour exécuter une *task* avec les droits de l’utilisateur *root*, il est nécessaire d’activer la clé *become*. Cette opération **n’est pas autorisée dans les Playbooks** développés chez GRDF. Un développeur devra décrire l’exécution de ses *tasks* avec les clés *become* et *become\_user*. Ceci peut être fait globalement à un *play*, et unitairement à une *task*.

Sous Windows, Tower utilise le protocole WinRM pour se connecter aux serveurs en tant que *S-DIO-ANS-PROD@nova.infra-grdf.com* ou *S-DIO-ANS-HPROD@nova-preprod.infra-grdf.com*. Cet utilisateur a les privilèges administrateur. Il n’est pour autant, **pas autorisé d’exécuter des *tasks* avec ces comptes techniques**. Comme pour Linux, un développeur devra décrire l’exécution de ses *tasks* avec les clés *become*, *become\_user* et également *become\_method*, égal à *runas*, différent de la valeur par défaut (*sudo*).

Le tableau ci-dessous liste les paramètres utilisés pour se connecter aux serveurs Linux et Windows. Le fact*ansible\_os\_family* peut être utilisé dans les Playbooks pour conditionner l’exécution d’une tâche au type d’OS.

|  |  |  |
| --- | --- | --- |
| Paramètre / fact | Linux | Windows |
| ansible\_connection | ssh (défaut) | winrm |
| ansible\_port | 22 (défaut) | 5986 |
| ansible\_winrm\_transport | N/A | credssp |
| ansible\_os\_family | RedHat | Windows |

|  |  |
| --- | --- |
|  | L’exécution de *task* avec les comptes techniques *tower*, *root* (sous Linux) ou *S-DIO-ANS-PROD@nova.infra-grdf.com* et *S-DIO-ANS-HPROD@nova-preprod.infra-grdf.com* (sous Windows) n’est pas autorisée |
|  | Un *pipeline* GitLab CI validera que les Playbooks font appel à la bonne élévation de privilèges pour chaque *task*/*play*. |

Exemples :

1. Activation de l’élévation des privilèges au niveau global. Toutes les *tasks* sont exécutées en tant que « appli\_user »

---

- name: play 1

hosts: all

**become: yes**

**become\_user: « ida »**

tasks:

- name: task 1

module\_name:

…

…

- name: task 2

module\_name:

…

…

1. Activation de l’élévation des privilèges au niveau unitaire. Toutes les *tasks* sont exécutées avec le même utilisateur sauf celles spécifiées avec un autre utilisateur

---

- name: play 1

hosts: all

become: yes

become\_user: « ida »

tasks:

- name: play 1 task 1

module\_name:

…

…

- name: play 1 task 2

module\_name:

…

…

- name: play 2

hosts: all

become: yes

become\_user: « ida »

become\_method: runas

tasks:

- name: play 2 task 1

module\_name:

…

…

**become\_user: autre\_user**

- name: play 2 task 2

module\_name:

…

…

### Description de l’application (role des serveurs)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Environnement** | **Nom du serveur** | **Fonction** | **CPU** | **Mémoire** | **Stockage** | **VLAN** | **Adresse IP** |
| **Homologation** | vlh1tow01.dc.infra-grdf.com | Nœud Tower | 4 | 8 | 50 | Front-3056 | 10.40.11.37 |
| **Homologation** | vlh1tow02.dc.infra-grdf.com | Nœud Tower | 4 | 8 | 50 | Front-3056 | 10.40.11.38 |
| **Homologation** | vlh1tow03.dc.infra-grdf.com | BDD | 4 | 8 | 100 | Front-3056 | 10.40.11.39 |
| **Production** | vlp1tow01.dc.infra-grdf.com | Nœud Tower | 4 | 8 | 50 | Front-3050 | 10.40.9.37 |
| **Production** | vlp1tow02.dc.infra-grdf.com | Nœud Tower | 4 | 8 | 50 | Front-3050 | 10.40.9.38 |
| **Production** | vlp1tow03.dc.infra-grdf.com | BDD | 4 | 8 | 100 | Front-3050 | 10.40.9.39 |

### Gestion des variables

Les inventaires Tower ne sont pas modifiables. Les développeurs ne peuvent que les utiliser pour exécuter leur Job Template.

Il est donc nécessaire de définir la valeur des variables dans le répertoire *group\_vars* prévu à cet effet. Pour chaque groupe de serveurs, les valeurs des variables d’inventaires doivent être positionnées dans les sous répertoires correspondants. Ces fichiers YAML doivent porter des noms correspondant à leur contenu. Par exemple, les fichiers de variables sans informations chiffrées peuvent s’appeler *vars.yml* tandis que ceux comprenant des informations chiffrées s’appelleraient *vaulted.yml*

$ tree group\_vars/

group\_vars/

front

app

datavm

|-- all/

| `-- vars.yml

| `-- vaulted.yml

|-- zookeeper/

| `-- vars.yml

| `-- vaulted.yml

|-- kafka\_broker/

| `-- vars.yml

| `-- vaulted.yml

Le sous répertoire *all* peut être utilisé pour contenir les variables globales à l’ensemble des serveurs.

|  |  |
| --- | --- |
|  | Aucune variable ne peut être située directement dans un fichier d’inventaire. |
|  | Les variables doivent être gérées au niveau *host scope* uniquement (répertoire *group\_vars*) |

### Liste des Rôles Ansible mis à disposition

Dans le cas où un développeur souhaite, dans son Playbook, exécuter une *task* nécessitant des privilèges plus élevés, un Rôle « système » est prévu à cet effet.

Ce Rôle est contenu dans le groupe GitLab *ansible-roles* et sera disponible sur TOWER en tant que formulaire:

| Nom du Role | Description du Role | Paramètre(s) du Role |
| --- | --- | --- |
| role\_manage\_sudo | En fonction des RIE RHEL et WIN, certaines commandes en fonction du user pourra se voir attribuer des élévations de privilèges | User générique habilité + Selection de la commande via une liste déroulante + durée de l’élévation de privilège |

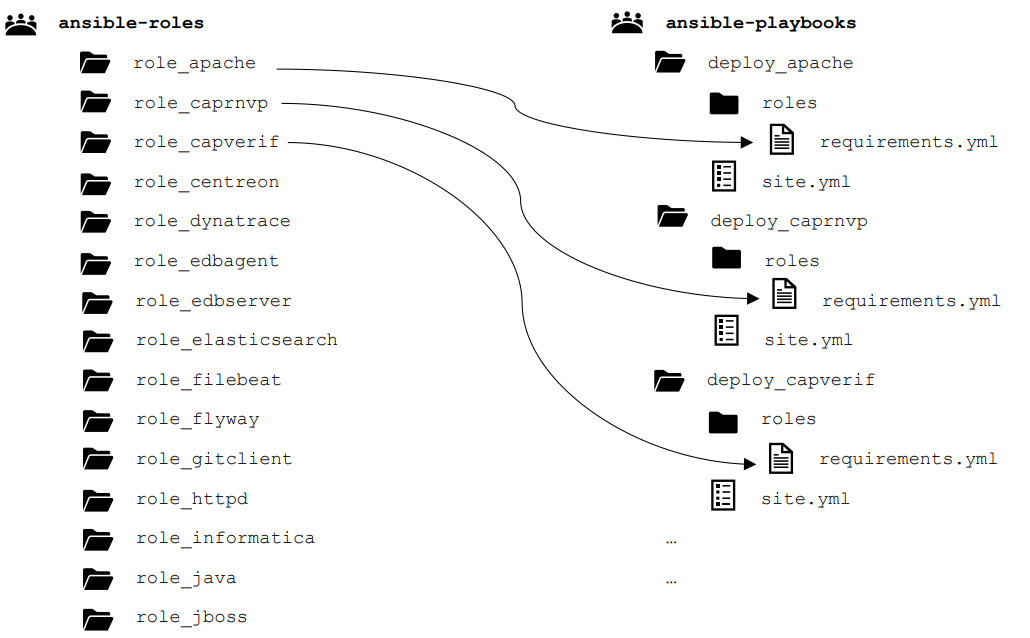
### Consommation d’un Rôle Ansible existant

Le groupe GitLab *ansible-roles* contient une collection de Rôles Ansible dont les standards d'écriture sont détaillés plus tard dans ce document. Chaque projet contenu dans ce groupe GitLab contient un unique Rôle qui assure une fonction unitaire permettant le plus souvent de déployer un outil ou un *middleware*, de configurer un service ou un ensemble de paramètres système ou autre.

Pour avoir accès en lecture au groupe GitLab *ansible-roles*, contacter l'équipe Ingénierie système à l'adresse suivante : [*GRDF-DSI-ING-EXPERTISE-LINUX@grdf.fr*](mailto:GRDF-DSI-ING-EXPERTISE-LINUX@grdf.fr).

On entend par *consommation* d'un Rôle Ansible, son utilisation à partir d'un Playbook issu d'un projet, normalement présent dans le groupe GitLab *ansible-playbooks*. Lorsqu'un développeur de Playbook souhaite utiliser un Role Ansible, il devra consulter le fichier README.md de ce dernier pour connaitre le comportement de ce Role et la fonction qu'il délivre et surtout la (ou les) variable(s) qu'il pourra (ou devra) renseigner lors de l’utilisation de ce Rôle.

Le schéma ci-dessous illustre les interactions entre ces deux groupes GitLab. Depuis le groupe GitLab *ansible-playbooks*, les développeurs de Playbooks peuvent consommer les Rôles présents dans groupe GitLab *ansible-roles*.



Pour ce faire, le projet GitLab doit contenir, en plus des différents fichiers Playbooks et répertoires divers (variables ou autres), un répertoire intitulé *roles* contenant lui-même un fichier *requirements.yml*. Dans ce fichier, sont listés l'ensemble des Roles que le développeur souhaite consommer dans son projet. Voici ci-dessous un exemple de fichier *roles/requirements.yml*

---

- name: role\_manage\_rpm

src: git+https://chef-gitlab.grdf.net/ansible-roles/role\_manage\_rpm

- name: role\_manage\_svc

src: git+https://chef-gitlab.grdf.net/ansible-roles/role\_manage\_svc

### Modules interdits dans les Playbook

Il n’est pas autorisé d’utiliser les modules suivants dans les Playbooks :

* yum
* package
* service
* firewalld
* raw
* command

|  |  |
| --- | --- |
|  | Un *pipeline* GitLab CI valide que les Playbooks ne font pas appel à des modules interdits |

### Description du git flow

Une branche main, protégée, existe par défaut dans chaque repository Gitlab.

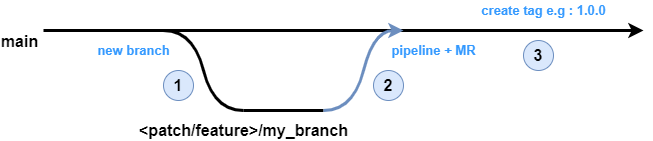
Cette branche représente les travaux qui ont été validés et mergés et celle utilisée par Tower pour déployer en Homologation ainsi qu'en Production.

Quand un développeur (membre du groupe AD GSL-APP-<TRI>-ANSIBLE-DEV) souhaite faire une modification du code Ansible il doit créer une nouvelle branche à partir de la branche main pour y faire ses modifications. Les modifications y sont alors testées et éxécutées via Tower depuis la branche d'évolution sur un inventaire de machines de tests et de conception.

Les modifications peuvent ensuite être fusionnées de manière automatique vers la branche main à condition que l’exécution du pipeline de vérifications du code ne remonte aucune erreur.

Pour déployer le code sur les serveurs d’Homologation et de Production, il faudra alors tagger le commit dans la branche main et utiliser ce tag dans Tower (e.g: '1.0.8').

Les merges et les créations de tags sont réservés aux membres du groupe AD GSL-APP-<TRI>-ANSIBLE-ADM.



1. Le développeur crée une nouvelle branche à partir de la branche main et y crée des modifications. Cette nouvelle branche doit respecter la norme de nommage suivante.

- [patch,feature]/champ\_libre ; avec :

- patch s’il s’agit du développement d’un correctif ; et

- feature s’il s’agit de l’ajout d’une nouvelle fonctionnalité

1. Il peut alors réaliser ses tests à partir de cette branche jusqu'à ce qu'il soit satisfait du résultat et que les pipelines de vérification soient OK.
2. Le dévéloppeur soumet ses modifications sous la forme d'une merge request vers la branche main. Cette dernière est passé en revue par les admins du projet et ensuite mergée.
3. Enfin, afin de réaliser la mise en PROD d'une ou plusieurs patchs/features, il faudra alors créer un tag sur le commit voulu et utiliser ce tag dans Tower pour la récupération du code.

|  |  |
| --- | --- |
|  | Chaque projet GitLab est composé d*’au moins une branche main* |
|  | Un.e développeur.se crée une nouvelle branche à partir de la branche *main* pour y faire ses modifications |
|  | Les fusions du code modifié vers les branches protégées sont traitées via un pipeline CI |
|  | Le projet Tower devra forcément pointer sur un tag créé sur la branche main |

## Ecriture d’un Role Ansible

Il arrive que le déploiement d’une application soit plus complexe et que l’utilisation des Roles soit nécessaire. Dans ce cas, il est possible d’écrire ses propres Roles en respectant les règles énoncées ci-après. Une revue de code est ensuite réalisée par l'équipe Ingénierie.

Les Roles existants sont disponibles dans le groupe GitLab *ansible-roles*. Pour être contributeur ou développer un nouveau Role, il est nécessaire de contacter l'équipe Ingénierie à l'adresse suivante : [*GRDF-DSI-ING-EXPERTISE-LINUX@grdf.fr*](mailto:GRDF-DSI-ING-EXPERTISE-LINUX@grdf.fr) qui se chargera de la création de ce nouveau Role.

|  |  |
| --- | --- |
|  | La création de nouveaux Roles Ansible et l’affectation des permissions sur ceux-ci et les Roles existants est sous la responsabilité de l'équipe Ingénierie uniquement |

### Standardisation et réutilisation des Roles

Il est important de garder à l'esprit, que lorsque l'on développe un Role Ansible, il doit pouvoir être réutilisable. Bien que parfois, cela ne soit pas nécessaire car le Role sera exécuté uniquement dans un environnement déterminé ayant certains standards (comme les noms ou adresses IP des serveurs DNS ou NTP) qui ne changeront pas, il reste néanmoins indispensable de respecter cette règle pour des raisons de confidentialité et d'organisation des données.

 Un Role Ansible est développé pour déployer une fonction précise (exemples : déployer Apache, déployer un *vhost*, configurer le *firewall*…)

Il est d’usage de faire appel à plusieurs Roles pour déployer une application, chaque Role réalisant une partie du déploiement

C'est également une bonne pratique que tous les développeurs de Roles Ansible suivent les mêmes règles d'écriture. Les liens ci-dessus renvoient vers ces bonnes pratiques.

Source : [*https://docs.ansible.com/ansible/latest/user\_guide/playbooks\_reuse\_roles.html*](https://docs.ansible.com/ansible/latest/user_guide/playbooks_reuse_roles.html)

|  |  |
| --- | --- |
|  | Chaque nouveau Role doit être développé de manière à être réutilisable facilement |

### Variables internes VS externes

Une variable interne d'un Role, définie dans le fichier *vars/main.yml* n'est pas censé être surchargée/modifiée. La changer pourrait altérer l'exécution du Role.

Une variable externe d'un Role, définie dans le fichier *defaults/main.yml*, peut (ou doit) être surchargée lorsque le Role est utilisée dans un Playbook.

D'après le second lien de référence, concernant la précédence des variables avec Ansible, on remarque que :

1. Une variable externe d'un Role a une précédence de bas niveau (peut être surchargée par une variable d'inventaire) tandis qu'une variable interne a une précédence de haut niveau (peut surcharger une variable d'inventaire)
2. Il existe 20 possibilités de surcharger une variable externe d'un Role. Le mieux est de le faire dans un fichier de variable situé dans le répertoire *groups\_vars* du projet ou lors de l'appel du Role dans le Playbook. La première manière suppose que notre inventaire est organisé en groupes correspondant par exemple à des environnements, des fonctions, des datacenters, des applications... Plus il y a de groupe, plus il est pratique de définir les valeurs de ces variables

Source : [*https://docs.ansible.com/ansible/latest/user\_guide/playbooks\_variables.html#variable-precedence-where-should-i-put-a-variable*](https://docs.ansible.com/ansible/latest/user_guide/playbooks_variables.html#variable-precedence-where-should-i-put-a-variable)

|  |  |
| --- | --- |
|  | Les variables d’un Role Ansible doivent être placées dans le répertoire approprié. |

### Proscrire tout type de *Hardcoding*

Un Role ne doit pas contenir d'informations en dur, notamment des *credentials* ou le nom d'un serveur tiers.

Ces informations doivent être variabilisées et leurs valeurs définies dans les fichiers appropriées (répertoires *defaults* ou *vars* du Role).

Si, parmi les variables externes d’un Role, certaines d’entre elles peuvent être considérées comme sensibles, elles doivent être chiffrées dans un vault Ansible et appelées à l’extérieur du Role uniquement.

Garder en tête que moins il y a de *hardcoding* dans un Role, plus il sera réutilisable.

|  |  |
| --- | --- |
|  | Un Role ne doit contenir aucune information en dur. |

### Nommage du Role et de ces variables

Le Role doit porter le nom *role\_<fonction\_du\_role>* (exemple : *role\_apache*). Le champ doit être court et parlant (que l'on sache rapidement son utilité). Ce nommage est réalisé à la création du Role par l’équipe Ingénierie.

Pour fonctionner avec un nombre de Roles important, les variables de chacun d'entre eux doivent être préfixées par le nom du Role. En faisant ça, il est facile de déterminer dans quel Role une variable est utilisée. La surcharge des variables est donc mieux maîtrisée.

Il n'est pas nécessaire que les noms des variables soient préfixés de *role\_*. Ils doivent porter un nom de type *<fonction\_du\_role>\_<nature\_de\_la\_variable>* (exemple : *apache\_package\_name*)

|  |  |
| --- | --- |
|  | Les variables internes et externes des Roles doivent être correctement nommées |

### Règles d'écriture

Un certain nombre de recommandations est donné ici. Pour faciliter la relecture, le troubleshooting et la modification des Roles Ansible, il est important de les respecter.

1. Le fichier *README* doit obligatoirement être renseigné. Il doit contenir une description des Roles, ses prérequis éventuels et la liste des variables externes et internes manipulées à l'intérieur de celui-ci

Une image contenant texte

Description générée automatiquement

1. L'extension des fichiers est *.yml*, et non *.yaml*

$ tree roles/role\_certificates/

roles/role\_certificates/

|-- README.md

|-- defaults

| `-- main.yml

|-- handlers

| `-- main.yml

|-- meta

| `-- main.yml

|-- tasks

| |-- RedHat.yml

| `-- main.yml

`-- vars

`-- main.yml

5 directories, 7 files

1. Tous les fichiers *YAML* doivent avoir en première ligne ---. La convention qui impliquerait que la dernière ligne soit ... n'est pas nécessairement à appliquer

$ head -n 5 roles/role\_certificates/defaults/main.yml

---

certificates\_certificates\_list: []

$ head -n 5 roles/role\_certificates/tasks/RedHat.yml

---

- name: Copies Certificates

ansible.builtin.template:

src: "{{ item }}.j2"

dest: "{{ certificates\_local\_dir }}/{{ item }}"

1. Utiliser l'indentation minimale (2 espaces) uniquement. Bien que *YAML* tolère des indentations de 4 espaces ou plus, ne pas privilégier cette option

$ cat roles/role\_certificates/tasks/RedHat.yml

---

- name: Copies Certificates

ansible.builtin.template:

src: "{{ item }}.j2"

dest: "{{ certificates\_local\_dir }}/{{ item }}"

owner: root

group: root

mode: 0644

loop: "{{ certificates\_certificates\_list }}"

become: yes

notify: Extracts certificates on Linux host

when:

- certificates\_certificates\_list is defined

- certificates\_certificates\_list | length > 0

1. Espacer les tâches ou les définitions de variables pour améliorer la lisibilité

- task 1

module

param1: value

param2: value

- task 2

- task 3

1. Dans un fichier YAML de type *task file* (répertoires *handlers* et *tasks*) :
   * Nommer les *task* de manière unique au sein d'un Role

|  |  |
| --- | --- |
|  | Chaque *task* d’un *play* doit avoir un nom unique |

* + La première ligne d'une tâche doit utiliser la directive *name* puis le nom long du module et ses attributs puis viennent dans cet ordre les directives *loop*, *become*, *when*, *changed\_when*, *failed\_when* ...

Exemple :

- **name: …**

**nom\_du\_module:**

param1: value

param2: value

…

loop: …

become: …

notify: …

when: …

register: …

failed\_when: …

ignore\_errors: …

…

|  |  |
| --- | --- |
|  | La première *key* d’une *task* est systématiquement la directive *name*. La seconde est le nom du module |

* + Chaque tâche nécessitant une élévation de privilège doit contenir la direction *become: yes*. RAPPEL : Pour Red Hat, une élévation de privilèges consiste à exécuter la tâche en tant que l'utilisateur *root*

|  |  |
| --- | --- |
|  | L’utilisation de l’élévation des privilèges en tant que « root » sera particulièrement contrôlée lors de la relecture du Role |

### Prise en main d'un nouveau Role Ansible

Depuis un serveur Linux disposant de l'utilitaire *git* et ayant les droits de se connecter en SSH au serveur GitLab. Par ailleurs, votre environnement utilisateur doit disposer de la clé SSH privée configurée votre compte GitLab.

Depuis son répertoire d'accueil, se placer dans le répertoire roles

$ cd ~/roles

Cloner le projet git en utilisant l'URL SSH copiée précédemment

$ git clone git@chef-gitlab.grdf.net:ansible-roles/role\_selinux.git

Cloning into 'role\_selinux'...

warning: You appear to have cloned an empty repository.

Avec la commande ansible-galaxy, créer une arborescence vide pour le nouveau Role

$ ansible-galaxy init role\_selinux/ --force

- role\_selinux/ was created successfully

Voici le résultat attendu

$ tree role\_selinux/

role\_selinux/

|-- README.md

|-- defaults

| `-- main.yml

|-- files

|-- handlers

| `-- main.yml

|-- meta

| `-- main.yml

|-- tasks

| `-- main.yml

|-- templates

|-- tests

| |-- inventory

| `-- test.yml

`-- vars

`-- main.yml

8 directories, 8 files

Pour faciliter la lecture d'un Role, il est très fortement recommandé de supprimer les répertoires non utiles (notamment le répertoire tests). L'arborescence minimale d'un Role Ansible doit être composée au minimum des fichiers et répertoires suivants :

role\_selinux/

|-- README.md

|-- meta

| `-- main.yml

|-- tasks

| `-- main.yml

Bien que laissé inchangé lors de la création de l'arborescence du Role, le répertoire *meta* ne doit pas être supprimé, au risque de ne plus pouvoir synchroniser le Role depuis Ansible Tower.

## Conformité

A instruire

## Exploitation

A instruire

# Utilisation de la brique technique

## Exemple de projet

L’ensemble des fichiers d’un projet GitLab (finalisé ou en cours de développement) est organisé comme décrit dans le tableau ci-après. Certains fichiers/répertoires sont obligatoirement présents. Les autres sont optionnels en fonction des besoins (utilisation de Roles, gestion des variables et des fichiers à déployer, factorisation en fichiers de *tasks*…)

| Fichier / Répertoire | | Description | Caractère |
| --- | --- | --- | --- |
| $ tree prj\_delivery\_tri/  prj\_delivery\_tri/ |  | | - |
| |-- README.md | Fichier README à renseigner | | Obligatoire |
| |-- files  | |-- myfile.cfg  | `-- myfile2.info | Répertoire contenant des fichiers plats | | Facultatif |
| |-- group\_vars  | |-- all  | | |-- vars.yml  | | `-- vaulted.yml  | |-- front  | | |-- vars.yml  | | `-- vaulted.yml  | `-- app  | |-- vars.yml  | `-- vaulted.yml  | `-- …  | |-- …  | `-- … | Répertoire contenant les variables d’inventaire des groupes de serveurs.  Les variables globales peuvent être définies dans le groupe *all*.  Toutes les autres variables doivent être organisées dans des groupes | | Obligatoire |
| |-- roles  | `-- requirements.yml | Répertoire contenant la liste des Roles à télécharger | | Facultatif |
| |-- site.yml | Playbook principal | | Obligatoire |
| |-- tasks  | |-- mytasks1.yml  | |-- mytasks2.yml  | `-- mytasks3.yml | Répertoire contenant les fichiers de *tasks* secondaires (appelées par le Playbook principal) | | Facultatif |
| |-- templates  | `-- mytemplate.j2 | Répertoire contenant les templates Jinja2 | | Facultatif |

## 

## Normes

Lors de la rédaction de playbooks ou de rôles Ansible, quelques normes sont à respecter afin d'assurer une cohérence de la plateforme.

Fichiers

Tout fichier de tasks, playbook, ou de variable est un fichier YAML et doit respecter les conventions suivantes :

- Les fichiers doivent utiliser l'extension '.yml'

- La première ligne est toujours "---" tandis que la dernière est "..."

- Les noms de variables sont en minuscules.

- Lorsqu'un booléen est utilisé, on utilise les formes "True" et "False" plutôt que les versions lowercase ou encore 0 et 1.

- On n'utilise pas de tabulations pour l'indentation du fichier mais des espaces, par saut de 2.

Structures

Les régles suivantes sont à respecter dans la structure de chaque projet Git Ansible :

- Le projet doit avoir un README.md permettant la compréhension et l'utilisation du projet/rôle

- Les playbooks doivent être à la racine d'un repo de projet

- Les dossiers possibles en fonction de vos besoins sont :

- templates => ajout de template jinja2

- tasks => ajout de fichiers de tasks

- files => ajout de fichiers statiques

## Connexion à Ansible Tower

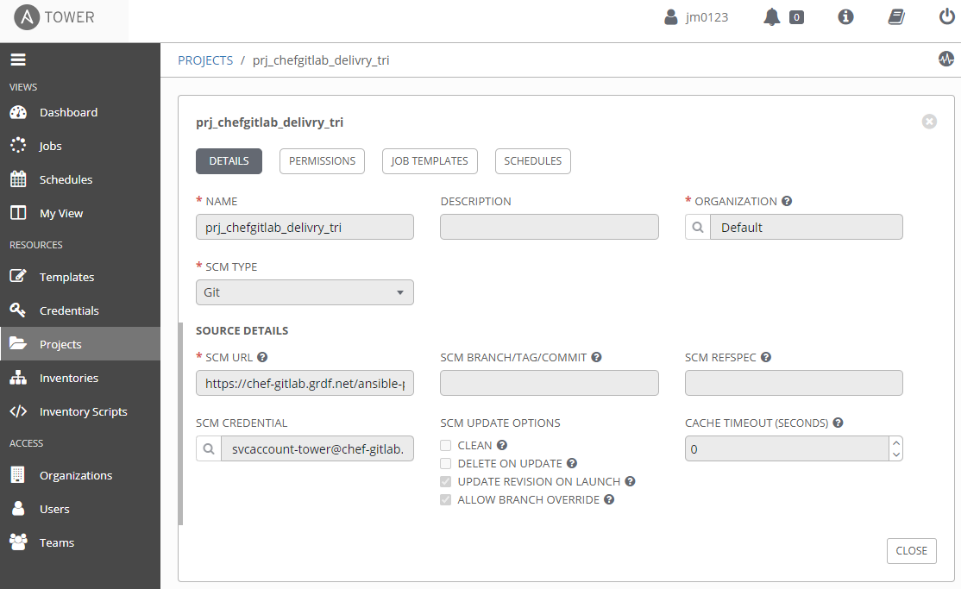
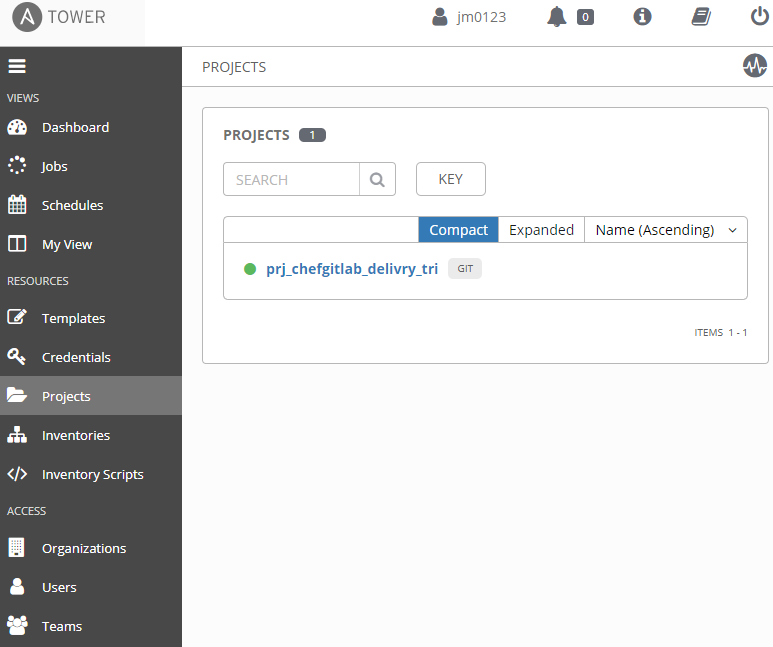
L’utilisateur.ice de Tower doit appartenir à l’un de ces groupes AD pour se connecter au Web UI et exécuter des Job Templates

* GSL-APP-<*TRI*>-ANSIBLE-ADM
* GSL-APP-<*TRI*>-ANSIBLE-DEV

## Projet

Dans le menu « Project », le projet GitLab est visible en lecture seule par l’utilisateur.ice. On doit y retrouver notamment :

* l’URL du serveur GitLab
* et l’option « Allow Branch override » qui permet de sélectionner la branche désirée dans le Job Template désactivée par défaut.



## Inventaires

Inventaire principal

Le principal inventaire est un inventaire VMWare présent afin d'identifier l'ensemble des VMs sur le parc.

Cet inventaire permet l'accès à toutes les machines hébergées sur l'infrastructure VMWare et de leur appliquer des groupes par défaut mais également des groupes custom via des règles, notamment :

- OFR (e.g PROD, HOMOLOGATION, CONCEPTION)

- AREA (ADMIN, EXTERNE, INTERNE)

- OS (RHEL, WINDOWS)

Cet inventaire est à destination des populations ayant l'accès sur le parc, à date il s'agit des POI et des équipes Ingénieries.

Inventaires applicatifs

Pour chaque trigramme applicatif, des inventaires seront créés dynamiquement à partir de l'inventaire principal, l'idée est de fournir un inventaire par Système d'exploitation et Environnement.

Par exemple pour un trigramme applicatif IDA possédant les environnements C1, C2, H1, H2 et P1, avec un mix de machines Windows et RedHat, les inventaires à disposition seront donc :

- ida\_rhel\_c1

- ida\_windows\_c1

- ida\_rhel\_c2

- ida\_windows\_c2

- ida\_rhel\_h1

- ida\_windows\_h1

- ida\_rhel\_h2

- ida\_windows\_h2

- ida\_rhel\_p1

- ida\_windows\_p1

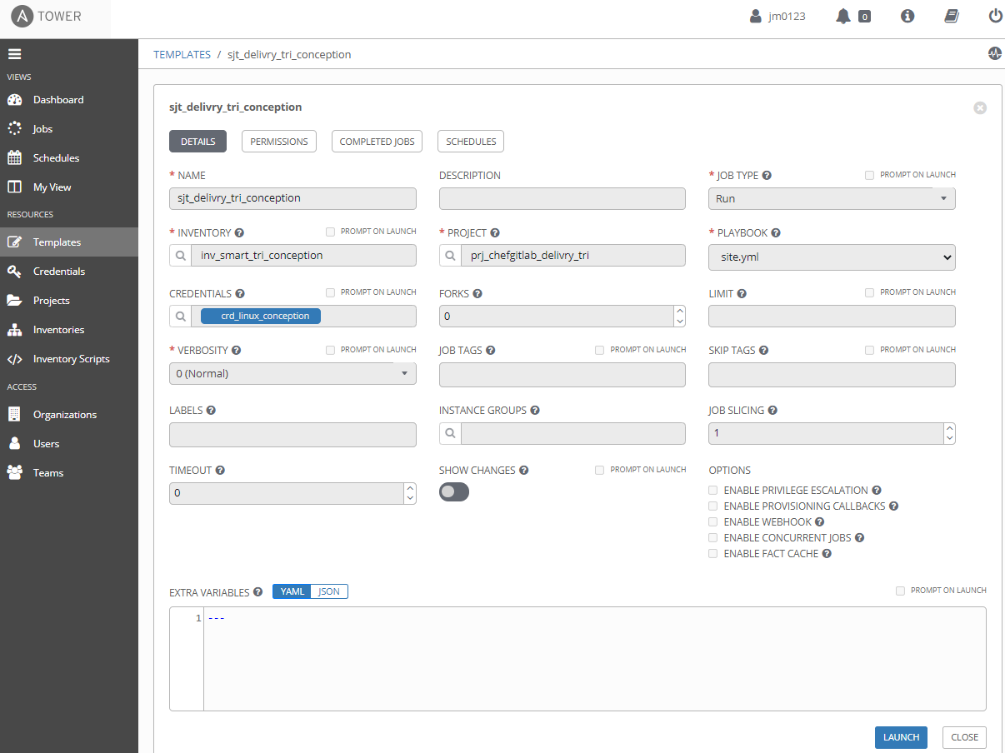
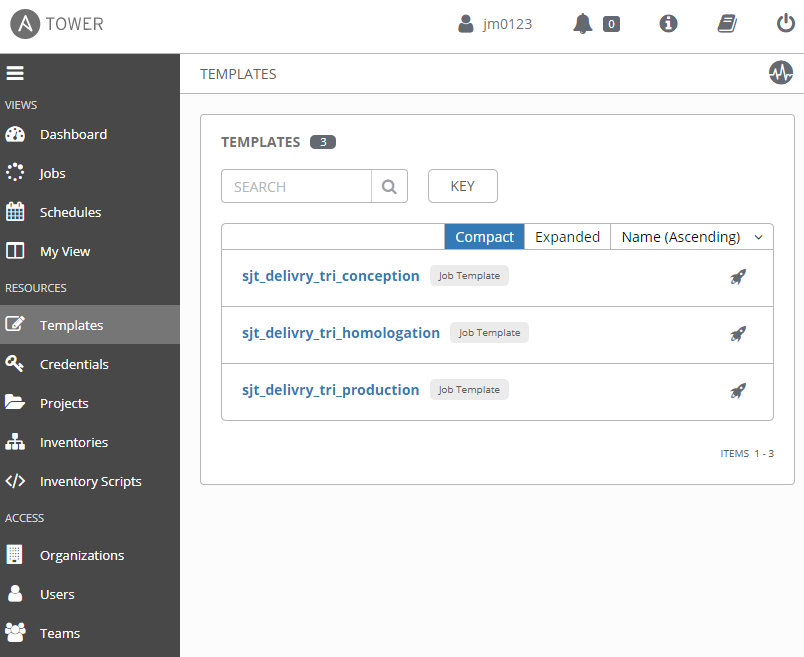
A noter que les inventaires de conceptions (eg. ida\_os\_cX) seront mis à disposition des domaines applicatifs tandis que les inventaires d'homologation et de production seront à destination des conduites.

## Job Templates

Dans le menu « Job Templates », un Job Template pour chaque Offre (Conception, Homologation et Production) sont visibles en lecture seule par l’utilisateur.ice. Chaque Job Template utilise :

* L’inventaire
* et le credential de l’offre correspondante

Bien que n’apparaissant pas, la branche, *main* ou *develop*, est spécifiée à ce niveau pour que Tower sache où récupérer le code sur le serveur GitLab. Il sera possible de le rendre la branche cible modifiable à l’exécution sur demande justifiée.



 L’exécution d’un Job Template se fait en cliquant sur l’icône suivant :

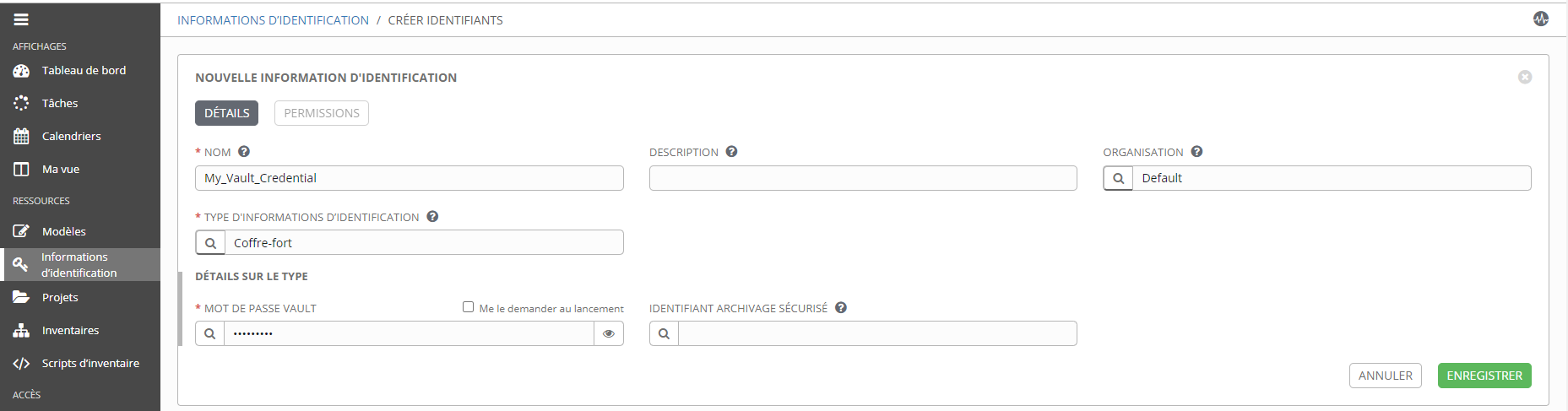
|  |  |
| --- | --- |
|  | Les utilisateur.ice.s de Tower n’ont accès qu’en lecture seule au ressources (Projects, Inventories and Job Templates). Seule l’exécution d’un Job Template est possible à partir du Web UI Tower |

## Credential Vault

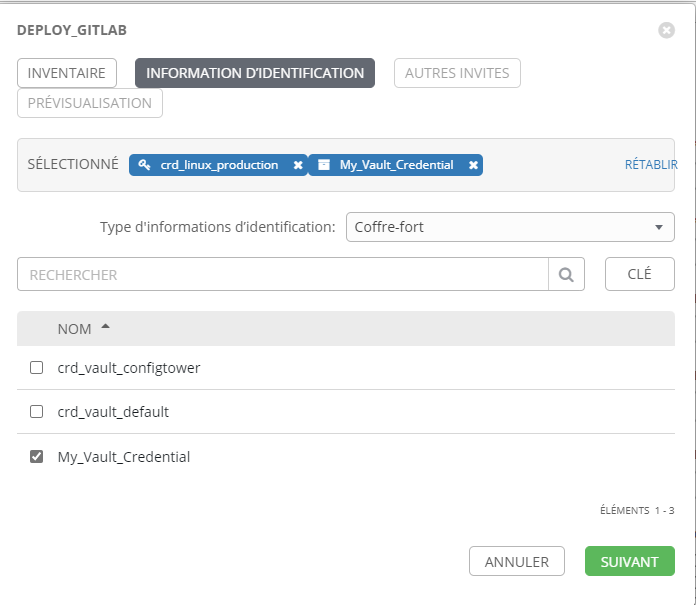
Lorsque le projet contient des informations sensibles, ces dernières doivent être chiffrées en utilisant ansible-vault (voir [ici](#_Ansible_Vault)).

Les job templates sont configurés par défaut avec l’option « demander au lancement » sur les credentials, ce qui permet au moment de l’exécution, de préciser un ou plusieurs credentials utilisés par le job.

Si des secrets existent dans vos projets, il faudra alors créer un credential de type « Coffre-Fort » (vault) dans le menu « Informations d’identification » où vous aller renseigner votre vault password (clé de chiffrement).



Et lors de l’éxécution de mon job template, j’aurais alors la possibilité de choisir le vault associé à mon projet :



Il est de votre responsabilité de créer votre Credential Vault dans Tower avec le mot de passe utilisé pour chiffrer vos secrets présents dans votre repository.