Les Smart Contracts Javascript

Ce document présente quelques notions générales aux smart contrats Kalima, et explique comment son créer des smart contrats Javascript pour Kalima. A noté que les smart contrats Javascript sont lancés par des nœuds client Java, et que ce document ne montre pas comment créer son propre nœud permettant de lancer des smart contrats.

Cependant un nœud a été installé et permet de lancer des smart contrats. L'un est lancé à l'arrivée de nouvelles transactions à l'adresse /sensors, l'autre est lancé à l'arrivée de nouvelles transactions à l'adresse /alarms/fire.

1. Prérequis

Pour créer un Smart Contract il est nécessaire d'avoir un éditeur de code et/ou un éditeur de texte :

- Exemple d'un éditeur de code : Visual Studio Code, lien de téléchargement : https://code.visualstudio.com/
- Exemple d'un éditeur de texte : notepad++, Gedit (Ubuntu), vi, nano…

Il faut également installer git, puisque les contrats sont stockés sur des répertoires git. Pour modifier un contrat, il suffit donc de pousser le contrat sur git, puis d'attendre quelques minutes, le temps que le contrat soit déployé, comme décrit dans la documentation API_Kalima.

2. Création d'un Smart Contract JavaScript

Instruction obligatoire :

Pour garantir le fonctionnement de votre Smart Contract JavaScript, il est nécessaire

```
load("nashorn:mozilla_compat.js");
```

d'ajouter la ligne suivante au début de votre programme :

Importation des packages :

Les smart contrats Javascripts sont lancés par des nœuds java. Depuis les smart contrats, nous avons accès à l'API Java Kalima. De plus, on peut importer n'importe quel package Java.

Pour importer un package Java vous pouvez utiliser la fonction « importPackage() »

```
importPackage(Packages.java.io);
importPackage(Packages.java.lang);
importPackage(Packages.java.util);
importPackage(org.kalima.kalimamq.message);
importPackage(org.kalima.cache.lib);
importPackage(org.kalima.util);
```

comme il est montré dans l'exemple suivant :

Création des objets Java :

Pour mettre suite à l'importation des packages Java, vous aurez besoin de déclarer les objets Java à utiliser dans votre Smart Contract de la même façon que l'exemple

```
var JString = Java.type("java.lang.String");
var KMsg = Java.type("org.kalima.cache.lib.KMsg");
```

suivant :

Le code du smart contrat :

Vous pouvez ensuite écrire le code de votre smart contrat. A noté que le nœud java qui sera chargé d'exécuter le smart contrat, exécutera obligatoirement une fonction du smart contrat. Votre code doit donc être à l'intérieur d'une fonction.

3. Exemple de Smart Contract

Dans le cadre des tutoriels, il existe comme dit au début de ce documents deux smart contrats. Ces smart contrats sont sur un serveur git privé, dans le repository « KalimaContractsTuto ». Si vous n'avez pas accès à ce répertoire, veuillez contacter un membre de notre équipe. Si vous avez l'accès, il vous est possible de modifier les smart contrats existants pour changer leurs comportements. Pour que vos modifications soient prises en compte, il vous suffira de commiter les changements sur le repository et d'attendre quelques minutes pour que le processus de publication des smart contrats soit terminé.

On retrouve également les deux exemples de smart contrats sur GitHub : https://github.com/Kalima-Systems/Kalima-Tuto/tree/master/KalimaSmartContracts.

Le Smart Contract « sensor.js » :

Voici le corps de l'exemple sensors.js :

```
function main(kMsg, clone, logger) {

        if(!kMsg.getKey().equals("temperature2")) return;

        var body = new JString(kMsg.getBody());

        var temperature = parseInt(body, 10);

        if(temperature == null)
```

```
                return "NOK";

        if(temperature >= 100) {

                clone.put("/alarms/fire", kMsg.getKey(), ("Temperature too high: " + temperature + "
°C").getBytes());

        }

}
```

Il est exécuté à chaque nouvelle transaction sur l'adresse /sensors. Trois paramètres lui sont passés :

- kMsg : Le message correspond à la transaction reçue. Il permettra notamment de récupérer les données utiles, ou encore la clé de la transaction
- clone : Le clone du nœud qui lance le smart contrat. Il permet de lire les transactions de la blockchain et de créer de nouvelles transactions par exemple
- logger : Logger interne de Kalima. Il n'est pas utile dans le cadre de ce tutoriel, car vous n'avez pas accès au nœud smart contrat. Cependant lorsque vous déployer votre propre nœud, il permet d'ajouter des logs dans des fichiers journalisés.

Le contrat est simple : On commence par vérifier la clé du message. Si elle n'est pas égale à « temperature2 », on ne fait rien. Ensuite on récupère les données utiles de la transaction en partant du principe qu'elle correspond à un entier qui correspond à une température par exemple. Enfin, on crée une nouvelle transaction à l'adresse « /alarms/fire » si la température dépasse un certain seuil.

Dans le cadre du tutoriel, vous pouvez par exemple modifier le seuil de température, puis envoyer des températures à l'aide de l'exemple java pour vérifier que le seuil a effectivement changé.

Le Smart Contract « fire.js » :

Le smart contrat fire.js ne fait rien pour le moment, à part enregistrer dans les logs le contenu des alarmes. Libre à vous de modifier ce smart contrat en guise d'entraînement.