



DOCUMENTATION 5

DÉPLOYER SON PROJET GRACE À ANSIBLE

SOMMAIRE

Historique.....	3
Acronymes.....	4
Introduction.....	5
1. Utiliser les collections RUCHE Ansible.....	6
1.1 Présentation.....	6
1.2 Utilisation.....	6
2. Déployer son SI.....	8
2.1 Déployer le Back de son SI.....	10
2.2 Déployer le Front de son SI.....	13
2.3 Utiliser le service d'Observabilité.....	16
3. Opérer le raccordement à MinDefConnect.....	20
3.1 Exposition des valeurs nécessaires au raccordement.....	20
3.2 Implémentation dans les playbooks.....	22
4. Vérifier l'architecture de son projet Ansible.....	25
5. Quelques astuces pour déboguer et faire un ticket Tuleap.....	27
5.1 Bonnes pratiques pour déboguer.....	27
5.2 Faire un ticket sur Tuleap.....	27
Conclusion.....	28

Historique

Tableau 1 : Historique

Version	Date	Auteur	Commentaire
1.0.0	30/08/2024	Maxime HEIM	
2.0.0	09/09/2024	Chloé PAGENEL	Refonte de la structure globale et écriture de la partie « Déployer son SI »
2.1.0	09/09/24	Romain GUYOT	Ajout de la partie MinDefConnect
2.1.1	10/09/2024	Chloé PAGENEL	Relecture
2.1.2	11/09/2024	Chloé PAGENEL	Relecture
2.1.3	12/09/2024	Cindy PEREIRA	Relecture
2.1.4	13/09/2024	Aurélie GRABAS-DOREMUS	Relecture
2.1.5	13/09/2024	Chloé PAGENEL	Relecture
2.1.6	07/10/2024	Cindy PEREIRA	Correction de coquilles
2.1.7	05/12/2024	Cindy PEREIRA	Précision sur le requirements.yml
2.1.8	22/01/2025	Romain GUYOT	Ajout d'une précision sur la récupération des secret Observability

Acronymes

Tableau 2 : Acronymes

Acronyme	Description
AAP	Ansible Automation Platform
COTS	Commercial Off-The-Shelf (<i>produit informatique standard</i>)
GDS	Gestion Des Secrets
OIDC	OpenID Connect
SMC	Service Métier Commun
TME	Tierce Maintenance d'Exploitation
VM	Machine Virtuelle

Introduction

L'objectif de ce **chapitre 5** est de parcourir un projet exemple pour appréhender les différentes étapes à réaliser afin de déployer son SI sur **PICSEL** à l'aide d'Ansible.

Ce chapitre va donc commencer par vous présenter les collections **RUCHE**, un élément essentiel du déploiement sur PICSEL.

Vous parcourrez ensuite le **projet exemple** à travers trois étapes clefs : déployer son Back, son Front et le Service Observabilité.

Puis, vous verrez comment intégrer, au sein de vos playbooks Ansible, l'authentification avec le service **MinDefConnect**.

Et enfin, vous passerez par une étape de vérification pour s'assurer que l'architecture de votre projet Ansible suit quelques règles communes qui ont été établies pour **simplifier** le déploiement sur **C1NP**.

En dernière partie de ce chapitre se trouve quelques astuces pour vous aider dans la **résolution d'erreurs communes**, ainsi que des indications sur comment rédiger un ticket **Tuleap**.

En résumé, voici les **cinq parties majeures** abordées par ce document :

Temps	Titre de la partie	Description
10%	Utiliser les collections RUCHE Ansible	Comment utiliser les collections clés en main de RUCHE ?
60%	Déployer son SI	Comment déployer son SI de A à Z.
15%	Utiliser MinDefConnect	Mettre en place l'authentification MinDefConnect.
10%	Vérifier son architecture projet Ansible	S'assurer de suivre quelques règles communes sur la structure de son projet.
5%	Quelques astuces pour déboguer	Vérifier des points clefs lors d'erreur rencontrées et créer un ticket Tuleap.

Tout au long de la documentation, les encadrés bleus correspondent aux NFR (Non Functional Requirements) liés aux parties traitées. Ces NFR ne sont pas obligatoires mais sont de bonnes pratiques qu'il est conseillé de suivre.

1. Utiliser les collections RUCHE Ansible

L'objectif de cette partie est de faire une présentation générale des collections **RUCHE** et de la marche à suivre pour les utiliser.

Cette partie couvre les NFR suivantes :

NFR_ANSIBLE_08 : Un SI ne doit en aucun cas développer le déploiement des COTS (Middleware) sauf s'il n'existe pas et n'est pas prévu. Dans ce cas le projet reversera les développements à RUCHE. Le playbook d'un SI doit s'appuyer sur les collections **ANSIBLE** du catalogue RUCHE.

1.1 Présentation

Les collections **RUCHE** sont des modules permettant d'effectuer des actions de façon simplifiée et normalisée. Si une collection RUCHE permet de réaliser une tâche nécessaire au déploiement de votre **SI**, il faut **obligatoirement** l'utiliser.

Par exemple, pour l'installation d'une base de données **PostgreSQL** nous allons pouvoir utiliser la collection « **minarm.postgresql** ».

Cette collection met à notre disposition un rôle « **postgresql_install** » qui configure et sécurise un ou plusieurs serveurs de base de données PostgreSQL ; en faisant simplement appel à ce rôle, l'installation de PostgreSQL se fait sans plus d'actions de notre part.

L'utilisation de cette collection permet d'uniformiser la façon d'installer PostgreSQL qu'importe le SI concerné, il est donc important de s'en servir.

1.2 Utilisation

Les collections RUCHE du **minarm** sont disponibles à cette URL : <https://ruche-galaxy.intradef.gouv.fr/ui/>

Ainsi, nous retrouvons la collection **postgresql** à l'adresse suivante : <https://ruche-galaxy.intradef.gouv.fr/ui/repo/published/minarm/postgresql>

Il y a différents points auxquels il faut prêter attention pour utiliser ces collections :

- Lire la partie « **Installer** » pour prendre connaissance des actions réalisées par la collection.
- Lire la partie « **Documentation** » pour prendre connaissance des différents rôles de la collection :

- Être attentif au « **Changelog** » pour prendre connaissance des dernières modifications de la collection.
- Être attentif à la documentation de chaque « **rôle** » pour prendre connaissance des **prérequis** et des **variables** nécessaires.
- Faire attention à toujours bien se référer à la dernière « **Version** » disponible car les collections sont amenées à évoluer dans le temps.

2. Déployer son SI

L'objectif de cette partie est de parcourir l'exemple du déploiement d'un SI sur PICSEL pour comprendre les éléments clefs. Cet exemple sera découpé en trois parties principales : **Back**, **Front** et **Observabilité**, qui s'incrémenteront au fur et à mesure.

L'exemple utilise les technologies de **Drupal** et de **PostgreSQL**. Il permet le déploiement d'un SI spécifique, il n'est donc pas destiné à être copié/collé mais plus à servir de guide pour avoir une idée et une structure de comment procéder étape par étape.

Le déploiement se fait avec des playbooks Ansible, hébergés sur Gitlab, et exécutés avec AAP.

Attention : Il est important de respecter, autant que possible, l'arborescence, l'architecture et la structure des dossiers et des fichiers de l'exemple afin de garder une cohésion globale entre tous les SI concernés par la migration.

Attention : Vous trouverez dans le projet exemple de la ZS_CONTINUITY une task « **fixC1NP** » permettant la mise en cache des métadonnées des dépôts yum afin de s'assurer que les dépôts présents sur artifactory soient disponibles sur la VM. Pensez à faire jouer cette task avant le reste de vos playbooks (voir **site.yml** à la racine du projet).

Cette partie couvre les NFR suivantes :

NFR_SECRET_01 : Les secrets utilisés par les scripts ANSIBLE doivent être stockés dans GDS. La récupération du secret dans le GDS doit être gérée par le code ANSIBLE du playbook (ou des rôles/collections sous-jacentes) via l'usage d'un rôle fourni par l'équipe GDS.

NFR_SECRET_02 : Les certificats HTTPS privés exposés par les frontaux WEB d'un SI doivent être récupérés en automatique auprès du serveur GDS. Le renouvellement des certificats doit aussi être pris en charge en automatique via l'usage d'un agent vault disponible au catalogue RUCHE

NFR_PROXY_C1NP : Un SI C1NP doit OBLIGATOIREMENT utiliser le PROXY C1NP pour réaliser une interconnexion avec un système/service exposé sur INTERNET. L'usage de ce proxy est à prendre en compte dans l'implémentation. L'interconnexion avec le PROXY est une variable de l'inventaire technique.

NFR_ANSIBLE_01 : L'interconnexion entre les composants logiciels hébergés sur les VMs du SI doit se faire en automatique via l'usage des « groupes » ANSIBLE apposés sur les hosts de l'inventaire. Par exemple, une VM Back hébergeant une BDD PostgreSQL pour le service MonSI doit être associé à un groupe « PGSQL_MonSI ». Le code Ansible de configuration de la VM Front récupère le Host de la BDD PostGreSQL via l'usage du groupe « PGSQL_MonSI ».

NFR_ANSIBLE_02 : Seules les variables nécessaires au déploiement doivent être présentées dans les inventaires techniques et fonctionnels. Les variables qui permettent une configuration technique avancée du service peuvent exister mais elles ne doivent pas être nécessairement présentées à l'exploitant. Le manuel d'installation doit uniquement reprendre les paramètres d'inventaires qui seront à modifier par l'exploitant

NFR_ANSIBLE_03 : L'inventaire "exemple" fourni dans le playbook de déploiement doit être documenté. Chaque variable doit posséder une ou des lignes de commentaires. Cet inventaire exemple doit correspondre à l'inventaire utilisé lors des phases de test du SI (cf. Chap 2.4).

NFR_ANSIBLE_04 : Un motif de valeur ne doit être saisi qu'une seule fois. Par exemple, ne pas avoir à saisir l'alias DNS ou FQDN dans plusieurs variables.

NFR_ANSIBLE_05 : Le nommage des variables doit être parlant. Par exemple, s'il est attendu un token et non un mot de passe, appeler la variable ma_var_token et pas ma_var_password

NFR_ANSIBLE_06 : Idempotence : les scripts de déploiement peuvent être exécutés plusieurs fois sans erreur. L'ensemble des paramètres de l'inventaire identifiés dans le manuel d'installation doivent pouvoir être modifié et pris en compte par Ansible sans erreur.

NFR_TLSISI_01 : Les échanges entre les VMs de la zone projet hébergeant le SI doivent être chiffrés. Par exemple : l'interconnexion entre un frontal Web (Drupal) et une base de données (PostgreSQL) doit être chiffré.
L'échange entre 2 logiciels exécutés sur la même VM n'a pas besoin d'être chiffré (exemple : NGINX vers Tomcat)

NFR_OBSSUP_01 : La mise sous observation système (logs et métriques systèmes, audit) est réalisée en automatique lors de la création de la VM..
La mise sous observation des middlewares est réalisée par les collections RUCHE. Le SI doit intégrer dans son playbook les rôles du service d'observabilité pour :
Collecter et formater les logs produits par son application
Collecter les traces (via OpenTelemetry)

2.1 Déployer le Back de son SI

Tout d'abord nous allons commencer par la partie Back du déploiement, qui consiste à mettre en place une base de données PostgreSQL.

L'exemple du projet que nous allons suivre se situe à l'adresse suivante : https://scm-intradef.picse1.defense.gouv.fr/ZS_CONTINUITY/samples/exemples-documentation-4-move2cloud/exemple-partie-back

N'hésitez pas à ouvrir l'exemple en même temps que vous suivez la documentation.

Pour la première étape de notre déploiement, nous allons renseigner le FQDN de la VM créée pour la base de données ainsi que celui de la VM créée pour la partie Front. Pour cela nous créons un fichier **hosts.yml** dans le dossier **/inventories/integration** et ajoutons nos FQDN :

```
/inventories/integration/hosts.yml
```

```
2 all:
3   children:
4     database:
5       hosts:
6         db1:
7           ansible_host: FQDNDB.picse1.defense.gouv.fr
8     front:
9       hosts:
10        front1:
11          ansible_host: FQDNFR.picse1.defense.gouv.fr
```

Figure 1 : Exemple hosts.yml

Nous ajoutons ensuite notre fichier **requirements.yml** pour déclarer les collections RUCHE nécessaires. Ici, pour le contexte général du déploiement, nous avons besoin des collections `community.general`, `minarm.gds`, `minarm.lvm`.

Pour le déploiement spécifique à la base de données, nous avons besoin des collections `community.postgresql` et `minarm.postgresql`. Pour chaque collection, il faut renseigner son `name`, sa `version` et son `type` :

Attention : Pour chaque collection, il faut utiliser la dernière version disponible. Aucune URL ne doit être appelée dans le fichier *requirements.yml*; le champ « source » ne doit pas être utilisé.

```
/collections/requirements.yml
```

```
1 collections:
2 ## GENERAL
3 # Utilisé pour de nombreux modules et plugins communs d'Ansible
4 | - name: community.general
5 |   version: 7.5.5
6 |   type: galaxy
7 # Utilisé pour l'installation et l'utilisation d'Ansible Vault
8 | - name: minarm.gds
9 |   version: 5.1.0
10 |   type: galaxy
11 # Utilisé pour le partitionnement des disques
12 | - name: minarm.lvm
13 |   version: 5.0.4
14 |   type: galaxy
```

Figure 2 : Ajout des collections au fichier requirements.yml

Nous pouvons ensuite instancier notre premier playbook **install_database.yml** dans le dossier **/playbooks**. Dans ce fichier nous allons notamment renseigner `name`, `hosts`, `environnement` ainsi que `collections` et `roles` :

/playbooks/install_database.yml

```
1 ---
2 - name: Configure database
3   ## Hosts sur lesquels appliquer les tâches (all, database, front, ...)
4   hosts: database
5   become: true
6   become_user: root
7   environment:
8     MINARM_YUM_REPO_NATIONAL_ORIGIN: "{{ minarm_yum_repo_norigin }}"
9     MINARM_ATF_REPO_MIDDLEWARE_BASEURL: "{{ minarm_atf_repo_mbaseurl }}"
10  ## Référencement des collections nécessaires
11  collections:
12    - minarm.gds
13    - minarm.postgresql
14    - minarm.lvm
15  ## Appel dans l'ordre d'exécution des roles minarm et des roles customs
16  roles:
17    - role: minarm.gds.vault_get_secret
18    - role: minarm.gds.vault_get_cert
19    - role: minarm.lvm.lvm_install
20    - role: ../roles/vault_restart
21    - role: minarm.postgresql.postgresql_install
22    - role: ../roles/bdd_config
23    tags:
24      - install_db
```

Figure 3 : Exemple playbook installation database

Ici, notre playbook va s'exécuter sur le groupe ansible « **database** » et fait appel à des rôles personnalisés (`roles/bdd_config` et `roles/vault_restart`) pour configurer PostgreSQL selon nos besoins.

Ansible fait le lien entre le groupe « **database** » et le fichier `/inventories/integration/group_vars/database.yml`, il est donc important que le nom soit identique des deux côtés.

Dans les fichiers du dossier `/inventories` nous mettons toutes les variables communes et nécessaires à la fois pour le déploiement du **back** et du **front**. Elles sont placées dans ce dossier car elles ne sont pas strictement liées au SI.

Donc, dans notre fichier d'inventaire `all.yml`, nous précisons les variables `{{ repository_login }}` et `{{ repository_password }}` pour les rôles ayant besoin des informations d'identifications Medusa :

```
/inventories/integration/group_vars/all.yml
```

```
42  ## Variables pour les collections nécessitant un accès à Medusa
43  repository_login: "{{ artifactory_user }}"
44  repository_password: "{{ artifactory_password }}"
--
```

Nous allons également retrouver les variables alimentées par le rôle `vault_get_secret` qui permet de récupérer les secrets stockés dans GDS. La variable `{{ si_gds }}` définie dans le même fichier fait référence à votre dossier (ex: `ansible`) GDS dans lequel se trouve vos secrets :

```
/inventories/integration/group_vars/all.yml
```

```
33  | # Récupération des secrets stockés dans gds
34  | gds_secret:
35  |   artifactory_password: "{{ si_gds }}:artifactory_password"
36  |   db_password: "{{ si_gds }}:db_password"
```

Les secrets **Observability** sont à récupérer de manière similaire, mais depuis le dossier « **observability** » au lieu de `{{ si_gds }}`.

Figure 4 : Exemple variables GDS

À l'inverse, dans le dossier `/playbooks`, nous retrouvons des variables qui sont définies seulement pour le fonctionnement interne du SI comme, par exemple, `{{ db_name }}` :

```
/playbooks/group_vars/all.yml
```

```
1 ---
2 ## Database Informations
3 postgresql_version: 15
4 db_name: drupal
```

Figure 5 : Exemple variables playbook

À noter que des dossiers **/production** et **/pre-production**, contenant les mêmes éléments que le dossier **/inventories/integration**, sont présents dans le dossier **/inventories**. Ils sont placés ici afin d'être modifiés plus tard par la TME pour l'intégration du SI sur C1NP.

Attention : Il est important de nommer les variables de la même façon qu'elles sont annotées dans l'exemple ainsi que de les placer dans les mêmes fichiers.

Finalement, nous créons le fichier **site.yml**, à la racine du projet, pour appeler notre playbook **install_database.yml** :

/site.yml

```
1 - name: Install Database
2   import_playbook: playbooks/install_database.yml
3   tags:
4     - database
```

Figure 6 : Exemple site.yml

C'est ce même fichier qui est renseigné dans AAP pour lancer l'exécution des playbooks (voir la documentation 4 « Déployer avec AAP et stocker ses secrets sur GDS »).

2.2 Déployer le Front de son SI

Nous allons maintenant pouvoir développer la partie Front, qui dans notre exemple consiste à mettre en place une application existante Drupal.

L'exemple du projet que nous allons suivre se situe à l'adresse suivante : https://scm-intradef.picsel.defense.gouv.fr/ZS_CONTINUITY/samples/exemples-documentation-4-move2cloud/exemple-partie-front

N'hésitez pas à ouvrir l'exemple en même temps que vous suivez la documentation.

De manière générale, nous allons procéder de la même façon que pour la partie Back.

Nous allons donc commencer par ajouter les collections RUCHE nécessaires à l'installation de Drupal, dans le fichier **requirements.yml** : `minarm.drupal`, `minarm.core`, `minarm.php` et `minarm.apache`.

/collections/requirements.yml

```
26  ## FRONT
27  # Utilisé pour l'installation de Drupal
28  - name: minarm.drupal
29    version: 5.1.3
30    type: galaxy
31  # Dépendances nécessaires au rôle minarm.drupal.drupal_install
32  - name: minarm.core
33    version: 5.0.16
34    type: galaxy
35  - name: minarm.php
36    version: 5.1.8
37    type: galaxy
38  - name: minarm.apache
39    version: 5.1.10
40    type: galaxy
```

Figure 7 : Requirements.yml avec partie Front

Rappel : Pour chaque collection, il faut utiliser la dernière version disponible.

Nous allons ensuite créer notre playbook **install_front.yml**, en respectant le même format que celui appliqué pour le Back :

/playbooks/install_front.yml

```
1 ---
2 - name: Install Front
3   ## Hosts sur lesquels appliquer les tâches (all, database, front, ...)
4   hosts: front
5   become: true
6   become_user: root
7   environment:
8     MINARM_YUM_REPO_NATIONAL_ORIGIN: "{{ minarm_yum_repo_norigin }}"
9     MINARM_ATF_REPO_MIDDLEWARE_BASEURL: "{{ minarm_atf_repo_mbaseurl }}"
10  ## Référencement des collections nécessaires
11  collections:
12    - minarm.lvm
13    - minarm.gds
14    - minarm.drupal
15  ## Appel dans l'ordre d'exécution des rôles minarm et des rôles customs
16  roles:
17    - role: minarm.lvm.lvm_install
18    - role: minarm.gds.vault_get_secret
19    - role: minarm.gds.vault_get_cert
20    - role: minarm.drupal.drupal_install
21    - role: ../roles/drupal_config
22    tags:
23      - install_front
```

Figure 8 : Exemple fichier `install_front.yml`

Ici, notre playbook va s'exécuter sur le groupe ansible « **front** » et fait appel à des rôles personnalisés (`roles/drupal_config`) pour configurer Drupal et Composer en fonction du besoin du SI que nous déployons.

Pour les différentes tâches à exécuter, nous allons avoir besoin de renseigner des variables. Celles-ci n'étant utiles que pour la partie Front, nous créons respectivement des fichiers **front.yml** dans les dossiers `/inventories/integration/group_vars` et dans `/playbooks/group_vars`.

Finalement, nous allons pouvoir appeler notre playbook **install_front.yml** dans le fichier **site.yml** :

/site.yml

```
1 - name: Install Database
2   import_playbook: playbooks/install_database.yml
3   tags:
4     - database
5
6 - name: Install Front
7   import_playbook: playbooks/install_front.yml
8   tags:
9     - front
```

Figure 9 : Exemple fichier `site.yml`

Une petite particularité est à prendre en compte pour les SI ayant besoin du **proxy sortant C1NP**, pour ce cas-là il est nécessaire de renseigner les variables suivantes (à laisser vide) :

```
/inventories/integration/group_vars/all.yml
```

```
45  ## Variables pour le proxy sortant C1NP. :  
46  proxy_host: ""  
47  proxy_port: ""
```

Figure 10 : Exemple inventories all.yml

Elles seront remplies par la TME lors du déploiement sur C1NP.

2.3 Utiliser le service d'Observabilité

Nous allons maintenant pouvoir nous attaquer à l'avant dernière partie de notre déploiement : le service **Observabilité**.

L'exemple du projet que nous allons suivre se situe à l'adresse suivante : https://scm-intredef.picsel.defense.gouv.fr/ZS_CONTINUITY/samples/exemples-documentation-4-move2cloud/exemple-partie-observabilite

N'hésitez pas à ouvrir l'exemple en même temps que vous suivez la documentation.

De manière générale, nous allons procéder de la même façon que pour les précédentes parties.

Nous commençons donc par renseigner le FQDN de la VM créée pour le service Observabilité. Pour cela nous ajoutons deux nouveaux groupes dans notre fichier **hosts.yml** ; le SI que nous déployons a fait le choix d'utiliser les briques heartbeat et logstash :

```
/inventories/integration/hosts.yml
```

```

1  ## Les groupes de hosts sont définis sur 'database' et
2  all:
3    children:
4      database:
5        hosts:
6          db1:
7            ansible_host: FQDNDB.picse1.defense.gouv.fr
8      front:
9        hosts:
10         front1:
11           ansible_host: FQDNFR.picse1.defense.gouv.fr
12         # ***** Nouveaux éléments
13       heartbeat:
14         hosts:
15           obs1:
16             ansible_host: FQDNOPS.picse1.defense.gouv.fr
17       logstash:
18         hosts:
19           obs1:
20             ansible_host: FQDNOPS.picse1.defense.gouv.fr

```

Figure 11 : Exemple hosts.yml

Puis nous écrivons notre playbook **install_obs.yml** dans lequel nous appelons le rôle `tenant_observability` de la collection `RUCHE minarm.observability`.

Nous rappelons également le rôle `vault_get_secret`, qui est une dépendance nécessaire au rôle `observability` :

```

/playbooks/install_obs.yml
1  ---
2  - name: Install observability
3    hosts: all
4    collections:
5      - minarm.gds
6      - minarm.observability
7    tags:
8      - observability
9    roles:
10     - name: minarm.gds.vault_get_secret
11     - name: minarm.observability.tenant_observability
12

```

Figure 12 : Exemple install_obs.yml

Pour notre SI Drupal nous avons fait le choix d'utiliser les modules **logstash**, **heartbeat** et **beats**. Les deux premiers modules nécessitent une VM spécifique, à l'inverse du module `beats` qui lui va se déployer sur les VMs back et front.

Nous allons pouvoir renseigner les variables nécessaires au service dans nos différents fichiers de variables :

```
/playbooks/group_vars/all.yml
20  ## Observability Information
21  obs_services_to_play:
22    - beats
23    - heartbeat
24    - logstash
25  obs_url_vip_apm: "https://apm.observ.{{ zone_url }}:9243"
```

Figure 13 : Exemple playbook all.yml

```
/playbooks/group_vars/database.yml
53  ## Observability
54  obs_service: postgresql
55  obs_role: database
56  obs_beats_modules:
57    - templates/database/metricbeat/postgresql.yml
58    - templates/database/filebeat/postgresql.yml
--
```

Figure 14 : Exemple playbook database.yml

```
/playbooks/group_vars/front.yml
102  # ## OBS information
103  obs_service: apache
104  obs_role: webserver
105  obs_beats_modules:
106    - templates/front/metricbeat/apache.yml
107    - templates/front/filebeat/apache.yml
108  obs_application_log_host: drupal-services
---
```

Figure 15 : Exemple playbook front.yml

Vous trouverez des informations complémentaires sur le **service Observabilité** au lien suivant : https://portail-ct-rns.intradef.gouv.fr/sites/Portail%20du%20Cloud/KB_PICSEL/layouts/15/start.aspx#/Pages%20Wiki%20PICSEL/Service%20Observabilit%C3%A9%20v2.aspx

Finalement, nous ajoutons la référence de notre playbook **install_obs.yml** dans notre fichier **site.yml** :

/site.yml

```
1  - name: Install Database
2    import_playbook: playbooks/install_database.yml
3    tags:
4      - database
5
6  - name: Install Front
7    import_playbook: playbooks/install_front.yml
8    tags:
9      - front
10
11 - name: Installation Observability
12   import_playbook: playbooks/install_obs.yml
13   tags:
14     - observability
```

Figure 16 : Exemple site.yml

Attention : Il est important de placer les variables du service Observabilité au même endroit que dans l'exemple du déploiement du SI Drupal.

3. Opérer le raccordement à MinDefConnect

L'objectif de cette partie est de faire le raccordement avec le SMC **MinDefConnect** (MDC) afin de lui déléguer les mécanismes **d'authentification**.

Tout d'abord, veuillez-vous assurer d'avoir effectué les **étapes d'enrôlement MDC** décrites dans la documentation « **2. Analyser et mettre en conformité son projet** ».

Dans le cas où vous deviez effectuer des **développements spécifiques**, car votre SI ne se base pas sur un COTS qui propose une extension qui gère l'authentification via OIDC par exemple, cette même documentation contient **un lien vers des documents techniques** de MinDefConnect qui pourront vous aider à implémenter le mécanisme.

Attention : La méthode exacte d'implémentation dépend fortement du COTS de votre SI, ou de la technologie sur laquelle il repose, le cas échéant.

Vous pouvez vous référer aux exemples de la ZS_CONTINUITY, qui implémentent MDC dans leur partie Exemple complet.

Cette partie couvre les NFR suivantes :

NFR_MDC : L'authentification des utilisateurs sur le SI sont OBLIGATOIREMENT réalisés via MindefConnect (Internet ou Intradef).

NFR_MDC_C1NP : L'interconnexion avec MDCi requiert 2 URL distinctes.

- L'URL Publique du MDCi utilisée par le front (ie le navigateur du poste utilisateur)
- L'URL Privé du MDCi qui sera utilisé par le Back.

3.1 Exposition des valeurs nécessaires au raccordement

Attention : Toute configuration côté client correspondant aux paramètres de raccordement à MinDefConnect (URL, scopes, client-id, secret...) doit être modifiable (IHM ou fichier de configuration) et non pas renseignée en dur dans le code.

3.1.1 Secrets GDS

Tout d'abord, vous devriez avoir obtenu en réponse à votre enrôlement un fichier json contenant – entre autre – le **realm**, le **secret_id**, l'**url de connexion** (intitulé « **auth-server-url** »), et un rappel de votre **client-id** (intitulé « ressource »).

La première étape est d'ajouter ces secrets dans votre GDS (comme décrit dans la documentation « **4. Déployer avec AAP et stocker ses secrets sur GDS** »).

À noter que seul le **secret_id** doit y être placé, les autres valeurs peuvent simplement être stockées dans les variables de votre fichier **/inventories/integration/group_vars/front.yml**.

La récupération des secrets depuis GDS passe par le rôle **vault_get_secret** de la collection **minarm.gds** (détaillé plus haut).

3.1.2 Inventaires

Nous allons donc pouvoir rajouter ces nouveaux secrets GDS dans notre fichier **all.yml** du dossier **/inventories** :

```
/inventories/integration/group_vars/all.yml
37 gds_secret:
38   mdc_client_secret: "{{ si_gds }}:mdc_client_secret"
39   mdc_id_realm: "{{ si_gds }}:id_realm"
```

Figure 17 : Exemple inventories all.yml

Les autres valeurs qui ne sont pas stockées dans GDS sont à placer dans le fichier **front.yml** du dossier **/inventories** :

```
/inventories/integration/group_vars/front.yml
29 ## MinDefConnect Information
30 mdc_url: "mdc-internet-v2.{{ zone_url }}"
31 mdc_realm: Internet
32 mdc_client_id: int-move2cdloud-internet
33 mdc_client_server : "https://mindefconnect.{{ zone_url }}"
34
```

Figure 18 : Exemple inventories front.yml

Attention : certaines valeurs sont sensibles à la casse, notamment le **realm**.
Veillez à bien positionner les majuscules (premier « i » dans le cas du realm « Internet »)

3.2 Implémentation dans les playbooks

Selon votre SI, vous devrez intégrer une **extension** ou réaliser un **développement spécifique** pour vous raccorder à **MinDefConnect**.

La méthode exacte dépend de votre SI et de sa technologie, nous ne sommes pas en mesure de vous expliquer précisément les étapes à suivre pour chaque technologie.

Sachez toutefois que vous aurez très probablement à manipuler des fichiers **template de configuration** de type Jinja2 (.j2), pour faire référence aux valeurs renseignées dans vos inventaires, et à les déposer à l'endroit approprié sur la VM applicative.

3.2.1 Exemple avec DRUPAL

À des fins **d'exemple**, la procédure du travail de raccordement pour un SI en DRUPAL a été détaillée ci-dessous.

- **Extension « OpenID Connect Client » :**

Les SI basés sur Drupal peuvent utiliser l'extension **OpenID Connect Client** (oidc) afin de déléguer leur authentification à un service RedHatSSO/Keycloak configuré, dont MDC fait partie.

Vous pouvez ajouter l'extension à votre SI manuellement avant de la packager, pour qu'elle soit incluse avec. La configuration sera faite par les playbooks Ansible dans les étapes suivantes.

- **Fichiers de configuration j2 :**

Pour l'extension susmentionnée de Drupal, la configuration passe par les fichiers templates **oidc.settings.yml.j2** et **oidc.realm.generic.yml.j2**.

Ils sont à placer dans votre projet Ansible, dans le dossier suivant :

```
/role/drupal_config/templates/oidc.settings.yml.j2
```

Vous pouvez consulter leurs contenus au lien suivant : https://scm-intradef.picisel.defense.gouv.fr/ZS_CONTINUITY/samples/exemples-documentation-4-move2cloud/exemple-projet-complet.

- **Task Configure MinDefConnect :**

L'étape suivante consiste à rédiger la tâche de configuration qui s'appuie sur ces fichiers de configuration.

La tâche ci-dessous est issue du rôle **site_config.yml**, appelé par le playbook **install_front.yml** :

```
/roles/drupal_config/tasks/site_config.yml
```

```
33 - name: Configure MinDefConnect
34   ansible.builtin.template:
35     src: "templates/{{ item.src }}"
36     dest: "{{ __drupal_folder }}/config/sync/{{ item.dest }}"
37     owner: "{{ drupal_owner }}"
38     group: "{{ drupal_group }}"
39   loop:
40     - {src: 'oidc.settings.yml.j2', dest: 'oidc.settings.yml'}
41     - {src: "oidc.realm.generic.yml.j2", dest: "oidc.realm.generic.{{ mdc_id_realm }}.yml"}
42
```

Figure 19: Exemple implémentation MDC

Cette tâche utilise les templates pour créer des fichiers de configuration remplis, en les plaçant dans le dossier **/{__drupal_folder}/config/sync/** de la VM Front ; le dossier par défaut où la configuration peut être manipulée sous forme de fichiers.

Les fichiers sont donc générés et remplis avec les valeurs issues de l'inventaire, et avec les bons droits.

- Task Chargement des fichiers de conf :

Une fois les fichiers placés au bon endroit, il faut demander à Drupal de les charger dans sa configuration active :

Vous pouvez vous référer à l'exemple de la ZS_CONTINUITY pour plus de détails sur les tâches à rédiger.

- Charger la configuration dans la Base de Données Drupal :

```
/roles/drupal_config/tasks/site_config.yml
```

```
51 - name: 'Drush: Import configuration'
52   ansible.builtin.shell: "{{ drush }} cim -y"
53   args:
54     chdir: "{{ __drupal_folder }}"
55   when: drupal_import_config
```

Figure 20: Exemple cmd drush cim

- Reconstruire le Cache de Drupal :

```
/roles/drupal_config/tasks/site_config.yml
```

```
64 ✓ - name: 'Drush: Clear cache'
65     ansible.builtin.command: "{{ drush }}" cr -y
66     register: tmp_command_output
67     changed_when: tmp_command_output.rc != 0
68
```

Figure 21 : Exemple cmd drush cr

- Task raccordement à la Mire :

Si vous n'avez pas modifié le chemin vers la page de login de Drupal (par défaut /user/login), vous devriez arriver sur la Mire de connexion de MinDefConnect.

Sinon, vous devrez faire pointer le chemin de votre page de connexion vers la Mire de connexion MDC :



Figure 22 : Mire de connexion MDC

4. Vérifier l'architecture de son projet Ansible

L'objectif de cette partie est de vérifier que l'architecture de votre projet Ansible suit quelques règles communes afin de fluidifier le passage de l'environnement PICSEL à C1NP.

Ainsi, la partie « **2. Déployer son SI** » terminée, vous devriez retrouver une architecture de projet similaire à celle présentée ci-dessous :

```
|__ collections
    |__ requirements.yml
|__ inventories :
    |__ production
        |__ group_vars
            |__ all.yml
            |__ database.yml
            |__ front.yml
        |__ hosts.yml
    |__ integration
        |__ group_vars
            |__ all.yml
            |__ database.yml
            |__ front.yml
        |__ hosts.yml
|__ playbooks
    |__ group_vars
        |__ all.yml
        |__ database.yml
        |__ front.yml
    |__ install_database.yml
    |__ install_front.yml
    |__ install_obs.yml
```

collections :

Contient les collections RUCHE liées au projet, déclarées dans le fichier « requirements.yml ».

inventories :

Contient les modèles des inventaires ainsi que les hosts. Les modèles sont répartis selon les groupes d'hosts définis (all, database, front, etc.).

playbooks :

Contient l'ensemble des « playbooks » qui exécutent les différentes actions de déploiement.
Contient également les groupes de variables spécifiques au fonctionnement du SI (dossier /group_vars).

```
|__roles
  |__bdd_config
    |__tasks
      |__main.yml
    |__requirements.yml
  |__README.md
  |__ansible.cfg
  |__site.yml
```

roles :

Contient les « rôles » applicatifs spécifiques au projet (dont la fonction ne peut pas être assurée par une collection)

README.md : Description de l'ensemble du projet, le contenu attendu des variables, les spécificités et le mode d'emploi des rôles. Il complète le manuel d'installation.

ansible.cfg : Contient des paramètres spécifiques à l'exécution du projet.

Ne pas modifier le paramètre « precedence ».

site.yml : « Playbook chapeau » appelant l'ensemble des playbooks du projet.

5. Quelques astuces pour déboguer et faire un ticket Tuleap

5.1 Bonnes pratiques pour déboguer

Il est parfois inévitable de rencontrer des erreurs de différentes formes lors de la réalisation des playbooks Ansible. Quelques indications peuvent vous aider à résoudre ces problèmes plus facilement.

Tout d'abord, des déploiements répétés sur une VM peuvent être à l'origine d'erreurs résiduelles. Il peut donc être pertinent de supprimer la VM concernée par des erreurs et d'en créer une nouvelle afin de repartir sur une base vierge.

Ensuite, il est nécessaire de vérifier que les collections RUCHE et/ou les templates impliqués dans l'erreur soient bien à jour. Pour cela, vous pouvez vérifier qu'il n'existe pas de version plus récente dans les branches Gitlab ou dans le champ version de la collection RUCHE ; il arrive que les erreurs rencontrées soient corrigées par le déploiement d'une nouvelle version.

Il est également possible d'augmenter la verbosité pour obtenir plus d'informations sur l'erreur rencontrée. Cela peut être fait en modifiant le champ verbosité dans le modèle de job sur AAP.

5.2 Faire un ticket sur Tuleap

S'il ne vous a pas été possible de résoudre l'erreur après maintes recherches, vous pouvez faire un ticket **Tuleap** à destination de l'équipe support **Move2Cloud** au lien suivant : <https://forge.intradef.gouv.fr/plugins/tracker/?tracker=8599&func=new-artifact>

Voici quelques indications à ajouter dans le ticket afin qu'il soit traité plus efficacement :

- Recopier l'erreur obtenue
- Indiquer la tâche sur laquelle l'erreur a lieu
- Montrer la section concernée du playbook et les variables ajoutées pour cette tâche
- Préciser la version utilisée de la collection RUCHE ou du template concernés
- Préciser les technologies utilisées par votre SI

N'hésitez pas à ajouter des captures d'écran, pièces jointes ou toutes autres infos qui vous paraissent utiles.

Conclusion

Vous avez terminé le chapitre 5 « **Déployer son projet grâce à Ansible** ». Grâce à celui-ci, vous avez pu remplir les objectifs suivants :

Avancement	Titre de la partie	Description
✓	Utiliser les collections RUCHE Ansible	Comment utiliser les collections clés en main de RUCHE ?
✓	Déployer son SI	Comment déployer son SI de A à Z.
✓	Utiliser MindefConnect	Mettre en place l'authentification MindefConnect.
✓	Vérifier son architecture projet Ansible	S'assurer de suivre quelques règles communes sur la structure de son projet.
✓	Quelques astuces pour déboguer	Vérifier des points clefs lors d'erreur rencontrées et créer un ticket Tuleap.

Maintenant que votre SI est déployé, vous allez pouvoir passer au dernier chapitre « la Promotion du SI ».

Courage, c'est la dernière ligne droite !