

连接并部署到 Ethereum 网络

内容出自**C2N数字游民部落**

加入社区获得更多材料: <https://discord.gg/R2qW7yqg>

连接并部署到 Ethereum 网络

了解 Ethereum 网络并将其用于开发、测试和生产。

学习目标

学完本模块后,你将能够:

- 识别 Ethereum 中提供的公用和专用网络。
- 说明如何为主网准备解决方案。
- 结合使用 MetaMask 和 Infura 将解决方案连接到网络。

先决条件

- 具有 C、Python 或 JavaScript 等编程语言的使用经验
- 对编程概念有基本的了解
- 能够使用命令行创建新目录并安装程序
- 安装了 Node.js
- 已安装 Hardhat
- 已安装 Visual Studio Code
- 已安装 Ethereum 的区块链开发工具包

简介

Ethereum 协议包括多个独立的环境(称为网络)。网络可用于开发和测试区块链解决方案,或其部署到生产环境。

要使用 Ethereum 网络,首先需要了解各网络之间的区别。我们将介绍连接并部署到每个网络环境的步骤。为了创建区块链,我们将了解如何选择将其设为专用还是公用。回顾如何从开发网络转到测试网络再转到生产网络。

此模块的目标是让你熟悉 Ethereum 协议中的网络选项。你将探索可用于与网络选项进行交互的工具，并了解如何使用不同的网络。

了解公用 Ethereum 网络

Ethereum 协议由多个公用网络组成。不同的 Ethereum 网络可以具有不同的特性、用途、功能和共识机制。Ethereum 当前有四个称为“测试网”的测试网络。它有一个称为“主网”的生产网络。

主网概述

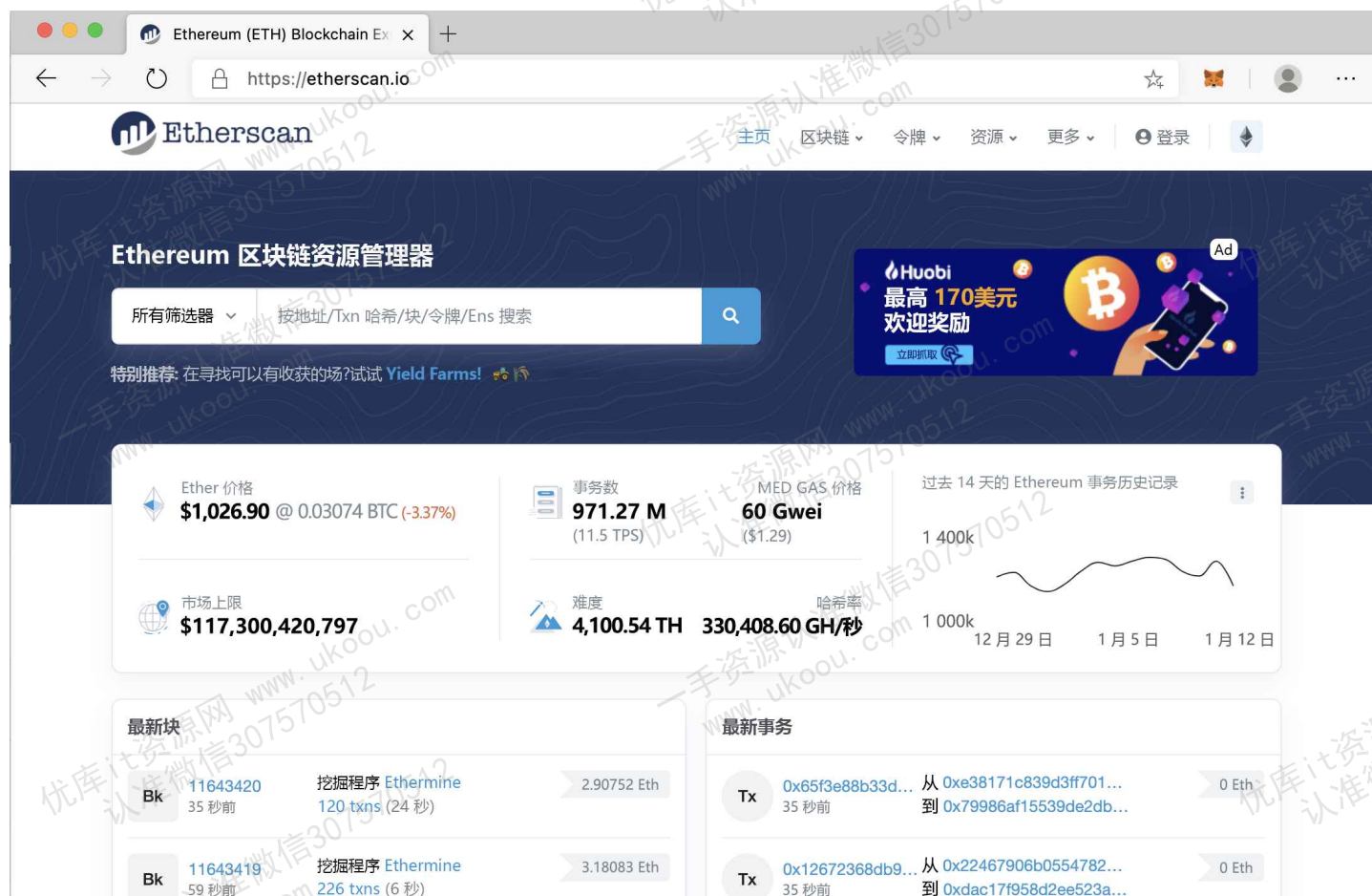
主网（“主网络”的简称）是一个真正的公用 Ethereum 区块链。部署到主网的应用程序可以交换和使用信息并彼此进行交互。

所部署的应用程序可以充分利用去中心化区块链的功能。没有“中央集权”。主网是完全去中心化的。

各种类型的代币和应用程序都可以部署到主网。交易部署到主网后，它们是不可变的，无法更改。

每项交易都有实际成本。这些成本需要以太 (ETH) 加密货币。

可以使用 **Etherscan** 查看 Ethereum 主网上的所有区块。Etherscan 会显示最新的已挖掘区块和交易。所有区块都可供检查。



Ethereum 测试网

Ethereum 有四个公用测试网。每个测试网都有不同的部署方法和流程。在将应用程序部署到主网之前，测试网会在一个实时公用环境中暂存并测试应用程序。

测试网使用[工作证明 \(PoW\)](#) 或[授权证明 \(PoA\)](#) 共识协议来确定如何将新的交易区块添加到网络中。下面是每个协议的简要概述：

- PoW：矿机通过解决加密哈希问题来挖掘新的区块并决定哪些交易是该区块的一部分。
- PoA：区块验证器在网络上验证其身份，以决定哪些交易将成为链中下一个区块的一部分。

测试网需要使用“测试以太币”。测试以太币是免费的，你可以从 faucets 访问它。你需要向 faucets 提供一个帐户地址来接收一定数量的测试以太币。

Faucets 已成为获取测试网的测试以太币的主要方式。为了开发人员和测试人员的利益，公用测试网络由社区管理。Faucets 保护测试网免受垃圾邮件攻击，因为该以太币由受信任参与方控制。

测试网比较（现在可用的是Sepolia，另外几个现在不可用了）

让我们看看 [Ethereum 测试网](#) 及其相关特性。

Sepolia

Sepolia 是以太坊的一个新测试网，用于替代使用PoW机制的Ropsten。自2022年起，Sepolia采用与以太坊主网相同的PoS共识机制，提供更环保的测试环境。

主要特征和用途

- **PoS共识机制**：允许开发者在无风险环境中测试PoS相关功能。
- **开发与测试**：适用于智能合约和应用的开发测试。
- **网络升级模拟**：用于模拟并评估主网升级影响。

使用方法

- **钱包配置**：需要配置MetaMask等钱包以连接到Sepolia，使用的RPC URL格式为 https://sepolia.infura.io/v3/{your_project_id}，链ID为11155111。
- **获取测试币**：通过Sepolia水龙头系统获取，用于测试网交易和部署。

Sepolia与其他测试网的比较

与 Goerli 和 Rinkeby 这类也使用 PoS 的其他测试网相比，Sepolia 提供了一个更接近于即将到来的以太坊网络升级状态的测试环境。随着以太坊生态系统继续发展，测试网络如 Sepolia 将发挥越来越重要的作用，以确保以太坊网络的平稳运行和升级。

总的来说，Sepolia 测试网是开发者在无风险环境中开发和测试以太坊项目的理想场所，特别是对于那些想要在 PoS 架构下进行实验的用户来说。

Sepolia提供接近未来以太坊网络升级状态的测试环境，使其成为PoS架构下实验的理想选择。

Ropsten (已失效)

Ropsten 是一个 PoW 共识协议。它的功能最接近主网。Ropsten 以瑞典的一个地铁站命名，自 2016 年以来就存在了。有人说它最好地重现了主网的状况。

更多详细信息：

- 支持的客户端：
 - [Geth](#)
 - [OpenEthereum](#)
- 区块时间：30 秒或更少
- Ropsten 特定的链接：
 - [GitHub](#)

Kovan (已失效)

[Kovan](#) 是一个 PoA 测试网，以新加坡的一个地铁站命名。其以太币必须从 faucet 申请，并由受信任参与方控制。由于这一特性，Kovan 对垃圾邮件攻击免疫。

更多详细信息：

- 支持的客户端：[OpenEthereum](#)
- 区块时间：4 秒
- Kovan 特定的链接：
 - [GitHub](#)
 - [网站](#)

Goerli (已失效)

PoA 跨客户端测试网 [Goerli](#) 以柏林的一个地铁站命名。此测试网的目标是在各种客户端上广泛使用。它足够强健，可以保证一致的可用性。它开始于 2018 年的 Goerli 计划。

更多详细信息：

- Goerli 支持大多数客户端，包括以下选项：
 - [Geth](#)
 - [OpenEthereum](#)
 - [Nethermind](#)
- 区块时间：平均为 15 秒
- Goerli 特定的链接：
 - [Faucet](#)
 - [资源管理器](#)

- [GitHub](#)
- [网站](#)

Ropsten 据说是与主网最相似的测试网。它是第一个主要测试网。。在部署到主网之前，请在 Sepolia 上进行部署和测试。

用于部署到测试网和主网的客户端和 API

Ethereum 设计为提供多个客户端。客户端可以由不同的团队开发并可使用不同的编程语言。此多元化使网络变得更强大且更多样化。这样做的目的是实现多样化，不让任何客户端处于支配地位。此设计降低了出现单一故障点的几率。

客户端

下面是一些常见的 [Ethereum 客户端](#)：

- [Geth 客户端](#)
- Go Ethereum（也称为 Geth）是协议最初的实现之一。此客户端拥有最大的用户群，也最为常用。Geth 为区块链开发人员和用户提供多种工具。
- Geth 客户端采用 GO 编写，并且是开源的。它根据 GNU 宽通用公共许可证 (LGPL) 版本 3 授予许可。
- [OpenEthereum](#)
- OpenEthereum 客户端对于使用协议的所有工作都很有用。它简化了自定义、产品集成、数据存储和内存管理等任务。它旨在成为最轻便、最快且最安全的 Ethereum 客户端。
- OpenEthereum 是采用 Rust 编程语言编写的。它根据 GNU 通用公共许可证 (GPL) 版本 3 进行许可。
- [Nethermind](#)
- Nethermind 提供了世界上速度最快的 .NET Core Ethereum 客户端和 P2P 数据市场。它为 Ethereum 区块链解决方案的开发人员提供咨询服务。

API

下面是一些常见的 Ethereum API：

- [Infura](#)
- Infura API 套件通过 HTTPS 和 WebSocket 提供对 Ethereum 和 IPFS 网络的即时访问。可以使用此直观的界面连接到所有测试网的终节点。Infura 支持 Truffle Suite 和适用于 Ethereum 的 Visual Studio Code 区块链开发工具包。
- [MetaMask](#)

- 部署到测试网或主网时，MetaMask 客户端提供了一个强健的界面和钱包，用于连接到 Ethereum 区块链并与之进行交互。
- 在测试网上使用 MetaMask 发送以太币和代币非常简单。此客户端提供了一个简单的界面，用于选择和使用不同的 Ethereum 网络。当你需要与开发网络进行交互时，可以使用 MetaMask，它简化了连接到 localhost 8545 或自定义 RPC 的操作，方便你使用 Ganache 和 Truffle 进行连接。同样，MetaMask 具有到公用测试网和主网的预定义连接。
- 如果连接到主网，请小心保护你的私钥。此连接使用真实的以太币。

了解专用 Ethereum 网络

如果网络的节点是隔离的，Ethereum 网络被视为专用网络。在专用网络中，节点永远不会连接到公用网络（如主网或测试网）。专用 Ethereum 网络包括开发网络或联盟网络。

开发网络

开发 Ethereum 应用程序时，一开始是在专用网络上运行应用。在将应用程序部署到生产环境之前，需要先了解它的工作原理。这种类型的专用网络称为开发网络。

在开发网络中构建区块链解决方案类似于在本地服务器上设计 Web 应用。可以在同一环境中设计、构建、测试和迭代解决方案。

在开发网络上，可快速创建、测试和迭代对解决方案的更改。与公用测试网相比，在开发网络中可以更顺畅、更快速地迭代解决方案。[Ganache](#) 和 [Hardhat](#) 等工具最常用来运行个人 Ethereum 开发网络。

适用于联盟网络的解决方案

联盟网络是经过授权的。联盟商需要受邀参与。联盟网络可确保安全性、隐私性、合规性和性能。

联盟区块链的众多选项包括 Hyperledger Besu、R3 Corda 和 Quorum。让我们探索两个常用的联盟区块链选项：Hyperledger Besu 和 R3 Corda。

Hyperledger Besu

[Hyperledger Besu](#) 是一个开源 Ethereum 客户端。它根据 Apache 许可证 2.0 开发并采用 Java 编写。

Besu 用来开发需要在专用网络中实现安全的高性能交易处理的企业应用程序。它有一个命令行界面和一个 JSON-RPC API。

Besu 运行、维护、调试和监视 Ethereum 网络中的节点。该 API 支持典型的 Ethereum 功能，例如：

- 以太币挖掘
- 智能合约开发
- 去中心化应用程序 (Dapp) 开发

Besu 是一个常用的 Ethereum 客户端。它的独特之处在于，它既可以用于公用网络，又可以用于专用的基于联盟的网络。

Besu 可以通过各种[方式](#)进行部署。有关配置和部署的详细信息，请参阅 [Hyperledger Besu 网站](#)。

R3 Corda

[Corda 平台](#)是一个经过授权的专用区块链。它侧重于支持实体之间受信任的通信、交互和交易。

[Corda Enterprise](#) 提供 Corda 开源代码库的核心属性。其中一项重要功能是支持许可的软件的企业业务需求。

在 Ethereum 中，客户端应用程序在交易生命周期中扮演着关键角色。在将标识提交到节点之前，客户端应用程序将获取签名密钥和签名交易。

在 Corda 平台中，客户端应用程序（通常称为 [CorDapp](#)）完全驻留在 Corda 节点上。密钥和交易签名标识仅存在于节点上，而不存在于客户端。客户端应用程序会触发已在目标 Corda 节点上注册的工作流，并监视工作流进度。

[Corda Visual Studio Code 扩展](#)支持 Corda 开发。若要安装该扩展，请在 Visual Studio Code 中选择“扩展”图标。然后输入“Corda”以查找并安装该扩展。

为部署到主网做准备

在部署到 Ethereum 主网之前，你需要全面测试并审核你的代码。在主网中工作需要真实的以太币，这需要花费真实的货币，并且费用可能会快速增加！

开发、测试并审核代码之后，请在至少一个测试网上运行项目，并解决任何问题。大多数项目在部署到主网之前都需要经历一个全面的流程。该流程包括审核、测试、安全性和治理。此流程将主网出错的风险和成本降到最低。

准备并部署到 Ethereum 主网的流程涉及一系列步骤：

1. 审核智能合约。
2. 验证源代码。
3. 管理密钥。
4. 处理项目治理。

以下各部分详细介绍了这些步骤。

审核智能合约

在将智能合约部署到公用网络之前，必须[审核并评估](#)其安全性。部署智能合约后，网络上的任何人都可以使用任何有效负载将交易直接发送到合约。合约的所有源代码和状态都是公开提供的。

由于区块链上的交易是不可变的，因此交易在提交后是永久性的。交易可能会导致资金被盗和其他恶意活动。因此，请在部署前审核智能合约。

验证源代码

在将合约部署到主网后，立即将 Solidity 代码提交给第三方，以[验证智能合约源代码](#)。公共服务（例如 [Etherscan](#)）会编译合约代码并根据已部署的程序集对其进行验证。这样，在区块链浏览器中查看合约代码的用户可以确认合约对应于在该地址运行的程序集。

若要验证并发布你的 Solidity 源代码，请首先输入合约源代码。如果生成的字节码与现有的创建地址字节码匹配，则合约得到验证。然后，合约源代码会被发布，任何人都可以对其进行公开验证。

安全地管理密钥

部署到主网时，必须[安全地管理私钥](#)。这一步需要严格的预防措施，以防你的私钥泄露、丢失或被盗。

密钥管理不善已导致了多起重大的盗窃和丢失事件。你用来部署智能合约并与之进行交互的帐户持有真实的以太币。它们是黑客的攻击目标。安全地存储私钥的常用方法包括[硬件钱包](#)和[冷存储](#)，即从不连接到任何网络的计算机。

处理项目治理

去中心化[项目的管理](#)采用各种不同的方式，具体取决于社区和用户群。通常会创建组织来决定如何管理运行中的去中心化系统的更新和其他方面。

可以通过多种方式来管理项目治理。一小群受信任的管理员可提供管理，也可以由所有的项目利益干系人以公开投票方式进行管理。这里没有固定的答案。此决策取决于你正在构建的解决方案以及你的社区和用户是谁。

练习 - 制作一个待办事项列表并将其部署到开发网络

在本练习中，你将开发一个任务管理器。你将使用它连接到 Ropsten 测试网络并在其上进行部署。

在本练习的第一部分，你的重点是创建项目，添加智能合约，然后将其部署到开发网络。

在本教程中，你将：

- 使用 Visual Studio Code 创建项目。
- 使用 Hardhat 进行编译和部署。
- 使用 Hardhat 本地网络作为区块链开发服务器。

创建一个新的hardhat项目

1. 在终端或命令提示符窗口中，通过输入 `mkdir todolist` 创建一个名为“todolist”的新目录。
2. 通过输入 `cd todolist` 转到新创建的目录。
3. 依次输入一下命令：


```
1 npm init -y //创建空项目
2 npm i -D hardhat //安装hardhat依赖
3 npx hardhat init // hardhat 初始化JavaScript项目
```

4. 在 Visual Studio Code 中, 打开 todolist 文件夹。

5. 在 Visual Studio Code 中的 contracts 目录中创建一个文件。将其命名为 TodoList.sol。在新文件中, 粘贴以下代码。

```
1 // SPDX-License-Identifier: MIT
2 pragma solidity >=0.4.22 <0.9.0;
3 contract TodoList {
4     uint public taskCount = 0;
5     struct Task {
6         uint id;
7         string taskname;
8         bool status;
9     }
10    mapping(uint => Task) public tasks;
11    event TaskCreated(
12        uint id,
13        string taskname,
14        bool status
15    );
16    event TaskStatus(
17        uint id,
18        bool status
19    );
20    constructor() {
21        createTask("Todo List Tutorial");
22    }
23    function createTask(string memory _taskname) public {
24        taskCount ++;
25        tasks[taskCount] = Task(taskCount, _taskname, false);
26        emit TaskCreated(taskCount, _taskname, false);
27    }
28    function toggleStatus(uint _id) public {
29        Task memory _task = tasks[_id];
30        _task.status = !_task.status;
31        tasks[_id] = _task;
32        emit TaskStatus(_id, _task.status);
33    }
34 }
```

部署到本地网络

1. 在 ignition/modules 文件夹中创建 TodoList.js 文件。在此文件中，粘贴以下代码以部署 TodoList 智能合约。

```
1 const { buildModule } = require("@nomicfoundation/hardhat-ignition/modules");
2
3 module.exports = buildModule("TodoListModule", (m) => {
4   const todoList = m.contract("TodoList", []);
5   return { todoList };
6 });
7
```

2. 在主项目目录中，打开 hardhat.config.js。代码应如下所示。

```
1 require("@nomicfoundation/hardhat-toolbox");
2
3 /** @type import('hardhat/config').HardhatUserConfig */
4 module.exports = {
5   solidity: "0.8.24",
6   networks: {
7     hardhat: {},
8   },
9 };
10
```

3. 启动本地网络: `npx hardhat node`

4. 新的命令行界面中部署 TodoList 到本地网络

```
npx hardhat ignition deploy ./ignition/modules/TodoList.js --network localhost
```

```
→ todolist npx hardhat ignition deploy ./ignition/modules/TodoList.js --network localhost
Compiled 2 Solidity files successfully (evm target: paris).
Hardhat Ignition 🚀

Deploying [ LockModule ]

Batch #1
  Executed LockModule#Lock
[ LockModule ] successfully deployed 🚀

Deployed Addresses

LockModule#Lock - 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

练习 - 部署到 Sepolia 测试网络

现在我们有了一个智能合约。我们已将其成功部署到我们的开发网络。接下来，我们将重点介绍如何部署到 Sepolia 测试网络。

练习概述

在本教程中，我们将通过 MetaMask 使用测试以太坊部署到 Sepolia。部署过程需要设置一个 [Infura](#) 帐户或者 [Alchemy](#) 账户，用于连接并部署到 Sepolia 测试网。部署后，可以使用 Sepolia 测试网浏览器检查已部署到测试网的区块。

练习设置

为了设置练习，你将执行以下操作：

1. 安装并设置 MetaMask。
2. 获取测试以太坊。
3. 安装 Alchemy 并将终结点链接到 Sepolia 测试网络。

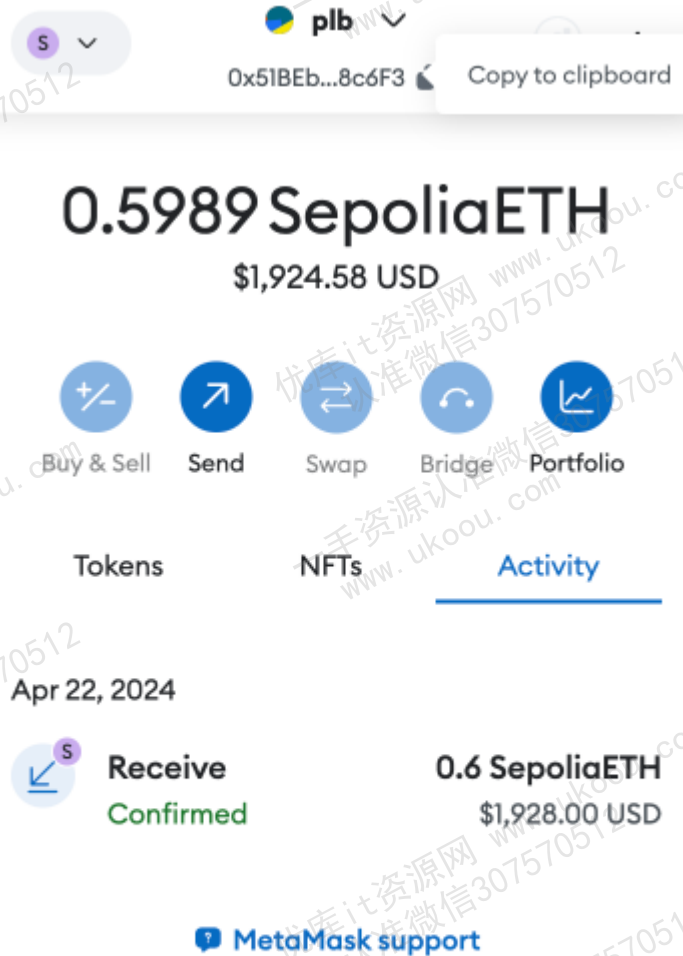
设置 MetaMask

[安装并设置 MetaMask](#)（如果尚未这样做）。然后，在浏览器中登录到你的帐户。

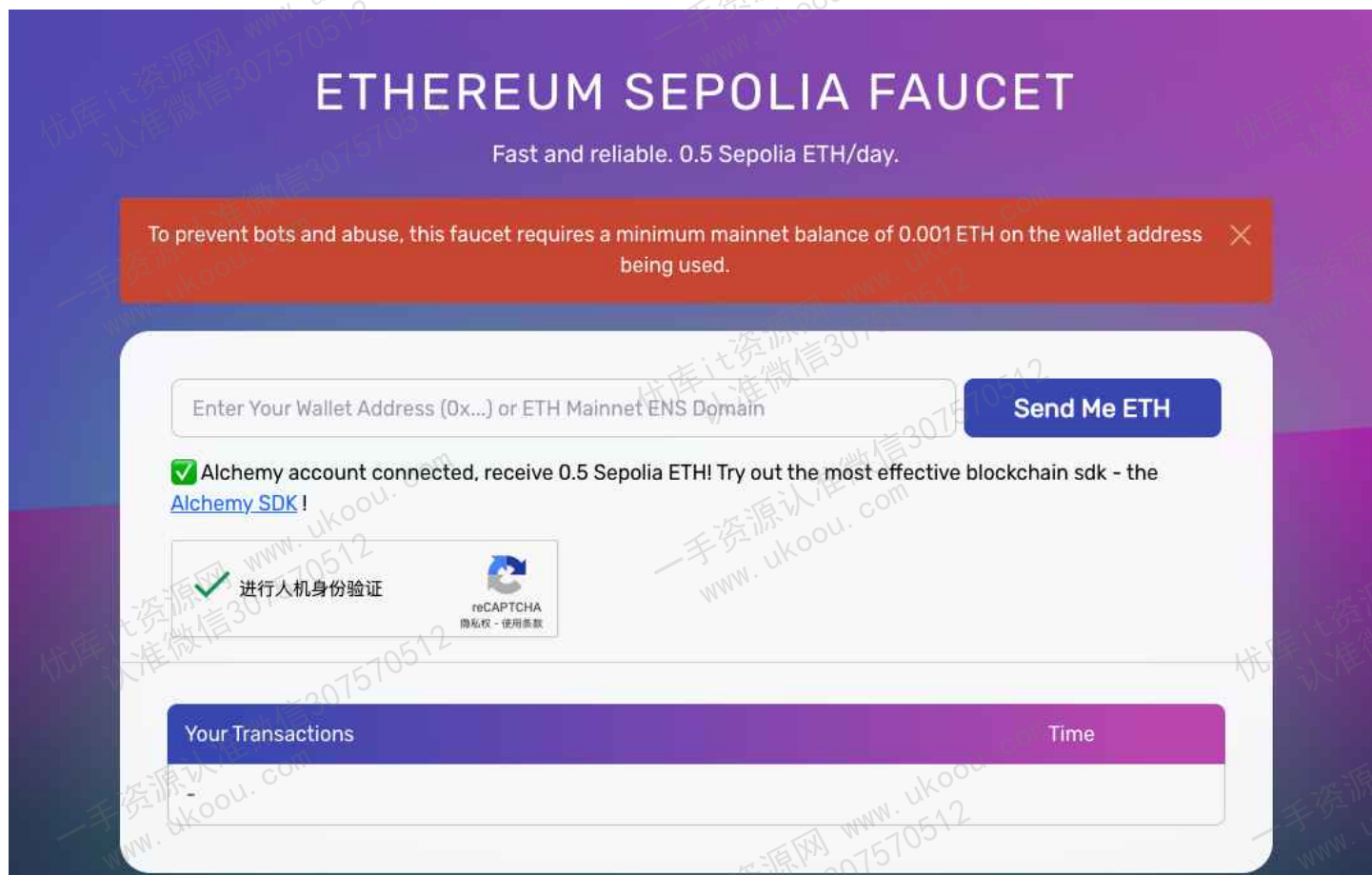
领水到 MetaMask Sepolia 测试网络

使用你的 MetaMask 帐户连接到 Sepolia 测试网络。首先从 Sepolia 测试 Faucet 获取测试以太坊：

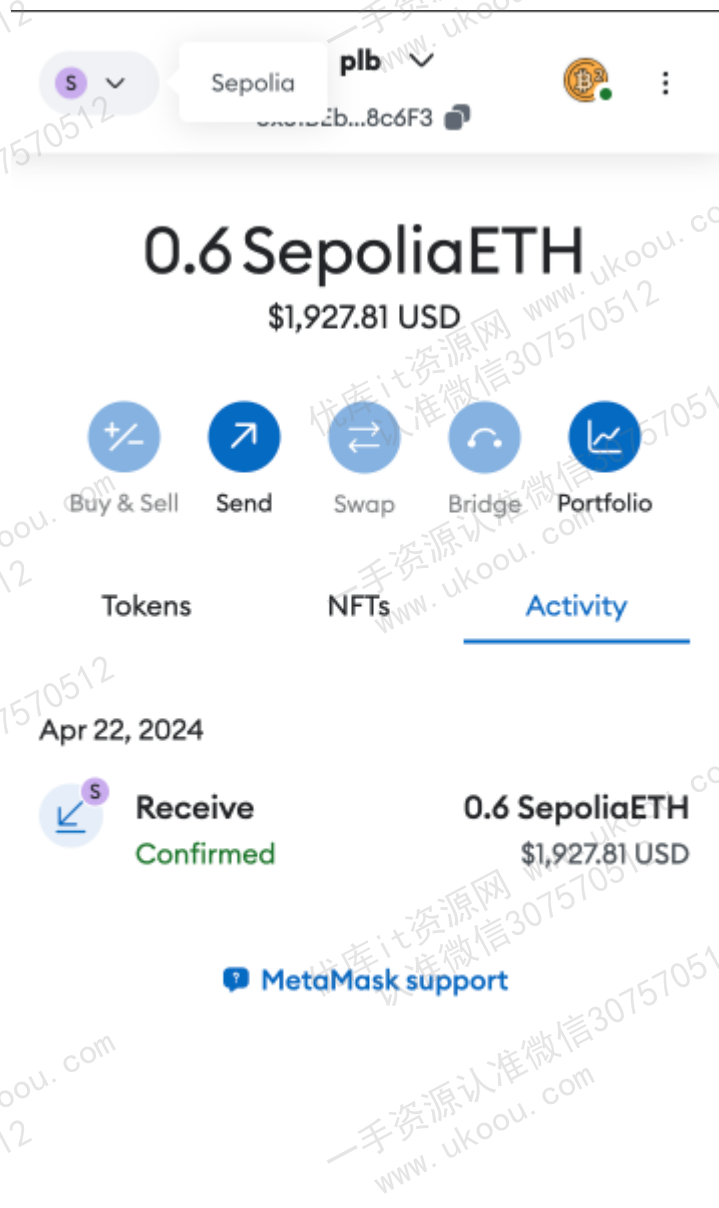
1. 打开 MetaMask。
2. 连接到 Sepolia。
3. 复制你的帐户的地址。



1. 打开一个浏览器窗口或选项卡，转到 [Sepolia faucet](#)。
2. 若要申请以太币，请输入你的测试网帐户地址，然后选择“Send me test Ether”（向我发送测试以太币）。



3. 在 MetaMask 中，验证你的帐户现在是否有以太币。



使用环境变量保存隐私信息

为了在hardhat中使用环境变量我们安装 dotenv

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

在你的项目根目录下，创建一个 `.env` 文件。这个文件将被用来存储敏感信息和配置变量，例如我们的账户私钥信息PRIVATE_KEY以及ALCHEMY_API_KEY:

```
1 ALCHEMY_API_KEY=***
2 PRIVATE_KEY=**
3
```

1. 如何从metamask获取我们的账户私钥: PRIVATE_KEY

2. 下一步我们将从Alchemy获取ALCHEMY_API_KEY

安装 Alchemy 并将终结点链接到 Sepolia 测试网络

Alchemy 开发套件提供对 Ethereum 网络的即时可缩放 API 访问。Alchemy 是一个托管的 Ethereum 节点群集，使用户能够在公用网络上运行应用程序。使用 Infura 部署到公用网络并与之进行交互。若要设置帐户，请转到Alchemy官网并执行以下步骤：

1. 注册
2. 创建项目

Create new app

Chain



Ethereum



Network



Ethereum Sepolia



Name ⓘ

e.g. Frontend Production

Description ⓘ

e.g. Our user facing website

Cancel

Create app

创建好之后可以获取API_KEY，并且将API_KEY粘贴到我们的 `.env` 文件中

Connect to Alchemy

API Key

fmUjn2Olxia0vUSFLC-xEM7U9wwYzItL

Copy

HTTPS

https://eth-sepolia.g.alchemy.com/v2/fmUjn2Olxia0vUSFLC-xEM7U9ww...

Copy

WebSocket

wss://eth-sepolia.g.alchemy.com/v2/fmUjn2Olxia0vUSFLC-xEM7U9ww...

Copy

Code examples

连接到 Sepolia

在 Visual Studio Code 中返回到 todolist 项目文件夹。

1. 在 Hardhat 配置文件 hardhat.config.js 中, 加入Sepolia的配置

```
1 require("@nomicfoundation/hardhat-toolbox");
2 require("dotenv").config();
3 /** @type import('hardhat/config').HardhatUserConfig */
4 module.exports = {
5   solidity: "0.8.24",
6   networks: {
7     hardhat: {},
8     sepolia: {
9       url:
10         "https://eth-sepolia.g.alchemy.com/v2/" + process.env.ALCHEMY_API_KEY,
11       accounts: [`0x${process.env.PRIVATE_KEY}`],
12     },
13   },
14 };
15
```

```
1 ropsten: {provider: () => new HDWalletProvider(mnemonic,
  'https://ropsten.infura.io/v3/${infuraKey}'), network_id: 3, // Ropsten's
  id
```



```
2   gas: 5500000,           // Ropsten has a lower block limit than mainnet
3   confirmations: 2,       // # of confs to wait between deployments. (default: 0)
4   timeoutBlocks: 200,     // # of blocks before a deployment times out
   (minimum/default: 50)
5   skipDryRun: true        // Skip dry run before migrations? (default: false for
   public nets )
6 },
```

部署到 Sepolia

若要部署到 Sepolia，请从 Visual Studio Code 终端运行以下命令：

```
1 npx hardhat ignition deploy ./ignition/modules/ToDoList.js --network sepolia
```

如果部署成功，则会看到以下输出：

```
→ todolist npx hardhat ignition deploy ./ignition/modules/ToDoList.js --network sepolia
✓ Confirm deploy to network sepolia (11155111)? ... yes
Hardhat Ignition 🚀

Resuming existing deployment from ./ignition/deployments/chain-11155111

Deploying [ ToDoListModule ]

Warning - previously executed futures are not in the module:
- LockModule#Lock

Batch #1
  Executed ToDoListModule#ToDoList
[ ToDoListModule ] successfully deployed 🚀

Deployed Addresses

LockModule#Lock - 0x8Bc5D4fd7586B3a2740725f3571DD8C95e48826f
ToDoListModule#ToDoList - 0x6542a4351000e3e24Bb80FAB60f8829c09469d19
```

在sepolia的区块链浏览器链接中拼接上面标识的地址可以查看我们部署的该合约：

<https://sepolia.etherscan.io/address/0x6542a4351000e3e24Bb80FAB60f8829c09469d19>

Sepolia Testnet

Search by Address / Txn Hash / Block / Token

7 * 🔍

Etherscan

Home Blockchain Tokens NFTs Misc

Contract

0x6542a4351000e3e24Bb80FAB60f8829c09469d19

Overview

ETH BALANCE
0 ETH

More Info

CONTRACT CREATOR
0x51BEbC1a...9E638c6F3 at txn 0x5d801a1056...

Multichain Info

N/A

Transactions



Token Transfers (ERC-20)

Contract

Events

Latest 1 from a total of 1 transactions

Download Page Data

Transaction Hash	Method	Block	Age	From	To	Value	Txn Fee
 0x5d801a1056...	0x60806040	5752118	3 mins ago	0x51BEbC1a...9E638c6F3	 Contract Creation	0 ETH	0.00073768

[Download: [CSV Export](#)]

A contract address hosts a smart contract, which is a set of code stored on the blockchain that runs when predetermined conditions are met. Learn more about addresses in our [Knowledge Base](#).

点击标识的transaction hash 可以查看部署的tx的详情