**Capítulo 10: Ethereum na Prática**

Este capítulo introduz o ambiente de desenvolvimento do Ethereum. Vários exemplos serão apresentados neste capítulo para complementar os conceitos teóricos fornecidos no capítulo anterior.

Começaremos considerando o processo prático de pagamento no Ethereum, e algumas inovações recentes e futuras para o blockchain. Em seguida, examinaremos redes de teste, que podem desempenhar um papel vital ao testar contratos inteligentes antes de implantá-los na rede principal. Depois, examinaremos a opção de rede privada no Ethereum, que permite a criação de uma rede privada independente. Essa rede privada pode ser usada como um livro-razão distribuído compartilhado entre entidades participantes e para o desenvolvimento e teste de contratos inteligentes.

Embora existam outros clientes disponíveis para o Ethereum, o Geth é o principal cliente para Ethereum e a ferramenta padrão preferida, e, como tal, este capítulo utiliza o Geth para os exemplos. Também usaremos o Remix IDE para escrever contratos inteligentes e o MetaMask para implantar contratos em uma rede privada que criaremos.

Ao longo do caminho, consideraremos os seguintes tópicos:

• Pagamentos em Ethereum  
• Inovações no Ethereum  
• Programação com Geth  
• Configuração de um ambiente de desenvolvimento  
• Introdução ao Remix IDE  
• Interação com o blockchain Ethereum com MetaMask

**Pagamentos em Ethereum**

Nesta seção, veremos como o Ethereum funciona do ponto de vista do usuário. Para esse propósito, apresentaremos o caso de uso mais comum de transferência de fundos — em nosso caso de uso, de um usuário (Bashir) para outro (Irshad). Usaremos dois clientes Ethereum, um para enviar fundos e outro para receber. Existem várias etapas envolvidas neste processo, conforme segue:

1. Primeiro, ou um usuário solicita dinheiro enviando o pedido ao remetente, ou o remetente decide enviar dinheiro ao receptor. Podemos usar qualquer software de carteira Ethereum. A solicitação pode ser enviada enviando o endereço Ethereum do receptor ao remetente.

Por exemplo, há dois usuários, Bashir e Irshad. Se Irshad solicitar dinheiro a Bashir, então ela pode enviar um pedido a Bashir usando um código QR. Assim que Bashir receber essa solicitação, ele irá escanear o código QR ou digitar manualmente o endereço Ethereum de Irshad e enviar o Ether para o endereço de Irshad. Esse pedido é codificado como um código QR, mostrado na seguinte captura de tela, que pode ser compartilhado por e-mail, texto ou estar disponível para escaneamento visual ou qualquer outro método de comunicação:

<IMAGEM>

1. Assim que Bashir receber essa solicitação, ele irá escanear esse código QR ou copiar ou digitar o endereço Ethereum no software da carteira e iniciar uma transação. Imagine que Bashir quer enviar fundos para Irshad. O remetente insere tanto o valor quanto o endereço de destino em um software de carteira para enviar o Ether ao receptor. Pouco antes de enviar o Ether, a etapa final é confirmar a transação.
2. Assim que o pedido (transação) de envio de dinheiro é construído no software da carteira, ele é então transmitido para a rede Ethereum. A transação é assinada digitalmente pelo remetente como prova de que ele é o proprietário do Ether.
3. Essa transação é então capturada por nós chamados mineradores na rede Ethereum para verificação e inclusão no bloco. Neste estágio, a transação ainda está não confirmada.

O fluxo de transação funciona fundamentalmente da mesma forma em todos os softwares de carteira, portanto, qualquer software de carteira pode ser usado. Existem muitos tipos diferentes de softwares de carteira disponíveis online para sistemas operacionais iOS, Android e desktop.

1. Uma vez verificada e incluída no bloco, o processo de Prova de Trabalho (PoW) começa.
2. Uma vez que um minerador encontra a resposta para o problema de PoW ao fazer hash repetidamente do bloco com um novo nonce, esse bloco é imediatamente transmitido para os outros nós, que então verificam o bloco e o PoW.
3. Se todas as verificações forem aprovadas, então esse bloco é adicionado ao blockchain, e os mineradores são pagos com recompensas conforme apropriado.
4. Finalmente, Irshad recebe o Ether, que será exibido no software da carteira em uso.

Com esse exemplo, concluímos nossa discussão sobre o uso mais comum da rede Ethereum: transferir Ether de um usuário para outro.

**Inovações no Ethereum**

Nesta seção, abordaremos algumas das inovações no Ethereum. O Ethereum está em constante desenvolvimento. Assim como os BIPs (Bitcoin Improvement Proposals), temos as propostas de melhoria do Ethereum (EIPs) no Ethereum para sugerir e implementar melhorias na plataforma. Você pode acompanhar os EIPs aqui: <https://eips.ethereum.org>. Agora abordaremos algumas melhorias importantes.

**Bomba de dificuldade (difficulty time bomb)**

Além do ajuste de dificuldade baseado na diferença de timestamp, há também outro elemento que aumenta a dificuldade de mineração exponencialmente a cada 100.000 blocos. Esta é a chamada bomba de dificuldade, ou era do gelo, introduzida na rede Ethereum, que tornará muito difícil minerar no blockchain Ethereum em algum momento no futuro.

Uma vez ativada, ao longo do tempo, a bomba de dificuldade torna a mineração no Ethereum 1.x tão proibitivamente lenta que se torna inviável, resultando na chamada “era do gelo”. Em outras palavras, esse mecanismo aumenta exponencialmente a dificuldade de mineração por PoW até um nível em que a geração de blocos se torna impossível, forçando assim os mineradores a migrar para o sistema de Prova de Participação (PoS) do Ethereum chamado Casper.

A atualização London (<https://ethereum.org/en/history/#london>) introduziu o EIP-1559. A atualização mais recente é a Paris (A Fusão: <https://ethereum.org/en/history/#paris>), que discutiremos em mais detalhes no Capítulo 13, *The Merge and Beyond*.

Na blockchain, esta transação é identificada pelo seguinte hash de transação:  
0xc63dce6747e1640abd63ee63027c3352aed8cdb92b6a02ae25225666e171009e.  
Isso pode ser visualizado no explorador de blocos em:  
<https://etherscan.io/tx/0xc63dce6747e1640abd63ee63027c3352aed8cdb92b6a02ae25225666e171009e>

Mais informações sobre o Casper estão disponíveis aqui:  
<https://github.com/ethereum/research/blob/master/papers/casper-basics/casper_basics.pdf>

**EIP-1559**

Uma maneira de mitigar, até certo ponto, as altas taxas de transação é o EIP-1559. No entanto, observe que o EIP-1559 não tem como objetivo reduzir a taxa de gas; ele apenas ajuda a prever melhor a taxa, o que pode significar que os usuários não acabarão pagando em excesso, resultando numa redução geral das taxas pagas ao longo do tempo.

O EIP-1559 foi implementado como parte da atualização London. O EIP-1559 impõe uma taxa base (por unidade de gas) que deve ser paga e tem como objetivo equilibrar oferta e demanda. Esta taxa é queimada permanentemente, o que pode diminuir a taxa de inflação do Ether. A taxa base se recalibra a cada bloco dependendo do “alvo”. Os blocos podem aumentar de tamanho alterando o MaxGasLimit, e se o bloco for maior que o alvo, então a taxa é mais alta, e mais baixa caso contrário. Os usuários também podem pagar uma gorjeta aos mineradores para priorizar suas transações, além da taxa base.

Com o aumento do tamanho do bloco, pode ser possível lançar um ataque de negação de serviço (DoS), onde a rede poderia ser sobrecarregada com blocos de tamanho grande sucessivos. O EIP-1559 torna o cálculo da taxa de gas mais previsível e melhora a experiência do usuário, o que pode resultar em os usuários não pagarem em excesso.

Observe que o EIP-1559 não é um mecanismo para reduzir a taxa de transação porque uma taxa alta é um problema devido a um problema de escalabilidade inerente, ou seja, um tamanho de bloco fixo, e, consequentemente, um mecanismo de oferta e demanda. Quando a rede está ocupada, um tamanho de bloco fixo resulta em os usuários aguardarem por períodos indefinidos para que sua transação seja incluída no bloco.

Soluções simples podem parecer possíveis para resolver esse problema, mas elas não são tão simples quanto parecem. Se pudermos, de alguma forma, impor uma taxa fixa e uma ordenação de transações universal impulsionada por consenso juntamente com tamanhos de bloco flexíveis, então esse problema pode ser resolvido. No entanto, se não houver mais incentivo para os mineradores devido às restrições de taxa fixa, então talvez os mineradores abandonem o blockchain.

**<IMAGEM>**

*Figura 10.2: Mecanismo do EIP-1559, antes e depois da atualização London*

O EIP-1559 funciona com base em três variáveis:

* **baseFeePerGas**: Este é um novo campo introduzido na estrutura de bloco. Representa o preço mínimo que deve ser pago para que uma transação seja incluída em um bloco. Com o EIP-1559, os blocos podem aumentar ou diminuir de tamanho conforme a demanda da rede até o limite de 30 milhões de gas.

A taxa base é calculada com base nos blocos anteriores, o que torna as taxas de gas mais previsíveis para os usuários. O cálculo baseia-se em uma fórmula que compara o tamanho do bloco anterior com o tamanho-alvo. A taxa base aumenta ou diminui com base na diferença entre o tamanho atual do bloco e o alvo, que é 15 milhões de gas por bloco. Se um bloco estiver 100% cheio, a taxa base aumentará 12,5%; se o bloco estiver apenas 50% cheio, a taxa base permanecerá a mesma; e se o bloco estiver 0% cheio, a taxa base diminui 12,5%. Isso significa que a rede atinge o equilíbrio em 50% de capacidade ao calibrar as taxas de acordo com o nível de utilização da rede.

Essa taxa de crescimento exponencial torna economicamente inviável que o tamanho do bloco permaneça alto indefinidamente. As carteiras podem calcular a taxa base de forma determinística com base nas informações disponíveis nos blocos anteriores. Além disso, a taxa base é queimada para evitar um ataque em que mineradores poderiam conspirar para aumentar permanentemente a taxa base.

* **maxPriorityFeePerGas**: Esta é uma variável controlada pelos usuários. Os usuários a definem e a adicionam às transações. Ela representa a parte da taxa de transação que vai para o minerador. Esta é a gorjeta do minerador que permite que as transações sejam executadas mais rapidamente.
* **maxFeePerGas**: Este é outro parâmetro definido pelo usuário. Ele representa o valor máximo que um usuário está disposto a pagar por sua transação. Ele inclui baseFeePerGas e maxPriorityFeePerGas. A diferença entre maxFeePerGas e baseFeePerGas + maxPriorityFeePerGas é reembolsada ao usuário.

Em resumo, o EIP-1559 torna as taxas de transação mais previsíveis, reduz os atrasos na inclusão e confirmação das transações, melhora a experiência do usuário por meio de um mecanismo automatizado de taxa/lance e ajusta as taxas com base na atividade da rede, ou seja, oferta e demanda.

Uma maneira de pensar sobre isso é que o EIP-1559 substitui a volatilidade dos preços das taxas pela volatilidade no tamanho dos blocos, e, com o tempo, o tamanho dos blocos se estabiliza porque os usuários que não conseguem pagar uma gorjeta ao minerador para priorizar seu bloco desistirão, e eventualmente o equilíbrio de blocos com 50% do tamanho será mantido novamente.

Em resumo, antes da atualização London, o cálculo do gas era:

Unidades de gas (limite) \* Preço por unidade de gas

Após a atualização London, o cálculo do gas passou a ser:

Unidades de gas (limite) \* (Taxa base + Gorjeta)

**A Fusão e as Próximas Atualizações**

As próximas atualizações são a chamada “fusão” (*merge*) e as cadeias de fragmentos (*shard chains*). A fusão uniu a atual rede principal do Ethereum em 15 de setembro de 2022 com a cadeia Beacon. A cadeia Beacon é uma cadeia PoS (Proof of Stake) e atualmente está ativa, fornecendo a cadeia principal para os fragmentos e para a cadeia Ethereum fundida.

Observe que agora não existe mais o Ethereum 2.0, como originalmente planejado. *Eth2* ou *Ethereum 2.0* era um termo usado para uma nova versão distinta do Ethereum, mas agora foi substituído por *The Merge*. A atualização *The Merge* fundiu tanto o Ethereum1 quanto o que deveria ser o Ethereum 2.0, e uma única plataforma surgiu, chamada simplesmente de *Ethereum*, não *Ethereum 2.0*. Um modelo mental para esclarecer isso é que o Eth1 é agora a chamada camada de execução (*execution layer*), onde ocorre a execução das transações, e o Eth2 é a camada de consenso (*consensus layer*) responsável pela Prova de Participação (PoS). Veremos mais sobre isso em um capítulo separado sobre a Fusão do Ethereum e futuras atualizações no Capítulo 13, *The Merge and Beyond*.

Em seguida, discutiremos o procedimento de instalação e uso de alguns dos clientes Ethereum.

**Programando com Geth**

Primeiramente, descreveremos o Geth e exploraremos várias operações que podem ser realizadas com este cliente.

**Instalando e configurando o cliente Geth**

O procedimento de instalação detalhado em <https://geth.ethereum.org/docs/getting-started/installing-geth> descreve a instalação de clientes Ethereum no macOS e Linux. Instruções para outros sistemas operacionais também estão disponíveis.

Uma vez concluída a instalação, o Geth pode ser iniciado simplesmente emitindo o comando geth no terminal. Ele vem pré-configurado com todos os parâmetros necessários para se conectar à rede Ethereum ao vivo (mainnet):

$ geth

Quando o cliente Ethereum é iniciado, ele começa a se sincronizar com o restante da rede. Existem três tipos de mecanismos de sincronização disponíveis, a saber: *snap*, *full* e *light*:

* **Snap**: Este é o modo padrão que mantém os estados dos 128 blocos mais recentes na memória, permitindo acesso imediato às transações nesse intervalo. O nó neste modo também armazena pontos de verificação entre o bloco de sincronização inicial (um bloco recente) e os 128 blocos mais recentes, o que permite reconstruir estados. Este modo começa baixando os cabeçalhos de um conjunto de blocos. Uma vez que os cabeçalhos são verificados, o restante dos dados, como recibos e corpos dos blocos, é baixado. Ele também baixa os dados brutos do estado e constrói as árvores de estado em paralelo.
* **Full**: Neste modo de sincronização, o cliente Geth gera o estado mais recente executando e verificando todos os blocos desde o bloco gênese. Neste modo, apenas os estados dos 128 blocos mais recentes são armazenados. Os blocos mais antigos são descartados regularmente e representados com pontos de verificação que podem ser usados para regenerar o estado, se necessário. Existem também nós “full archive” que mantêm todos os dados desde o bloco gênese, e os dados antigos nunca são excluídos. Atualmente, o tamanho do blockchain Ethereum é muito grande para ser baixado e mantido sem dificuldades em hardware de entrada. Portanto, geralmente recomenda-se o uso de SSDs para um nó *full archive*, de modo que a latência de disco não cause atrasos no processamento.
* **Light**: Este é o modo mais rápido e baixa e armazena apenas a árvore de estado atual. Neste modo, o cliente não baixa nenhum bloco histórico e apenas processa blocos mais recentes.

O modo de sincronização é configurável no cliente Geth via o parâmetro:

--syncmode valor

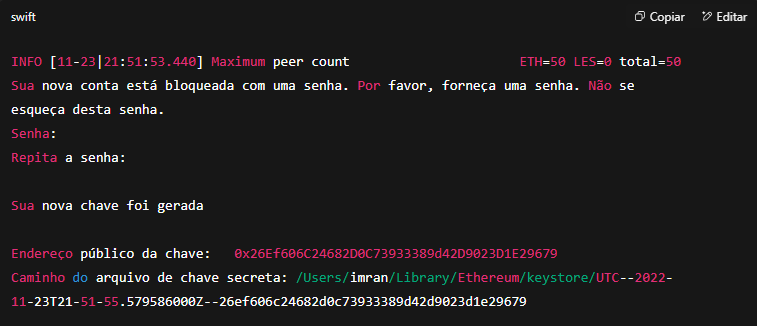
Aqui, o valor pode ser *snap*, *full* ou *light*, por exemplo, --syncmode light inicia o cliente Geth no modo de sincronização leve.

**Criando uma nova conta no Geth**

Novas contas podem ser criadas via linha de comando usando o Geth ou qualquer outra interface de linha de comando de cliente. Execute o seguinte comando para adicionar uma nova conta:

$ geth account new

Este comando produzirá uma saída semelhante à seguinte:

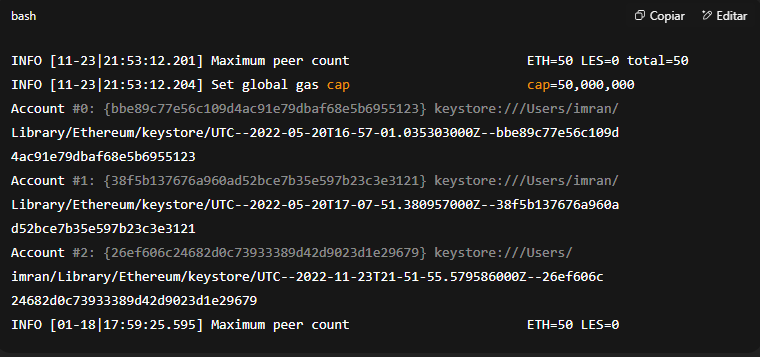


* Você pode compartilhar seu endereço público com qualquer pessoa. Outros precisam dele para interagir com você.
* Você NUNCA deve compartilhar a chave secreta com ninguém! A chave controla o acesso aos seus fundos!
* Você deve FAZER BACKUP do seu arquivo de chave! Sem a chave, é impossível acessar os fundos da conta!
* Você deve LEMBRAR sua senha! Sem a senha, é impossível descriptografar a chave!

A lista de contas pode ser exibida usando o cliente Geth emitindo o seguinte comando:

$ geth account list

Este comando produzirá uma saída semelhante à seguinte:



A seguir, veremos diferentes métodos para interagir com o blockchain.

**Consultando o blockchain usando o Geth**

Há diferentes métodos disponíveis para consulta ao blockchain. Primeiro, para conectar-se à instância em execução do cliente, pode-se usar uma API local IPC ou RPC.

Há três métodos para interagir com o blockchain usando o Geth:

* Console do Geth
* Geth attach
* Geth JSON-RPC

O console do Geth e o Geth attach são usados para interagir com o blockchain usando um ambiente JavaScript REPL. O JSON-RPC é um mecanismo de chamada de procedimento remoto que utiliza o formato de dados JSON para codificar suas chamadas. Em termos mais simples, é uma RPC codificada em JSON.

**Console do Geth**

O Geth pode ser iniciado no modo console executando o seguinte comando:

$ geth console

Isso iniciará o ambiente JavaScript interativo no qual comandos JavaScript podem ser executados para interagir com o blockchain Ethereum, por exemplo, para obter saldo, número de blocos e muitos outros comandos.

**Geth attach**

Quando um cliente Geth já estiver em execução, o console JavaScript interativo pode ser invocado conectando-se a essa instância. Isso é possível executando o comando geth attach.

O console JavaScript do Geth pode ser usado para executar várias funções. Por exemplo, uma conta pode ser criada conectando o Geth.

O Geth pode ser conectado ao daemon em execução, como mostrado na captura de tela a seguir:

$ geth attach

Este comando produzirá uma saída semelhante à seguinte:

Bem-vindo ao console JavaScript do Geth!

instância: Geth/v1.10.14-stable/darwin-arm64/go1.17.5

coinbase: 0xbbe89c77e56c109d4ac91e79dbaf68e5b6955123

Observe que você verá endereços e caminhos de diretórios diferentes ao executar isso em seu computador.

em bloco: 0 (Thu Jan 01 1970 01:00:00 GMT+0100 (BST))

datadir: /Users/imran/Library/Ethereum

módulos: admin:1.0 debug:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

Para sair, pressione ctrl-d ou digite exit

Assim que o Geth for conectado com sucesso à instância em execução do cliente Ethereum, ele exibirá o prompt de comando >, que fornece uma interface interativa de linha de comando para interagir com o cliente Ethereum usando notações JavaScript.

Por exemplo, uma nova conta pode ser adicionada usando o seguinte comando no console do Geth:

> personal.newAccount()

Senha:

Repita a senha:

"0x76bcb051e3cedfcc9c7ff0b25a57383dacbf833b"

A lista de contas também pode ser exibida de forma semelhante:

> eth.accounts

["0x07668e548be1e17f3dcfa2c4263a0f5f88aca402",

"0xba94fb1f306e4d53587fcdcd7eab8109a2e183c4",

"0x1df5ae40636d6a1a4f8db8a0c65addce5a992a14",

"0x76bcb051e3cedfcc9c7ff0b25a57383dacbf833b"]

**API JSON-RPC do Geth**

JSON significa *JavaScript Object Notation*. É um formato de texto leve e fácil de entender usado para transmissão e armazenamento de dados. Uma chamada de procedimento remoto (*remote procedure call* – RPC) é um conceito de sistemas distribuídos. É um mecanismo usado para invocar um procedimento em outro computador. Parece como se uma chamada local estivesse sendo feita porque é necessário escrever código para lidar com interações remotas.

Observe que os leitores verão um endereço diferente.

Mais detalhes sobre RPC e JSON podem ser encontrados aqui: <https://www.jsonrpc.org>

Para este capítulo, é suficiente saber que a API JSON-RPC é amplamente usada no Ethereum para permitir que usuários e *dApps* interajam com o blockchain.

Há vários métodos disponíveis para interagir com o blockchain. Um é usar o console do Geth, que faz uso da API Web3 para consultar o blockchain. A API Web3 faz uso da API JSON-RPC. Outro método é fazer chamadas JSON-RPC diretamente, sem usar a API Web3. Nesse método, chamadas RPC diretas podem ser feitas ao cliente Geth via HTTP. Por padrão, o RPC do Geth escuta na porta TCP 8545.

Agora, veremos alguns exemplos envolvendo o uso da API JSON-RPC. Usaremos uma ferramenta comum chamada curl (<https://curl.haxx.se>) para esse propósito.

Para que esses exemplos funcionem, primeiro o cliente Geth precisa ser iniciado com os parâmetros apropriados. Se houver uma sessão existente do Geth em execução com outros parâmetros, interrompa essa instância e execute o comando geth, como mostrado abaixo, que iniciará o Geth com o RPC disponível. O usuário também pode controlar quais APIs são expostas:

$ geth --http --http.api "eth,net,web3,personal"

Neste comando, o Geth é iniciado com as flags --rpc e --rpcapi, junto com uma lista de APIs que são expostas:

* A flag --rpc ativa o servidor HTTP-RPC.
* A flag --rpcapi é usada para definir quais APIs estarão disponíveis pela interface HTTP-RPC.  
  Isso inclui várias APIs como eth, net, web3 e personal.

Para cada um dos exemplos a seguir, execute o comando curl no terminal, conforme mostrado em cada um dos exemplos abaixo.

A lista de contas pode ser obtida emitindo o seguinte comando:

$ curl -X POST --insecure -H "Content-Type: application/json" --data \

'{"jsonrpc":"2.0","method":"eth\_accounts","params":[], "id":64}' <http://localhost:8545>

Isso exibirá a seguinte saída JSON, que lista todas as contas Ethereum possuídas pelo cliente:

{

"jsonrpc":"2.0",

"id":64,

"result":[

"0x07668e548be1e17f3dcfa2c4263a0f5f88aca402",

"0xba94fb1f306e4d53587fcdcd7eab8109a2e183c4",

"0x1df5ae40636d6a1a4f8db8a0c65addce5a992a14",

"0x76bcb051e3cedfcc9c7ff0b25a57383dacbf833b"

]

}

Podemos verificar se a rede está ativa usando o comando mostrado aqui:

$ curl -X POST --insecure -H "Content-Type: application/json" --data \

'{"jsonrpc":"2.0","method":"net\_listening","params":[], "id":64}' <http://localhost:8545>

Isso exibirá a seguinte saída com o resultado true, indicando que a rede está ativa:

{"jsonrpc":"2.0","id":64,"result":true}

Podemos encontrar a versão do cliente Geth usando este comando:

$ curl -X POST --insecure -H "Content-Type: application/json" --data \

'{"jsonrpc":"2.0","method":"web3\_clientVersion","params":[], "id":64}' <http://localhost:8545>

Isso exibirá a versão do cliente Geth:

{"jsonrpc":"2.0","id":64,"result":"Geth/v1.9.9-stable-01744997/linux-amd64/go1.13.4"}

Para verificar o status mais recente de sincronização, podemos usar o seguinte comando:

$ curl -X POST --insecure -H "Content-Type: application/json" --data \

'{"jsonrpc":"2.0","method":"eth\_syncing","params":[], "id":64}' <http://localhost:8545>

Isso exibirá os dados relativos ao status de sincronização ou retornará false:

{

"jsonrpc":"2.0",

"id":64,

"result":{

"currentBlock":"0x1d202",

"highestBlock":"0x8a61c8",

"knownStates":"0x23b9b",

"pulledStates":"0x2261a",

"startingBlock":"0xa5c5"

}

}

O endereço coinbase pode ser consultado usando:

$ curl -X POST --insecure -H "Content-Type: application/json" --data \

'{"jsonrpc":"2.0","method":"eth\_coinbase","params":[], "id":64}' <http://localhost:8545>

Isso exibirá a saída conforme mostrado abaixo, indicando o endereço coinbase do cliente:

{"jsonrpc":"2.0","id":64,"result":"0x07668e548be1e17f3dcfa2c4263a0f5f88aca402"}

Esses são apenas alguns exemplos das APIs extremamente ricas que estão disponíveis no cliente Geth do Ethereum.

Nesta seção, cobrimos as APIs JSON-RPC do Geth e vimos alguns exemplos de como consultar o blockchain via interface RPC. Com isso, completamos a introdução ao Geth.

**Configurando um ambiente de desenvolvimento**

Uma abordagem comum e sensata para desenvolver e testar contratos inteligentes do Ethereum é dentro de uma rede privada local ou um ambiente simulado como o Ganache. Depois que todos os testes relevantes forem bem-sucedidos em uma rede de teste pública, os contratos podem então ser implantados na rede principal pública. Existem, no entanto, variações nesse processo.

Mais informações e documentos oficiais sobre as APIs RPC do Geth estão disponíveis no seguinte link:  
<https://eth.wiki/json-rpc/API>

Muitos desenvolvedores optam por desenvolver e testar contratos apenas em um ambiente simulado local e, em seguida, implantá-los diretamente na rede principal pública ou em suas redes privadas/empresariais de produção. Desenvolver primeiro em um ambiente simulado e depois implantar diretamente em uma rede pública pode levar a um tempo de produção mais rápido, já que configurar redes privadas pode demorar mais do que configurar um ambiente local de desenvolvimento com um simulador de blockchain.

Existem novas ferramentas e frameworks disponíveis, como o **Truffle** e o **Ganache**, que tornam o desenvolvimento e teste para o Ethereum mais fáceis. Exploraremos essas ferramentas com mais profundidade no Capítulo 11, *Tools, Languages, and Frameworks for Ethereum Developers*, mas primeiro, usaremos uma abordagem manual na qual desenvolveremos um contrato inteligente e o implantaremos manualmente via linha de comando em uma rede privada. Frameworks e ferramentas tornam o desenvolvimento mais fácil, mas ocultam a maioria dos detalhes internos que são úteis para iniciantes construírem uma base sólida de conhecimento.

Vamos começar conectando-nos a uma rede de teste.

**Conectando-se a redes de teste**

O cliente Go do Ethereum (<https://geth.ethereum.org>), Geth, pode ser conectado a uma rede de teste usando uma das seguintes opções de linha de comando disponíveis com o Geth. O Geth pode ser executado com uma flag para conectar-se à rede desejada:

* --goerli: Rede Goerli — rede de teste pré-configurada com prova de autoridade
* --rinkeby: Rede Rinkeby — rede de teste pré-configurada com prova de autoridade
* --ropsten: Rede Ropsten — rede de teste pré-configurada com prova de trabalho
* --sepolia: Rede Sepolia — rede de teste pré-configurada com prova de trabalho

Este comando conecta-se à rede Sepolia:

$ geth –sepolia

Um explorador de blocos para a rede de teste Sepolia está localizado em:  
<https://sepolia.etherscan.io>  
e pode ser usado para rastrear transações e blocos na rede de teste Ethereum.

Agora vamos experimentar a construção de uma rede privada e, em seguida, veremos como um contrato pode ser implantado nesta rede usando ferramentas de linha de comando.

**Criando uma rede privada**

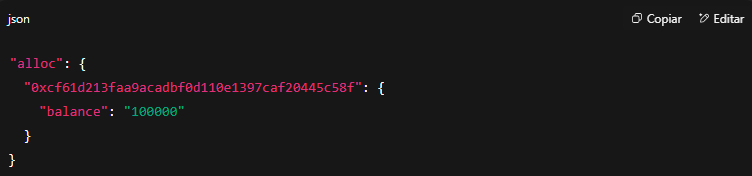
Nesta seção, iniciaremos uma rede privada e a prepararemos para uso. Utilizaremos os seguintes componentes, conforme especificado no capítulo anterior:

* **ID da Rede**: O ID da rede 786 foi escolhido para a rede privada de exemplo.
* **Arquivo Gênese**: Um arquivo gênese personalizado que será usado está mostrado aqui:



Este arquivo é salvo como um arquivo de texto com a extensão .json, por exemplo, privategenesis.json. Opcionalmente, o Ether pode ser pré-alocado especificando os endereços dos beneficiários e a quantidade de Wei, mas isso geralmente não é necessário, pois estando em uma rede privada, o Ether pode ser minerado muito rapidamente.

Para pré-alocar uma conta com Ether, uma seção pode ser adicionada ao arquivo gênese, como mostrado aqui:



* **Diretório de dados**: No exemplo a seguir, é ~/etherprivate/.

Além dos três componentes mencionados, é desejável desativar a descoberta de nós para que outros nós na internet não descubram sua rede privada e para que ela seja segura. Isso pode ser feito executando o Geth com a flag --nodiscover, que desativa o mecanismo de descoberta de pares. Se outras redes tiverem o mesmo arquivo gênese e ID de rede, elas podem se conectar à sua rede privada, o que pode resultar em problemas de segurança. A chance de haver o mesmo ID de rede e bloco gênese é muito baixa, mas mesmo assim, desativar a descoberta de nós é uma boa prática e é recomendado.

A conexão de outra rede ou nó à sua rede privada pode resultar em violações de segurança, vazamento de informações e outros incidentes indesejáveis. No entanto, observe que redes privadas geralmente são executadas dentro de ambientes empresariais e são protegidas por práticas de segurança padrão da empresa, como firewalls.

Desabilitar a descoberta de pares também nos permite definir uma lista de pares estáticos que temos em nossa rede. Isso nos dá a habilidade adicional de controlar quem pode entrar em nossa rede privada.

Para configurar essa lista, os IDs dos nós são adicionados a um arquivo de configuração chamado static-nodes.json. Este arquivo geralmente é colocado no diretório de dados do Geth (cliente Ethereum) executável. Esse diretório também é onde os arquivos chaindata (banco de dados) e keystore são salvos. Por padrão, o diretório de dados está localizado em <diretório pessoal do usuário>/Library/Ethereum, mas pode ser configurado usando a flag --datadir.

O nome do arquivo deve ser static-nodes.json, sob o diretório de dados. Isso é valioso em uma rede privada porque dessa forma, a rede é limitada apenas a nós conhecidos. Um exemplo de arquivo static-nodes.json é mostrado a seguir:

[

"enode://44352ede5b9e792e437c1c0431c1578ce3676a87e1f588434aff1299d30325c233c8d426fc57a25380481c8a36fb3be2787375e932fb4885885f6452f6efa77f@xxx.xxx.xxx.xxx:TCP\_PORT"

]

Aqui, xxx é o endereço IP e TCP\_PORT pode ser qualquer porta TCP válida e disponível no sistema. A longa string hexadecimal é o ID do nó.

Agora que entendemos os vários aspectos e componentes necessários para configurar uma rede privada, incluindo o arquivo gênese e outros arquivos de configuração relevantes, vamos prosseguir com a configuração da nossa própria rede privada usando Ethereum.

**Inicializando a rede privada**

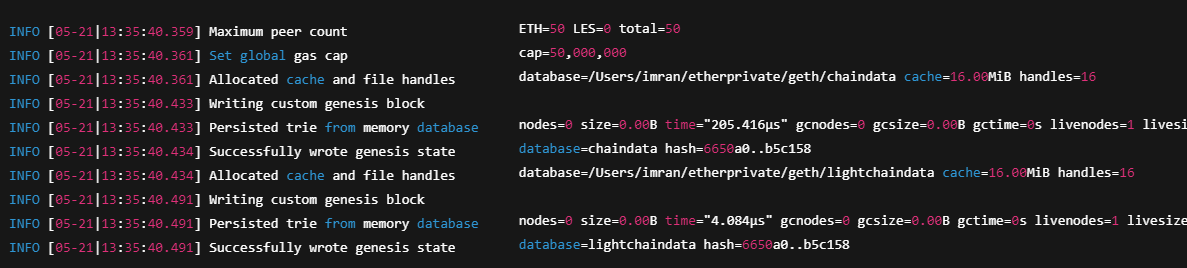
A primeira etapa é criar um diretório chamado etherprivate no diretório home do usuário:

$ mkdir ~/etherprivate

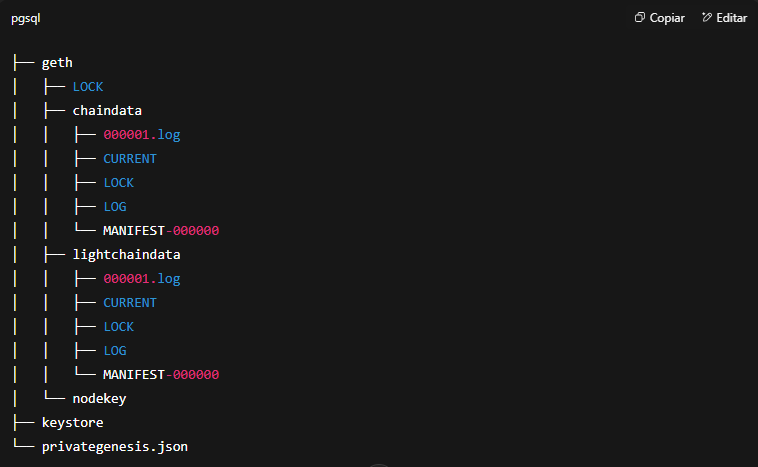
Esse comando criará o diretório. Depois que o diretório for criado, coloque o arquivo privategenesis.json mostrado anteriormente dentro dele. Neste ponto, armazenado sob o diretório pessoal do usuário, temos um diretório chamado ~/etherprivate, que contém o arquivo gênese chamado privategenesis.json. Estamos prontos para iniciar nossa rede. O comando inicial para iniciar a rede privada é o seguinte:

$ geth init ~/etherprivate/privategenesis.json --datadir ~/etherprivate

Isso produzirá uma saída semelhante à seguinte:



Se você vir a mensagem Successfully wrote genesis state, então tudo está certo e você deve ver uma estrutura de diretório como a mostrada abaixo:



Essa estrutura de diretórios contém os dados da cadeia (*chaindata*), logs e um *keystore*. Para que o Geth inicie, o seguinte comando pode ser emitido:

**Iniciando a rede privada**

Para que o Geth seja iniciado, o seguinte comando pode ser emitido:

$ geth --datadir ~/etherprivate/ --networkid 786 --http --http.api 'web3,eth,net,debug,personal' --http.corsdomain '\*'

Isso produzirá várias mensagens de saída conforme o Geth é iniciado. Uma parte importante do log a ser observada são as seguintes linhas:

INFO [05-21|17:12:54.789] IPC endpoint opened url=/Users/imran/etherprivate/communication

INFO [05-21|17:12:54.790] HTTP server started endpoint=127.0.0.1:8545 prefix= cors=\* vhosts=localhost

Essas linhas mostram informações sobre o endpoint de Comunicação entre Processos (*Inter-Process Communication – IPC*), o endpoint HTTP e as informações da conta *coinbase* (base de Ether). Essas informações são úteis para os exemplos fornecidos posteriormente nesta seção.

Agora o Geth pode ser conectado via IPC à instância do cliente Geth em execução na rede privada usando o seguinte comando:

$ geth attach ~/etherprivate/geth.ipc

Esse comando abrirá o console interativo JavaScript para executar a sessão da rede privada:

Bem-vindo ao console JavaScript do Geth!

instância: Geth/v1.10.14-stable/darwin-arm64/go1.17.5

coinbase: 0x6e94bdb15141491bc3b9de3a9cab9d87ae2af82f

em bloco: 188 (Sat May 21 2022 17:47:53 GMT+0100 (BST))

datadir: /Users/imran/etherprivate

módulos: admin:1.0 debug:1.0 eth:1.0 ethash:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 txpool:1.0 web3:1.0

Para sair, pressione ctrl-d ou digite exit

Você pode notar que uma mensagem de aviso aparece quando o Geth é iniciado:

AVISO: Nenhuma conta etherbase definida e nenhuma conta encontrada como padrão.

Essa mensagem aparece porque atualmente não há contas disponíveis na nova rede de teste e nenhuma conta está definida como *etherbase* para receber recompensas de mineração. Esse problema pode ser resolvido criando uma nova conta e definindo essa conta como *etherbase*. Isso também será necessário quando a mineração for realizada na rede de teste.

**Mecanismos IPC**

Mecanismos de Comunicação entre Processos (IPC) são usados para permitir a comunicação entre diferentes processos ou threads executando localmente em um sistema operacional. Existem diferentes mecanismos de IPC, incluindo *pipes* (tubulações) e *signals* (sinais). Um *pipe* é um buffer limitado entre processos realizado, na prática, por um descritor de arquivo compartilhado entre os processos. Um *signal* é um método usado para chamar a atenção de um processo. O Geth usa *pipes* para habilitar a comunicação entre instâncias do Geth.

Isso é mostrado nos comandos a seguir. Observe que esses comandos são inseridos no console JavaScript do Geth. O comando a seguir cria uma nova conta. Nesse contexto, a conta será criada na rede privada com ID 786, pois esta é a rede que criamos anteriormente:

> personal.newAccount("Password123")

"0x6e94bdb15141491bc3b9de3a9cab9d87ae2af82f"

Depois que a conta é criada, a próxima etapa é defini-la como a conta *etherbase*/*coinbase*, para que a recompensa de mineração vá para essa conta. Isso pode ser feito usando o seguinte comando:

> miner.setEtherbase(personal.listAccounts[0])

True

Atualmente, a conta *etherbase* não tem saldo, como pode ser visto usando o comando:

> eth.getBalance(eth.coinbase).toNumber();

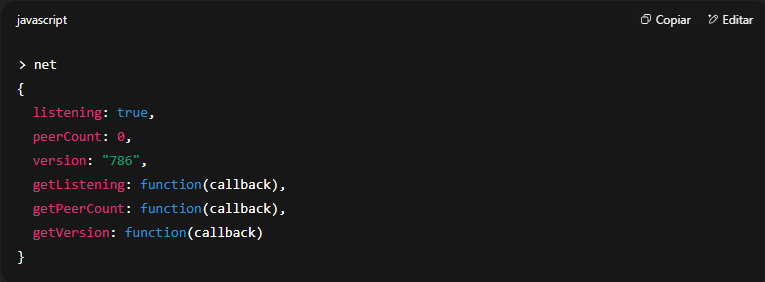
0

Nesta seção, criamos uma rede privada com um arquivo gênese personalizado. Também criamos uma nova conta em nossa rede privada. Agora, antes de explorar mais métodos, vamos entender alguns fundamentos do uso do console JavaScript.

**Experimentando o console JavaScript do Geth**

No console JavaScript, podemos realizar várias operações. Uma dica geral é que, se pressionarmos duas vezes a tecla Tab no teclado, uma lista completa dos objetos disponíveis será exibida.

Além disso, quando um comando é digitado, ele pode ser autocompletado pressionando a tecla Tab duas vezes. Se a tecla Tab for pressionada duas vezes, então a lista de métodos disponíveis também será exibida. Além do comando mencionado, para obter uma lista de métodos disponíveis de um objeto, basta pressionar Enter. Um exemplo é mostrado abaixo, que mostra uma lista de todos os métodos disponíveis para o objeto net:



Também está disponível o objeto eth, que possui vários métodos. Embora existam muitos métodos nesse objeto, o mais comum é o getBalance, que podemos usar para consultar o saldo atual de Ether. Isso é mostrado no exemplo a seguir:

> eth.getBalance(eth.coinbase)

150000000000000000000

Após a mineração, uma quantidade significativa pode ser vista aqui. A mineração é extremamente rápida, pois trata-se de uma rede privada sem competição para resolver o PoW, e também no arquivo gênese, a dificuldade da rede foi configurada como muito baixa.

O saldo acima é mostrado em Wei. Se quisermos ver a saída em Ether, podemos usar o objeto web3, como mostrado aqui:

> web3.fromWei(eth.getBalance(eth.coinbase), "ether")

150

Existem alguns outros comandos que podem ser usados para consultar a rede privada. Alguns exemplos são mostrados a seguir:

**Obter o preço atual do gas:**

> eth.gasPrice

1000000000

Obter o número mais recente de bloco:

> eth.blockNumber

30

**O objeto debug pode ser útil ao depurar problemas.** Um comando de exemplo é mostrado aqui; no entanto, há muitos outros métodos disponíveis. Uma lista desses métodos pode ser visualizada digitando debug.

O método a seguir retornará o RLP do bloco 0:

> debug.getBlockRlp(0)

"0xf901f8f901f3a00000000000000000000000000000000000000000000000000000000000...<dados truncados>"

A saída acima mostra o bloco 0 no formato codificado RLP (Recursive Length Prefix).

Agora estamos prontos para iniciar a mineração em nossa rede privada.

**Mineração e envio de transações**

Agora que iniciamos nossa rede privada, a mineração pode ser iniciada simplesmente emitindo o seguinte comando. Esse comando recebe um parâmetro: o número de *threads*. No exemplo a seguir, dois *threads* serão alocados para o processo de mineração ao especificar 2 como argumento para a função start:

> miner.start(2)

Aqui também podemos fornecer um parâmetro inteiro. Por exemplo, se fornecermos 1, ele usará apenas um núcleo da CPU para mineração, o que ajuda com questões de desempenho, caso o uso de todos os recursos da CPU esteja reduzindo o desempenho do sistema. Um comando de exemplo usando apenas uma CPU é miner.start(1). Em sistemas com apenas uma CPU, emitir esse comando inevitavelmente usará apenas uma CPU. No entanto, em um sistema multicore, fornecer o número de núcleos que podem ser usados para mineração ajuda a lidar com preocupações de desempenho, pois múltiplos *threads* de mineração podem ser executados em paralelo em múltiplos núcleos.

Depois que o comando acima for emitido como preparação para mineração, o processo de geração do DAG começa, produzindo uma saída semelhante à seguinte:

INFO [05-21|17:17:54.075] Commit new mining work number=1 sealhash=d18bb2..2acb58 uncles=0 txs=0 gas=0 fees=0 elapsed=2.161ms

INFO [05-21|17:17:54.798] Generating DAG in progress epoch=0 percentage=0 elapsed=335.435ms

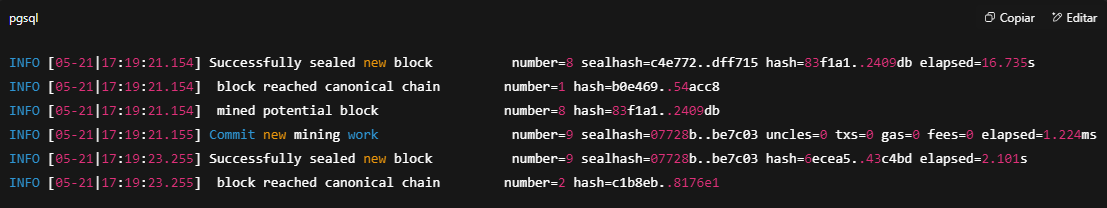
INFO [05-21|17:17:55.135] Generating DAG in progress epoch=0 percentage=1 elapsed=673.142ms

INFO [05-21|17:17:55.524] Generating DAG in progress epoch=0 percentage=2 elapsed=1.062s

INFO [05-21|17:17:55.926] Generating DAG in progress epoch=0 percentage=3 elapsed=1.463s

INFO [05-21|17:17:56.343] Generating DAG in progress epoch=0 percentage=4 elapsed=1.880s

Uma vez que a geração do DAG é concluída, o processo de mineração começa e blocos são produzidos. O Geth exibirá uma saída semelhante à mostrada a seguir. Pode-se ver que os blocos estão sendo minerados com sucesso com a mensagem *mined potential block*:



A mineração pode ser interrompida com o seguinte comando:

> miner.stop()

Null

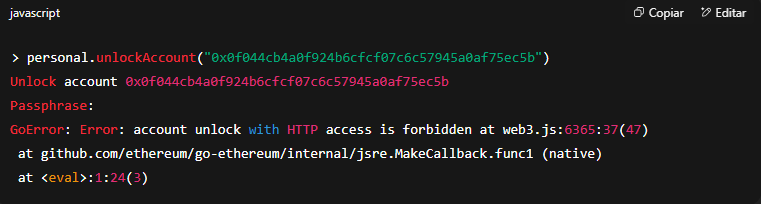
Agora vejamos como criar uma nova conta e enviar uma transação em nossa rede privada.

• **Criar uma nova conta.** Observe que "Password123" é a senha escolhida como exemplo, mas você pode escolher qualquer uma:

> personal.newAccount("Password123")

"0x0f044cb4a0f924b6cfcf07c6c57945a0af75ec5b"

• **Desbloquear a conta antes de enviar transações.** Usamos a opção de desbloqueio inseguro; caso contrário, as contas não podem ser desbloqueadas com acesso HTTP. Se for esse o caso, você verá uma mensagem de erro como a seguinte:



Para contornar o erro, reinicie o Geth usando a flag --allow-insecure-unlock:

$ geth --datadir ~/etherprivate/ --allow-insecure-unlock --networkid 786 --http --http.api 'web3,eth,net,debug,personal' --http.corsdomain '\*'

Agora desbloqueamos ambas as contas que criamos anteriormente, uma no início da criação da rede privada e outra agora há pouco:



Agora que temos duas contas disponíveis na rede privada, e a mineração foi ativada, podemos enviar Ether da conta coinbase para a nova conta.

Primeiro, iniciamos a mineração:

> miner.start(1)

Em seguida, usamos o comando eth.sendTransaction para transferir Ether da conta coinbase para a nova conta:

> eth.sendTransaction({from:eth.coinbase, to:"0x0f044cb4a0f924b6cfcf07c6c57945a0af75ec5b", value: web3.toWei(100, "ether")})

Isso exibirá um hash de transação semelhante a este:

"0x20c3b1c6dc51d7a836019d89c55a451d0cda7cd7cf358fa56e3708b4e7882ca6"

Depois que o bloco for minerado e a transação incluída, o saldo da conta receptora será atualizado:

> eth.getBalance("0x0f044cb4a0f924b6cfcf07c6c57945a0af75ec5b")

100000000000000000000

E usando a função web3.fromWei:

> web3.fromWei(eth.getBalance("0x0f044cb4a0f924b6cfcf07c6c57945a0af75ec5b"), "ether")

100

Assim, vimos como configurar uma rede privada e usá-la para criar contas, minerar e transferir Ether entre contas.

**Introdução ao Remix IDE**

O Remix IDE é um ambiente de desenvolvimento baseado na web usado para escrever, compilar, testar e implantar contratos inteligentes Ethereum escritos em Solidity. Ele oferece uma interface rica com suporte à linguagem Solidity, detecção de erros, compilação, implantação, interação e até mesmo análise de gas.

Pode ser acessado em <https://remix.ethereum.org>.

O Remix também pode ser executado localmente. O código-fonte está disponível no GitHub em <https://github.com/ethereum/remix-project>, e instruções de instalação estão disponíveis lá.

Ao acessar <https://remix.ethereum.org>, o ambiente é carregado diretamente no navegador, como mostrado na imagem a seguir:

<IMAGEM>

As seções principais do Remix são:

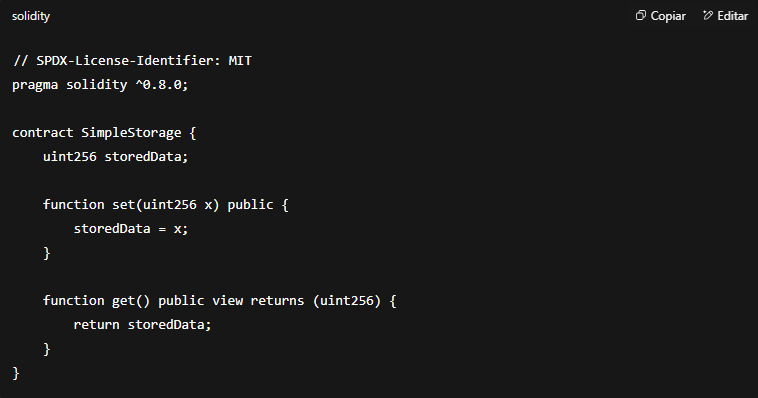
* **Painel do Explorador de Arquivos** (à esquerda): onde os arquivos são organizados.
* **Editor de Código** (centro): para escrever contratos inteligentes Solidity.
* **Painel de Terminal e Console** (abaixo): para visualizar logs, executar comandos e interagir com contratos.
* **Painéis de Plugins** (direita): onde estão os compiladores, implantações e interações.

Vamos agora escrever um contrato simples.

**Escrevendo e implantando um contrato no Remix**

Agora escreveremos e implantaremos um contrato inteligente simples usando o Remix. A interface Remix deve estar acessível por meio de <https://remix.ethereum.org>. Após carregar a página, siga os seguintes passos:

1. No painel do explorador de arquivos, clique em **"contracts"**, e depois clique em **"New File"**. Dê um nome ao arquivo, como SimpleStorage.sol.
2. No editor central, digite o seguinte código:



Este é um contrato simples com duas funções:

* set(uint256 x) define o valor de storedData.
* get() retorna o valor armazenado.

1. Em seguida, clique no botão **"Solidity Compiler"** no painel à direita.
2. Escolha a versão apropriada do compilador (correspondente à versão declarada no contrato).
3. Clique em **"Compile SimpleStorage.sol"**.

Se a compilação for bem-sucedida, você verá uma marca verde ao lado do botão.

**Implantando o contrato**

Depois de compilar o contrato com sucesso, prossiga com os seguintes passos para implantá-lo:

1. Clique no botão **"Deploy & Run Transactions"** no painel à direita.
2. Certifique-se de que **"Environment"** está configurado como **"JavaScript VM"** (uma blockchain simulada no navegador).
3. Clique no botão **"Deploy"**.

O contrato será implantado, e você verá seu endereço e interface no painel inferior.

1. Você verá os botões **"get"** e **"set"** abaixo do contrato implantado:
   * Para testar, digite um número (por exemplo, 42) no campo de entrada de set e clique no botão.
   * Depois clique no botão get para recuperar o valor.

Você verá que o valor armazenado foi atualizado e recuperado corretamente.

**Interagindo com o contrato**

O Remix facilita a interação com os contratos inteligentes:

* Cada chamada à função é exibida no console inferior.
* As funções de leitura como get() são chamadas gratuitamente (sem custo de gas).
* As funções que alteram o estado como set() geram transações e consomem gas — mesmo no ambiente simulado.

Com isso, mostramos como escrever, compilar, implantar e interagir com um contrato inteligente Solidity básico no Remix.

**Resumo**

Neste capítulo, introduzimos o Ethereum com detalhes, incluindo:

* Arquitetura básica do Ethereum e o conceito de contas e endereços.
* Contratos inteligentes e a Máquina Virtual Ethereum (EVM).
* Clientes Ethereum, especialmente o Geth.
* Modos de sincronização, criação de contas e mineração.
* Execução de transações via linha de comando.
* Uso da API JSON-RPC.
* Criação de redes privadas com arquivos gênese personalizados.
* Desenvolvimento, implantação e testes de contratos inteligentes com Remix IDE.

Esses conceitos formam a base para o desenvolvimento Ethereum. No próximo capítulo, exploraremos ferramentas, linguagens e frameworks amplamente utilizados no desenvolvimento Ethereum, incluindo **Truffle**, **Hardhat**, **Brownie**, **Foundry**, entre outros.