Écrire un Jenkinsfile

Pour écrire un Jenkinsfile qui intègre une bibliothèque partagée, un paramètre de type chaîne de caractères, un bloc try-catch, un bloc node et une instruction if, vous pouvez utiliser l'approche suivante. Les Jenkinsfiles sont écrits en Groovy et peuvent définir un pipeline pour automatiser les processus de construction, de test et de déploiement. Voici un exemple complet d'un Jenkinsfile scripté qui répond à ces exigences.

Tout d'abord, pour utiliser une bibliothèque partagée, vous devez l'importer avec l'annotation `@Library`. Les bibliothèques partagées dans Jenkins contiennent du code réutilisable qui peut être appelé à partir de plusieurs pipelines. Par exemple, si votre bibliothèque partagée s'appelle `my-shared-library`, vous l'incluez en haut du Jenkinsfile comme ceci :

```
@Library('my-shared-library') _
```

Le tiret bas (_) après l'annotation est nécessaire pour importer correctement la bibliothèque.

Ensuite, pour définir un paramètre de type chaîne de caractères, utilisez l'étape `properties`. Cela permet aux utilisateurs de passer une valeur de chaîne au pipeline lorsqu'il s'exécute. Voici comment ajouter un paramètre de type chaîne nommé `MY_STRING` :

```
properties([
    parameters([
        string(name: 'MY_STRING', defaultValue: 'default', description: 'Un paramètre de type chaîne')
    ])
])
```

Le bloc `node` spécifie où le pipeline s'exécute, par exemple sur un agent disponible. À l'intérieur de ce bloc, vous pouvez inclure la logique de votre pipeline :

```
node {
    // Les étapes du pipeline vont ici
}
```

Pour gérer les erreurs potentielles, enveloppez vos étapes dans un bloc `try-catch`. Cela garantit que si quelque chose échoue, vous pouvez attraper l'exception et la gérer avec grâce. De plus, une instruction `if` peut être utilisée pour prendre des décisions en fonction de la valeur du paramètre de type chaîne (`params.MY_STRING`).

Voici le Jenkinsfile complet combinant tous ces éléments :

```
@Library('my-shared-library') _

properties([
```

```
    parameters([
        string(name: 'MY_STRING', defaultValue: 'default', description: 'Un paramètre de type chaîne')
    ])
])

node {
    try {
        // Appeler une fonction de la bibliothèque partagée (en supposant qu'elle existe)
        myLibraryFunction()

        // Utiliser une instruction if basée sur le paramètre de type chaîne
        if (params.MY_STRING == 'test') {
            echo "Exécution du test"
        } else {
            echo "Ne pas exécuter le test"
        }
    } catch (Exception e) {
        echo "Une erreur est survenue : ${e.message}"
    }
}
```

Explication des composants

- `@Library('my-shared-library') _` : Importe une bibliothèque partagée nommée `my-shared-library`. Remplacez `my-shared-library` par le nom réel de votre bibliothèque. La fonction `myLibraryFunction()` est supposée être définie dans cette bibliothèque.
- `properties` : Définit un paramètre de type chaîne `MY_STRING` avec une valeur par défaut de `'default'` et une description. Vous pouvez accéder à ce paramètre dans le pipeline en utilisant `params.MY_STRING`.
- `node` : Exécute le pipeline sur un agent disponible. Vous pouvez spécifier un agent particulier (par exemple, `node('linux')`) si nécessaire.
- `try-catch` : Exécute les étapes à l'intérieur du bloc `try` et attrape toute exception dans le bloc `catch`, imprimant un message d'erreur si quelque chose ne va pas.
- `if` : Vérifie si `MY_STRING` est égal à `'test'` et imprime un message en conséquence. Vous pouvez modifier la condition et les actions selon vos besoins.

Notes

- Cet exemple suppose un pipeline scripté, qui offre plus de flexibilité avec la syntaxe Groovy. Si vous avez besoin d'un pipeline déclaratif à la place, la structure serait légèrement différente (par exemple, en utilisant `pipeline` et `script` blocs pour le code Groovy).

- L'appel à `myLibraryFunction()` est un espace réservé. Assurez-vous que votre bibliothèque partagée définit cette fonction, ou remplacez-la par une fonction réelle de votre bibliothèque.
- Vous pouvez étendre ce Jenkinsfile en ajoutant des étapes (par exemple, `stage('Build')`) ou une logique plus complexe selon vos besoins.

Ce Jenkinsfile fournit une base solide qui inclut tous les éléments que vous avez demandés : une bibliothèque partagée, un paramètre de type chaîne, un bloc try-catch, un bloc node et une instruction if. Ajustez les détails (nom de la bibliothèque, appels de fonction ou conditions) pour qu'ils correspondent à votre cas d'utilisation !