Architecture Document

Equipe Projet

Axel BERNARD
Moussa BERTHE
Yao KONAN

SOMMAIRE

I. Principe de mise en œuvre de la solution (comment) :

La solution implémentée est un plugin pour Jenkins, une plateforme d'automatisation de construction et de déploiement. Le plugin surveille et enregistre la consommation d'énergie pendant l'exécution des pipelines Jenkins, puis affiche les données collectées sous forme de graphiques dans l'interface utilisateur de Jenkins. Pour ce faire, le plugin utilise des hooks fournis par Jenkins pour écouter les événements clés du cycle de vie des pipelines, tels que le démarrage, l'exécution et l'achèvement.
Afin de pouvoir obtenir les informations énergétiques, nous utilisons PowerAPI.
PowerAPI récupère la consommation de l'ordinateur à l'aide de cgroup.
Pour pouvoir obtenir les consommations du build et aussi des étapes du pipeline , nous sommes censés utiliser des cgroups et ajouter les pids des processus de la pipeline dans les cgroups correspondantes. Afin de pouvoir faciliter le regroupement des consommation par pipeline, le plugin récupère uniquement la consommation des pipelines s'exécutant dans des agents dockers.

II. Règles d'architecture :

1. **Séparation des préoccupations** :
    - Les différentes parties du projet sont conçues pour avoir des responsabilités clairement définies, favorisant ainsi la modularité et la réutilisabilité du code.

    CompletionLatch : Gère la synchronisation entre différentes parties du système en utilisant un mécanisme de verrouillage. Cette classe se concentre uniquement sur la gestion de la synchronisation et ne contient pas de logique métier spécifique.

    DisplayChart : Gère l'affichage des graphiques de consommation d'énergie à partir des données collectées. Elle encapsule la logique d'affichage et ne se préoccupe pas de la récupération des données ou de leur traitement.

    InfluxDBData : Se concentre sur l'interaction avec la base de données InfluxDB et le traitement des données de consommation d'énergie. Elle encapsule cette logique et ne se préoccupe pas des détails d'affichage ou de configuration.

    VariablesConsumptionsPreviousBuild : Gère les données de consommation d'énergie et de puissance des builds précédents. Elle encapsule ces données et ne se préoccupe pas de leur traitement ou de leur affichage.

    ValuesEnergetic : Gère les valeurs énergétiques calculées, telles que la consommation d'énergie et la puissance. Elle encapsule ces valeurs et ne se préoccupe pas de leur affichage ou de leur stockage persistant.

    ListenerExecution : Gère l'écoute de l'exécution du workflow Jenkins. Elle encapsule la logique d'écoute et de traitement des événements d'exécution.

    ListenerStages : Gère l'écoute des étapes du pipeline Jenkins. Elle encapsule la logique d'écoute et de traitement des événements de progression des étapes.

ListenerRunEnergyMonitoring : Gère la surveillance de la consommation d'énergie lors de l'exécution d'un build. Elle encapsule la logique de surveillance et de traitement des données énergétiques.

ValuesStagesData : Gère la collecte et le stockage des données relatives à chaque étape (stage) de l'exécution du pipeline Jenkins. Elle encapsule la logique de gestion des données de chaque étape.

ValuesExecutionData : Gère la collecte et le stockage des données relatives à l'exécution globale du pipeline Jenkins. Elle encapsule la logique de gestion des données globales d'exécution.

2. **Gestion des dépendances** :
    - Les classes dépendent principalement des événements du pipeline Jenkins et des données énergétiques collectées à partir d'une source externe, probablement une base de données InfluxDB, pour leur fonctionnement.
3. **Utilisation de principes SOLID** :
    - Les classes respectent les principes de responsabilité unique, d'ouverture/fermeture, de substitution de Liskov, d'inversion de dépendance et d'interface ségrégée, favorisant ainsi une architecture robuste et facile à maintenir.

CompletionLatch : Respecte le principe de responsabilité unique en se concentrant uniquement sur la gestion de la synchronisation, ce qui facilite sa compréhension et sa réutilisation. Elle est également conforme au principe d'encapsulation en masquant les détails de mise en œuvre du mécanisme de verrouillage.

DisplayChart : Respecte le principe de responsabilité unique en se concentrant uniquement sur l'affichage des données. Elle est ouverte à l'extension pour prendre en charge différents types de graphiques, mais fermée à la modification de sa logique d'affichage.

InfluxDBData : Respecte le principe de responsabilité unique en se concentrant uniquement sur l'interaction avec InfluxDB. Elle est ouverte à l'extension pour prendre en charge d'autres bases de données, mais fermée à la modification de sa logique de traitement des données.

VariablesConsumptionsPreviousBuild : Respecte le principe de responsabilité unique en se concentrant uniquement sur la gestion des données de builds précédents. Elle est ouverte à l'extension pour prendre en charge d'autres types de données historiques, mais fermée à la modification de sa logique de gestion des données.

ValuesEnergetic : Respecte le principe de responsabilité unique en se concentrant uniquement sur la gestion des valeurs énergétiques. Elle est ouverte à l'extension pour prendre en charge d'autres types de valeurs énergétiques, mais fermée à la modification de sa logique de gestion des valeurs.

ListenerRunEnergyMonitoring : Respecte le principe de responsabilité unique en se concentrant uniquement sur la surveillance de la consommation d'énergie. Elle est ouverte à l'extension pour prendre en charge différents types de surveillance énergétique, mais fermée à la modification de sa logique de surveillance.

ListenerExecution : Respecte le principe de responsabilité unique en se concentrant uniquement sur l'écoute de l'exécution du workflow. Elle est ouverte à l'extension pour prendre en charge d'autres types d'événements, mais fermée à la modification de sa logique d'écoute.

ListenerStages : Respecte le principe de responsabilité unique en se concentrant uniquement sur l'écoute des étapes du pipeline.

ValuesStagesData : Respecte le principe de responsabilité unique en se concentrant uniquement sur la gestion des données de chaque étape. Elle est ouverte à l'extension pour prendre en charge différents types de données d'étape, mais fermée à la modification de sa logique de stockage des données.

ValuesExecutionData : Respecte le principe de responsabilité unique en se concentrant uniquement sur la gestion des données globales d'exécution.

III. Modèle Statique :

Le projet est organisé en plusieurs packages, chacun contenant des classes ayant des responsabilités spécifiques :

- **Package `io.jenkins.plugins`** : Contient les classes principales du plugin, telles que `VariablesConsumptionAction`, `CompletionLatch`, `DisplayChart`, `ListenerExecution`, `ListenerStages`, `ListenerRunEnergyMonitoring`, `ValuesExecutionData`, `ValuesStagesData`, `InfluxDBExample`, `ValuesEnergetic`.
    - `VariablesConsumptionAction` : Gère les données de consommation d'énergie pour chaque build. Stocke les temps de début et de fin de la consommation d'énergie.
    - `CompletionLatch` : Fournit un mécanisme de synchronisation pour attendre la fin du traitement.
    - `DisplayChart` : Affiche les graphiques de consommation d'énergie dans l'interface utilisateur de Jenkins. Récupère et affiche les données énergétiques sous forme graphique.
    - `ListenerExecution` : Écoute les événements d'exécution du pipeline et enregistre les données énergétiques. Mesure et enregistre les données de consommation d'énergie pendant l'exécution du pipeline.

- `ListenerStages` : Écoute les événements des étapes du pipeline et enregistre les données énergétiques spécifiques à chaque étape. Mesure et enregistre les données de consommation d'énergie pour chaque étape du pipeline.
- `ListenerRunEnergyMonitoring` : Écoute les événements du cycle de vie des builds et enregistre les données énergétiques globales. Déclenche les actions nécessaires au début et à la fin des builds.
- `ValuesExecutionData` : Stocke et fournit des données sur l'exécution du pipeline. Utilisé pour agréguer et accéder aux données de consommation d'énergie des différentes exécutions.
- `ValuesStagesData` : Stocke et fournit des données sur les étapes du pipeline. Utilisé pour agréguer et accéder aux données de consommation d'énergie des différentes étapes.
- `InfluxDBExample` : Gère l'interaction avec la base de données InfluxDB pour récupérer et traiter les données énergétiques. Utilise les données de connexion pour interroger InfluxDB et récupérer les mesures de consommation d'énergie stockées.
- `ValuesEnergetic` : Fournit des modèles de données pour représenter les valeurs énergétiques calculées. Représente les valeurs énergétiques et les puissances mesurées et calculées.

IV. Modèle dynamique :

Le flux des événements dans le plugin suit le cycle de vie des builds et des étapes du pipeline Jenkins, intégrant l'utilisation de PowerAPI pour mesurer la consommation d'énergie.

- **Lancement de PowerAPI** : PowerAPI est lancé à l'aide de Docker Compose avant le démarrage du build Jenkins. Il commence à mesurer la consommation des différents cgroups et containers Docker sur l'ordinateur.
- **Flux des événements nominaux** :
  - **Démarrage du build** : Lorsqu'un build est lancé, `ListenerRunEnergyMonitoring` initialise les actions et enregistre les temps et consommations de début.
  - **Exécution des étapes du pipeline** : `ListenerStages` enregistre les données énergétiques pour chaque étape. Pendant cette période, PowerAPI mesure continuellement la consommation d'énergie.
  - **Collecte des données par PowerAPI** : PowerAPI stocke les données mesurées dans la base de données InfluxDB.
  - **Récupération des données d'InfluxDB** : `InfluxDBExample` se connecte à InfluxDB en utilisant les informations de connexion prédéfinies. Il récupère les données mesurées correspondant au temps de début et calcule les consommations et puissances énergétiques spécifiques à l'exécution du pipeline.

    - **Traitement des données** : Les données récupérées et calculées par `InfluxDBExample` sont ensuite fournies aux autres classes pour stockage (`ValuesExecutionData`, `ValuesStagesData`) et affichage (`DisplayChart`).
- **Flux des événements sur erreur** : En cas d'erreur ou d'exception pendant l'exécution du pipeline, les classes telles que `ListenerExecution` et `ListenerRunEnergyMonitoring` enregistrent les erreurs et les données pertinentes pour le diagnostic.
- **Démarrage et arrêt du plugin** :
    - **Démarrage** : Initialisation des composants du plugin et démarrage de l'écoute des événements du pipeline.
    - **Arrêt** : Enregistrement des données finales, nettoyage des ressources et arrêt de PowerAPI.

V. Explication de la prise en compte des contraintes d'analyse :

Pour répondre aux différentes contraintes du projet, plusieurs approches ont été adoptées lors de la conception et de l'implémentation de la solution.

1. **Facilité de téléchargement et de configuration** :
    - **Plugin Jenkins** : Le plugin est empaqueté en tant que fichier JAR, facilement téléchargeable et installable via l'interface utilisateur de Jenkins. Une fois installé, il peut être configuré à partir de l'interface de Jenkins sans nécessiter des étapes de configuration complexes.
    - **PowerAPI** : PowerAPI est distribué avec une configuration Docker Compose, simplifiant son déploiement et son démarrage. Les utilisateurs peuvent lancer PowerAPI avec une simple commande Docker Compose, sans avoir à gérer manuellement les dépendances et configurations internes.
2. **Visualisation des données** :
    - **Interface utilisateur Jenkins** : Le plugin comprend des fonctionnalités pour afficher les données de consommation d'énergie sous forme de graphiques compréhensibles. La classe `DisplayChart` se charge de récupérer les données énergétiques et de les afficher sous forme de graphiques historiques, permettant aux utilisateurs de visualiser facilement leur consommation d'énergie et de comparer les performances entre les builds.
    - **Données compréhensibles** : Les graphiques affichent des étiquettes claires, des valeurs énergétiques et des puissances, facilitant ainsi la compréhension des utilisateurs sur leur consommation d'énergie.

3. **Impact minimal sur les performances des pipelines CI/CD** :
    - **Mesures asynchrones** : Les mesures de consommation d'énergie sont effectuées de manière asynchrone en utilisant PowerAPI et stockées dans InfluxDB. Cela permet de minimiser l'impact sur les performances des pipelines, car les mesures ne bloquent pas l'exécution des étapes du pipeline.
    - **Traitement en arrière-plan** : Les classes telles que `ListenerExecution` et `ListenerStages` effectuent des traitements et des enregistrements de données en arrière-plan, évitant ainsi de ralentir le processus de build principal.
4. **Compatibilité et intégration transparente** :
    - **Intégration avec Jenkins** : Le plugin est conçu pour s'intégrer de manière transparente dans Jenkins, en utilisant les API et les événements de Jenkins pour enregistrer et traiter les données énergétiques. Les classes de Listener (comme `ListenerExecution` et `ListenerStages`) écoutent les événements du pipeline Jenkins et capturent les données pertinentes sans nécessiter des modifications substantielles du workflow existant des développeurs.
    - **Utilisation de PowerAPI** : PowerAPI fonctionne indépendamment et interagit avec le système via Docker, permettant ainsi une compatibilité avec divers environnements CI/CD sans nécessiter des modifications du système hôte.

VI. Cadre de Production:

Pour le développement, la configuration et la livraison du plugin Jenkins, plusieurs outils et technologies ont été utilisés afin de garantir une intégration fluide, une gestion efficace et une utilisation optimale des ressources. Voici une description détaillée de ces outils et de leur rôle dans le projet.

Outils de développement

1. **Visual Studio Code (VSCode)** :
    - **Description** : VSCode est un éditeur de code source léger mais puissant, développé par Microsoft. Il offre de nombreuses fonctionnalités comme la coloration syntaxique, l'autocomplétion, le débogage intégré, ainsi que des extensions pour divers langages et outils.
    - **Utilisation dans le projet** : VSCode a été utilisé pour écrire et gérer le code du plugin Jenkins, faciliter le débogage et assurer une intégration avec les outils de contrôle de version comme Git.
2. **Jenkins** :

    - **Description** : Jenkins est un outil d'intégration continue et de livraison continue (CI/CD) open-source, utilisé pour automatiser les tâches répétitives liées au développement logiciel.
    - **Utilisation dans le projet** : Jenkins a été utilisé comme plateforme cible pour développer, tester et déployer le plugin. Les fonctionnalités de Jenkins ont permis de tester le plugin dans un environnement réel et de s'assurer de son bon fonctionnement.
3. **Docker** :
    - **Description** : Docker est une plateforme permettant de créer, déployer et exécuter des applications dans des conteneurs. Les conteneurs Docker encapsulent une application et ses dépendances dans un environnement isolé, facilitant la portabilité et la gestion des versions.
    - **Utilisation dans le projet** : Docker a été utilisé pour exécuter PowerAPI et ses composants (InfluxDB, MongoDB, HWPC Sensor, SmartWatt, et Grafana). Cela a permis de simuler un environnement de production réel pour tester la consommation énergétique des builds Jenkins.
4. **PowerAPI** :
    - **Description** : PowerAPI est un ensemble d'outils pour mesurer la consommation d'énergie des processus et des conteneurs sur un système. Il est constitué de cinq conteneurs Docker : HWPC Sensor, SmartWatt, InfluxDB, MongoDB et Grafana.
        - **HWPC Sensor** : Collecte les données de consommation d'énergie.
        - **SmartWatt** : Analyse et traite les données collectées.
        - **InfluxDB** : Base de données pour stocker les données de consommation énergétique.
        - **MongoDB** : Base de données pour stocker les données complémentaires.
        - **Grafana** : Outil de visualisation pour afficher les données énergétiques.
    - **Utilisation dans le projet** : PowerAPI a été utilisé pour mesurer et analyser la consommation d'énergie des builds Jenkins. Les données collectées par PowerAPI sont stockées dans InfluxDB, qui sont ensuite récupérées par le plugin pour générer des rapports énergétiques.

Outils de configuration

1. **Jenkins** :
    - **Description** : Jenkins fournit une interface utilisateur graphique (GUI) pour configurer et gérer des plugins, des jobs, des pipelines, et d'autres aspects du serveur Jenkins.
    - **Utilisation dans le projet** : Jenkins a été utilisé pour configurer le plugin développé, spécifier les détails de connexion à la base de données InfluxDB, et définir les paramètres nécessaires pour la collecte et l'analyse des données énergétiques.
2. **Configuration de PowerAPI** :
    - **Docker Compose** : Docker Compose a été utilisé pour orchestrer le démarrage et la configuration des conteneurs PowerAPI. Le fichier

`docker-compose.yml` définit les services nécessaires, leurs dépendances et leurs configurations.
    - **Variables d'environnement** : Les détails de connexion à InfluxDB et d'autres paramètres nécessaires pour PowerAPI sont définis dans le fichier `stack-powerapi.yml` ou dans des fichiers d'environnement, facilitant ainsi le déploiement et la configuration.

Outils de livraison

1. **Fichier HPI** :
    - **Description** : Le plugin Jenkins est empaqueté en tant que fichier HPI (Hudson Plugin Interface), qui est le format standard pour les plugins Jenkins.
    - **Utilisation dans le projet** : Le plugin développé est compilé et empaqueté en un fichier HPI à l'aide de Maven (`mvn clean install`). Ce fichier est ensuite prêt à être installé sur un serveur Jenkins via l'interface d'administration des plugins.
2. **Maven** :
    - **Description** : Maven est un outil de gestion de projet et d'automatisation de build, principalement utilisé pour les projets Java.
    - **Utilisation dans le projet** : Maven a été utilisé pour gérer les dépendances, compiler le code source, exécuter les tests et empaqueter le plugin en un fichier HPI.