Scaffold-ETH Deep Dive

Introduction to Build Guild and Scaffold-ETH

Build Guild Dow focuses on education, prototyping, and grant-giving within the Ethereum space.

- **Speedrun Ethereum:** A condensed curriculum that helps technical individuals quickly grasp essential concepts in Ethereum development. It emphasizes not only the 'how' but also the 'why' of building on Ethereum.
- **Scaffold-ETH:** A tool designed for prototyping and building applications rapidly on Ethereum.

Tools Built with Scaffold-ETH

- **ABI Ninja:** Provides an instant front end for interacting with any smart contract.
- **Hacked Wallet Recovery:** A tool leveraging flashbots to sweep remaining assets from compromised wallets.
- **Address Vision:** A utility that is a public good for the Ethereum ecosystem.

Getting Started with Scaffold-ETH 2

1. **Forking the Repository:** You can fork the Scaffold-ETH repository from GitHub.
2. **Using MPX:** Alternatively, you can use the command `MPX create eth`. This command initiates a "Choose Your Own Adventure" setup, allowing you to configure your project.
    - You'll be prompted to name your project.
    - You can choose between Hardhat or Foundry as your development environment.
3. **Running Scaffold-ETH:** After setting up, you need to run three commands:
    - `yarn start`: Starts the front end.
    - `yarn chain`: Starts the local blockchain.
    - `yarn deploy`: Deploys your template contract to the local network.

Live Smart Contract Interaction

Scaffold-ETH allows you to modify your smart contract and see the changes reflected in the front end in real time. For example:

1. **Adding a Public Address Delegate:** Add a new state variable to your smart contract.
2. **Real-time Updates:** Upon saving and redeploying, the front end automatically adapts to reflect the new functionality in your smart contract.

```
// Example: Adding a delegate address
address public delegate;
```

Template Smart Contract Overview

The template smart contract includes various primitives and data types to quickly get developers started with Solidity.

- If you're new to Solidity, you can refer to Solidity by Example and copy examples into your contract to experiment with them.
- The template includes a greeting contract with a payable function. If the value sent is greater than zero, a premium flag is set.

```
// Example: Payable greeting function
function setGreeting(string memory _greeting) public payable {
    greeting = _greeting;
    if (msg.value > 0) {
        premium = true;
    }
}
```

Interacting with the Greeter Contract

1. **Sending Transactions:** Interact with the contract by sending transactions. For example, changing the greeting.
2. **Paying Ether:** You can send Ether to the contract. Note that you need to convert Ether to Wei, as Solidity does not support floating-point math.
    - `$1 Eth = 10^18$ Wei`
    - This conversion is a UX consideration for developers to understand the underlying values.

Building Your Hackathon Project

Let's say you're building a hackathon project:

1. **Modifying the Smart Contract:** Add new functionalities to your smart contract.
2. **Implementing a Set Delegate Function:** Add a function to update the delegate address.

```
// Example: Set delegate function
function setDelegate(address _delegate) public {
    delegate = _delegate;
}
```

Integrating with the Front End

After deploying the updated contract, the new function will be available in the front end. You can then:

1. **Set the Delegate:** Use the front end to set the delegate address.
2. **ENS Avatar Preview:** The front end provides a nice ENS (Ethereum Name Service) avatar preview and a blocky preview for addresses.

Shipping Your App

Once you're satisfied with your smart contract, you can shift focus to building a user-friendly front end. The goal is to integrate the delegate and set delegate functionalities into the React app using hooks.

Building a DApp: Reading and Writing to the Smart Contract

When building a DApp, the focus is split between the **smart contract** and the **user interface (UX)**. A significant amount of time is spent on the front end (e.g., React) to create a user-friendly experience.

Reading from the Contract

To display data from the smart contract on the front end, you need to use a hook. A hook in this context is a prebuilt function that handles the complex logic.

> **Hook**: A function from wagmi (or a similar library) that handles wallet connections, network interactions, and other complex logic, simplifying the process of reading from the blockchain.

For instance, the `useAccount` hook from wagmi provides the connected address, which can be displayed using components like the `Address` component, which handles ENS resolution.

To read a specific value from the contract, such as the delegate address, you can use the `useScaffoldContractRead` hook.

```
const { data: delegate } = useScaffoldContractRead({
  contractName: "YourContract",
  functionName: "delegate"
});
```

This hook takes the contract name and the function name (e.g., "delegate") as arguments and returns the data, which can then be displayed in the UI.

```
<Address value={delegate} />
```

Writing to the Contract

To enable users to write to the smart contract, you'll typically need an input form to collect the data and a button to trigger the write operation.

1. **Input Form**: Create an input field using a component like `AddressInput` to allow users to enter a new delegate address. This input should track the state using the `useState` hook.

```
const [newDelegate, setNewDelegate] = useState("");

<AddressInput
  value={newDelegate}
  onChange={(address) => setNewDelegate(address)}
/>
```

2. **Button**: Add a button that, when clicked, triggers a function to write the new delegate address to the contract. This is done using the `useScaffoldContractWrite` hook.

```
const { writeAsync: setDelegate } = useScaffoldContractWrite({
  contractName: "YourContract",
  functionName: "setDelegate",
  args: [newDelegate],
});
```

   This hook takes the contract name, function name (e.g., "setDelegate"), and arguments (the new delegate address) as parameters. The `writeAsync` function returned by the hook is then called when the button is clicked.

```
<button onClick={() => setDelegate()}>Set Delegate</button>
```

Deploying the DApp

Once the smart contract and front end are ready, the next step is to deploy the DApp.

1. **Local Deployment**: When running locally, deployment is free.
2. **Testnet/Mainnet Deployment**: To deploy to a testnet (e.g., Base or Optimism) or the mainnet, you will need to pay gas fees.
    - First, generate a new account using `yarn generate`. This creates a local account that is isolated for deployments.
    - Send enough funds to this account to cover the gas fees.
    - Use a command like `yarn deploy --network base` to deploy the smart contract to the specified network.

Deploying to Base and Switching Chains

The lecture demonstrated how to **deploy a smart contract** to the **Base** network and how to switch a front end application between different chains using **Scaffold-ETH**.

Cost of Deployment

Deploying the smart contract to Base cost approximately **two cents**. The professor jokingly stated that a new era of **mass adoption** is upon us.

Switching Chains with Scaffold-ETH

One advantage of using **Scaffold-ETH** is the ability to easily switch the front end application between different chains. This is done via the **scaffold.config.ts** file. By changing the **targetNetwork** to the desired chain and saving, the application automatically adjusts to the new network.

For example, changing the target network to **Base** updates the app to interact with the Base network. When a user interacts with the app (e.g., clicking on an address), they are redirected to the appropriate block explorer (e.g., **Base Scan** for Base, **Etherscan** for Ethereum).

Metamask and Wallet Providers

The lecture touched upon how Metamask and other wallet providers handle network switching. Metamask automatically prompts users to switch to the correct network if they are on the wrong one.

Deploying the Front End

The command `yarn vercel YOLO prod` is used to deploy the front end application to Vercel.

Scaffold-ETH as a Hackathon Tool

Scaffold-ETH is described as a valuable tool for hackathon projects, providing the necessary features out of the box. It is designed to be clean, lightweight, and powerful, making it suitable for building production-ready applications.

Key characteristics of Scaffold-ETH:

- Forkable: Easy to start new projects based on the existing structure.
- Quick and Lean: Efficient and avoids unnecessary bloat.
- Powerful: Capable of building production-level applications.