Documentation EMS - partie Exploitation

youen froger

May 2023

1 Introduction

La documentation est séparée en deux parties (Développeur et Exploitation), dont les sujets sont répartis ainsi :

1.1 Doc exploitation

- Comment est constitué l'EMS : description générale des modules et de leur fonctionnement logique (simplifié)
- Sources de données de l'EMS format et dispo / où mettre les droits d'accès
- Sorties de l'EMS par ordre d'importance et leur signification
- Fonctionnement nominal (chemin macro) de l'EMS et résilience
- Installation et mise en route de l'EMS (+ contraintes sur les appareils et le contexte technique)
- Exploitation (paramètres et robustesse/fragilité du code) = A FAIRE
- FAQ utilisateurs/exploitant + Résolution des problèmes = A FAIRE

1.2 Doc développeur

- Comment est constitué l'EMS : description générale des modules et de leur fonctionnement logique (plus précis)
- Doc solveur simplifiée : détail des consommateurs, fonction objectif...
- Chemin de la donnée plus précis + comment rajouter de nouvelles sources.
- Comment rajouter un nouveau type de consommateur + Préciser les choix/arbitrages faits pour les différents consommateurs => *à renseigner par l'ENS*
- Comment modifier la source des prévisions d'entrée pour ELFE
- Comment relancer un scénario passé
- Compléments sur les sources de données
- FAQ développeur

2 Sources de données de l'EMS

2.1 Sources

2.1.1 Base de données de coordination

Il s'agit de la base de données PostgreSQL qui contient tous les appareils pilotés par ELFE ainsi que les contraintes des utilisateurs. L'EMS va récupérer à date anniversaire les informations dans les différentes tables afin de savoir si il doit planifier la consommation d'un appareil ou non.

2.1.2 Zabbix

Il s'agit d'une base de données PostgreSQL qui contient les consommations et productions des différents appareils connectés au projet ELFE. Ces données sont historisées et exposées par un service Zabbix qui fonctionne dans une machine virtuelle dédiée. L'EMS va venir récupérer ici les différentes consommations des appareils type "machine à laver" pour pouvoir apprendre leurs consommations au moyen du service "machine_cycle_learner" et stocker les résultats dans une base de données interne. La température actuelle des chauffages asservis est également acquise depuis cette base de données. De plus, le service de prévision de consommation - production va également interagir avec cette source de données afin de faire la prévision par persistance. Le mécanisme de persistance est décrit dans la section qui traite du service power_prediction

2.1.3 Meteo_concept

Il s'agit d'une API web qui permet de récupérer des données de prévision à l'échelle de l'heure pour les 12 prochaines heures, puis à l'échelle du quart de journée. Le service Meteo_concept est chargé de faire cette acquisition et de stocker les données dans la table initialweather de la base de données interne de l'EMS. Chaque prédiction récupérée est également historisée dans la table historyinitialweather de la base de donnée interne. Voici les données entrantes :

- weather_timestamp : le timestamp unix auquel la prévision est censé se réaliser
- forecast_timestamp (historique uniquement) : le timestamp unix auquel la prévision a été récupérée sur météo concept.
- temperature : prévision de température à 2m en Kelvin.
- wind_speed: prévision de la vitesse du vent en km/h à 10m de hauteur.
- gust_speed: prévision de la vitesse du vent pendant les rafales en km/h à 10m de hauteur.
- wind_direction: prévision de la direction du vent en ° de 0 à 360 à 10m de hauteur
- sun_hours : non récupérées pour le moment, la durée total d'ensoleillement de la journée.

Actuellement, seules les données de températures sont effectivement prises en compte pour les chauffages asservis.

2.1.4 Base de données interne de l'EMS

La base de données interne de l'EMS a pour rôle principal celui de tampon qui permet le transfert d'information entre les différents services. Cela permet entre autre de parer à l'éventualité que l'une ou l'autre des sources de données se tarisse temporairement. Un autre avantage de cette architecture est de permettre de remplacer les sources par des nouvelles, et tant qu'elles produisent des données dans les mêmes champs, les autres services continueront de fonctionner normalement. Il est donc possible de substituer à MétéoConcept une source de données jugée plus fiable à l'avenir sans avoir à modifier quoi que ce soit d'autre que le service qui écrit dans cette table. Voici la liste des tables internes à l'EMS et de leur rôle.

- machine, cycle et cycledata : ces tables contiennent la représentation interne d'une machine, identifiée par l'ID Zabbix de la partie puissance de l'équipement de mesure électrique. La table machine contient également des informations importantes pour l'acquisition du cycle de la machine décrit plus bas dans ce document
- devicetemperaturedata : ici est stockée la dernière température connue (en 100e de K) de chaque chauffage asservi connu, de concert avec le timestamp unix d'acquisition et l'ID Zabbix de l'équipement de mesure de température.
- ems_ecs : il s'agit ici de stocker la dernière consommation d'énergie (en Wh) sur 24h connue d'un ballon ECS afin de pouvoir planifier sa prochaine consommation. Plus de détails dans la section {section}. Les consommations sont indexées par l'ID Zabbix de la partie mesure de puissance de l'appareil de mesure de consommation du ballon ECS

- history_curve / history_model / history_prediction : Il s'agit de la sortie d'un module optionnel qui permet d'historiser des courbes de prédiction. Ce module optionnel a été réalisé afin de pouvoir comparer différentes méthodes de prédiction. Cela permet d'avoir pour une même courbe différentes méthodes de prédiction.
- initialweather/historyinitialweather : voir section meteo concept
- prediction: Il s'agit de la prévision de production - consommation utilisée par l'EMS. Elle est sous la forme : data_timestamp (timestamp unix auquel le point est valide) et power (prévision de la surproduction prévue, en W, initialement prévue pour être le déficit de production)
- ems_modele_thermique: Cette table contient les différentes valeurs des différents modèles thermiques.
- result / result_ECS : il s'agit de la sortie de l'EMS qui lui permet de savoir si il doit ou non planifier un appareil

2.2 Droits d'accès

2.2.1 Base de données de coordination

Les données des droits d'accès sont stockées dans le fichier db_credentials.py du module credentials, dans le champs "ELFE". Elles se séparent en 5 champs:

- "host": l'adresse ip/ le nom auquel l'EMS peut accéder à la base de données
- "database" : le nom de la base de données dans laquelle la base de donnée de coordination réside
- "user" : le nom d'utilisateur de l'EMS pour la base de données de coordination.
- "password" le mot de passe de l'utilisateur de l'EMS pour la base de données de coordination.
- "options" : ce champ est optionnel si le schéma est "public", sinon il doit contenir : "-c search_path=nom_du_schema,public"

2.2.2 Zabbix

Les données des droits d'accès sont stockées dans le fichier zabbix_credentials.py du module credentials Ce fichier initialise une variable zabbix_credentials avec un objet qui contient les champs suivants :

- "username" : le nom d'utilisateur pour l'EMS dans la base de données Zabbix
- "password": le mot de passe pour l'utilisateur de l'EMS dans la base de données Zabbix.
- "url": l'url du serveur Zabbix.

2.2.3 Meteo_concept

Les données des droits d'accès sont stockées dans le fichier meteo_concept_credentials.py du module credentials. Ce fichier initialise une variable avec pour valeur la clef meteo_concept (voir https://api.meteo-concept.com)

2.2.4 Base de données interne de l'EMS

Les données des droits d'accès sont stockées dans le fichier db_credentials.py du module credentials, dans le champs "EMS". Elles se séparent en 4 champs:

- "host": l'addresse ip/ le nom auquel l'EMS peut accéder à la base de données
- "database" : le nom de la base de données dans laquelle la base de donnée de coordination réside
- "user" : le nom d'utilisateur de l'EMS pour sa base de données interne.
- "password" le mot de passe de l'utilisateur de l'EMS pour sa base de données interne.

3 Sorties de l'EMS

Les sorties de l'EMS ont lieux dans la base de données pointée par le champ ems_sortie du fichier de configuration db_credentials.py du module credentials.

L'EMS publie une planification pour la prochaine journée des consignes pour chacun des appareils qui ont demandé à être planifiés. La planification est publiée dans la table result et est sous la forme suivante :

- Une ligne par appareil, par planification et par méthode de planification, qui contient
  - Un id qui sert de clef primaire (id)
  - Le timestamp à partir duquel le plan devrait être appliqué (first_valid_timestamp).
  - L'id de la machine considérée (machine_id)
  - Le type de résultat (heuristique ou simulation complète, cela n'a pas été implémenté, result_type)
  - Le type de machine (machin_type)
  - Les 96 décisions pour les prochaines 24h au pas du 1/4 d'h (decisions_0 jusqu'à decisions_95)
- On pourrait avoir deux EMS en parallèle en changeant la valeur du champs de la méthode de planification et choisir celui dont on applique le résultat. Ce champ est rempli par la valeur qui se trouve dans le fichier de configuration de l'EMS. Ce mécanisme n'a pas été testé complètement, mais a priori les deux EMS pourraient cohabiter avec la même base de données interne.

L'EMS publie également deux courbes : la courbe de production - consommation initialement prévue avant le placement des consommateurs flexibles (sans les inclure) (p_c_without_flexible_consumption) et celle après placement des consommateurs flexibles (p_c_with_flexible_consumption). Ces deux courbes sont stockées au fil de l'eau dans les tables susmentionnées et dont les champs sont data_timestamp (le timestamp à partir duquel la valeur est valide) et power (la valeur de la prédiction en W)

Finalement l'EMS publie un rapport d'exécution qui est stocké dans la table ems_run_info et qui contient :

- Un champs timestamp qui correspond au timestamp unix du round simulé par l'EMS (Si l'EMS est lancé à 14h02, donc planifie les machines pour 14h15, ce timestamp correspondra à 14h00 car c'est la date théorique de lancement de l'EMS).
- un champs run_time_ms qui contient le **temps de résolution du problème uniquement**, pas le temps de récupérer les données, de le formuler et de les poster.
- un champs consumer_count qui contient le nombre de machine qui ont été placées par l'EMS durant cette simulation
- des champs conso_min_hour et conso_min_hour_timestamp (respectivement conso_max_hour et conso_max_hour_timestamp) qui contiennent la valeur (en W) de la prévision de déficit minimal (respectivement maximal) de production sur un heure et le timestamp associé.

4 Description générale des modules, des services et leur fonctionnement

L'EMS est réparti entre différents modules python et services. Les modules sont des bibliothèques de fonctionnalités qui peuvent être partagées entre différents services alors que les services sont différents programmes qui sont lancés périodiquement.

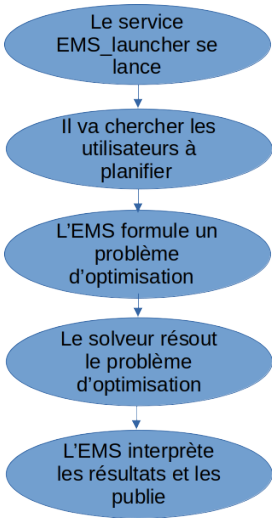


Figure 1: Chemin d'execution du service EMS_launcher, central à l'EMS

4.1 Modules

Pour ce qui est des modules, les voici :

- database: ce module continent la description des différentes bases de données et tables ainsi que les fonctionnalités pour s'y connecter et échanger des données avec.
- solution: ce module contient le solveur python ainsi que les différentes classes de consommateurs afin de pouvoir formuler et résoudre le problème d'optimisation dont les résultats correspondent au placement optimal des consommateurs flexibles afin de minimiser l'import. Les 5 consommateurs sont :
  - Machine générique : basé sur un cycle appris à l'avance.
  - ECS (eau chaude sanitaire) : prise en compte de l'énergie nécessaire à la chauffe complète, tout en calculant la prévision sur la base de la consommation de la veille, et gestion d'une période d'heures creuses.
  - Voiture électrique : apprentissage de la courbe et modélisation de la dynamique de charge, pour gérer une charge partielle.
  - Chauffage non-asservi : pilotés par ELFE au niveau du fil pilote (mode éco ou confort). On se donne de la flexibilité via un "ratio" : temps confort sur période de chauffe souhaitée.
  - Chauffage asservi : lié à un modèle thermique, on suit la température en pilotant directement la puissance de chauffe.
- elfe_interfaces: ce module contient deux fichiers :
  - ELFE_Database_populator qui était chargé de peupler une réplication locale de la base de données de coordination pour tests. Ce fichier n'est plus maintenu et ne devrait pas être utilisé
  - ELFE_data_gatherer qui est chargé de récupérer les données depuis la base de données de coordination et de les traduire sous forme de contraintes que le module Solution comprend
- services : ce module continent les programmes python et les fichiers systemd qui servent au fonctionnement des différents services.
- learning: ce module contient les fonctionnalités d'apprentissage de l'EMS. Il continent notamment la fonctionnalité d'apprentissage des cycles des machines de l'utilisateur, ainsi que le sous-module d'interfaçage avec zabbix.

- predict: ce module contient les fonctionnalités de prédiction de l'EMS. Il contient notamment la fonctionnalité de prédiction par persistance et celle qui permet de récupérer les données météo.
- logger: ce module contient la fonction qui permet de sauvegarder les conditions initiales d'un problème afin de pouvoir le relancer ensuite avec une courbe de consommation différente.
- utils: ce module contient des fonctionnalités qui n'ont pas de module propre. Il contient notamment dans un sous-module time le calcul de la date actuelle, du minuit de la date actuelle et d'autres fonctionnalités utiles.
- data: ce module n'est en réalité qu'un dossier qui sert à héberger les cycles appris des machines et les conditions initiales dans lesquelles le calcul a été lancé. Ces informations contiennent notamment l'heure à laquelle le calcul a été lancé, l'heure à laquelle il devra être appliqué et la courbe de $consommation - production$ qui a été prise en compte.
- config: ce module contient les valeurs de configuration utiles pour l'utilisation de l'EMS.
- credentials: ce module contient les différentes clefs et mots de passe nécessaires au fonctionnement de l'EMS.
- bodge: ce module contient le code qui n'est là que pour adapter la sortie de l'EMS lorsque la sortie brute de l'EMS ne permet pas le pilotage direct des équipements. Ce n'est actuellement le cas que pour l'eau chaude sanitaire (ECS).
- cli_tools: ce module contient les outils et programmes jugés pertinents pour l'utilisation de l'EMS. Il contient actuellement un outil utile pour customiser la configuration de l'apprentissage des cycles des machines.
- tests: ce module contient les tests unitaires de l'EMS.

4.2 Services

Pour son fonctionnement, l'EMS ajoute des services *systemd* qui se lancent avec une certaine périodicité. Ces services ont pour rôle d'assurer l'acheminement et le traitement des données. Les fichiers chargés de lancer ces services sont stockés dans le dossier "systemd_files" du module "services" de l'EMS. Lors de l'installation, un fichier de configuration de l'environnement de lancement des services est créé dans le dossier /etc/ems. Ce fichier se nomme "EMS_ systemd_config.txt". Chacun de ces services lance un fichier python du module service dont le nom correspond au nom du service, à quelques exceptions près. En voici une description par ordre d'importance :

- EMS_launcher : Ce service se lance tous les quarts d'heure (14m30s, 29m30s, 44m30s et 59m30s ± 20s) et est chargé de la formulation, de la résolution et de la publication des résultats du problème d'optimisation. Il faut voir cela comme la partie centrale de l'EMS, qui optimise le placement des consommations des appareils flexibles afin de minimiser l'import requis. Ce service va chercher les appareils qu'il doit piloter dans la base de données de coordination, détermine ceux qui doivent être planifiés (voir section 4.3) et les ajouter au solveur pour ce round. Il va également récupérer les informations relatives à la prévision de déficit de production dans la table prediction (en prenant l'opposé de la donnée stockée) de la base de donnée interne (voir 2.1.4), les cycles des machines concernées (voir 2.1.4), les dernières consommations connues des ECS dans la table ecs_ems (voir 2.1.4), les dernières températures connue des chauffages asservis pilotés (voir 2.1.4), les résultats de la dernière planification (voir 3) et la prédiction météo de température (voir 2.1.3).
- Power_prediction : Ce service se lance tous les quarts d'heure (14m, 29m, 44m, 59m ± 20s) et est chargé de la prédiction de la courbe de "production - consommation" qui sera utilisée par l'EMS et stocké dans la table prediction (voir 2.1.4)
- Meteo_concept : Ce service se lance toutes les heures à la 58e minute $\pm 20s$. Ce service lance actuellement le fichier meteo_concept_gatherer.py du module predict. Son rôle est de récupérer une prédiction météorologique depuis l'api météo concept (voir 2.1.3), de constituer une prédiction météo et de l'historiser (voir 2.1.3).

- machine_cycle_learner : Ce service se lance toutes les heures à la 58e minute ±20s. Ce service lance le fichier machine_cycle_learner.py du module learning. Son rôle est de récupérer depuis zabbix les consommations des équipements pilotés de type machine et de faire l'acquisition de leur dernier cycle de consommation connu. Pour se faire, le service va utiliser des valeurs liées à l'équipement dans la table interne "machine" de l'EMS pour détecter et acquérir un cycle de consommation en utilisant la méthodologie définie dans 8.1. L'acquisition crée un nouveau fichier csv dans le dossier in_use du module data, copie l'ancien dans le dossier old et le marque pour suppression. Il est donc important de sortir et de nettoyer le dossier old de temps en temps.

- ECS_acquisition: Ce service se lance tous les jours à 16h00m30s ± 20s. Son rôle est de récupérer la consommation de l'ECS des participants sur les dernières 24h afin de pouvoir planifier leur consommation. Ce service s'appuie sur la partie "energie" des appareils de mesure pour déduire la quantité d'énergie consommée et va l'écrire dans la table ECS_EMS de la base de données interne de l'EMS (voir 2.1.4). **Attention :** ce service dépend du nommage des éléments dans l'outil Zabbix. Il peut faire le calcul de l'énergie consommée par différence de l'élément énergie, ou par intégration de l'élément puissance lorsque l'énergie n'est pas disponible. Il a donc besoin que l'élément soit correctement taggué "appareil:ECS" et que le nom (*item name*) se termine par "_energie", "_puissance", " energie" ou " puissance" (recommandation de ma part d'utiliser l'espace, je ne suis pas certain que les _ ne provoquent pas de bugs car je n'ai pas eu l'occasion de le tester).

- user_temp: Ce service se lance toutes les minutes et va récupérer depuis zabbix la dernière valeur de température connue pour tous les chauffages asservis. Il stocke cette valeur dans la table devicetemperaturedata de la base de données interne de l'EMS (voir 2.1.4)

4.3 Règles de prise en compte des différents appareils par l'EMS

Le champs equipement_pilote_ou_mesure_mode_id de la table equipement_pilote_ ou_mesure doit avoir la valeur qui correspond au mode pilote (30). Ensuite, par catégorie :

- Le champs id de la table equipement_ pilote_ ou_mesure correspond à une valeur de equipement_ pilote _ou_mesure_id dans la table equipement_pilote_ machine_générique, auquel cas on planifie la machine si et seulement si la planification précédente n'a pas planifiée la machine pour démarrer au début de ce pas de temps

- Le champs id de la table equipement_pilote_ou_mesure correspond à une valeur de equipement _pilote_ou_mesure_id dans la table equipement_pilote_ ballon_ecs auquel cas on planifie le ballon ECS si et seulement si la planification précédente n'a pas planifiée la machine pour démarrer au début de ce pas de temps, le timestamp actuel est supérieur d'au moins 12h à la valeur du champs timestamp_dernière_mise_en_marche de la table equipement_pilote_ou_mesure et le ballon ECS possède au moins une heure creuse active. Dans ce cas le ballon est planifié en se basant sur le plus gros créneaux actif d'heures creuses renseigné par dans la table equipement_pilote_ballon_ecs_heures_creuses.

- Le champs id de la table equipement_pilote_ou_mesure correspond à une valeur de equipement _pilote_ou_mesure_id dans la table equipement_pilote_ vehicule_electrique_generique auquel cas on planifie la charge du véhicule électrique si et seulement si il n'a pas été planifié pour commencer maintenant au round précédent.

- Le champs id de la table equipement_pilote_ou_mesure correspond à une valeur de equipement _pilote_ou_mesure_id dans la table equipement_pilote_ chauffage_asservi auquel cas on planifie le chauffage asservi en se basant sur le résultat du round d'avant.

- Le champs id de la table equipement_pilote_ou_mesure correspond à une valeur de equipement _pilote_ou_mesure_id dans la table equipement_pilote_ chauffage_non_asservi auquel cas on planifie le chauffage asservi en se basant sur le résultat des rounds précédents.

4.4 Journalisation

Les services publient leurs journaux de fonctionnement dans syslog, et accessibles via la commande *journalctl -u [nom du module]*.

Les principaux messages à suivre sont pour le service EMS_launcher :

- [$WARN\_CONSTRAINT$] erreur sur les contraintes : si l'utilisateur fournit des contraintes jugées impossibles du point de vue de l'EMS, celui-ci va les modifier de manière à respecter au mieux possible les contraintes de l'usager
- [$WARN\_IGNORED$] machine ignorée : si la machine était planifiée pour débuter dans ce quart d'heure et est toujours en mode piloté dans la base de données zabbix ou que son timestamp_dernière_mise_en_marche n'a pas été mis à jour (dans le cas de l'ECS uniquement)
- [$WARN\_DATABASE$] erreur d'interaction avec la base de données.
- [$WARN\_IMPOSSIBLE$] erreur de contraintes impossibles

5 Documentation simplifiée du solveur

Le solveur est un programme qui résout un problème mathématique d'optimisation. Ce programme va minimiser la valeur d'une "fonction coût" en fonction de "variables d'optimisations" en respectant des contraintes sur les valeurs possibles de ces variables d'optimisation.

La fonction coût actuelle correspond à l'énergie qui seras importée une fois le placement des consommation flexibles effectuée dans le cas ou le placement des consommations flexibles est suivi et que la prévision de consommation-production qui lui est fournie en entrée est correcte.

Le solveur travaille à partir de *consommateurs*, qui ont des contraintes (indiquées par les utilisateurs réels de ELFE) et d'une prédiction de consommation non-flexible - production.

Ces consommateurs et leurs contraintes sont ensuite traduits sous forme d'un problème MILP (Mixed Integer Linear Programming) qui est ensuite fourni à un solveur adapté à ce type de problèmes. L'une des entrées de ce problème est la prédiction du déficit d'énergie (Consommation - production), sans considérer les appareils flexibles. Ainsi les consommations flexibles sont ajoutées à la courbe initiale (on ne *déplace* pas les consommations flexibles, on les *place*).

La valeur à minimiser est la somme des valeurs positives de la courbe d'équilibre (déficit d'énergie). C'est le seul objectif exprimé de la fonction. *WARN* Il est très complexe mathématiquement d'optimiser pour plusieurs objectifs différents (pic d'imports par exemple ...), car le problème devient "non-linéaire" et donc trop calculatoire pour être résolu en un temps acceptable pour le pilotage des appareils du projet ELFE.

Le solveur travaille chaque 15min, avec les appareils déclarés à ce moment-là et la dernière courbe d'équilibre disponible.

Le solveur MILP ne donne un résultat que si les contraintes sont mathématiquement "possibles". Le code surveille les contrainte exprimées, et les modifie si besoin quand une demande est impossible (pour revenir à un problème possible). Une alerte est alors indiquée dans le journal de fonctionnement de l'EMS.

Le solveur est déjà publié en logiciel libre, sous la licence BSD 3-clauses.

6 Installation et mise en route de l'EMS

L'installation se fait au moyen du dépôt github et du script *install.sh*. Pour commencer, récupérer la dernière version à jour du code de l'EMS, avec la commande :

$>git clone "https://github.com/Energies-citoyennes-en-Pays-de-Vilaine/EnergyManagmentSystem.git"

Il faut ensuite déplacer le dossier ainsi obtenu dans le dossier dans lequel on veut installer l'EMS. Une fois cette opération effectuée, il faut éditer le script d'installation (*install.sh*) afin d'installer et de configurer les composants correctement. Tous les paramètres d'installation sont situés au début du fichier. **Ne pas éditer** CONFIG_FILENAME et CONFIG_FOLDER, ces variables ont été prévues pour pouvoir permettre au script d'installation d'être plus dynamique mais ce mécanisme n'est pour l'instant pas supporté.

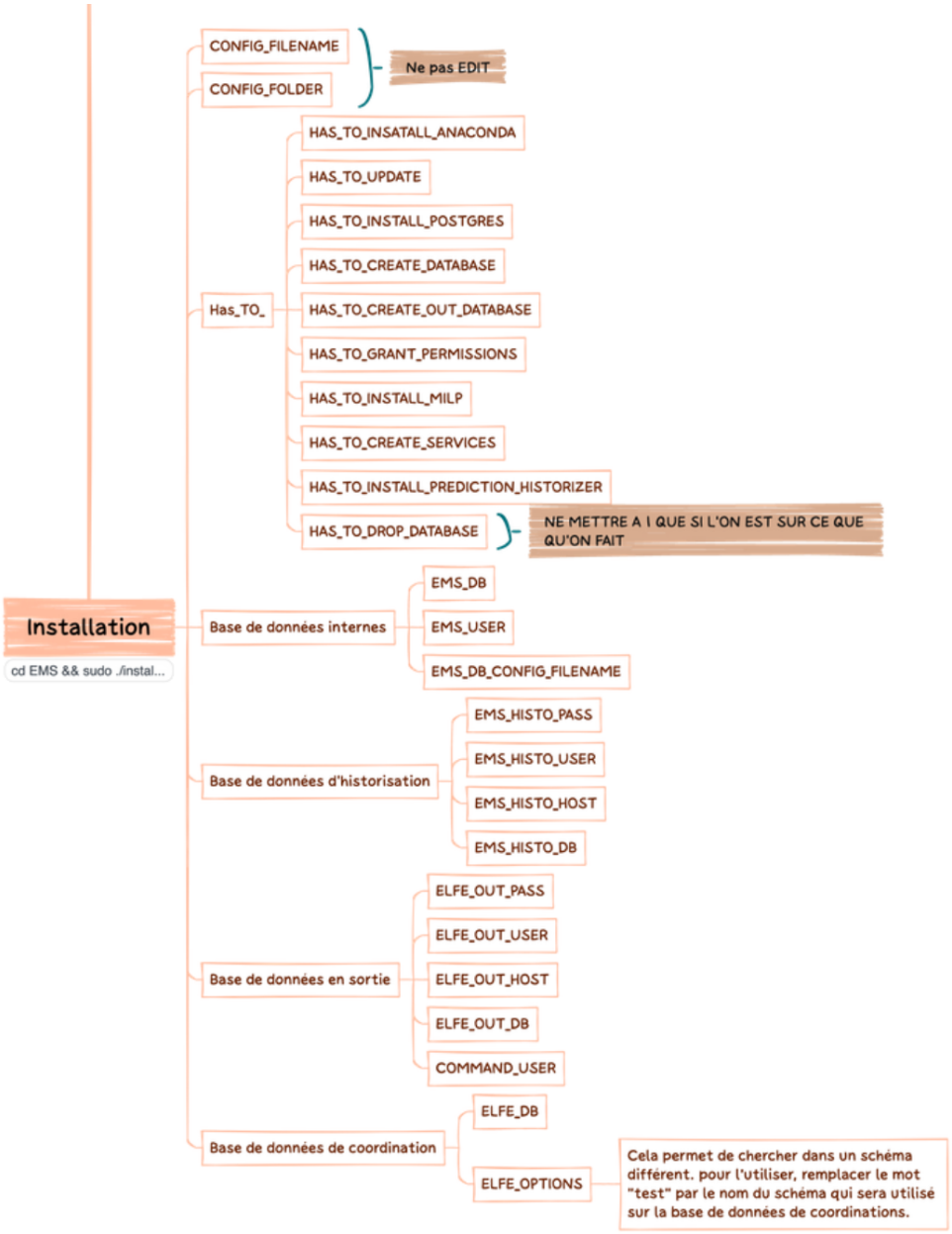


Figure 2: schéma d'installation, par Thomas Lesaunier

Ensuite viennent une liste de paramètres d'installation préfixés par HAS_TO_ . Ces paramètres servent à déterminer quelles parties du script d'installation doivent être lancées ou non. Un valeur de 1 veut dire que la partie doit être lancée et une valeur de 0 qu'elle ne doit pas l'être. Voici leur rôle :

- HAS_TO_INSTALL_ANACONDA : ce paramètre détermine si le script doit installer anaconda et l'environnement python requis pour l'utilisation de l'EMS. Valeur recommandée sur une première installation : 1.
- HAS_TO_UPDATE : ce paramètre détermine si le script d'installation doit lancer les commandes apt update et apt upgrade. Valeur recommandée : 1
- HAS_TO_INSTALL_POSTGRES : détermine si le script doit installer postgresql (entre autres pour la base de données internes de l'EMS). Valeur recommandée pour une première installation : 1.
- HAS_TO_CREATE_DATABASE : ce paramètre détermine si le script doit créer la base de données interne de l'EMS. Valeur recommandée pour une première installation : 1
- HAS_TO_CREATE_OUT_DATABASE : ce paramètre définit si le script doit créer la base de données de sortie de l'EMS. Il ne faut pas mettre cette valeur à 1 si cette base de données a déjà été créée en amont.
- HAS_TO_GRANT_PERMISSIONS : ce paramètre définit si le script d'installation doit donner des permissions d'accès à la base de données de sortie de l'EMS. A priori, il faut mettre cette valeur à 1 si et seulement si le script a créé la base de donnée de sortie de l'EMS.
- HAS_TO_INSTALL_MILP : Ce paramètre détermine si le script d'installation doit installer les packages pythons nécessaires à l'utilisation de l'EMS au moyen d'anaconda. Si l'EMS a dû installer anaconda, ce paramètre est ignoré et le script installera ces packages quoi qu'il advienne. Valeur recommandée pour une première installation : 1
- HAS_TO_CREATE_SERVICES : Ce paramètre sert à déterminer si le script doit installer les services *systemd* qui permettent à l'EMS de fonctionner. Valeur recommandée : 1. Si les services sont déjà installés, les réinstaller n'a aucun impact, donc je ne vois aucune raison de ne pas mettre ce paramètre à 1.
- HAS_TO_INSTALL_PREDICTION_HISTORIZER : Ce paramètre définit si le script doit installer le service optionnel d'historisation des prédictions. Pas de valeurs recommandées.
- HAS_TO_DROP_DATABASE : ce paramètre définit si l'EMS doit détruire la base de données précédente. **NE METTRE A 1 QUE SI L'ON EST SÛR DE CE QU'ON FAIT**

Lors de l'installation, si l'EMS doit créer des bases de données ou installer postgres, un nouveau fichier contenant les clefs d'accès aux base de données est crée à partir du fichier *credentials.db_credentials_example.py* et en remplaçant les champs de ce nouveau fichier par les valeurs connues du script d'installation. Ne pas oublier de faire une copie de l'existant si l'on ne fait pas une installation fraîche afin d'éviter des problèmes futurs.

Les paramètres suivants sont des paramètres de configuration de l'installation de l'EMS.

- base de données interne
  - EMS_DB : le nom de la base de données interne de l'EMS. Cela sera utilisé pour sa création. Valeur conseillée : "ems"
  - EMS_USER : le nom de l'utilisateur de l'EMS pour sa base de données interne. Valeur conseillée : "ems_user"
  - EMS_DB_CONFIG_FILENAME : le nom du fichier dans lequel seront conservées les clefs créées par ce run d'installation. Ne pas changer la valeur de "db_credentials.py", le mécanisme n'est pas complètement supporté.
- base de données d'historisation. Ces champs n'ont d'importance que si le service d'historisation de prédiction de l'EMS est activé. Laisser les valeurs. Voici les paramètres:

  - EMS_HISTO_PASS : le mot de passe de la base de données dans lesquelles seront historisées les courbes de prévisions de l'EMS, si ce service est activé. Si la valeur est "0", ce champs prendra le même mot de passe que celui généré par l'EMS pour sa base de données internes (à ne changer que si l'on veut historiser dans une base de données externe)
  - EMS_HISTO_USER : le nom d'utilisateur de l'EMS dans la base de données d'historisation. Par défaut, il s'agit du nom d'utilisateur de l'EMS dans sa base de données interne
  - EMS_HISTO_HOST : l'adresse ip / nom d'hôte du serveur qui héberge la base de données d'historisation.
  - EMS_HISTO_DB : Le nom de la base de données dans laquelle le script doit créer les tables nécessaires au service d'historisation.

- Base de données de sortie. Ces champs servent à l'EMS à savoir où écrire les résultats de ses calculs et où publier la planification de consommation.
  - ELFE_OUT_PASS : Le mot de passe de l'utilisateur de l'EMS sur le serveur qui contient la base de données de sortie. Ce mot de passe sera réutilisé pour la base de données de coordination.
  - ELFE_OUT_USER : le nom d'utilisateur de l'utilisateur de l'EMS sur le serveur qui contient la base de données de sortie. Ce nom d'utilisateur sera réutilisé pour la base de données de coordination.
  - ELFE_OUT_HOST : le nom d'hôte / l'adresse ip du serveur qui contient la base de données de sortie. Ce champs sera réutilisé pour la base de données de coordination.
  - ELFE_OUT_DB : le nom de la base de données de sortie de l'EMS
  - COMMAND_USER: le nom de l'utilisateur auquel on doit donner les permissions si on crée la base de données de sortie.

- base de données de coordination : certains des champs utilisés ici sont partagés avec la base de donnée de sortie, notamment le moyen d'accéder au serveur. Les champs qui sont spécifiques sont
  - ELFE_DB : le nom de la base de données de coordination
  - ELFE_OPTIONS : cela permet de chercher dans un schéma différent. pour l'utiliser, remplacer le mot "test" par le nom du schéma qui sera utilisé sur la base de données de coordinations.

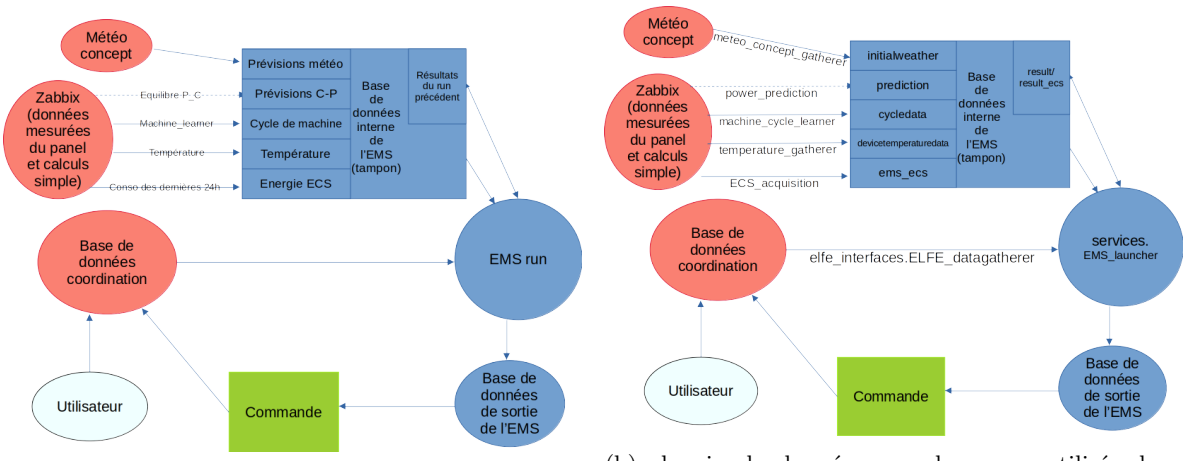
Une fois tous ces paramètres configurés, il suffit de lancer le script *install.sh* en tant qu'administrateur :

```
$>cd EnergyManagementSystem
$>sudo ./install.sh
```

Une fois ces étapes réalisées, il faut copier credentials/zabbix_credentials_example.py vers le fichier credentials/zabbix_credentials.py et le remplir (voir 2.2.2). Il faut faire de même avec le fichier credentials/meteo_concept_credentials_example.py (voir 2.2.3)

7 Chemin de données

On peut voir dans la figure 3 le chemin normal des données utilisées pour placer les différentes consommations. Les données de sources externes sont acquises périodiquement pour remplir une base de donnée interne à l'EMS qui lui assure la disponibilité des informations. Lorsque la donnée acquise forme une courbe de prévision, celle-ci est acquise sur la durée maximale disponible afin d'assurer une continuité de service si la source venait à ne plus être disponible. La prédiction de production - consommation est réalisée par persistance sur 24h à partir d'une courbe agglomérée par un service de zabbix (externe à l'EMS), et cette persistance est recopiée deux fois afin d'avoir une prévision sur 48h. Les prévisions météo sont acquises sur 12h au pas de l'heure, puis sur 14j au pas du quart de journée. Les données sont ensuite ré-échantillonnées pour avoir une prévision horaire sur 14j, en donnant priorité aux données initialement horaires.



(a) chemin de données avec des noms simplifiés

(b) chemin de données avec les noms utilisés dans l'EMS

Figure 3: chemin des données qui parviennent à l'EMS pour traitement

8 Outils de l'EMS

8.1 Configuration de l'outil d'apprentissage des courbes de consommations

Pour configurer l'outil d'apprentissage des courbes de consommations, il faut utiliser le module cli_tools et un outil de connexion à la base de données. Cet outil s'utilise dans l'environnement MILP, et uniquement quand l'EMS est configuré pour pouvoir se connecter à un zabbix fonctionnel. Pour se placer dans l'environnement MILP, avec un terminal connecté à l'EMS, taper :

```
$>conda activate milp
```

une fois cette commande executé, se placer dans le dossier d'installation de l'EMS et lancer l'outil en utilisant la commande

```
$>python -m cli_tools.machine_learner_conf ZabbixID [threshold_start] [threshold_end] [period_count] [period]
```

Les arguments entre crochets sont optionnels. Si on n'utilise qu'un zabbixID, l'outil va se connecter à la base de donnée interne de l'EMS et afficher les résultats selon ce qui est actuellement configuré pour ce consommateur. Sinon, les valeurs manquantes seront complétées avec les valeurs par défaut contenues dans le module config.config (les valeurs par défaut de la class MachineLearnerConfig). Voici l'utilité des différentes valeurs :

- ZabbixID : l'ID zabbix de la mesure de puissance de l'appareil pour lequel il faut configurer l'outil d'apprentissage de la courbe de consommation
- threshold_start : la valeur au dessus de laquelle la consommation doit passer pour que l'acquisition d'un cycle commence
- threshold_end : la valeur au dessous de laquelle la consommation moyennée sur une période period doit rester pendant period_count périodes pour que l'acquisition soit considérée complète.
- period_count le nombre de périodes durant lesquelles la consommation de l'appareil doit rester en dessous de threshold_end pour que l'acquisition soit considérée complète
- period : la période d'acquisition en secondes.

9 Divers

9.1 Nettoyage

L'EMS produit des données qui peuvent être sauvegardées et extraites, notamment :

- dans le dossier data/old, les anciennes courbes de consommation des appareils type machine en csv. Celles encore en cours d'utilisation sont dans le dossier in_use
- le dossier data/run_conditions contient les données de simulation passée. Il est possible de les récupérer sur un stockage externe puis de les supprimer.
- dans la base de données, les courbes historyinitialweather qui ne sont la que pour l'historisation des données de météo concept
- dans la base de données, les données des courbes de production/prévision du passé.

10 Foire aux Questions

Quand l'EMS produit sa liste de planification à l'instant t :

Q: S'agit-il seulement d'actions à faire dans le quart d'heure suivant ?

R:Non, il s'agit d'un plan d'actions pour les prochaines 24h. Cela dit, il faut toujours appliquer le dernier plan en date.

Q: Si ce n'est pas le cas, certaines actions à plus longue échéance peuvent-elles être replanifiées dans les traitements suivants de l'EMS ?

R: les actions sont replanifiées à chaque nouveau traitement de l'EMS.

Q: Avez-vous considéré une priorité entre les types de machines ? selon le jour ou la nuit ? Existe-t-il une priorité entre les participants pro et les résidentiels ?

R: Il n'existe pas de priorité entre les types de machines. L'outil cherche uniquement à minimiser l'import en plaçant les machines flexibles à sa disposition, tout en respectant les contraintes des utilisateurs. Il est possible que les machines dont les contraintes sont plus fortes soient lancées en priorité, mais c'est une conséquence de la minimisation et non une priorité manuellement choisie.

Q: Toutes les demandes (valides ou invalides) sont-elles forcément satisfaites ? Est-ce qu'un traitement contrôle ça, dans l'EMS ou ailleurs ?

R: Toutes les demandes valides sont forcément satisfaites. Lorsqu'une machine est invalide, l'EMS fait au mieux pour la rendre valide et respecter la demande. Si l'EMS n'a pas réussi à la rendre valide par les mécanismes identifiés et que celle-ci bloque l'exécution du programme, l'EMS n'ayant pas de moyen d'identifier une machine qui pose problème autres que ceux utilisés pour tenter de la rendre possible, l'EMS va éliminer les groupes de machines des moins sûrs aux plus sûrs jusqu'à ce que le problème soit possible. Dans ce cas, toutes les machines valides ne seront pas forcément planifiées. Il s'agit d'un mécanisme d'urgence, pas du fonctionnement nominal.

Q: que se passe-t-il si le serveur est coupé ? si les tables en entrée sont incorrectes/pas interrogeables/non rafraîchies ?

A: le rôle de la base de données interne est de faire tampon pour pallier à ce genre de scénario. Cela dépend de la durée de la coupure et de quel serveur/service ne fonctionne plus. De toute manière si l'EMS n'arrive pas a accéder à la base de donnée de coordination, il ne saura pas quoi planifier. C'est également le rôle de la planification publiée sur 24h de pouvoir pallier à un dysfonctionnement de l'EMS.