
MRS Documentation

Release 0.0

jp

Mar 10, 2021

Contenu:

1	Ressources	3
1.1	Tutoriel MRS	3
1.2	Tutoriel intégration iframe	7
1.3	Glossaire	11
1.4	Opérations	11
2	Index	17
	Index	19

Ceci est le projet de simplification de demande administrative de remboursements en ligne.

- [Documentation](#)
- [Intégration Continue](#)
- [Code Coverage](#)

1.1 Tutoriel MRS

Ce chapitre décrit les diverses manières d'exécuter un serveur HTTP avec le site MRS. Peut être utilisé soit à des fins de développement d'intégration via iframe, soit pour contribuer au code source.

La solution la plus simple est d'utiliser docker, que nous utiliserons dans ce tutoriel.

Note: Si vous ne souhaitez pas utiliser docker vous pouvez vous référer au `Dockerfile` situé à la racine du code source pour configurer votre environnement de développement manuellement, ainsi qu'au fichier `config.yml` dans le dossier `.circleci` pour voir les commandes utilisées dans la pipeline de test.

1.1.1 Démarrer MRS: Container éphémère

On peut démarrer le serveur avec seulement une commande:

```
docker run -e DEBUG=1 --rm -p 8000:8000 -it betagouv/mrs:master mrs dev 0:8000
```

On peut ensuite ouvrir l'URL `http://localhost:8000` pour accéder au site.

Pour éviter de démarrer le projet en mode développement ce qui causerait de certains problèmes de sécurité, l'option `-e` exécute l'image dans un container avec la variable d'environnement `DEBUG` à `1` ce qui équivaut à `True` une fois casté en booléen.

L'option `--rm` efface le container automatiquement après exit de la commande, nous verrons d'autres possibilités dans la section suivante.

L'option `-p` utilisée ici permet d'exposer le port 8000 du container au port 8000 de l'hôte.

Les options `--interactive` et `--tty`, abrégées en `-it` permettent au container de démarrer avec le support du standard input ainsi qu'une émulation de `tty`. Sans ces options impossible d'utiliser une commande interactive.

L'image utilisée ici est `betagouv/mrs` avec le tag `master` qui correspond à la branche `master` du code source, qui est soit déjà utilisée en production, soit en cours de validation chez nous sur l'environnement de pré production: c'est la branche de base.

La commande par défaut de l'image exécute un serveur `uwsgi` utilisé en production. Ici, nous surchargeons la commande par défaut en exécutant la commande qui démarre un serveur de développement, `mrs dev`, avec comme argument `0:8000` pour qu'il écoute sur le port 8000 de toute interface du container.

Note: La commande `mrs` exécute la commande `django manage.py`.

1.1.2 Démarrer MRS: Container persistant

Pour hacker sur MRS, nous recommandons de démarrer l'image sur un container persistant:

```
docker run -e DEBUG=1 --name mrs -d -p 8000:8000 -it betagouv/mrs:master mrs dev ↵  
↪0:8000
```

Par rapport à la commande ci-dessus, non seulement sans l'option `--rm` mais également avec l'option `--name mrs` en plus permet de donner un nom au container afin d'y référer dans d'autres commandes, mais également avec l'option `-d` pour démarrer le container en tâche de fond.

On peut le voir tourner avec la commande `docker ps`:

```
$ docker ps  
CONTAINER ID          IMAGE                COMMAND              CREATED              NAMES                ↵  
↪STATUS              PORTS                ↵  
954e922c3fd8        betagouv/mrs:master "mrs dev 0:8000"    44 minutes ago     Up ↵  
↪15 seconds         6789/tcp, 0.0.0.0:8000->8000/tcp  mrs
```

On peut ensuite exécuter différentes opérations sur le container `mrs`, par exemple:

- `docker start mrs`
- `docker stop mrs`
- `docker restart mrs`
- `docker rm mrs`
- `docker inspect mrs`

On peut ainsi rentrer dans le container avec `docker exec`:

```
docker exec -it mrs bash
```

Ou encore directement créer un nouveau super utilisateur en base de données avec la commande `mrs createsuperuser` ce qui permettra de s'authentifier pour accéder à l'interface d'administration sur l'url `/admin`:

```
docker exec -it mrs mrs createsuperuser
```

Ci-dessus, on exécute donc avec standard input et `tty`, sur le container `mrs`, la commande `mrs createsuperuser`.

Note: toute modification faite dans le container n'est pas sauvegardée dans l'image, donc si vous effacez le container avec `docker rm` et que vous en re-creez un à partir de l'image avec `docker run`, il faudra re-créer votre utilisateur.

1.1.3 Hacker MRS: monter sa branche de code dans le container

Pour hacker MRS, rien de tel que de démarrer le container avec un bind mount du dossier:

```
git clone https://github.com/betagouv/mrs
docker run -v $(pwd)/mrs/src:/code/src -e DEBUG=1 --name mrs -d -p 8000:8000 -it_
↪betagouv/mrs:master mrs dev 0:8000
```

Ainsi, toute modification faite dans le code source sera visible dans le container, et le serveur de développement devrait recharger le python, et toute modification de fichier JS, JSX ou SCSS causera une re compilation des bundles par le watcher webpack.

Danger: Attention cependant, la base de données SQLite de développement se trouve dans le dossier `mrs/src/db.sqlite3`, vous pouvez aussi bien l'effacer et redémarrer le container lorsque vous souhaitez repartir à zéro.

1.1.4 Hacker MRS: tout faire en local

Autrement, il suffit d'une toolchain nodejs et python normale à jour sur son système, avec un utilisateur qui a les droits de création sur postgres.

JavaScript

Installer le paquetage Node yarn avec `sudo npm install -g yarn`, puis et exécuter à la racine du code source qui contient `package.json`:

- `yarn install` pour installer les dépendances dans le dossier `node_modules`, et compiler les bundles webpack,
- `yarn test` pour exécuter les tests,
- `yarn run lint` pour exécuter le linter.

Python

Vous pouvez installer MRS et les dépendances dans `~/.local` avec `pip install --user -e /chemin/vers/mrs`, ensuite vous pouvez exécuter la commande:

```
PATH=~/.local/bin:$PATH mrs dev
```

Cela exécutera un serveur de développement sur `localhost:8000` ainsi qu'un watcher webpack, il faut donc que la commande `yarn install` décrite ci-dessus fonctionne.

Danger: Aussi, cela effectuera automatiquement les migrations de database. En dev, c'est un fichier `db.sqlite3` dans le dossier `src`. N'hésitez pas à l'effacer et à relancer la commande pour le recréer en cas de problème avec la DB.

Postgres

Les tests ont besoin d'une base de données Postgres (notamment pour les jsonfields).

Pour que votre utilisateur shell ait les droits de création et de suppression de tables pendant les tests:

```
sudo -u postgres createuser -drs $USER
# -d: l'utilisateur a le droit de créer des BDs
# -r: il peut créer des rôles
# -s: superutilisateur
# $USER doit etre votre username PAM
```

et tant qu'on y est:

```
sudo -u postgres createdb --owner $USER -E utf8 mrs
```

(et si besoin, voyez dropuser).

Jeu de data de tests

Nous maintenons un jeu de data utilisées par les tests d'acceptance dans `src/mrs/tests/data.json`. Il est censé contenir un minimum de data pour activer un max de use-case.

Pour charger en DB:

```
export DB_ENGINE=django.db.backends.postgresql DB_NAME=mrs DEBUG=1 DJANGO_SETTINGS_
↪MODULE=mrs.settings
sudo -u postgres dropdb mrs
sudo -u postgres createdb -E utf8 -O $USER mrs
mrs migrate
djcli delete contenttypes.ContentType
mrs loaddata src/mrs/tests/data.json
```

Pour sauvegarder la db dans le fichier de data, on veut grosso modo mettre à jour les mêmes modèles, rien de plus facile avec une incantation shell:

```
mrs dumpdata --indent=4 $(grep model src/mrs/tests/data.json | sort -u | sed 's/./
↪*model": "\([^"]*\)",*/\1/') > src/mrs/tests/data.json
```

Vérifier en testant que cela n'impacte pas d'autres jeux de données, tels que `test_mrsstat.json`, auquel cas le supprimer et relancer les tests.

Tests

Pour tester le Python, installer le packaging Python tox avec `pip install --user tox`.

Créer la base de données de test postgres `mrs_test`, puis lancez les migrations (`mrs migrate`) en spécifiant bien le nom de la BD et le type de la BD en variables d'environnements: `DB_NAME=mrs_test DB_ENGINE=django.db.backends.postgresql` (voir le fichier `tox.ini`).

Enfin, exécuter à la racine du code source qui contient `tox.ini`:

- `PATH=~/.local/bin:$PATH tox -e qa` pour lancer l'analyse statique
- `PATH=~/.local/bin:$PATH tox -e py36-dj20` pour exécuter les tests dans un environnement python 3.6 avec Django 2.0.

Tox fera le baby sitting des environnements dans le dossier `.tox`, par exemple dans le dossier `.tox/py36-dj20` l'environnement `-e py36-dj20`.

En outre, les tests exécutés par notre pipeline sont définis dans `.circleci/config.yml`.

1.2 Tutoriel intégration iframe

1.2.1 Introduction

Vous mettez un site internet a disposition de vos patients et vous souhaitez leur faciliter l'usage du formulaire MRS. MRS vous propose:

- d'uploader leur *PMT*,
- d'afficher le formulaire pré rempli (Nom, Prénom, *NIR*, email, date de naissance) dans une iframe,
- de recevoir l'*UUID* de la demande lorsque l'utilisateur valide le formulaire,
- de récupérer le statut d'une demande (Nouvelle, Validée, Refusée).

1.2.2 Pre-requis

Pour le développement, vous aurez peut-être besoin de démarrer un serveur MRS en local ou sur votre environnement de preprod. Pour cela, merci de vous référer au tutoriel en question.

Il vous suffit de contacter MRS et de leur envoyer:

- votre numéro *FINESS*,
- l'URL a autoriser en headers *CORS*, ou alors si vous voulez utiliser le hack pour l'"autorisation *CORS* dynamique".

1.2.3 CORS crash course

Vous pouvez toujours exécuter une requête AJAX, cependant le navigateur décidera si oui ou non il permet votre appel d'utiliser la réponse de la ressource HTTP en question.

Avant toute chose, le navigateur fait une requete de "preflight" *CORS*, faisons quelques tests ensemble pour comprendre le fonctionnement:

```
$ curl \
  -H "Origin: http://localhost:9000" \
  -H "Access-Control-Request-Method: POST" \
  -H "Access-Control-Request-Headers: X-Requested-With" \
  -X OPTIONS \
  -I -v \
  'http://localhost:8000/institution/310000000/mrsrequest/iframe/?origin=http://lol/
↪test'
```

```
X-Frame-Options: ALLOW-FROM http://lol/test
Access-Control-Allow-Origin: *
```

Si vous n'avez pas activé l'option de *CORS* dynamique, alors MRS n'autorise que l'URL définie par les admins et est insensible au paramètre origin:

```
X-Frame-Options: ALLOW-FROM https://example.com/iframe.html
Access-Control-Allow-Origin: https://example.com
```

Le choix vous appartient donc. A noter que toute mesure de sécurité qui n'est pas prise dès le départ en coûtera si un jour MRS décide de ne plus autoriser le CORS dynamique. Nous conseillons de prendre ses dispositions et partir sur des bonnes pratiques dès le départ, plutôt que de prendre un crédit technique sur la question.

1.2.4 Configurer l'instance de dev

Pour autoriser l'affichage de l'iframe, connectez vous sur l'interface d'administration de votre instance de MRS, et ajoutez un "Établissement" avec comme numéro de *FINESS* le "310000000" et cochez la case "Autorisation *CORS* dynamique".

Note: En production vous pouvez contacter MRS pour indiquer l'URL que vous voulez autoriser en CORS pour afficher votre iframe.

The screenshot shows the MRS Admin interface. At the top, there is a dark blue header with the text "MRS Admin" in yellow. Below the header is a light blue breadcrumb trail: "Accueil > Établissement > Établissements > 310000000". The main content area is titled "Modification de Établissement". There are three main sections: 1. "Finess :" with a numeric input field containing "3100000" and up/down arrow buttons. 2. "Origin :" with a text input field containing an asterisk (*). Below this field is the label "URI du site patients". 3. A checkbox labeled "Autorisation CORS dynamique" which is checked. Below the checkbox is the text "Cocher pour les hebergeurs non-HDS qui ne veulent pas de controle d'origine".

Vous pourrez ensuite ouvrir:

```
http://localhost:8000/institution/310000000/mrsrequest/iframe/
```

Passez l'URL que vous voulez utiliser dynamiquement grâce au paramètre GET "origin":

```
$ curl -I http://localhost:8000/institution/310000000/mrsrequest/iframe/?origin=http://localhost:8000/your/url
X-Frame-Options: ALLOW-FROM http://localhost:8000
Access-Control-Allow-Origin: http://localhost:8000
```

1.2.5 Afficher l'iframe

Vous pouvez afficher l'iframe soit depuis l'URL configurée dans l'admin pour votre *FINESS*, soit en passant le paramètre "origin" en paramètre GET, à condition que la cache "Autorisation CORS dynamique" soit cochée pour ce *FINESS*.

Vous pouvez également passer les paramètres GET suivants:

- first_name, le prénom,
- last_name, le nom de famille,
- birth_date, la date de naissance, exemple 2000-12-31,
- nir, le numéro de sécurité sociale,
- email, l'email de l'assuré,
- hidePMT=1, pour cacher le champs d'upload de *PMT* du formulaire initial si vous comptez la fournir à la place de l'assuré,

Example

Note: Si le *CORS* dynamique est actif alors vous devrez passer le paramètre origin.

```
<iframe
  id="mrsrequest"
  src="https://www.mrs.beta.gouv.fr/institution/310000000/mrsrequest/iframe/?
  ↪origin=http://votreserveur/iframe.html&first_name=Test%20Étienne&last_name=∞&birth_
  ↪date=2000-12-31&nir=1234567890123&email=exemple@exemple.com"
  width="100%"
  height="800"
  style="border: none"
></iframe>
```

Résultat:

1.2.6 Uploader la PMT

Ci dessus, le champs d'upload de *PMT* est affiché dès le départ dans le formulaire pour que l'utilisateur puisse l'uploader. Ici, nous allons traiter le cas dans lequel nous avons le fichier et nous voulons le soumettre à la place de l'utilisateur pour lui faciliter la demande. Nous cacherons donc le champs en ajoutant le paramètre GET `hidePMT=1`.

Et nous passerons l'URL de la *PMT* à l'iframe dans un message. Dans ce cas, c'est l'iframe qui va télécharger le fichier et l'uploader ensuite pour la demande ouverte dans l'iframe. Exécutez un tel appel avec une fonction de ce type:

```
function uploadPMT() {
  document.getElementById('#iframe').contentWindow.postMessage(
    // votre pmt
    '{"pmt_url": "http://www.mrs.beta.gouv.fr/institution/example.jpg"}',
    // origine de l'iframe, peut aussi etre '*'
    'https://www.mrs.beta.gouv.fr'
  )
}
```

Si l'upload échoue pour une raison ou pour une autre, le formulaire ne validera pas et demandera a ce moment la a l'utilisateur d'uploader sa *PMT* dans le formulaire.

Danger: Pour telecharger le fichier, l'iframe a besoin que l'URL passée réponde avec le header `Access-Control-Allow-Origin` correspondant a l'URL de l'instance de MRS in question.

1.2.7 Recevoir l'UUID de demande

C'est l'iframe qui enverra l'*UUID* de la demande via un message lorsque le formulaire sera validé. Vous pouvez le recevoir par exemple en exécutant ce code avant que l'utilisateur ne soumette le formulaire avec succès:

```
function receiveMessage(event) {
  console.log('mrsrequest_uuid:', event.data.mrsrequest_uuid)
}
window.addEventListener('message', receiveMessage)
```

1.2.8 Recuperer le statut de la demande

Pour récupérer le statut d'une demande, il suffit d'appeler l'URL de statut de demande avec votre *FINESS*, par exemple:

```
$ curl https://mrs.beta.gouv.fr/institution/310000000/mrsrequest/470f3dbe-1f0a-4dfc-
↪8bf1-95f8d504deb3/status/
{"status": 0}
```

Les codes de statut sont:

- 0: Soumise
- 100: Annulée par l'assuré
- 999: Rejetée
- 1000: En cours
- 2000: Validée

Note: Si le *CORS* dynamique est actif alors vous devrez passer le parametre origin.

1.3 Glossaire

CORS Acronyme de [Cross Origin Resource Sharing](#). C'est un protocole de couche supérieure au HTTP qui permet au navigateur de vérifier si une réponse est partageable avec une [origine](#). Une mauvaise compréhension du CORS mènera a des problèmes incompris lors de l'intégration de l'iframe.

PMT Allez voir la definition sur le [Wiki](#)

FINESS Allez voir la definition sur le [Wiki](#)

UUID Chaîne de caractères servant d'identifiant unique à chaque demande en interne.

NIR Allez voir la definition sur le [Wiki](#)

1.4 Opérations

Ce chapitre vise les DevOps en vue de maintenir un déploiement de MRS automatisé, comme c'est le cas pour la production, staging, ecole, ainsi que pour les déploiements de dev ephemeres faits a partir des branches.

Ce chapitre a donc vocation d'expliquer tous les types de déploiements possibles ainsi que les opérations courantes, plutot dans la pratique tel un "cookbook" qu'un document expliquant les concepts et les pourquoi, sujet traité globalement dans un article sur l'[eXtreme DevOps](#) et le [hacking d'operations avec ansible/bigsudo/docker/compose](#) ainsi que la *distribution de roles Ansible yourlabs* <<https://galaxy.ansible.com/yourlabs/>>.

1.4.1 Architecture systeme

Utilisateurs

Il est considéré que vous avez un accès avec le meme nom d'utilisateur, sudo sans mot de passe et une authentification SSH par clef, vous pouvez vous en assurer avec la commande suivante que j'utilise:

```
bigsudo yourlabs.ssh @host
```

Vous pouvez m'ajouter en root sur un serveur en recuperant ma clef ssh sur github.com/jpic.keys et m'ajouter en root si vous avez besoin d'aide ou alors ajoutez moi avec la commande suivante:

```
bigsudo yourlabs.ssh adduser usergroups=sudo username=jpic
```

Partitions

/home Les données persistentes des utilisateurs comme des instances MRS déployées sur le serveur.

/home/nomdinstance Données persistentes d'une instance de MRS, exemple: `/home/mrs-production:` pour la prod sur le serveur de prod

/var/lib/docker Docker est vraiment lent si il ne peut pas exploiter le CoW, monter une partition btrfs sur ce dossier le rend effectivement plus rapide.

/mnt/backup Sur le serveur de prod, un espace sur un autre RAID pour stocker les backups.

Services

Les docker-composes contiennent les annotations nécessaires pour que Traefik sache générer les certificats HTTPS et router les domaines vers le container Django/uWSGI.

Exemple de commande qui installe traefik (ainsi que docker et un firewall):

```
bigsudo yourlabs.traefik @host
```

Vous avez accès aux instances de NetData et de Prometheus pour accéder au monitoring en passant par `netdata.fqdn` ou `prometheus.fqdn` (remplacez `fqdn` par le nom de domaine complet du serveur). Exemple de commande pour installer `netdata` et `prometheus`:

```
bigsudo yourlabs.netdata @host
```

Notez que NetData est configuré pour alerter l'équipe via un webhook Slack (ChatOps) dans `/etc/ansible/facts.d/yourlabs_netdata.fact`.

1.4.2 Architecture MRS

Compose

Docker-compose préfixe les containers et volume d'une installation à partir d'un préfixe. Pour docker-compose, les containers et volumes d'une stack partagent préfixe soit passé à compose avec `--project-name`, soit c'est le nom du dossier qui contient le fichier compose (comportement par défaut).

La différence entre un déploiement persistant et un déploiement de dev réside principalement là:

- pour un env persistant (prod, staging ...), on passe un `home=/home/mrs-production` et on fusionne `docker-compose.yml` avec `docker-compose.persist.yml`,
- pour un env éphémère (branche dev), on passe `project=test-$BRANCHNAME`, et `yourlabs.compose` garde automatiquement une version dans `~/.yourlabs.compose/test-$BRANCHNAME/docker-compose.yml` en vue de gérer l'instance plus tard.

En plus, le déploiement éphémère se fait avec l'argument `lifetime=604800` ce qui crée un fichier `~/.yourlabs.compose/test-$BRANCHNAME/removeat` contenant la somme de l'argument et du timestamp. `yourlabs.compose` installe également un timer `systemd` pour effacer les vieux déploiements éphémères.

Dans tous les cas, si le serveur a un load-balancer fonctionnel (déploiable avec `bigsudo yourlabs.traefik` ou manuellement), alors on veut aussi fusionner `docker-compose.traefik.yml`.

Enfin, utile pour les envs de dev et staging, on peut aussi fusionner `docker-compose.maildev.yml` pour avoir un serveur de mail de test.

Docker

Le Dockerfile de MRS construit une image de container avec webpack pour compiler le front et uWSGI pour servir le code Python.

Persistence

Les données sont persistées dans un dossier dans `/home` tels que `/home/mrs-production` ou `/home/mrs-staging`. On y trouve:

- `./backup.sh`: le script pour faire une backup,

- `./restore.sh`: le script pour restaurer une backup,
- `./prune.sh`: le script pour appliquer la politique de retention de backup,
- `./log`: logs django, inclus dans les backups,
- `./postgres`: les données de postgres
- `./spooler`: dossier de background jobs uWSGI,
- `./restic`: dossier qui contient le repo de backups, sur le serveur de production c'est dans `/mnt/backup/mrs-production/restic` pour éviter de stimuler une copie de la DB sur les disques de runtime.

Cron

systemd.timer est utilisé en guise de cron, MRS en deploie pour chaque instance persistente (voir section suivante "Compose"):

- `nomdinstance-backup`: execute une backup de nuit
- `nomdinstance-prune`: execute la politique de retention de backups

1.4.3 Operations courantes

Il faut un acces sudo sans mot de passe sur l'un des serveurs pour pouvoir effectuer l'une de ces operations.

Ajouter un utilisateur sudo

Pour ajouter un utilisateur en sudo sans mot de passe, avec son nom d'utilisateur github, et la clef ssh publique correspondante a cet utilisateur sur github:

```
bigsudo yourlabs.ssh adduser usergroups=sudo username=github_username @mrs.beta.gouv.fr
↪fr @staging.mrs.beta.gouv.fr
```

Pour choisir un nom d'utilisateur ou clef qui n'est pas sur github:

```
bigsudo yourlabs.ssh adduser usergroups=sudo username=your_username key=https://
↪gitlab.com/your_gitlab_username.keys @mrs.beta.gouv.fr @staging.mrs.beta.gouv.fr
```

Après vous pouvez bien entendu le faire manuellement a l'ancienne, mais perso je trouve cette maniere plus rapide car elle encapsule des operations autrement repetitives.

Envoyer un mail de test

Typiquement pour tester la configuration du serveur de mail:

```
docker-compose exec django mrs sendtestemail
```

Copier les données de prod en staging

Cette opération se passe en deux temps:

- la copie des données d'une base de données à l'autre à travers ssh
- l'execution du script d'anonymisation des données, car staging n'a pas vocation d'etre particulièrement protegée

```
ssh -A staging.mrs.beta.gouv.fr
```

1.4.4 Developpement

Local

Pour executer la meme operation de deploiement et d'installation de prod en local, en vue de la bidouiller, sans le load-balancer.

Du coup, on va pas mal tordre l'execution qui est faite en CI dans cet objectif:

```
export LFTP_DSN=ftp://localhost
export RESTIC_PASSWORD=lol
export RESTIC_REPOSITORY=/tmp/backup/mrs-production-restic
export POSTGRES_BACKUP=/tmp/backup/mrs-production-postgres
export SECRET_KEY=notsecret
export BASICAUTH_DISABLE=1
export HOST=localhost:8000
export ALLOWED_HOSTS=127.0.0.1,localhost
bigsudo ansible/deploy.yml home=/tmp/testmrs compose_django_ports='["8000:8000"]'
↪compose_django_build= compose_django_image=betagouv/mrs:master compose=docker-
↪compose.yml,docker-compose.persist.yml
```

LFTP_DSN Le DSN de connection a passer a LFTP pour qu'il upload les backups chiffrées

RESTIC_PASSWORD Le mot de passe de chiffrement de backups

RESTIC_REPOSITORY Le chemin vers le repo de backups

POSTGRES_BACKUP Le chemin dans lequel postgres doit dumper ses data

SECRET_KEY La clef secrete avec laquelle les mots de passes sont chiffrés

ALLOWED_HOSTS La liste des hostnames que le serveur est censé accepter. Toute requete recue par le serveur dont le host name ne correspond pas prendra direct une 403.

HOST Le host que le healthcheck doit verifier.

BASICAUTH_DISABLE Desactiver le HTTP Basic Auth sur ce deploiement, a noter que le Basic Auth se base sur les utilisateurs en base de données.

bigsudo Le generateur de ligne de commandes Ansible, a installer avec pip

ansible/deploy.yml C'est le script de deploiement en ansible

home=/tmp/testmrs Que le deploiement persiste dans ce dossier (en prod: /home/mrs-production)

compose_django_ports='["8000:8000"]' Cela permet au deploiement d'etre utilisable sans load balancer en l'exposant sur le port 8000 de localhost

compose_django_build= Annule la configuration de build: on ne veut pas qu'il essaye de builder en production

compose_django_image=betagouv/mrs:master Image a deployer: vu qu'on ne veut pas la builder en prod

compose=docker-compose.yml,docker-compose.persist.yml Liste des fichiers compose a fusionner pour la configuration finale de ce deploiement

Vagrant

Vagrant necessite VirtualBox, c'est une alternative au developpement local qui permet d'avoir la meme version que centos sur les serveurs, et vous permet de tester sans ajouter des crons inutiles sur votre systeme local grace a l'isolation d'une machine virtuelle.

Pour allger le workflow de commandes, des raccourcis sont mis dans le script `./do` a la racine du repo:

- `./do vagrant apply` enchaîne un `destroy` et un `up`
- `./do vagrant up` fait un `up` et puis genere la config ssh dans `./.vagrant-ssh`
- `./do vagrant bigsudo` se substitue a `bigsudo` et passe `./.vagrant-ssh` en argument `ansible` et vise bien la VM default

Exemples:

```
./do vagrant bigsudo yourlabs.traefik -v
./do vagrant bigsudo ansible/deploy.yml
```


CHAPTER 2

Index

- genindex
- modindex
- search

C

CORS, **11**

F

FINESS, **11**

N

NIR, **11**

P

PMT, **11**

U

UUID, **11**