Sprint 1:

- *Establish a mechanism to initiate consumption measurement with each Jenkins build:*

On a défini dans un fichier pipeline toutes les instructions de collecte d'information à travers des build steps ("/Sprint_1/1.consumption/code_consumption.groovy" de notre repository Github). Le contenu de ce fichier sera mis dans les builds steps d'un projet Jenkins et la consommation sera affichée dans la console de sortie.

- *Know how to collect informations about a build during running:*

De même, grâce à un fichier pipeline qui peut être intégré comme build step d'un «Item» Jenkins (anciennement «Job»), cela permet d'obtenir les différents PIDs d'un build de cet Item Jenkins. ("/Sprint_1/2.pids/code_pids.groovy" de notre repository Github)

- *Develop a way to run the bash script with a jenkins plugin:*

Nous avons créé un projet Maven permettant d'exécuter un fichier bash sur le système ("/Sprint_1/3.lecture_script/lecture_script_project" de notre repository Github). Nous avons mis en place un script bash donnant des informations sur le système à titre d'exemple. Nous avons aussi créer un fichier de test pour ce projet.

-Pour exécuter les tests :
mvn test

-Pour run le projet :
mvn exec:java -Dexec.mainClass="com.example.BashScriptExecutor"

- *Develop a way to format energy consumption data in watts before storing it:*

La puissance est la dérivée de l'énergie en fonction du temps, il en résulte que :

watts = Δjoules/Δsecondes.

Il faut aussi prendre en compte qu'on a au départ du microJoule (e-6).
On a donc directement pu inclure cela dans notre fichier de build steps.

- *Show total energy consumption in the Jenkins build output console:*

Des essais ont été réalisés pour utiliser le fait d'exécuter un script depuis Jenkins afin de lire la consommation ainsi, nous avons rencontré des problèmes en lien avec le workspace dans lequel Jenkins s'exécute.

Sprint 2

- *Create a blank plugin and set up the environment*

Initialisation et configuration du plugin, et mise en place de projets pour pouvoir le tester par la suite.

- *Integrate the feature that gives the consumption into this plugin*

Utilisation d'un RunListener (notre classe ListenerRunEnergyMonitoring) qui permet de définir du code qui s'exécute avant (onInitialize() et onStarted()) et après (onCompleted()) le build. Nous avons ainsi pu intégrer au plugin la mesure de consommation que nous avions sous forme de pipeline au sprint 1.

Concernant PowerAPI, nous avons créer un classe InfluxDBData qui effectue la connexion à la base de donnée , récupere les informations correspondantes à la pipeline en fonction du temps de début de la pipeline et effectue le calcul de la puissance et de l'énergie

- *Create a process to give access to this plugin in any project*

Cela se fait en lançant la commande mvn package pour générer le répertoire target et en récupérant le fichier .hpi qui sera à upload sur Jenkins dans la page «paramètres avancés» de la gestion des plugins.

- *Develop UI components for energy graphs*

La page que nous ajoutons à Jenkins est définie dans le fichier index.jelly (/src/main/ressources/io/jenkins/plugins/DisplayChart de notre plugin). Pour développer un affichage graphique nous avons ajouté du javascript à cette page avec la librairie Chart.js. Cette fonctionnalité fonctionne conjointement avec la classe DisplayChart de notre plugin qui est une RunAction2 Jenkins et qui lorsque ajoutée au run (run.addAction) permet de rendre effective la page que l'on ajoute. Cette classe permet aussi d'effectuer des traitements notamment pour des données qu'on aurait besoin d'utiliser pour notre affichage graphique.

- *Manage the integration of consumption data*

La classe DisplayChart permet d'effectuer des traitements notamment pour des données qu'on aurait besoin d'utiliser pour notre affichage graphique. Notamment nous pouvons y définir des méthodes renvoyant des JSON afin qu'elles soient appelées par notre page dans via son script Javascript de cette manière : "${it.getEnergyHistoryAsJsonAll()}" (la méthode "getEnergyHistoryAsJsonAll()" est définie dans la classe Java DisplayChart qui est reconnu comme étant «it» du fait de l'arborescence dans laquelle se trouve le fichier index.jelly de cette page.

Sprint 3

- *Install and configure InfluxDB*

    1. suivre le tutoriel d'installation de influxdb sur www.influxdata.com/downloads.
    2. Lancer influxdb avec : sudo service influxdb start ou sudo systemctl start influxdb
    3. Aller sur le localhost:8086 de son navigateur créer un premier user avec l'interface graphique un token sera généré. Il faut bien conserver car c'est lui qui accorde les droits administrateurs
    4. Ajouter la dépendance influxdb dans votre pom.xml

- *Develop a database integration for data insertion and retrieval*

    1. Utiliser ces identifiants pour initialiser un client influxdb
    2. On utilise les méthodes writeData() et retrieveMessage() pour respectivement écrire et lire des données dans la base de données

- *Processing historical data (sorting and aggregating it)*

  On utilise un mécanisme de récursion : il y a une méthode qui permet de récupérer les données du précédent build (run.getPreviousBuild()), grâce à cela dans chaque nouveau build nous récupérons les données du sprint précédent auxquelles nous ajoutons les données du build en cours. Ainsi par récursion nous avons tout l'historique de données.

- *Set up the graphical display*

  Comme expliqué précédemment nous utilisons la classe ChartDisplay pour définir des méthodes qui seront récupérées par le Javascript.

- *Propose display options: date range to view etc.*

  Nous définissons une méthode Javascript updateChart() qui permet de mettre à jour le graphique en fonction de variables correspondant à des boutons que nous avons ajoutés sur la page. Pour ce faire nous avons dû développer d'autres listener tel qu'un FlowExecutionListener (notre classe ListenerExecution) qui nous permet d'exécuter du code tout au long du build (chaque listener constitue en fait un thread différent) grâce à la méthode onRunning().

- *Obtain the different time range of the different stages*

  Grâce à un GraphListener Jenkins (notre classe ListenerStages) nous pouvons récupérer les temps auxquels nous changeons de stage, de cette manière nous pouvons séparer les données selon les stages. Il faut noter que sont pris en compte des stages que Jenkins crée en plus que ceux définis dans la pipeline

- *Apply PowerAPI to each time range to be able to separate consumption according to steps*

Ici nous utilisons la méthode ValuesEnergetic<Double, Double> processInfluxDBData() de la classe influcDBData afin d'obtenir les informations énergétiques du build en fonction de son temps de début et de la durée du stage souhaité si c'est un stage sinon calcule la puissance et énergie pour toute la pipeline.