如果您是区块链开发的初学者，还不知道如何开始，或者您只是想了解怎样部署智能合约并与之进行交互，这篇教程就是为您准备的。 通过使用虚拟钱包 (Metamask)、Solidity、Hardhat 和 Alchemy（如果您不理解这些名词的含义，不用担心，后续我们会进行解释），我们将演示在 Ropsten 测试网络上创建并部署一个简单的智能合约。

在本教程的第 2 部分，我们将学习在智能合约部署后如何与之进行交互，第 3 部分将学习如何在 Etherscan 上发布智能合约。

如果您有任何问题，请随时在 Alchemy Discord 中提出！

步骤 1：连接到以太坊网络

有多种方法可以向以太坊链发起连接请求。 为了简单起见，我们将使用一个 Alchemy 上的免费帐户。 Alchemy 是一个区块链开发者平台和 API，允许我们与以太坊进行通信，而无需运行我们自己的节点。 平台还有用于监测和分析的开发者工具，我们将在本教程中利用这些工具来了解我们智能合约部署中的情况。 如果您还没有 Alchemy 账户，您可以在这里免费注册。

步骤 2：创建应用程序（和应用程序接口密钥）

创建了 Alchemy 账户后，您可以通过创建应用程序来生成应用程序接口密钥。 我们可以用它向 Ropsten 测试网络发起请求。
如果您不熟悉测试网络，请查看此页面。

1. 在 Alchemy 仪表板中，将鼠标悬停在导航栏中的“应用程序”上，单击“创建应用程序”并前往此页面。

2. 将您的应用命名为“Hello World”，提供简短的描述，选择“Staging”作为环境（用于您的应用程序簿记），选择“Ropsten”网络。

3. 点击“创建应用程序”即可！ 您的应用程序应该会出现在下面的表格中。

步骤 3：创建一个以太坊账户（地址）

我们需要一个以太坊帐户来发送和接收交易。 在本教程中，我们将使用 MetaMask——浏览器中的虚拟钱包，用来管理您的以太坊账户地址。
关于交易的详细信息。

您可以在这里免费下载并创建一个 Metamsk 账户。
创建账户时，或者如果您已经有一个账户时，确保切换到右上方的“Ropsten 测试网络”（这样我们就不会用实际货币进行交易）。

步骤 4：从水龙头添加以太币

为了将我们的智能合约部署到测试网络，我们需要一些虚拟以太币。
要获得以太币，您可以访问 Ropsten 水龙头，输入您的 Ropsten 帐户地址，然后点击“发送 Ropsten 以太币”。
由于网络原因，您接收虚拟以太币可能需要一些时间。 您应该会很快在您的 MetaMask 帐户中看到以太币！

步骤 5：查看账户余额

为了核实我们的余额，可以使用 Alchemy 的创作者工具发出 eth_getBalance 请求。 这将返回我们钱包中的以太币金额。 输入您的 Metamask 帐户地址并单击“发送请求”后，您应该会看到这样的响应：

```
{ "jsonrpc": "2.0", "id": 0, "result": "0x2B5E3AF16B1880000"
 }
```

> **注意：**结果以 wei 为单位，而非 ETH。 Wei 是以太坊最小的面额。 将 wei 转换为 ETH 的公式为：1 eth = 1018 wei。 因此，如果我们将 0x2B5E3AF16B1880000 转换为十进制，我们会得到 5*10¹⁸，即 5 ETH。
>
> 呼！ 我们的虚拟货币到账了<Emoji text="" size={1} />。

步骤 6：初始化我们的项目

首先，需要为我们的项目创建一个文件夹。 导航到您的命令行，然后输入：

```
mkdir hello-worldcd hello-world
```

现在我们进入了项目文件夹，我们将使用 npm init 来初始化项目。
如果你尚未安装 npm，请遵循这些说明（我们还需要 Node.js，所以请一并下载！）。

```
npm init
```

如何回答安装中的问题并不重要，以下提供一个回答的样例供参考：

```
package name: (hello-world)version: (1.0.0)description: hell
o world smart contractentry point: (index.js)test command:gi
t repository:keywords:author:license: (ISC)About to write to
 /Users/.../.../.../hello-world/package.json:{  "name": "hel
lo-world",  "version": "1.0.0",  "description": "hello world
 smart contract",  "main": "index.js",  "scripts": {    "te
st": "echo \\"Error: no test specified\\" && exit 1"  },  "a
uthor": "",  "license": "ISC"}
```

批准 package.json，我们就可以进行下一步了！

步骤 7：Hardhat

安全帽是一个用于编译、部署、测试和调试以太坊软件的开发环境。
在部署到实时链上之前，它帮助开发者在本地构建智能合约和去中心化应用程序。

在我们的 Hello-world 项目中运行：

```
npm install --save-dev hardhat
```

查看此页面，了解更多有关安装说明的详细信息。

步骤 8：创建安全帽项目

在我们的项目文件夹中运行：

```
npx hardhat
```

然后应该能看到一条欢迎消息和选项，用于选择您想要做的事情。
选择“创建一个空的 hardhat.config.js”：

```
888    888                      888 888               888888
    888                      888 888                 888888
 888                      888 888               888888888888
```

```
8  8888b.  888d888 .d88888 88888b.   8888b.  888888888    88
8     "88b 888P"  d88" 888 888 "88b     "88b 888888    888 .
d888888 888    888  888 888  888 .d888888 888888    888 888
 888 888    Y88b 888 888  888 888  888 Y88b.888    888 "Y888
888 888     "Y88888 888  888 "Y888888  "Y888 Welcome to Hard
hat v2.0.11 ??What do you want to do? …Create a sample proje
ct? Create an empty hardhat.config.jsQuit
```

这将生成一个 hardhat.config.js 文件，我们将用以配置项目的所有设置（步骤 13）。

步骤 9：添加项目文件夹

为了使我们的项目有条理，我们将创建两个新的文件夹。
在您的命令行中导航到项目的根目录，然后输入：

```
mkdir contractsmkdir scripts
```

- contracts/ 是保存我们的 Hello world 智能合约代码文件的位置
- scripts/ 是我们存放脚本的位置，用于部署我们的合约和与之交互。

步骤 10：编写合约

您可能要问自己，天啊，到底什么时候才能写代码？？ 答案是，就是现在，步骤 10。

在您最喜欢的编辑器中打开 hello-world 项目（我们喜欢的是 VSCode）。
智能合约是用一种叫 Solidity 的语言编写的，我们将用它来编写我们的 HelloWorld.sol 智能合约。

1. 导航到“contracts”文件夹并创建一个名为 HelloWorld.sol 的新文件
2. 下面是我们将用于本教程的来自以太坊基金会的 Hello World 智能合约样例。 复制下面的内容并粘贴到 HelloWorld.sol 文件，并一定要阅读注释以了解该合约的工作内容：

```
// Specifies the version of Solidity, using semantic version
ing.// Learn more: https://solidity.readthedocs.io/en/v0.5.1
0/layout-of-source-files.html#pragmapragma solidity ^0.7.0;/
```

```
/ Defines a contract named `HelloWorld`.// ??????????????????? Once deployed, a contract resides at a specific address on the Ethereum blockchain. Learn more: https://solidity.readthedocs.io/en/v0.5.10/structure-of-a-contract.htmlcontract HelloWorld {   // Declares a state variable `message` of type `string`.   // ????????????????????? The keyword `public` makes variables accessible from outside a contract and creates a function that other contracts or clients can call to access the value.   string public message;   // Similar to many class-based object-oriented languages, a constructor is a special function that is only executed upon contract creation.   // ????????????? Learn more:https://solidity.readthedocs.io/en/v0.5.10/contracts.html#constructors   constructor(string memory initMessage) {      // Accepts a string argument `initMessage` and sets the value into the contract's `message` storage variable).      message = initMessage;   }   // A public function that accepts a string argument and updates the `message` storage variable.   function update(string memory newMessage) public {      message = newMessage;   }}
```

这是一
个非常简单的智能合约，创建时存储了一条消息，而且可以通过调用 update
功能来更新消息。

步骤 11：将 Metamask 和 Alchemy 连接至您的项目

我们创建了 Metamask 钱包、Alchemy
帐户，并且编写了一个智能合约，现在是将这三者连起来的时候了。

从虚拟钱包发送的每笔交易都需要使用您独有的私钥签名。
为了给程序提供此项许可，我们可以安全地将私钥（和 Alchemy
应用程序接口密钥）存储在一个环境文件中。

> 如需了解更多关于发送交易的信息，请查看关于使用 web3 发送交易的本教程。

首先，在项目目录中安装 dotenv 软件包：

```
npm install dotenv --save
```

然后在我们的项目根目录中创建 .env 文件，并将您的 Metamask 私钥和超文本传输协议 Alchemy 应用程序接口网址加入其中。

- 遵循这些说明导出您的私钥
- 请从下方获取超文本传输协议 Alchemy 应用程序接口网址

复制 Alchemy 应用程序接口网址

您的 .env 文件应该类似：

```
API_URL = "https://eth-ropsten.alchemyapi.io/v2/your-api-key"PRIVATE_KEY = "your-metamask-private-key"
```

为了将这些变量和代码连接，我们将在步骤 13 中调用 hardhat.config.js 文件中的这些变量。

<InfoBanner isWarning=>
不要提交 .env！ 请确保永远不要与任何人共享或公开您的 .env 文件，因为这样做会泄露您的秘密。 如果您使用版本控制，请将您的 .env 添加到 gitignore 文件中。

步骤 12：安装 Ethers.js

Ethers.js 是一个软件库，通过以更加方便用户的方法打包标准 JSON RPC 方法，从而更容易与以太坊互动，并向以太坊提出请求。

安全帽使我们更容易将插件集成到工具和扩展功能中。 我们将利用 Ethers 插件完成合约部署（Ethers.js 有非常简洁的部署方法）。

在您的项目目录中输入：

```
npm install --save-dev @nomiclabs/hardhat-ethers "ethers@^5.0.0"
```

下一步中，我们还将在 hardhat.config.js 中使用 ethers。

步骤 13：更新 hardhat.config.js

到目前为止，
我们已经添加了几个依赖库
和插件，现在我们需要更新 hardhat.config.js，以便项目使用所有这些新的组件。

按如下所示更新您的 hardhat.config.js 代码：

```
require('dotenv').config();require("@nomiclabs/hardhat-ether
s");const { API_URL, PRIVATE_KEY } = process.env;/** @type
import('hardhat/config').HardhatUserConfig*/module.exports =
 {   solidity: "0.7.3",   defaultNetwork: "ropsten",   netwo
rks: {      hardhat: {},      ropsten: {         url: API_UR
L,         accounts: [`0x${PRIVATE_KEY}`]      }   },}
```

步骤 14：编写合约

为了确保一切正常，我们来编译一下合约。 compile
任务是安全帽的内部任务之一。

在命令行中运行：

```
npx hardhat compile
```

您可能会看到关于 SPDX license identifier not provided in source file
的警告，但不用担心，希望其它的均看起来正常！ 如果遇到问题，您可以随时在 Alchemy cord 社区中发消息询问。

步骤 15：编写部署脚本

合约已经写完，配置文件也准备妥当，现在是写合约部署脚本的时候了。

转到 scripts/ 文件夹，创建一个名为 deplos.js 的新文件，在其中添加以下内容：

```
async function main() {   const HelloWorld = await ethers.ge
tContractFactory("HelloWorld");   // Start deployment, retur
ning a promise that resolves to a contract object   const he
llo_world = await HelloWorld.deploy("Hello World!");   conso
```

```
le.log("Contract deployed to address:", hello_world.address)
;}main()  .then(() => process.exit(0))  .catch(error => {
 console.error(error);    process.exit(1);  });
```

安全帽在合约教程中对这些代码的每一行均提供了很好的解释，我们在这里直接引用他们的解释。

```
const HelloWorld = await ethers.getContractFactory("HelloWor
ld");
```

ethers.js 中的 ContractFactory 是用于部署新智能合约的抽象对象。因此这里的 HelloWorld 是我们 hello world 合约实例的工厂。 使用 hardhat-ethers 插件时，ContractFactory 和 Contract 实例默认与第一个签名账户相连。

```
const hello_world = await HelloWorld.deploy();
```

调用 ContractFactory 代码中的 deploy() 函数会启动合约部署，然后返回解析为 Contract 的 Promise。 这个对象包括我们智能合约中每个函数的对应调用方法。

步骤 16：部署合约

我们终于准备好部署我们的智能合约啦！ 导航到命令行后运行：

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

您会看到类似以下所示的信息：

```
Contract deployed to address: 0x6cd7d44516a20882cEa2DE9f205b
F401c0d23570
```

如果我们访问 Ropsten etherscan
并搜索我们的合约地址，应该能够看到它已经成功部署。 交易将类似以下：

From 地址应该和您的 Metamask 账户地址相同，而 To 地址会显示“Contract Creation”，但如果我们点击进入交易，我们会在 To 字段看到我们的合约地址：

恭喜！ 您刚刚在以太坊区块链上部署了一个智能合约

为了更深入了解到底发生了什么，我们转到 Alchemy 仪表板中的 Explorer

选项卡。 如果您有多个 Alchemy
应用程序，请确保按应用程序筛选，然后选择“Hello World”。

在这里您会看到一系列的 JSON-RPC 调用，都是在我们调用 .deploy()
函数时，Hardhat/Ethers 替我们在后端完成的。 这里有两项重要调用，一个是 eth_sendRawTransaction，这是实际将我们的合约写入 Ropsten 链的请求，另一个是 eth_getTransactionByHash
，此为读取有关我们交易给定哈希值的请求（即 交易时的典型模式）。
如需了解更多关于发送交易的信息，请查看关于使用 Web3 发送交易的本教程