TP - Déployer un Smart Contract sur Ethereum Testnet

> Créer votre premier smart contrac sur une vraie blockchain

Prérequis

- Connaissances de base en JavaScript
- Node.js installé
- Rabby installé
- Compte Infura
- 2 ou 3 comptes MetaMask avec des ETH de test (Sepolia faucet)

Objectifs du cours

- Comprendre le rôle d’un smart contract Ethereum
- Configurer son environnement de développement (Node.js, Hardhat, Metamask)
- Écrire et compiler un smart contract en Solidity
- Le déployer sur un testnet (Goerli, Sepolia, etc.)
- Interagir avec le contrat après le déploiement

💡

Qu'est-ce qu'un smart contract?

Un smart contract est simplement un petit programme qui vit sur la blockchain. Une fois déployé, il reste accessible en permanence et exécute exactement ce pour quoi il a été programmé, sans possibilité de modification ou d'arrêt.

C'est comme un distributeur automatique: vous envoyez des crypto-monnaies et il exécute automatiquement une action prédéfinie.

Étape 1 : Préparer l’environnement de développement

💡

Qu'est-ce que Hardhat?

Hardhat est un environnement de développement pour Ethereum. Il facilite la compilation, le déploiement et le test de smart contracts.

→ https://hardhat.org/

Installer Hardhat

```
mkdir hello-world
cd hello-world
npm init -y
# pnpm init
npm install --save-dev hardhat
# pnpm npm install --save-dev hardhat
```

💡

Qu’est-ce que pnpm ?

C’est simplement npm en plus optimisé : il prend moins de place et est plus rapide.

→ https://pnpm.io/fr/

Initialiser Hardhat

```
npx hardhat
# pnpm dlx hardhat
```

Choisissez : *Create a JavaScript project*

Étape 2 : Ajouter le contrat HelloWorld

💡

Qu'est-ce que Solidity?

Solidity est un langage de programmation créé spécifiquement pour développer des smart contracts sur Ethereum. On peut le comparer à JavaScript ou Python, mais conçu pour écrire des programmes qui s'exécutent sur la blockchain.

Le projet Hardhat contient déjà un exemple de contrat, mais nous allons créer le nôtre. Supprimez le fichier `contracts/Lock.sol` et créez un nouveau fichier `HelloWorld.sol` dans le dossier `contracts/`.

contracts/HelloWorld.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.7.3;
```

```
contract HelloWorld {
    event UpdatedMessages(string oldStr, string newStr);
    string public message;

    constructor(string memory initMessage) {
        message = initMessage;
    }

    function update(string memory newMessage) public {
        string memory oldMsg = message;
        message = newMessage;
        emit UpdatedMessages(oldMsg, newMessage);
    }
}
```

Expliquons ce code:

- `pragma solidity >=0.7.3;` indique la version du compilateur Solidity à utiliser
- `contract HelloWorld { ... }` définit notre contrat, comme une classe en programmation orientée objet
- `event UpdatedMessages(...)` déclare un événement qui sera émis quand le message change
- `string public message;` crée une variable publique qui stocke notre message
- Le `constructor` initialise cette variable quand le contrat est déployé
- La fonction `update` permet de modifier le message et émet un événement pour notifier ce changement

Étape 3 : Configurer le testnet (ex. Sepolia)

Qu'est-ce qu'un testnet?

Un testnet (réseau de test) est une version de la blockchain Ethereum qui fonctionne exactement comme le réseau principal, mais avec de l'argent factice. C'est parfait pour tester vos smart contracts sans risquer de vrais fonds. **Sepolia** est l'un des testnets les plus populaires d'Ethereum.

Pour déployer notre contrat sur un testnet, nous devons configurer Hardhat pour qu'il se connecte au réseau Sepolia.

Installer les dépendances nécessaires

```
npm install --save-dev @nomicfoundation/hardhat-toolbox dotenv
# pnpm install --save-dev @nomicfoundation/hardhat-toolbox dotenv
```

Le package `dotenv` nous permettra de stocker nos clés privées en sécurité.

Créer un fichier `.env` et ajouter dedans :

```
PRIVATE_KEY=ta_cle_privee_metamask_sans_0x
SEPOLIA_RPC_URL=https://sepolia.infura.io/v3/ta-cle-api
```

> Vous pouvez obtenir une clé API gratuite sur https://infura.io

> Vous pouvez trouver votre clé privée dans Rabby.

💡 Infura.io joue un rôle clé dans le développement de smart contracts en fournissant un accès simple et rapide à la blockchain Ethereum (et d'autres réseaux comme Polygon ou Arbitrum) via une API. Plutôt que de devoir exécuter **un nœud Ethereum complet** localement, les développeurs peuvent utiliser Infura pour interagir avec la blockchain — déployer des contrats, lire ou écrire des données — de manière fiable et évolutive. Cela facilite grandement le développement, les tests et la mise en production d'applications décentralisées (dApps).

Modifier `hardhat.config.js`

```
require("@nomicfoundation/hardhat-toolbox");
require("dotenv").config();

module.exports = {
  solidity: "0.8.28",
  networks: {
    sepolia: {
      url: process.env.SEPOLIA_RPC_URL,
      accounts: [process.env.PRIVATE_KEY],
    },
  },
};
```

Étape 4 : Créer le script de déploiement

Maintenant, nous avons besoin d'un script pour déployer notre contrat. Créez un fichier `deploy.js` dans le dossier `scripts/` :

```
const hre = require("hardhat");

async function main() {
```

```
  const HelloWorld = await hre.ethers.getContractFactory("HelloWorld");
  const hello = await HelloWorld.deploy("Bonjour, Blockchain !");
  await hello.waitForDeployment();
  console.log(`Contrat déployé à l'adresse : ${hello.target}`);
}

main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

Ce script:

1. Charge notre contrat HelloWorld
2. Le déploie avec un message initial
3. Attend que le déploiement soit terminé
4. Affiche l'adresse du contrat déployé

Étape 5 : Obtenir des ETH de test (faucet)

Pour déployer notre contrat sur le testnet Sepolia, nous avons besoin d'ETH de test. Un "faucet" (robinet) est un service qui distribue gratuitement des tokens de test.

Allez sur un faucet Sepolia comme :

- https://faucet.triangleplatform.com/ethereum/sepolia
- https://sepolia-faucet.pk910.de/#/

Collez votre adresse de Rabby, récupérez des ETH de test.

Vous recevrez quelques ETH de test qui vous permettront de payer les frais de déploiement et d'interaction avec votre contrat.

Étape 6 : Déployer sur Sepolia

Maintenant que tout est configuré, déployons notre contrat:

```
npx hardhat run scripts/deploy.js --network sepolia
```

Après quelques secondes, vous verrez l'adresse de votre contrat s'afficher. Exemple :

```
Contrat déployé à l'adresse : 0x123abc...
```

Vous pouvez vérifier que votre contrat est bien déployé en visitant https://sepolia.etherscan.io/ et en recherchant l'adresse de votre contrat.

> 💡 Un explorateur blockchain est un outil en ligne qui permet de visualiser et rechercher des données enregistrées sur une blockchain. Il offre une interface pour consulter des informations comme les transactions, les blocs, les adresses de portefeuilles, les smart contracts, ou encore l'état du réseau. Par exemple, sur Ethereum, un explorateur comme Etherscan permet de vérifier si une transaction a été confirmée, de suivre les mouvements d'un token, ou de consulter le code d'un contrat intelligent.

Étape 7 : Interagir avec le contrat

Maintenant que notre contrat est déployé, interagissons avec lui. Créez un fichier `interact.js` dans le dossier `scripts/` :

```
const hre = require("hardhat");

async function main() {
  // Remplacez par l'adresse de votre contrat déployé
  const contractAddress = "VOTRE_ADRESSE_DEPLOYEE";
```

```
  // On se connecte à notre contrat existant
  const HelloWorld = await hre.ethers.getContractFactory("HelloWorld");
  const hello = await HelloWorld.attach(contractAddress);

  // On lit le message actuel
  const currentMessage = await hello.message();
  console.log("Message actuel :", currentMessage);

  // On met à jour le message
  const tx = await hello.update("Nouveau message !");
  await tx.wait();
  console.log("Message mis à jour !");

  // On vérifie que le message a bien été mis à jour
  const newMessage = await hello.message();
  console.log("Nouveau message :", newMessage);
}

main().catch((error) => {
  console.error(error);
  process.exitCode = 1;
});
```

> N'oubliez pas de remplacer `"VOTRE_ADRESSE_DEPLOYEE"` par l'adresse de votre contrat.

Executer le code.

Chaque interaction de modification (comme l'appel à la fonction `update`) crée une transaction sur la blockchain qui doit être executée, ce qui peut prendre quelques secondes.

Étape 8 : Rendu du TP

Pour finaliser ce TP, vous devez faire une transaction avec votre **prénom** en **data** et envoyez l'URL de transaction sur l'explorer dans le formulaire de rendu.

L'explorer : https://sepolia.etherscan.io/

Le formulaire : https://forms.cloud.microsoft/e/62A9A7muwV

Conclusion

Félicitations! Vous venez de :

1. Écrire votre premier smart contract en Solidity
2. Configurer un environnement de développement blockchain
3. Déployer votre contrat sur un vrai réseau de test Ethereum
4. Interagir avec votre contrat déployé