Blockchain Honor Degree Sem VII

HBCC701 : Blockchain Development

Module - 4 : Blockchain Deployment (10 Hours)

Instructor : Mrs. Lifna C S

Topics to be covered

- **Ethereum client,**
- **Ethereum Network,**
- **Introduction to Go Ethereum (Geth),**
  - **Geth Installation and Geth CLI,**
  - **Setting up a Private Ethereum Blockchain.**
- **Introduction to Truffle**
  - **Smart Contract deployment on a Private Blockchain.**
- **Introduction to Ganache**
- **Introduction to DApp,**
  - **DApp architecture,**
  - **DApp Scalability,**
  - **DAppTesting**
- **Introduction to Web3 JS**
- **DApp & Web3 JS : Connecting to the Blockchain and Smart Contract and Deployment**

Self-learning Topics: Smart Contract deployment using Ganache

Ethereum Client

- **software program** that is used **to implement the Ethereum specification**
- **connect itself with other Ethereum clients** over a peer-to-peer network.
- Different Ethereum clients can communicate with one another if they follow the reference specification and the defined communication protocols.
- These interactions among different clients in the network **take place using various programming languages**
  - like Geth (Go), OpenEthereum (Rust),
  - Nethermind (C#, .NET), Besu (Java),
  - Erigon (Go/Multi).
- The **yellow paper** is the Ethereum protocol that allows anybody to run a client to construct a node.
- **Ethereum sets standard behaviors that all Ethereum clients must adhere to**
- **Ethereum's specs** enabled the blockchain to allow for different, but interoperable, software implementations of an Ethereum client by providing standard documentation and simple language.

Ethereum Client

1. **Full Client:**
   - **save the complete Ethereum blockchain**,
   - which **might take several days to synchronize** and
   - takes a **massive amount of disc space** – more than 1 Terabyte, according to the most recent estimates.
   - **Enable connected nodes to conduct all network functions**, including as
     - mining,
     - transaction
     - block-header validation,
     - smart contract execution.

Ethereum Client

2. **Light Client:**

- **do not always need to necessary keep all of the data,**
- when data storage and performance are concerns, developers utilize the “light clients”.
- Light clients **provide a portion of full client capability.**
- they can provide quick delivery and free up data storage space.
- The functionality of a light client is adapted to the purposes of the Ethereum client.
- **widely used within wallets to maintain private keys and Ethereum addresses.**
- **manage smart contract interactions and transaction broadcasts**.
- **useful for web3 instances** within JavaScript objects, Dapp browsers
- **obtaining the exchange rate data.**

Ethereum Client

3. **Remote Client:**
   - A remote client is **much like a light client**.
   - The primary distinction is that a **remote client does not keep its own copy of the blockchain or validate transactions or block headers.**
   - Remote clients, on the other hand, rely entirely on a full or light client to have access to the Ethereum blockchain network.
   - These clients are **mostly used as wallets for transmitting and receiving transactions.**

Ethereum Networks

- groups of connected computers that communicate using the Ethereum protocol.
- There is **only one Ethereum Mainnet,**
- But **independent networks**
  - conforms to the **same protocol rules** can be created
  - for testing and development purposes.
  - These independent "networks" that conform to the protocol **without interacting with each other.**

Note :

- One can even start one locally on your own computer for testing your smart contracts and web3 apps.
- Eg : Geth & Ganache Private Networks created during the Lab Experiments
- Yout **Ethereum account will work across the different networks**,
- But your account balance and transaction history won't carry over from the main Ethereum network.

For testing purposes :

- it's useful to know which networks are available
- how to get testnet ETH to play around with.

For security considerations : it's not recommended to reuse mainnet accounts on testnets or vice versa.

Courtesy : Ethereum

Ethereum Networks

- **PUBLIC NETWORKS**
  - Public networks are accessible to anyone in the world with an internet connection.
  - Anyone can read or create transactions on a public blockchain and validate the transactions being executed.
  - The consensus among peers decides on the inclusion of transactions and the state of the network.
- **Ethereum Mainnet**
  - Mainnet is the primary public Ethereum production blockchain,
  - where actual-value transactions occur on the distributed ledger.
  - When people and exchanges discuss ETH prices, they're talking about Mainnet ETH.

Courtesy : Ethereum

Ethereum Networks

- **Ethereum Testnets**
  - These are networks used by protocol developers or smart contract developers to test both protocol upgrades as well as potential smart contracts in a production-like environment before deployment to Mainnet.
  - Test any contract code you write on a testnet before deploying to Mainnet.
  - Among dapps that integrate with existing smart contracts, most projects have copies deployed to testnets.
  - Most testnets started by using a permissioned proof-of-authority consensus mechanism.
  - This means a small number of nodes are chosen to validate transactions and create new blocks – staking their identity in the process.
  - Alternatively, some testnets feature an open proof-of-stake consensus mechanism where everyone can test running a validator, just like Ethereum Mainnet.
  - **ETH on testnets is supposed to have no real value**;
  - Most people get testnet ETH for free from faucets.
  - Most faucets are webapps where you can input an address which you request ETH to be sent to.

Ethereum Networks

- **Which Testnet should I use?**
  - The two public testnets that client developers are **Sepolia** and **Goerli**.
  - **Sepolia**
    - recommended default testnet for application development.
    - Features :
      - Closed validator set, controlled by client & testing teams
      - New testnet, less applications deployed than other testnets
      - Fast to sync and running a node requires minimal disk space
  - **Goerli (long-term support)**
    - the Goerli testnet is deprecated and will be replaced by Holesovice in 2023.
    - Features :
      - Open validator set, stakers can test network upgrades
      - Large state, useful for testing complex smart contract interactions
      - Longer to sync and requires more storage to run a node

Courtesy : Ethereum

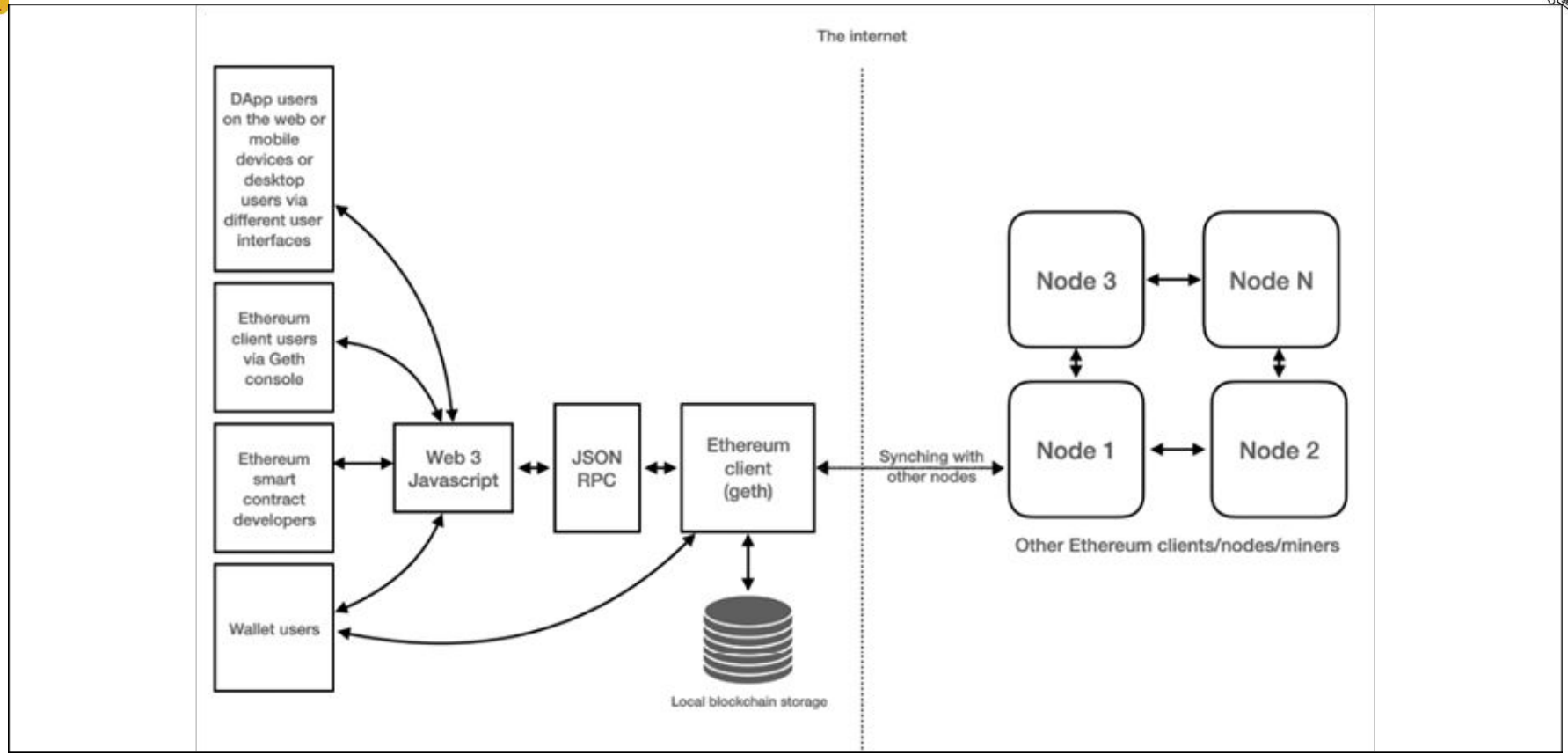
Ethereum Networks

- **PRIVATE NETWORKS**
  - An Ethereum network is a private network if its nodes are not connected to a public network (i.e. Mainnet or a testnet).
  - Private only means reserved or isolated, rather than protected or secure.
- **Development networks**
  - To develop an Ethereum application, run it on a private network to see how it works before deploying.
  - Create a local blockchain instance to test your dapp.
  - This allows for much faster iteration than a public testnet.
- **Consortium networks**
  - The consensus process is controlled by a pre-defined set of nodes that are trusted.
  - For example :
    - a private network of known academic institutions that each govern a single node,
    - and blocks are validated by a threshold of signatories within the network.

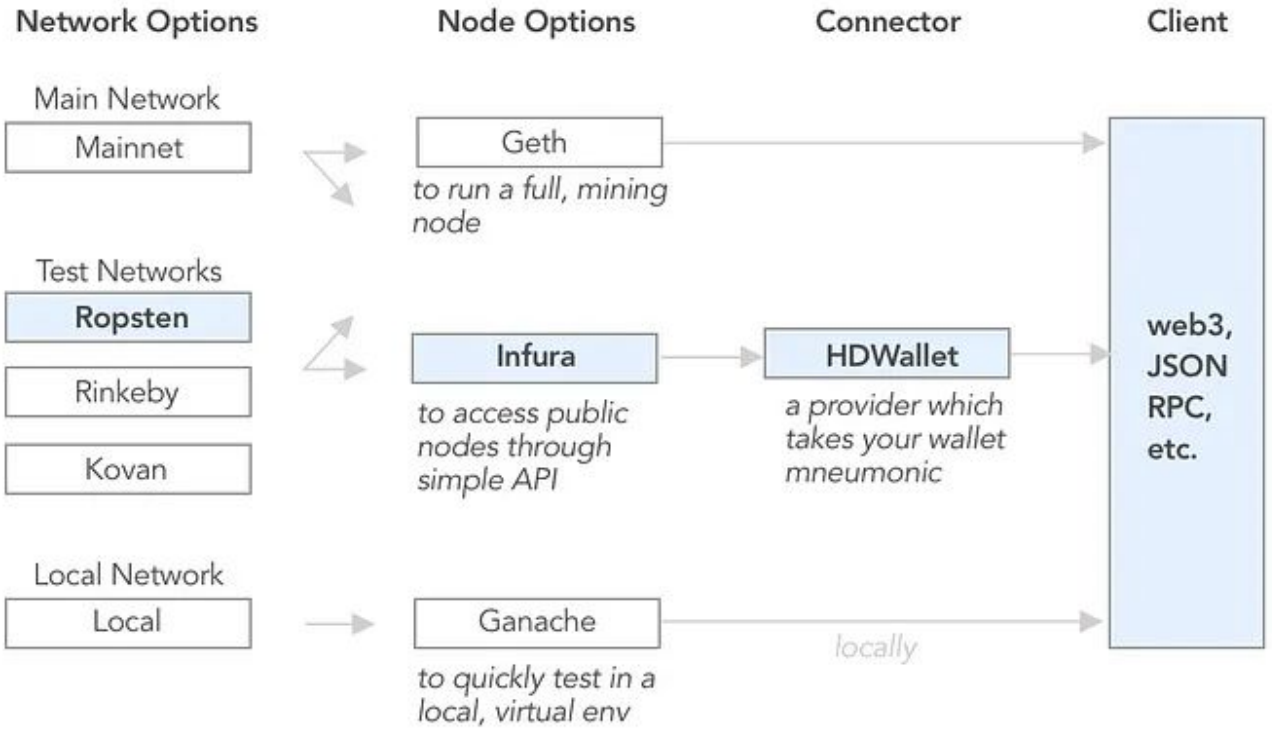
Note :

If a **public Ethereum network ⇒ public internet**, a **consortium network ⇒ private intranet.**

Ethereum High level Ecosystem

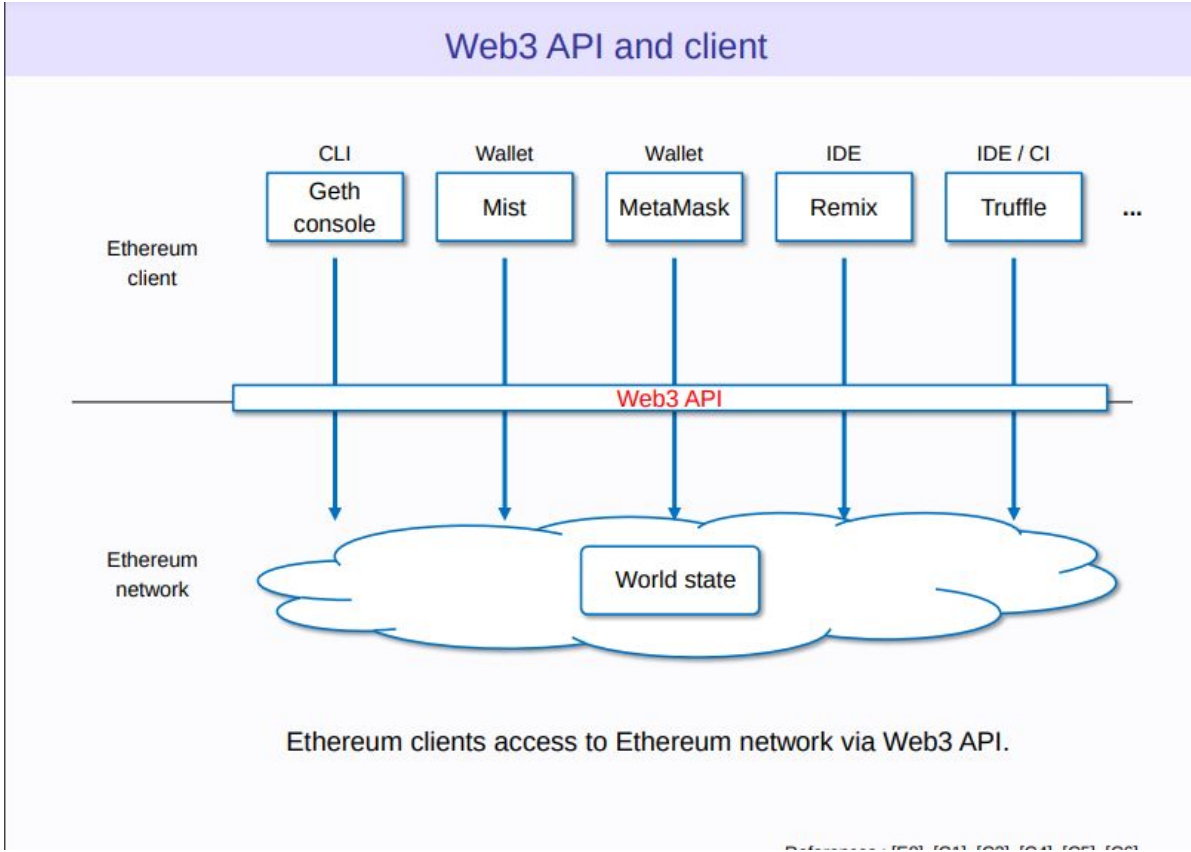


Ethereum Smart Contract Deployment Options



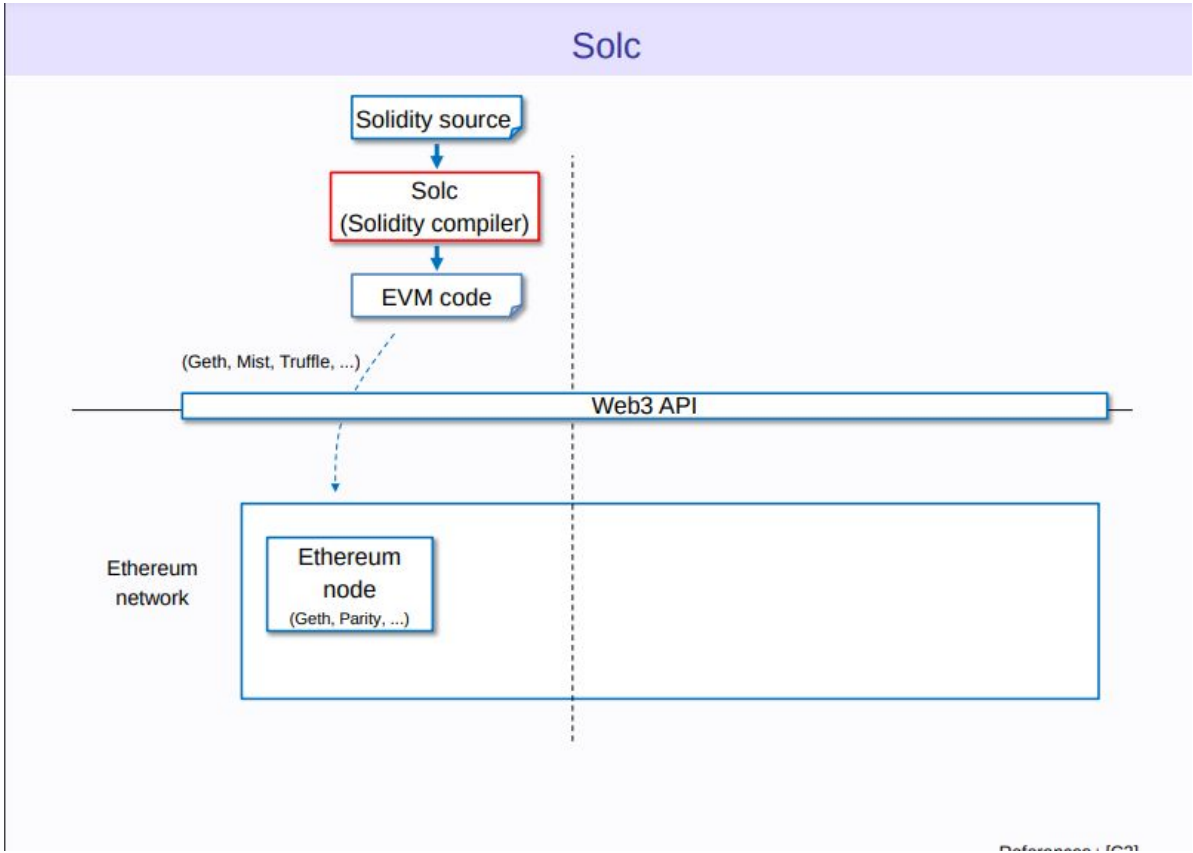
Courtesy : Medium - Coinmonks

Ethereum Blockchain Deployment via Web3 API

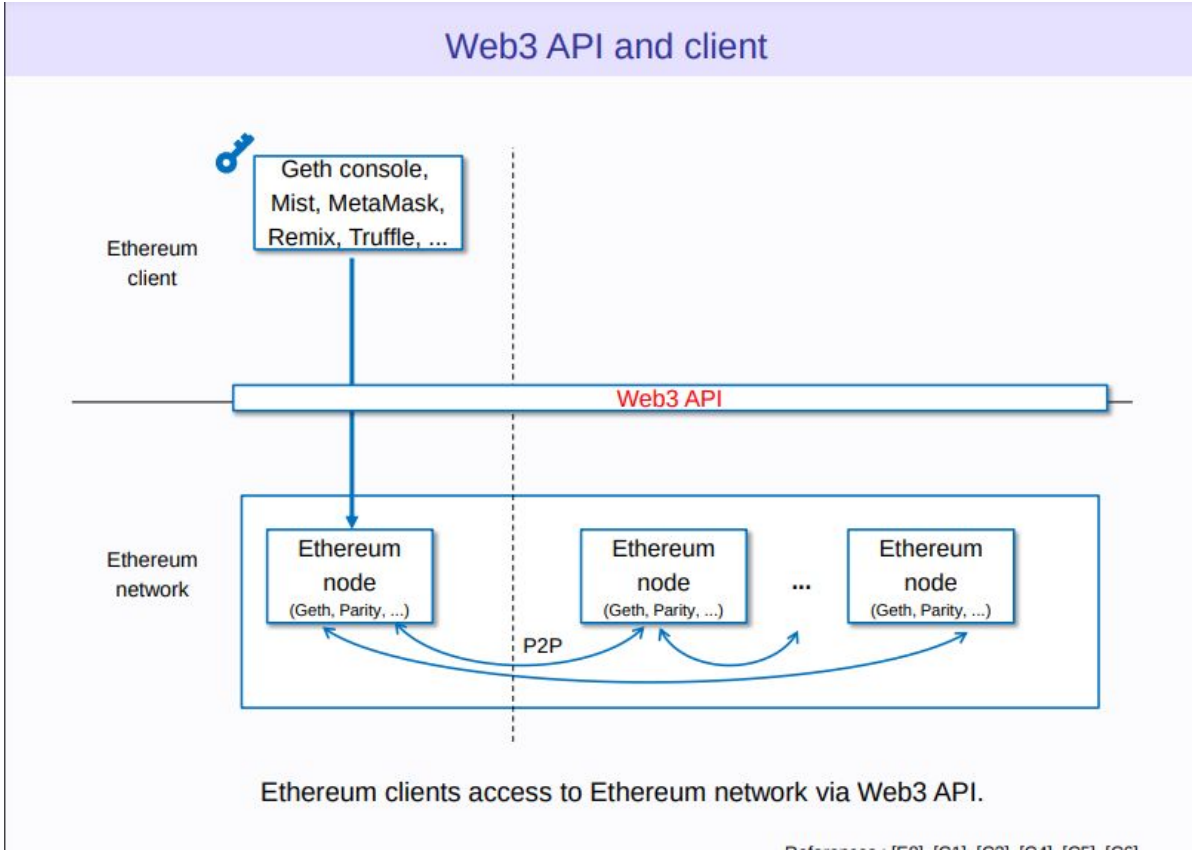


Courtesy : Takenobu T

Ethereum Blockchain Deployment via Web3 API

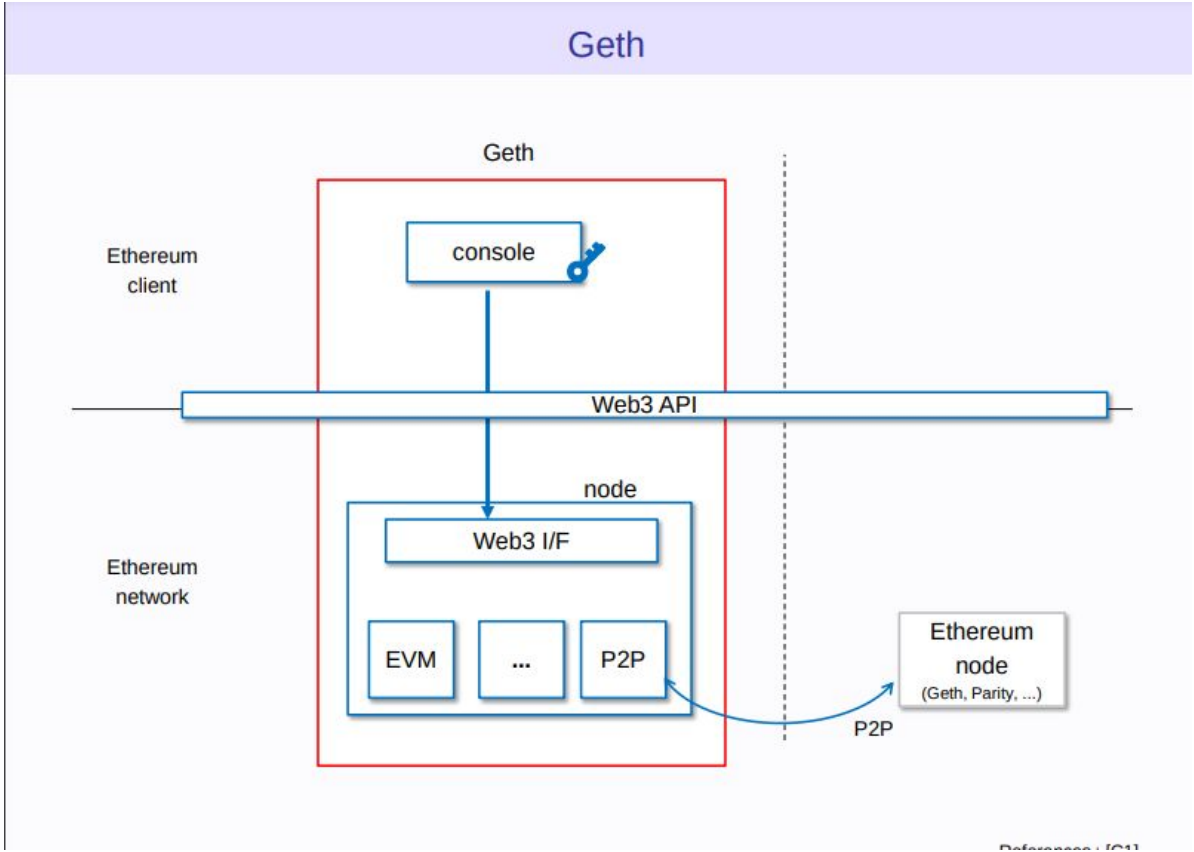


Ethereum Blockchain Deployment via Web3 API and Client



Courtesy : Takenobu T

Ethereum Blockchain Deployment via Web3 API and Geth



Courtesy : Takenobu T

Introduction to Go Ethereum (Geth)

- Most popular and widely used implementations of the Ethereum protocol.
- It serves as the official Go language implementation for the Ethereum blockchain.
- Geth plays a crucial role in the Ethereum ecosystem, as it allows nodes to participate in the network, mine, interact with smart contracts, and perform other essential functions.
- Features of Geth
  1. **Ethereum Node Implementation:**
     a. Geth is a software client that allows computers to connect to the Ethereum network and participate as nodes. Nodes are essential components of the Ethereum network, as they maintain a copy of the entire blockchain and help validate and relay transactions and smart contract interactions.
  2. **Developed in Go Language:** Efficient, fast, and well-suited for distributed systems like Ethereum.
  3. **Node Operations:**
     a. Geth can operate in multiple modes, including:
        i. Full Node: Maintains a complete copy of the Ethereum blockchain and validates all transactions and smart contract executions. Full nodes are crucial for network security and decentralization.
        ii. Light Node: Synchronizes with the blockchain more quickly by relying on other nodes for data. Light nodes are suitable for resource-constrained devices and mobile applications.
        iii. Archive Node: Stores historical data, including all past states of the Ethereum blockchain. Archive nodes are used for in-depth data analysis and research.

Introduction to Go Ethereum (Geth)

4. **Mining and Consensus:** Geth allows users to participate in the Ethereum network's consensus mechanism, which is transitioning from proof-of-work (PoW) to proof-of-stake (PoS) with Ethereum 2.0. In PoW, Geth can be used for mining to secure the network and validate transactions.
5. **Interacting with Smart Contracts:** Developers and users can interact with Ethereum smart contracts through Geth. It provides a command-line interface (CLI) and JSON-RPC API for sending transactions, deploying contracts, and querying contract data.
6. **Network Connectivity:** Geth can connect to different Ethereum networks, including the mainnet (the public Ethereum network), testnets (Ropsten, Rinkeby, Goerli, etc.), and private Ethereum networks. This flexibility is essential for development and testing.
7. **Security and Performance:** Geth is designed to be secure and highly performant, ensuring that it can handle the high demands of the Ethereum network while maintaining data integrity and consistency.
8. **Open Source and Community-Driven:** Geth is an open-source project, and its development is guided by the Ethereum community. Contributors and developers from around the world actively work on improving and maintaining the software.
9. **Continuous Development:** Geth is subject to continuous development and updates to adapt to changes in the Ethereum protocol, improve performance, and enhance security.

Geth Installation and Geth CLI

Steps for installing Geth on Ubuntu 22.04 LTS

sudo add-apt-repository -y ppa:ethereum/ethereum
sudo apt-get update
sudo apt-get install ethereum
sudo apt-get upgrade geth

After installation, check the status of the geth version

```
lifna@lifna-ThinkCentre-neo-50s-Gen-3:~$ geth version
Geth
Version: 1.12.0-stable
Git Commit: e501b3b05db8e169f67dc78b7b59bc352b3c638d
Architecture: amd64
Go Version: go1.20.3
Operating System: linux
GOPATH=
GOROOT=
```

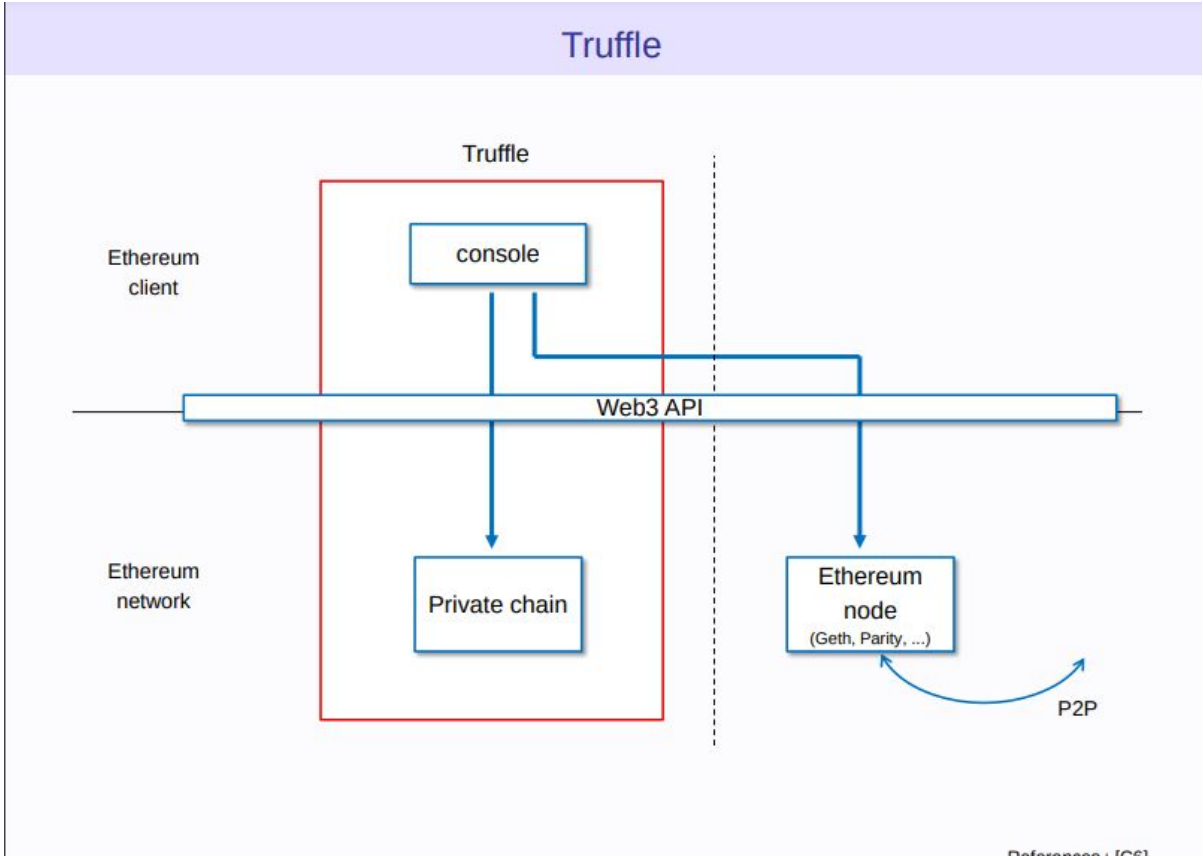

Courtesy : Github - Geth

Setting up a Private Ethereum Blockchain

1. Create a folder named, private_ethereum_setup
2. Create 2 subfolders named node1 and node2 in the folder private_ethereum_setup
3. Create 2 accounts in the folder corresponding to node1 and node2
4. Create a genesis.json file in the folder, private_ethereum_setup
5. Initialize the nodes with the genesis file
6. Configure the bootnode
7. Establish a Peer-Peer Connection between the nodes along with the bootnode
    a. On Terminal - 1 : Run the bootnode
    b. On Terminal - 2 : Run Node 1 as a miner
    c. On Terminal - 3 : Run Node 2 as a peer
8. On Terminal 4 : Attach JavaScript Console with Node 1
    a. Check the network status
    b. Chcek the balances of accounts associated with Node 1 & Node 2
    c. Perform Transactions
    d. Check the status Mempool

Courtesy : Github - Geth

Ethereum Blockchain Deployment via Web3 API and Truffle



Courtesy : Takenobu T

Introduction to Truffle

Truffle : A world class **development environment, testing framework and asset pipeline for blockchains using the Ethereum Virtual Machine (EVM)**, aiming to make life as a developer easier.

- Built-in smart contract compilation, linking, deployment and binary management.
- Advanced debugging with breakpoints, variable analysis, and step functionality.
- Use console.log in your smart contracts
- Deployments and transactions through MetaMask with Truffle Dashboard to protect your mnemonic.
- External script runner that executes scripts within a Truffle environment.
- Interactive console for direct contract communication.
- Automated contract testing for rapid development.
- Scriptable, extensible deployment & migrations framework.
- Network management for deploying to any number of public & private networks.
- Package management with NPM, using the ERC190 standard.
- Configurable build pipeline with support for tight integration.

Benefits of Building with Truffle

1. **Insight into Transaction Details:** Truffle Dashboard lets you inspect transactions before they occur by giving you more digestible information to more confidently approve or reject transactions.
2. **Diagnose and fix errors quickly:** The Truffle Debugger together with Truffle's testing suite allows you to surgically diagnose errors when they occur and fix them quickly
3. **Focus on your dapp's uniqueness:** Truffle offers a workflow that gets out of your way and lets you focus on your dapp's unique functionality. It's prescriptive enough to provide guidance, yet flexible enough to adapt to your needs.
4. **Operate safely:** With Ganache's zero-config mainnet forking, you can easily test against live networks without spending real Ether. You can execute risky transactions in a sandbox and test integrations with production smart contracts, all with the same human-readable information and debugging tools you use for development networks.
5. **Applicable at all stages of development:** Regardless of whether your project began with Truffle CLI or another framework, Truffle can provide tooling and insights. We also support a number of different protocols and offer the ability to easily add support for new ones.

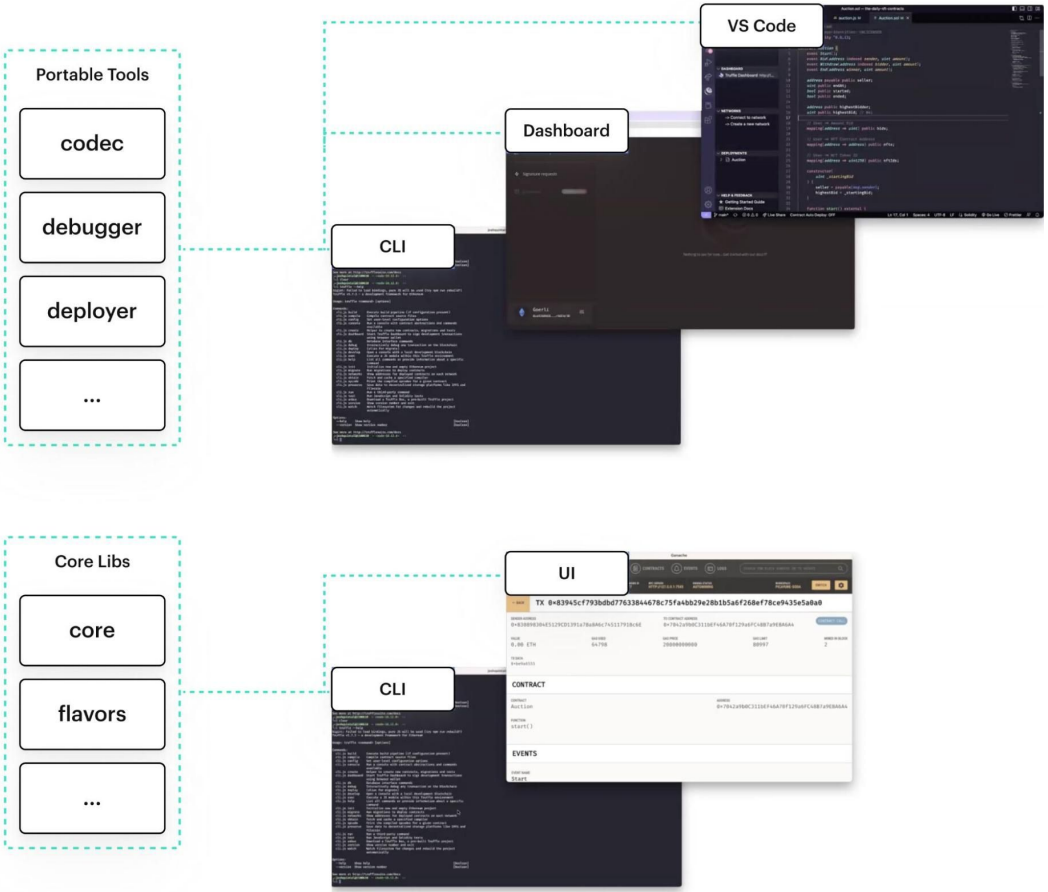
Truffle Suite Core Features and Tooling

- a suite of tools that come together to create a smooth development experience.
- These tools include Truffle CLI, Truffle Dashboard, Ganache, and Truffle for VS code.
- Underlying libraries that power these tools.
    - **Truffle Debugger** (@truffle/debugger)
        - Portable Solidity debugger library that can be used with or without Truffle.
        - It is a standalone package that is available on NPM,
        - Truffle's debugger is at the heart of every Truffle tool to integrate it into Truffle Dashboard.
    - **Truffle Encoder & Decoder** (@truffle/codec)
        - provides an interface for decoding Solidity smart contract state and information sent to, or from smart contracts using the Solidity ABI.
        - It produces output in a machine-readable form that avoids losing any information.
        - This is a low-level package for encoding, decoding, and data representation.
    - **Truffle Fetch & Compile** (@truffle/fetch-and-compile)
        - used to obtain externally verified sources and compile them.

Courtesy : Truffle Suite

Truffle Suite Core Features and Tooling

Portable Libraries are combined to form various tools within Truffle Suite



Courtesy : Truffle Suite

Truffle Command Line Interface

- A powerful command-line tool that provides developers with a wide range of functionality for building, testing, and deploying smart contracts on the Ethereum blockchain.
- At its core, the Truffle CLI provides a suite of commands for compiling, testing, and deploying smart contracts, as well as for interacting with the Ethereum network.
- These commands can perform a wide range of tasks, from compiling contracts and generating boilerplate code to testing contracts and deploying them to the blockchain.
- One of the key features of the Truffle CLI is its integration with other Truffle tools and frameworks, such as
  - **Ganache UI (a local blockchain simulator)**,
  - **Drizzle (a front-end library for building dapps)**, and
  - **Web3.js (a library for interacting with the Ethereum network)**. This integration makes it easy for developers to build, test, and deploy complex dapps using a single, unified toolset.
- **Prerequisites:** Node.js (version 18 or later), NPM (Node Package Manager)

Courtesy : Truffle Suite

Truffle Basic Commands

1. **`truffle init`**: Initializes a new Truffle project in the current directory.
2. **`truffle compile`**: Compiles your Solidity smart contracts.
3. **`truffle migrate`**: Deploys your smart contracts to the blockchain.
4. **`truffle test`**: Runs unit tests for your smart contracts.
5. **`truffle console`**: Opens the Truffle console, allowing you to interact with your contracts using JavaScript.
6. **`truffle develop`**: Starts a development blockchain locally for testing and development.
7. **`truffle networks`**: Lists the available network configurations in your Truffle project.
8. **`truffle version`**: Displays the Truffle version installed.

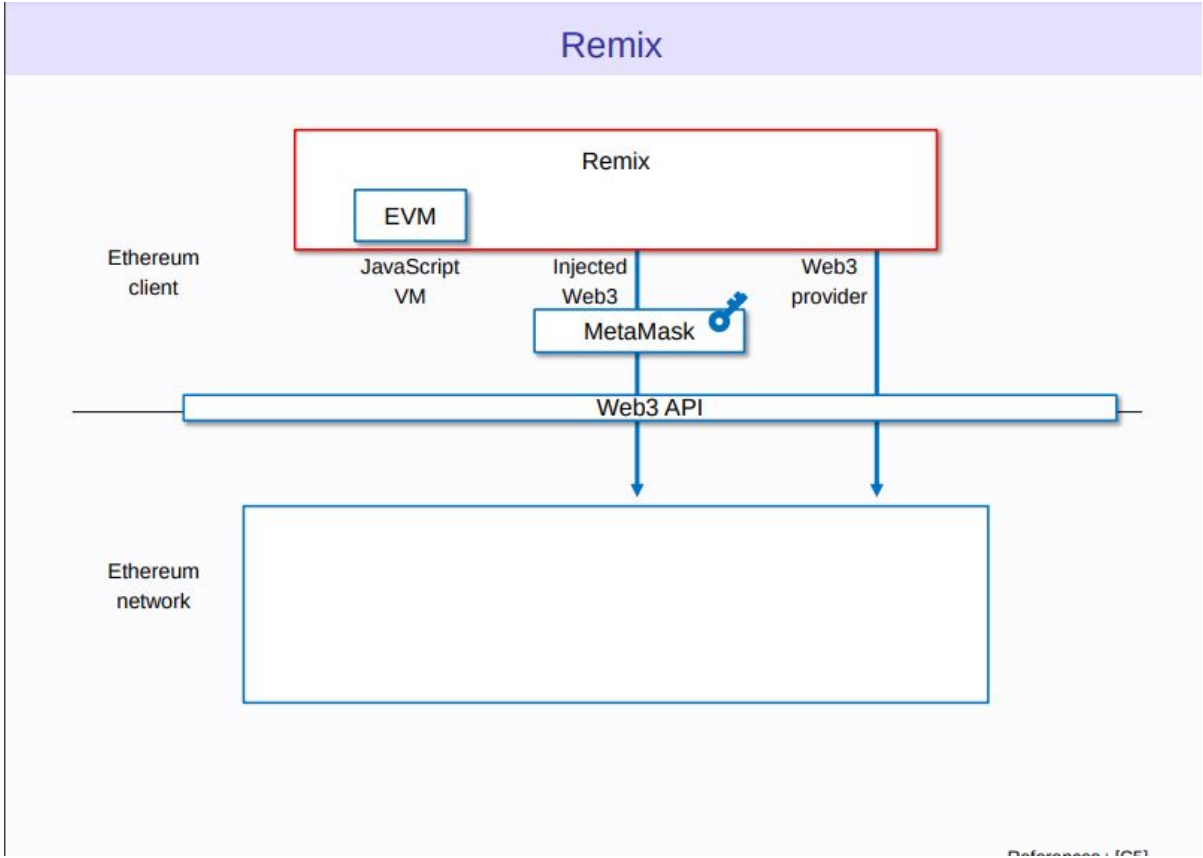
Courtesy : Truffle Suite

Truffle - Smart Contract deployment on a Private Blockchain

1. Install Truffle Suite (Windows / Linux)
2. Create a Truffle Project / Unbox the existing Truffle project
3. Compile the Truffle Project
4. Test the Smart Contracts
5. Deploy the Smart Contract on Ganache
    a. Update the network settings in **truffle-config.js** file
    b. Create a Workspace in Ganache - append the truffle-config.js file
    c. Use **truffle migrate --reset** to compile and deploy the contracts on Ganache

(Github Repo - Tutorial)

Ethereum Blockchain Deployment via Web3 API, Remix IDE & Metamask



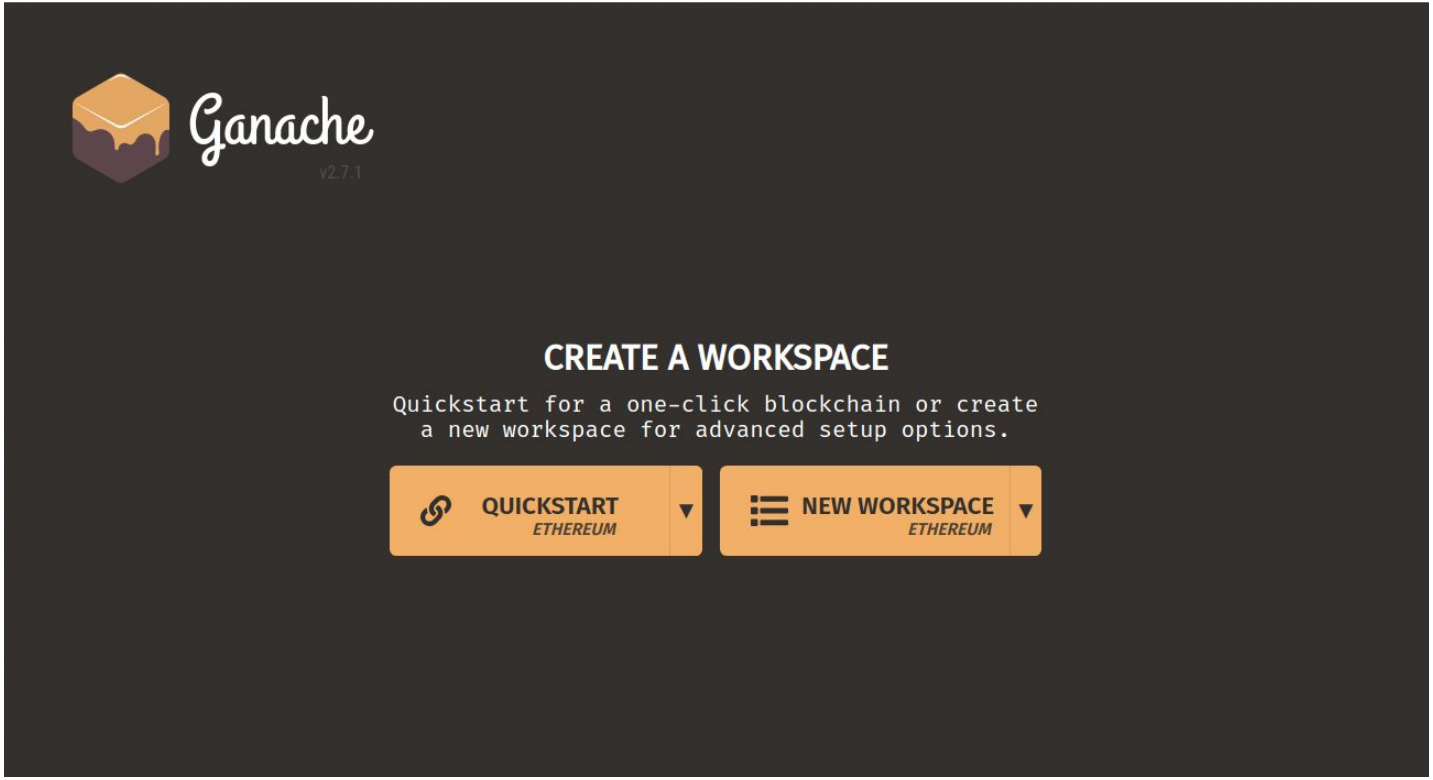
Introduction to Ganache for Ethereum blockchain

- Ganache is a personal blockchain for rapid Ethereum and Filecoin distributed application development.
- One can use Ganache across the entire development cycle;
- enabling you to develop, deploy, and test your dApps in a safe and deterministic environment.
- Ganache comes in two flavors: a UI and CLI.
    - Ganache UI is a desktop application supporting Ethereum and Filecoin technology.
    - More robust command-line tool, ganache, is available for Ethereum development. It offers:
        - console.log in Solidity
        - Zero-config Mainnet and testnet forking
        - Fork any Ethereum network without waiting to sync
        - Ethereum JSON-RPC support
        - Snapshot/revert state
        - Mine blocks instantly, on demand, or at an interval
        - Fast-forward time
        - Impersonate any account (no private keys required!)
        - Listens for JSON-RPC 2.0 requests over HTTP/WebSockets
        - Programmatic use in Node.js
        - Pending Transactions

All versions of Ganache are available for Windows, Mac, and Linux.

Introduction to Ganache for Ethereum blockchain

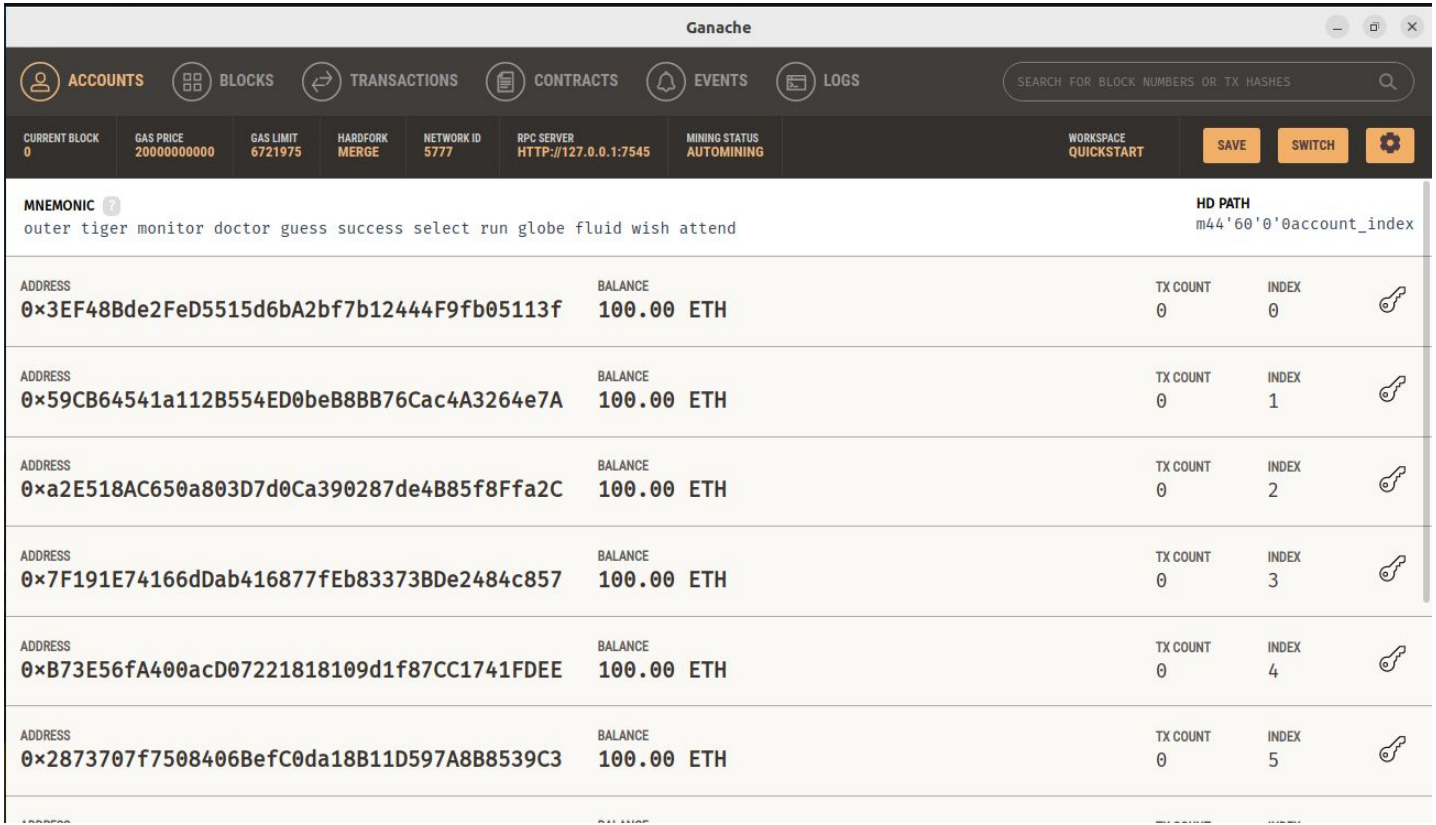
Flash screen of the Ganache after installation, Click on **Quickstart** to create a blockchain



Courtesy : Github - Ganache

Introduction to Ganache for Ethereum blockchain

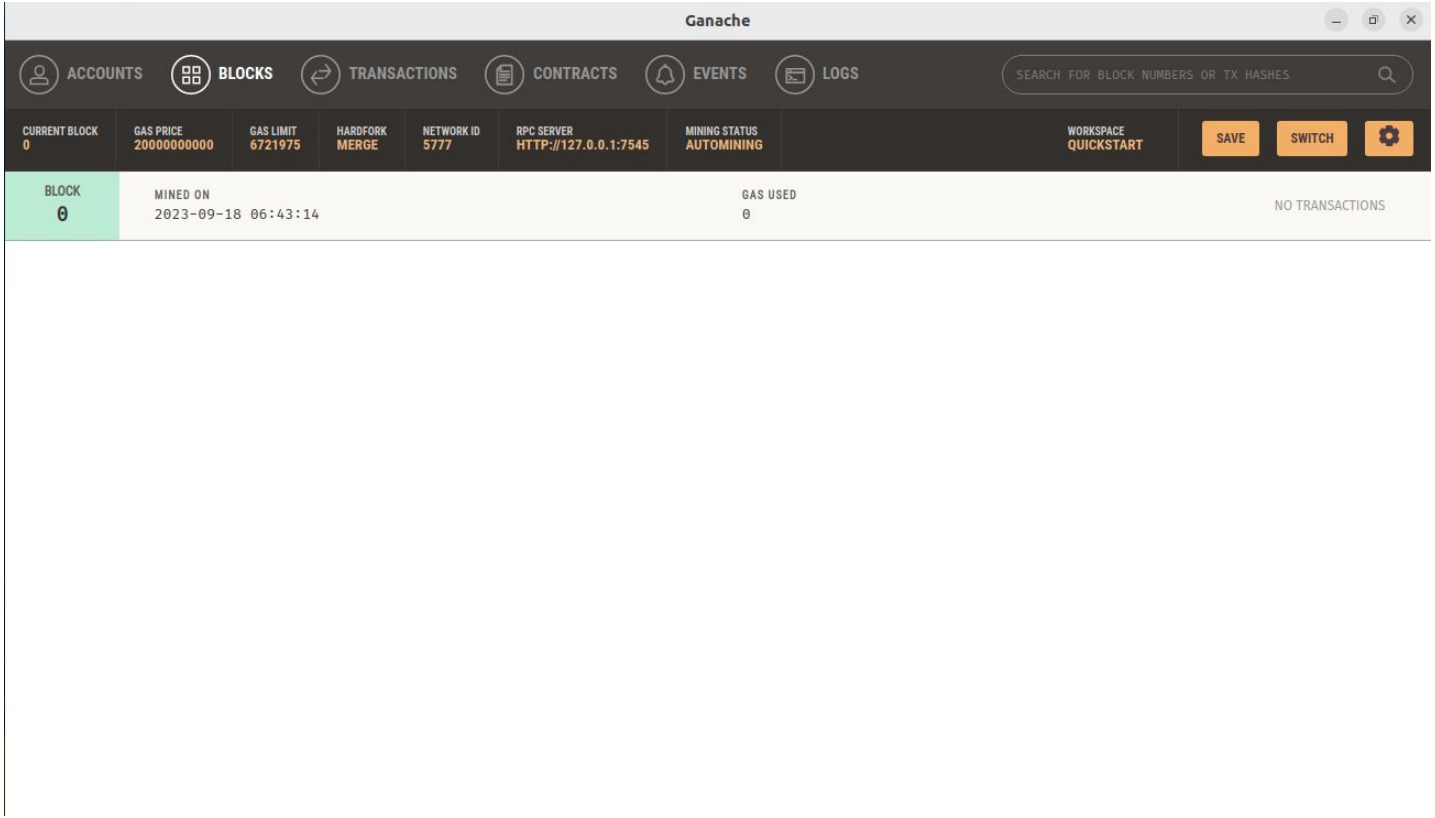
On the Accounts tab, enlists the 10 Accounts created each with 100 Ethers



Courtesy : Github - Ganache

Introduction to Ganache for Ethereum blockchain

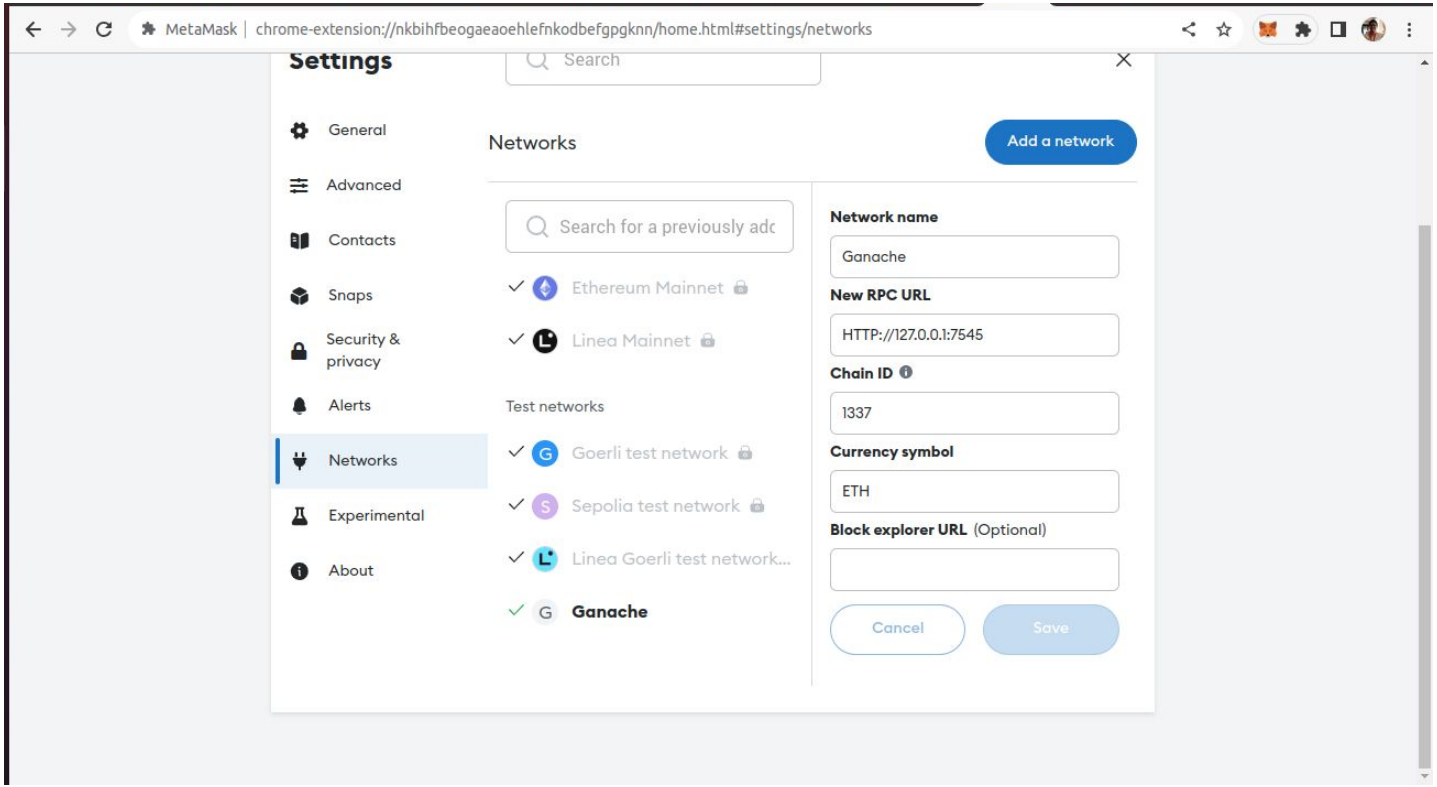
On the Blocks Tab, we could see the Genesis Block created



Courtesy : Github - Ganache

Introduction to Ganache for Ethereum blockchain

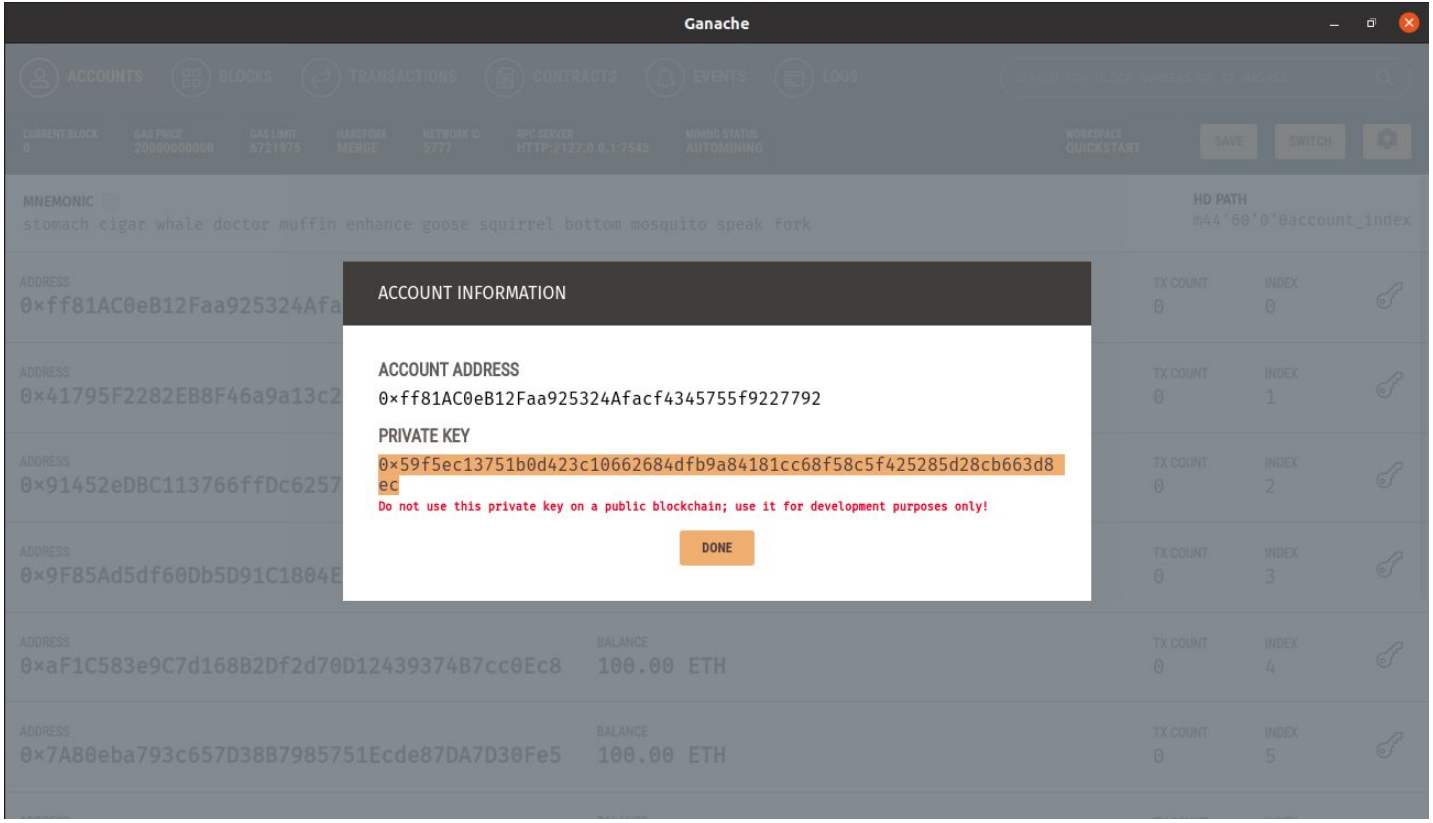
On the Metamask, add Ganache Network as follows:



Courtesy : Github - Ganache

Introduction to Ganache for Ethereum blockchain

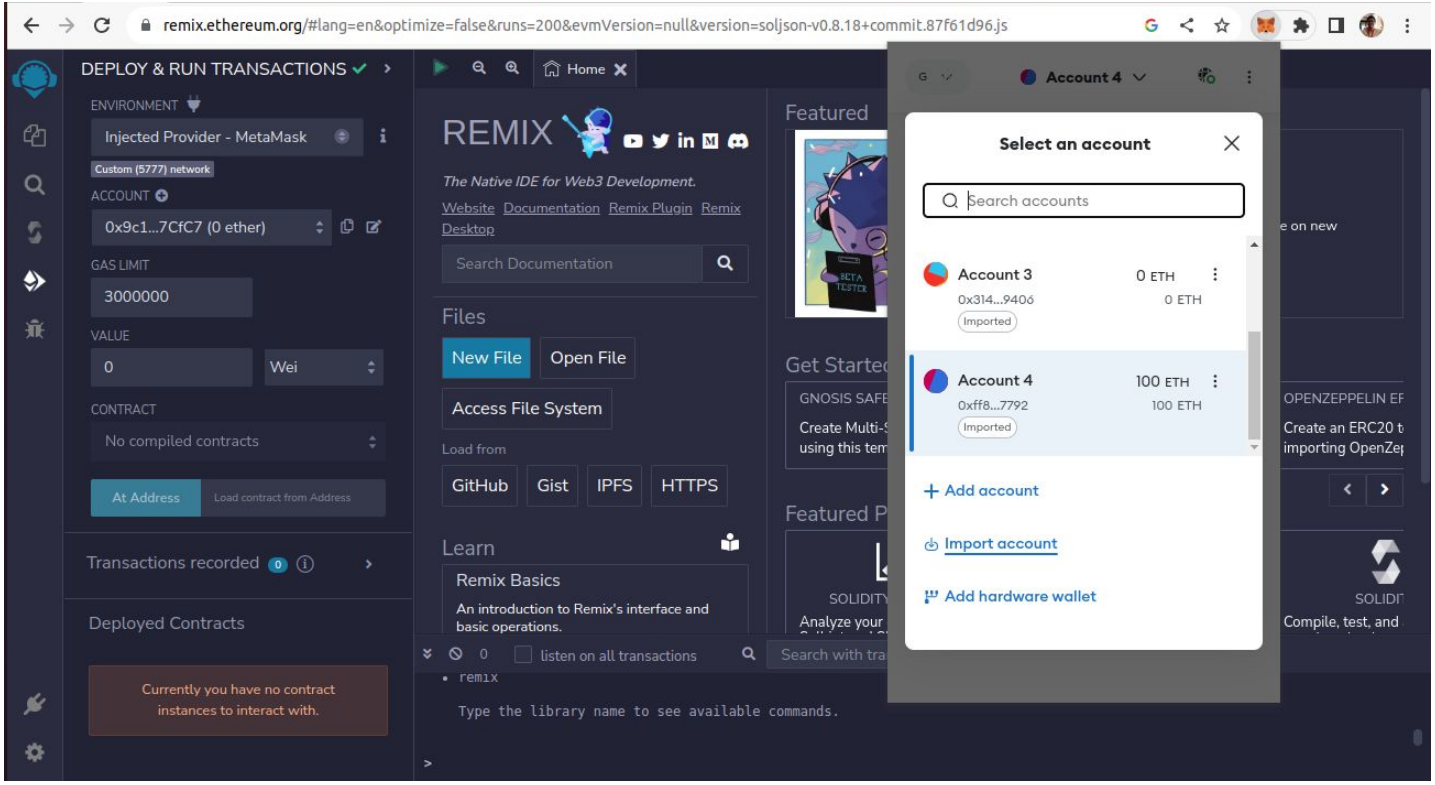
From on account, copy the Private key



Courtesy : Github - Ganache

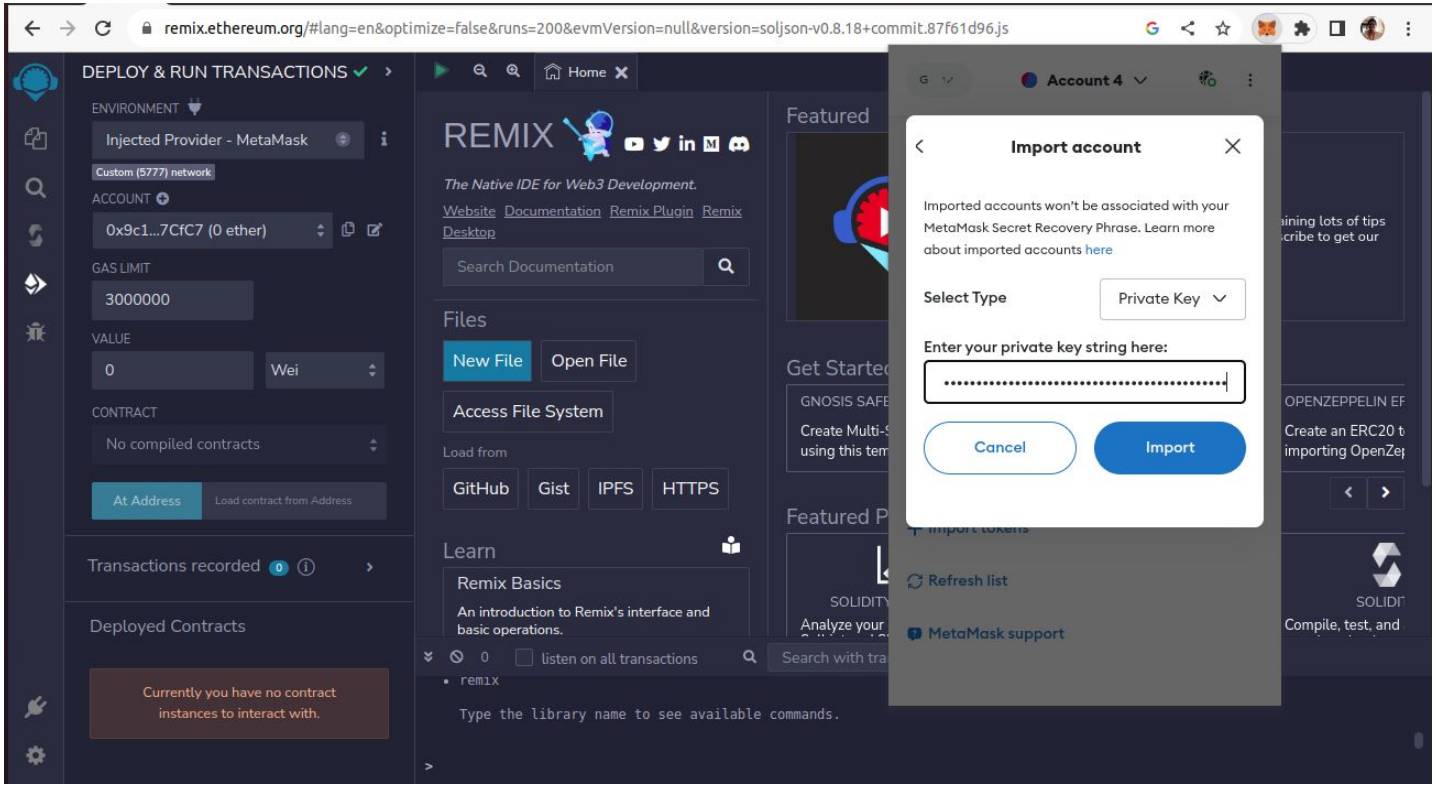
Introduction to Ganache for Ethereum blockchain

On the Metamask, select **Import Account**



Introduction to Ganache for Ethereum blockchain

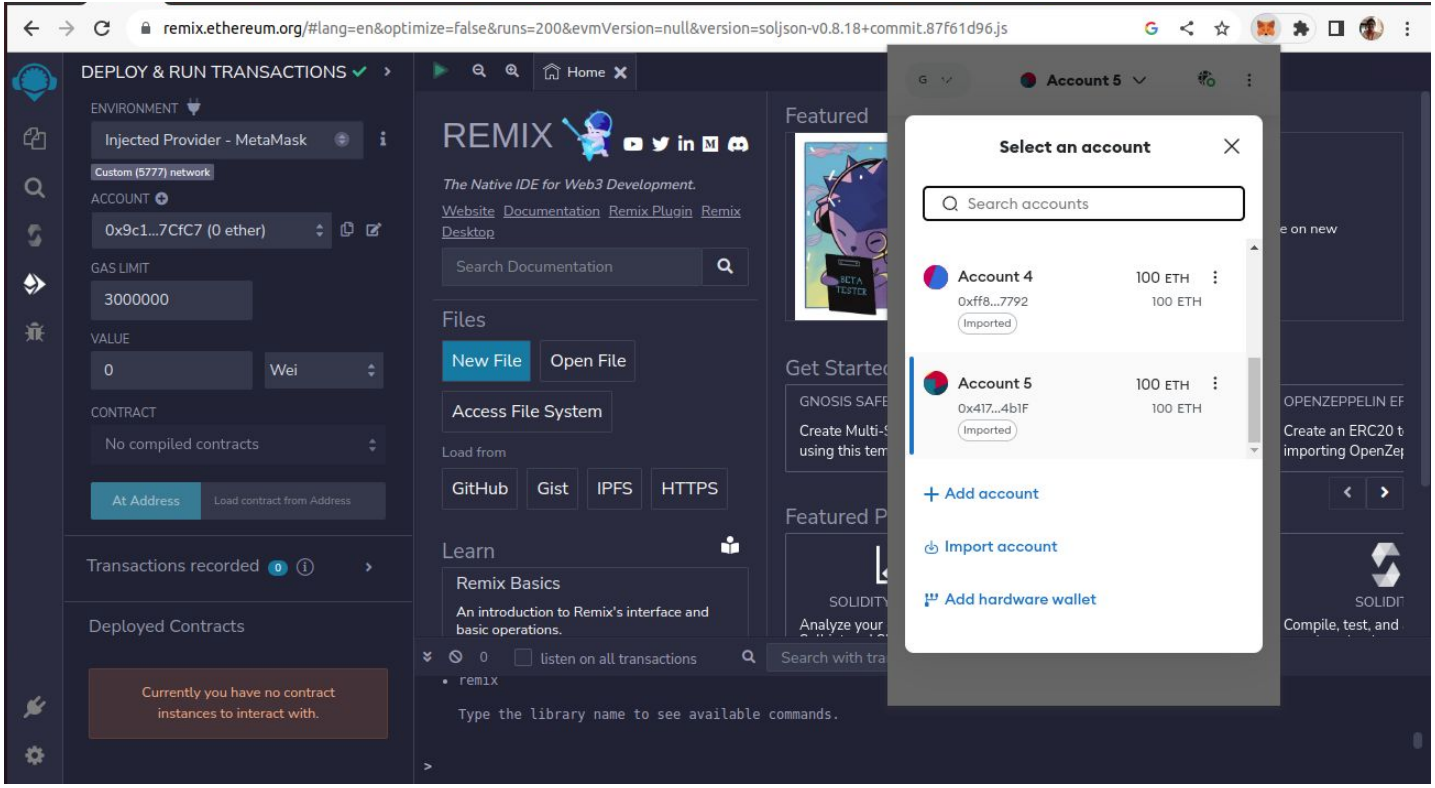
While importing Account on the Metamask, paste the Private key



Courtesy : Github - Ganache

Introduction to Ganache for Ethereum blockchain

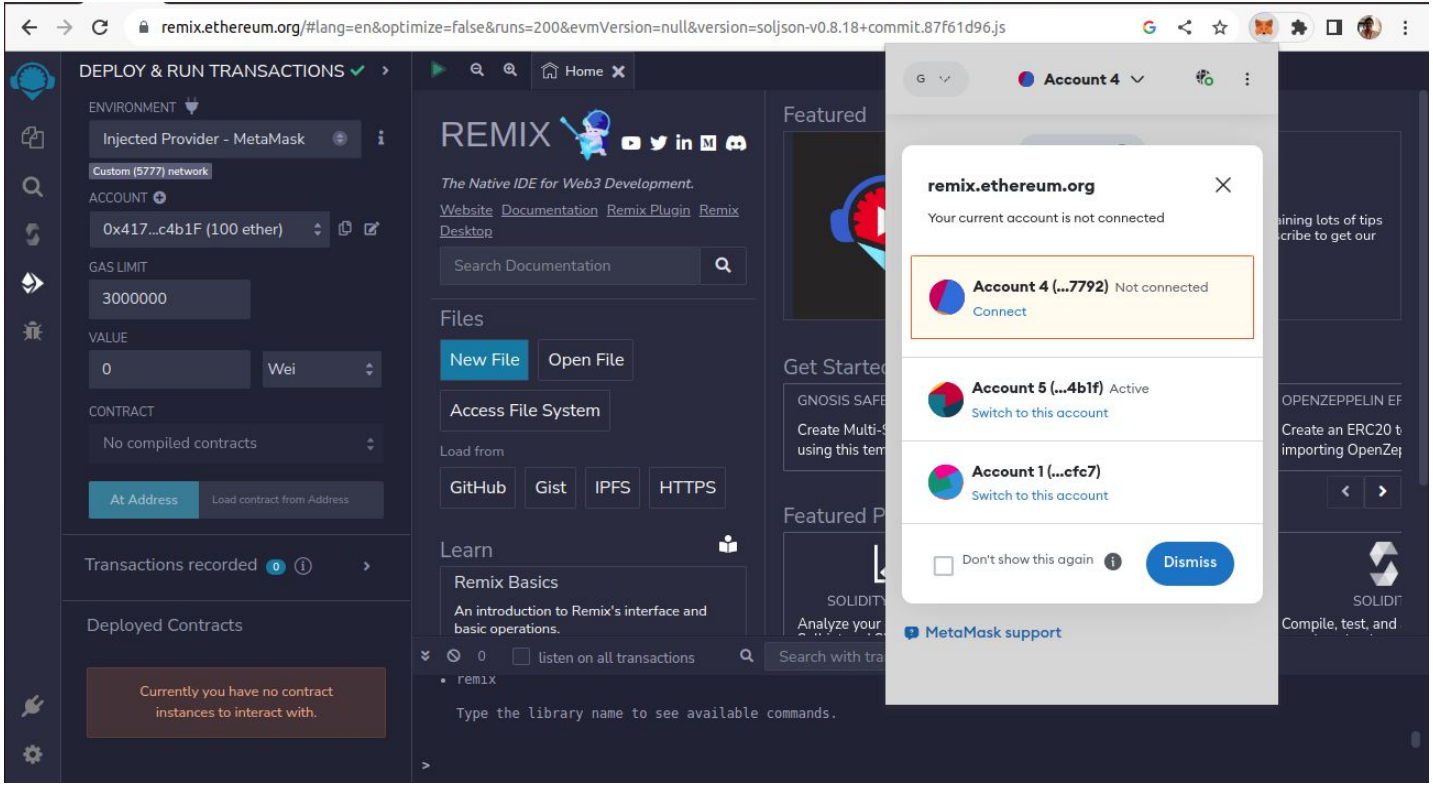
Enlists the Account imported



Courtesy : Github - Ganache

Introduction to Ganache for Ethereum blockchain

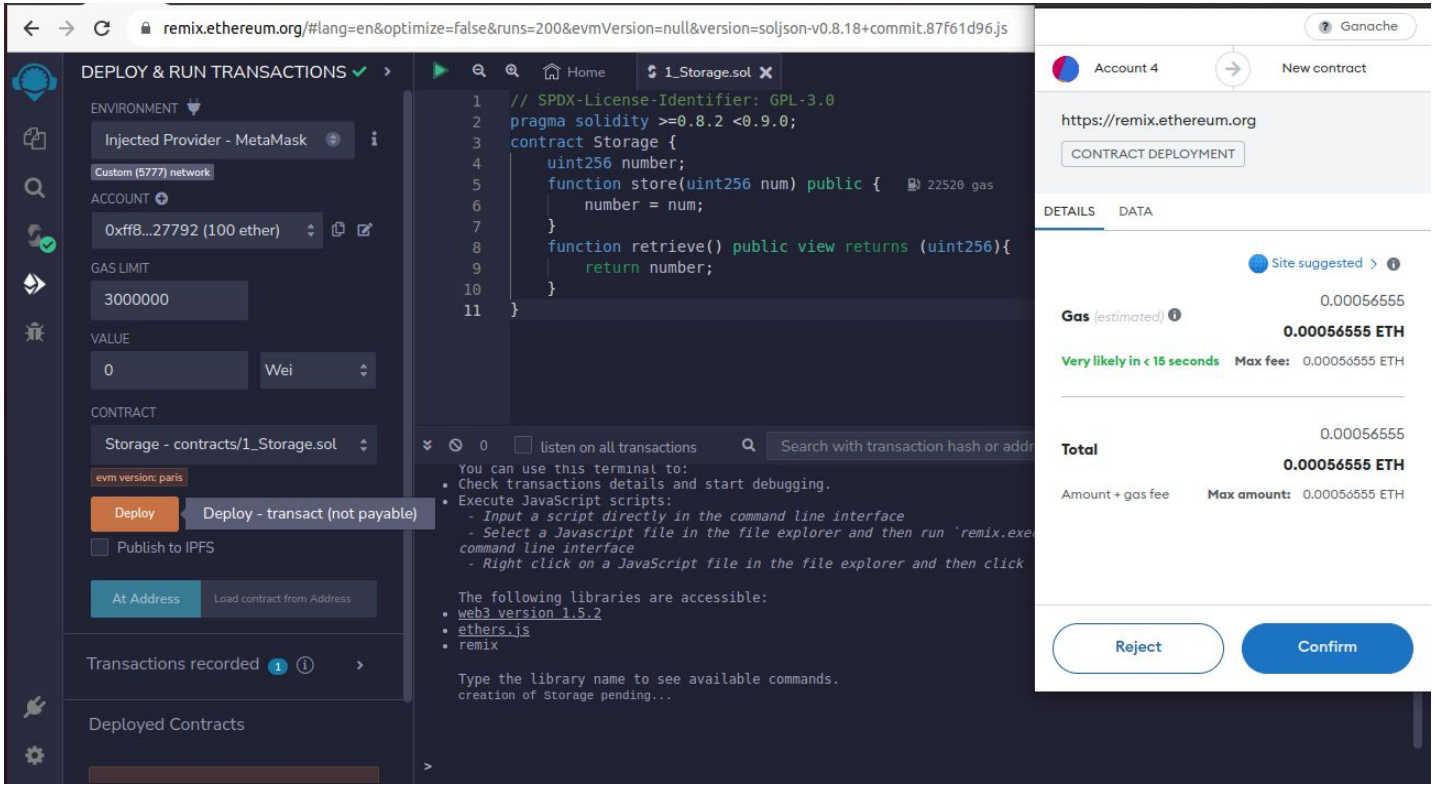
Select the Account from Metamask to be connected with Remix IDE



Courtesy : Github - Ganache

Introduction to Ganache for Ethereum blockchain

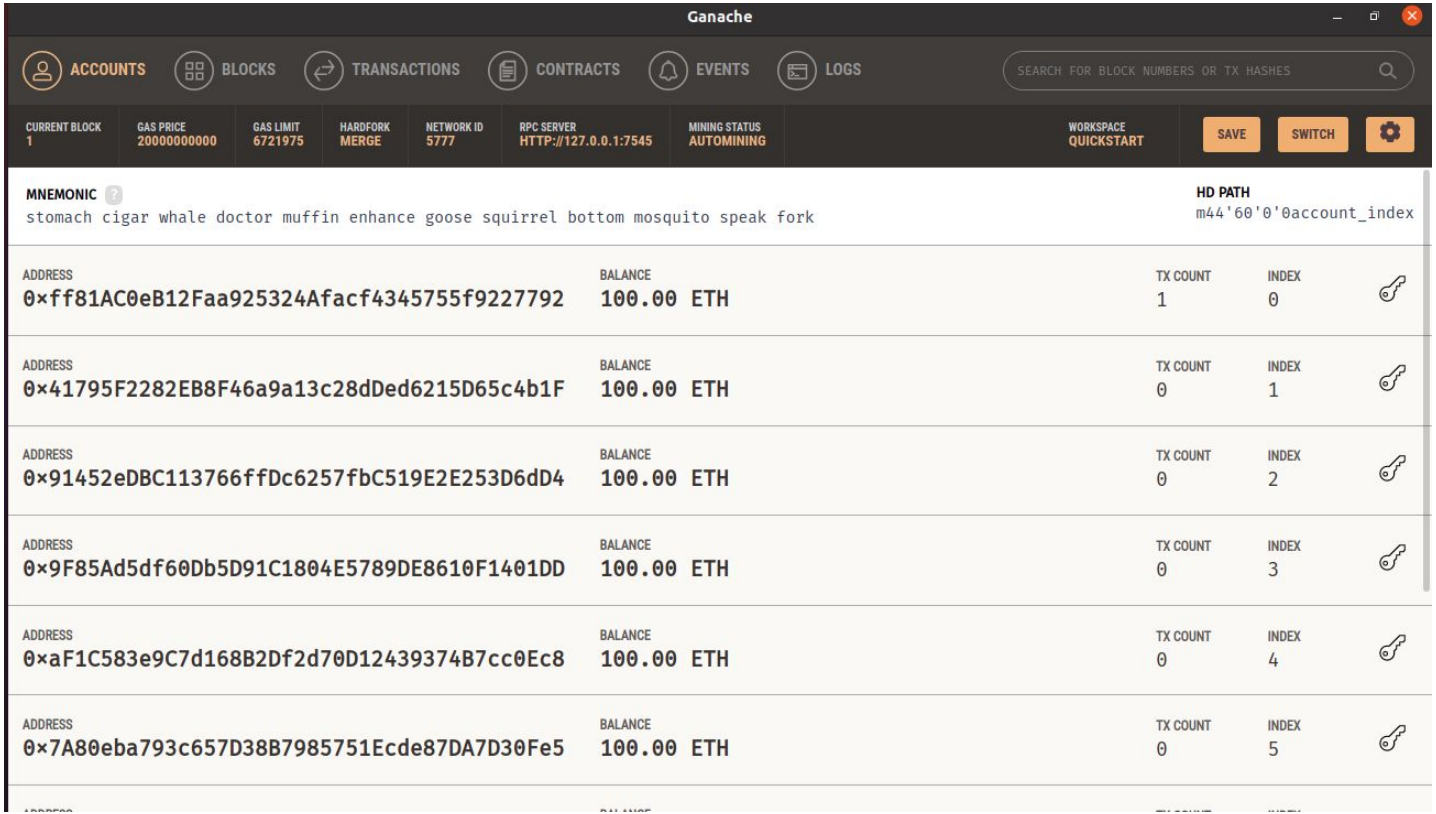
Deploy the Smart Contract



Courtesy : Github - Ganache

Introduction to Ganache for Ethereum blockchain

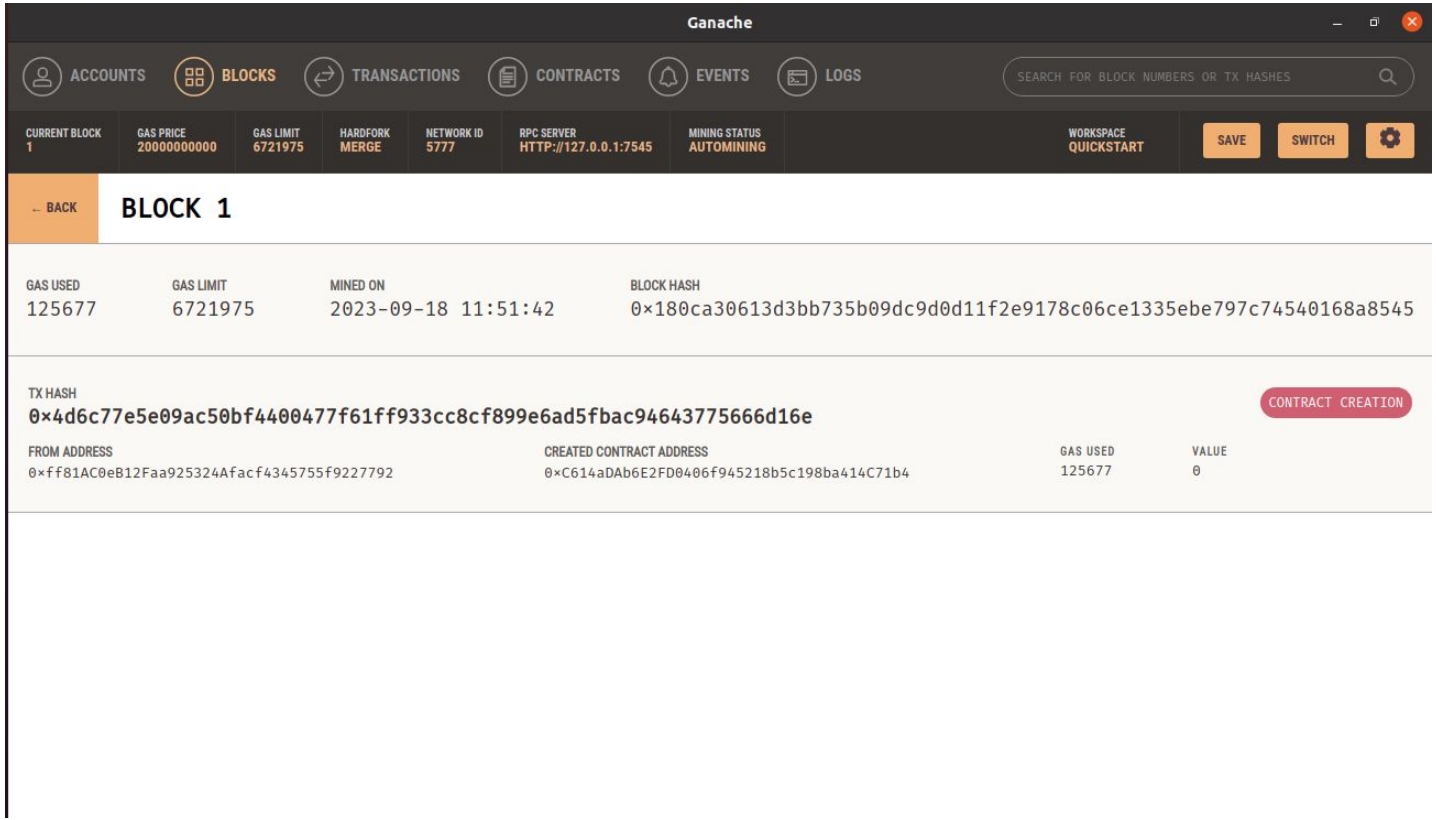
On the Ganache Environment, the Transaction count is updated



Courtesy : Github - Ganache

Introduction to Ganache for Ethereum blockchain

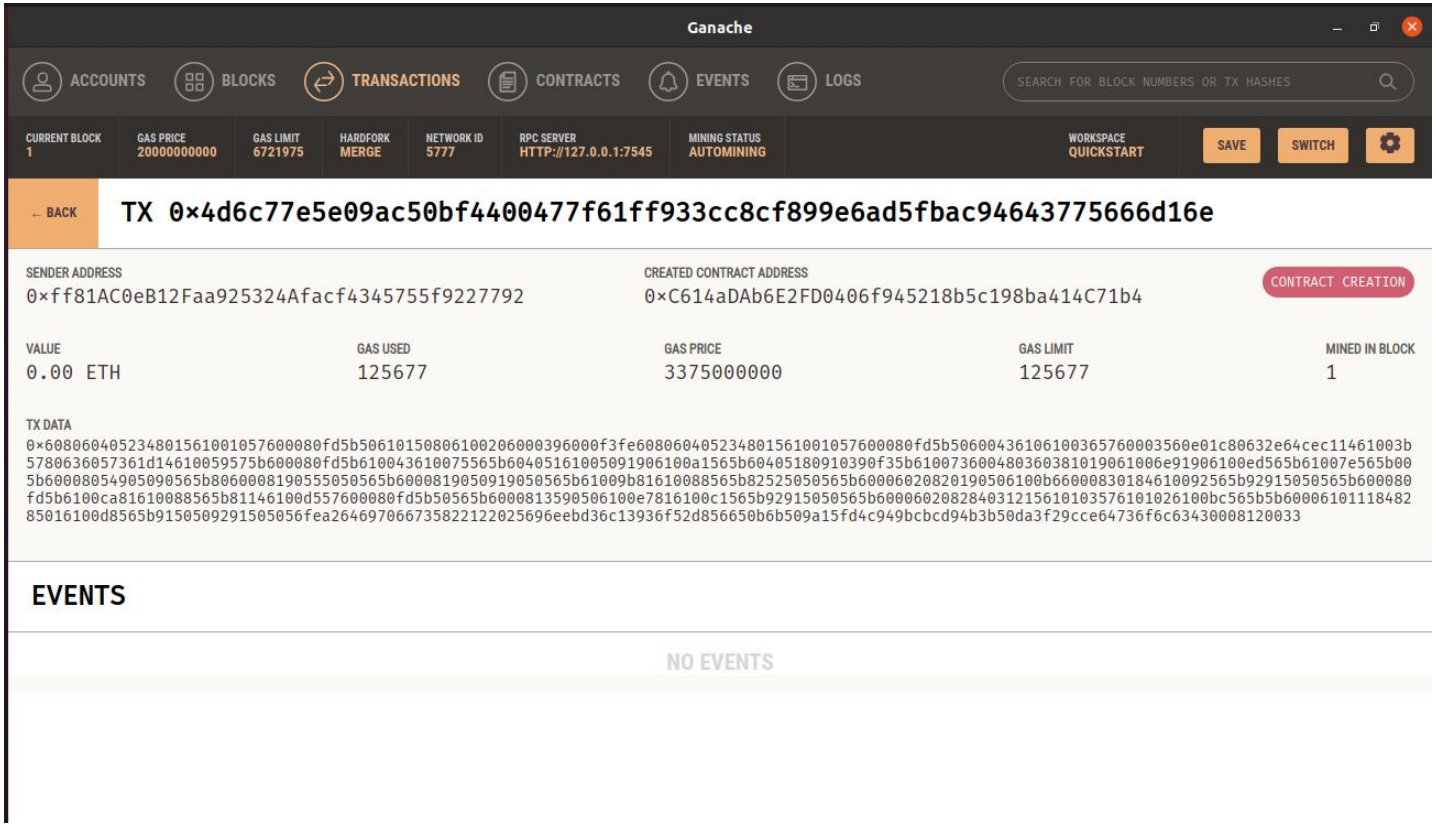
Block 1 is added to the Blockchain which displays the Contract Created



Courtesy : Github - Ganache

Introduction to Ganache for Ethereum blockchain

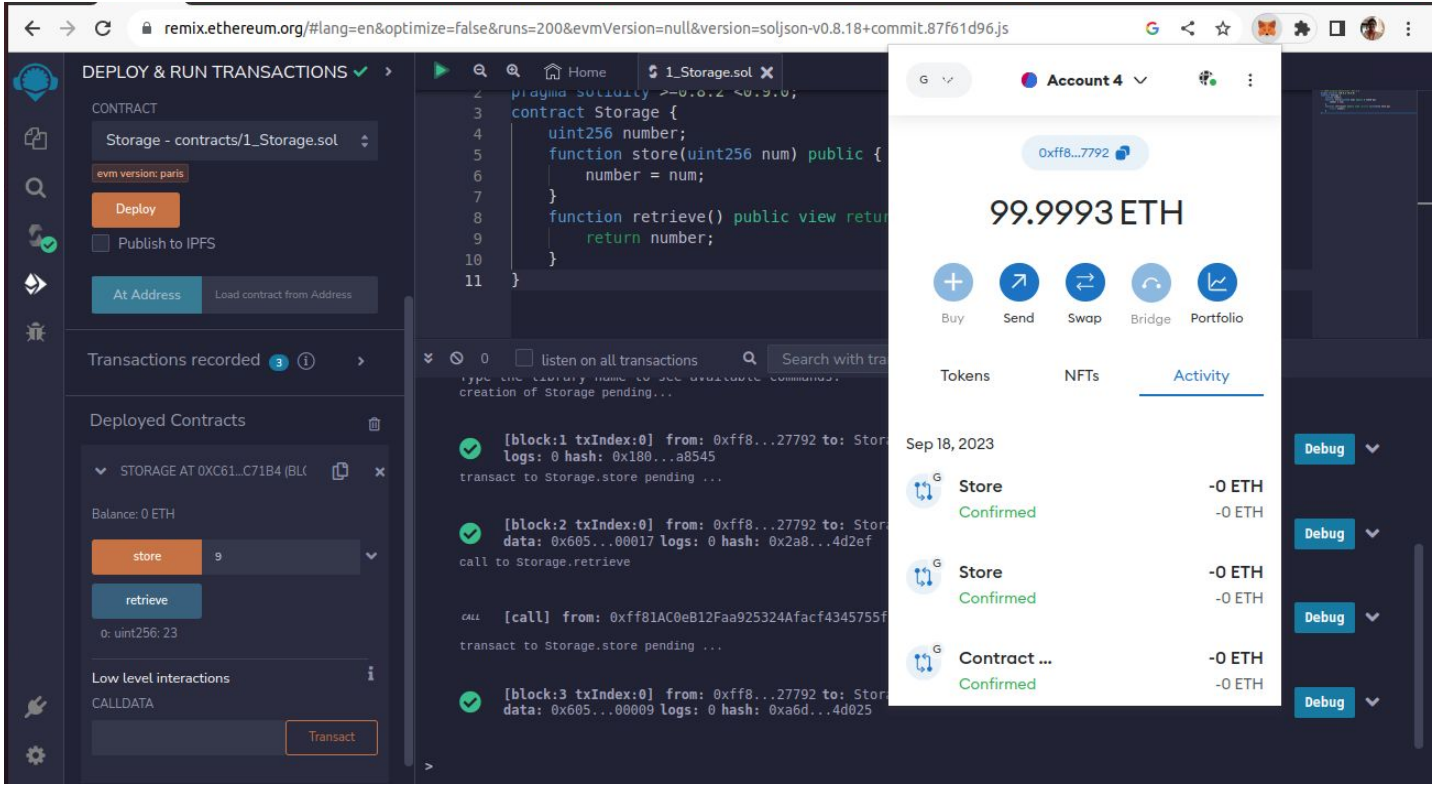
Transaction details of the Contract is displayed on Ganache Environment



Courtesy : Github - Ganache

Introduction to Ganache for Ethereum blockchain

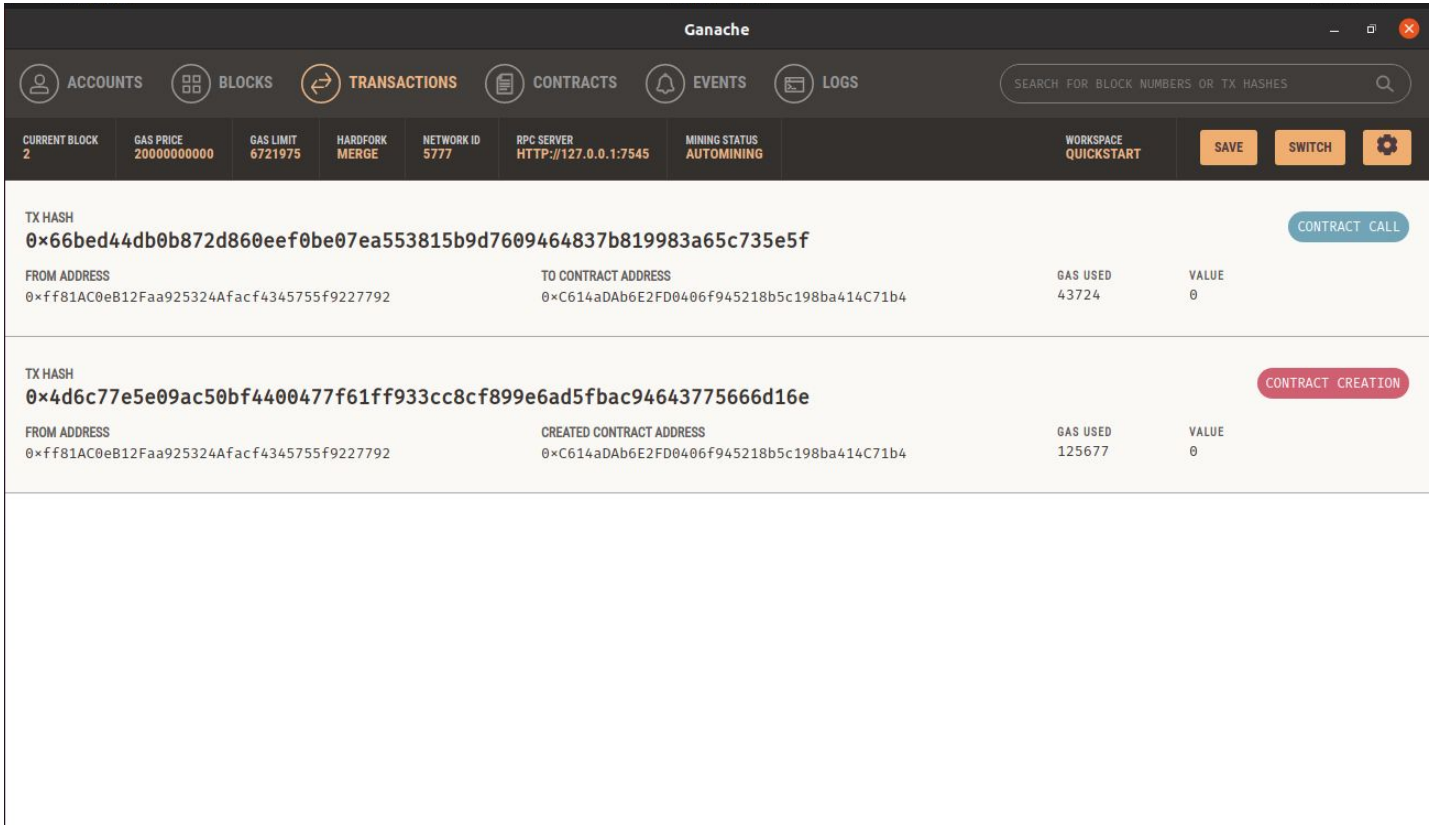
After interacting with the Smart Contract, funds are updated on the Metamask



Courtesy : Github - Ganache

Introduction to Ganache for Ethereum blockchain

On the Ganache, the Contract call is listed



Courtesy : Github - Ganache

Introduction to Dapp (Decentralized Application)

- A software application that operates on a decentralized network, typically using blockchain technology.
- Compared to traditional applications that rely on centralized servers, Dapps leverage the security, transparency, and trustlessness of blockchain to function.
- Salient Features of DApps
  1. **Decentralization:**
     - Dapps run on a decentralized network of computers, often referred to as a blockchain.
     - This network is distributed across many nodes, making it resistant to single points of failure and censorship.
  2. **Smart Contracts**:
     - Dapps typically use smart contracts, which are self-executing contracts with the terms of the agreement directly written into code.
     - Smart contracts automatically execute actions when specific conditions are met, without the need for intermediaries.
  3. **Transparency:**
     - All transactions and actions within a Dapp are recorded on a public ledger (the blockchain).
     - This transparency ensures that anyone can verify transactions and data, enhancing trust in the application.

Introduction to Dapp (Decentralized Application)

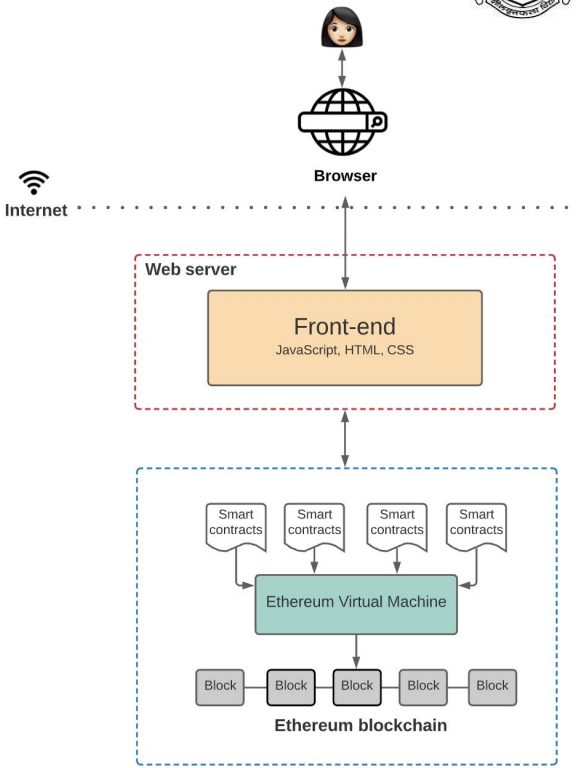
4. **Security:**
   - Blockchain technology, with its cryptographic techniques, provides a high level of security.
   - Transactions are immutable, meaning once recorded on the blockchain, they cannot be altered.
   - This makes Dapps resilient to fraud and hacking.
5. **Trustlessness:**
   - Dapps aim to operate without relying on a central authority or intermediary.
   - Users can interact with the application and each other directly without needing to trust a third party.
6. **Token Economy:**
   - Many Dapps have their own native tokens or use established cryptocurrencies like Ethereum's Ether.
   - These tokens can be used for various purposes within the application, such as paying for services, participating in governance, or earning rewards.
7. **Use Cases:**
   - Dapps have a wide range of use cases, including decentralized finance (DeFi), non-fungible tokens (NFTs), supply chain management, voting systems, gaming, and more.
   - Each Dapp is designed to solve specific problems or provide new functionalities in a decentralized manner.

Introduction to Dapp (Decentralized Application)

8. **User Interface:**
   - Dapps often have user interfaces (UIs) similar to traditional apps or websites, making them accessible to a broader user base.
   - Users may not even be aware that they are interacting with blockchain technology.
9. **Challenges:**
   - While Dapps offer numerous advantages, they also face challenges, including scalability issues, user adoption barriers, and regulatory compliance.
   - These challenges are actively being addressed by the blockchain community.
10. **Development:**
   - Building a Dapp typically involves programming smart contracts using languages like Solidity (for Ethereum) and developing a frontend interface.
   - Popular Dapp development frameworks like Truffle and web3.js make the development process more accessible.

Dapp Architecture

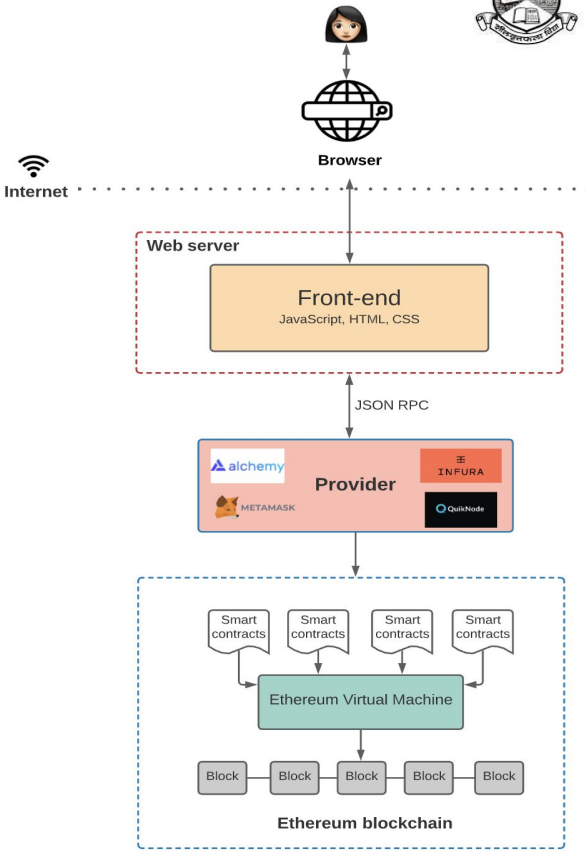
1. **Ethereum blockchain:**
   - Securely executes and verifies application code, called smart contracts.
   - DApps use the Ethereum blockchain for data storage.
2. **Smart Contracts:**
   - DApps use smart contracts to define the state changes happening on the blockchain.
   - A smart contract is a collection of code and data that resides at a specific address on the Ethereum Blockchain and runs on the Ethereum blockchain.
3. **Ethereum Virtual Machine(EVM):**
   - Global virtual computer that executes the logic defined in the smart contracts and processes the state changes that happen on this Ethereum network.
4. **Front-end:**
   - The part of the DApps, that users can see and interact with such as the graphical user interface(GUI),
   - Communicates with the application logic defined in smart contracts.



Courtesy : Preeti - DApp

DApp - How Frontend Code Communicate ?

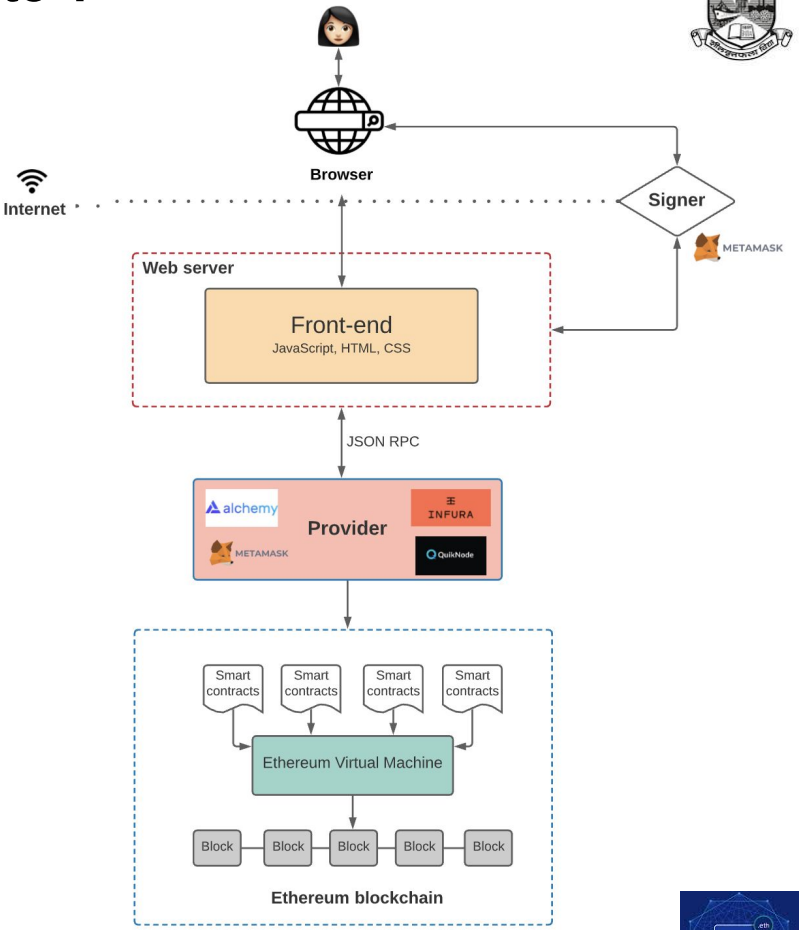
- When we need to interact with the data and code on a blockchain, we need to interact with one of these nodes.
- As any node can broadcast a request for a transaction to be executed on the EVM.
- A miner will then execute the transaction and propagate the resulting state change to the rest of the network.
- There are two ways to broadcast a new transaction:
  1. Set up your own node which runs the Ethereum blockchain software
  2. Use nodes provided by third-party services like Infura, Alchemy, and Quicknode
- Every Ethereum client (i.e. provider) implements a JSON-RPC specification. This ensures that there's a uniform set of methods when frontend applications want to interact with the blockchain.



Courtesy : Preeti - DApp

DApp - How Frontend Code Communicate ?

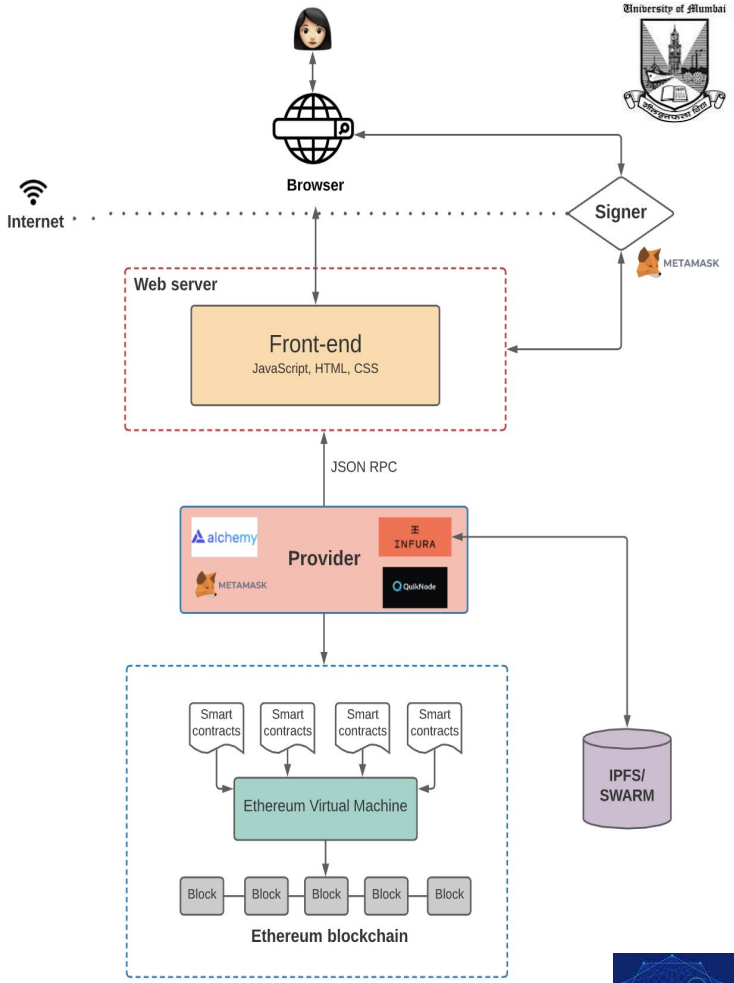
- When a user wants to publish a new post onto the chain, our DApp would ask the user to “sign” the transaction using their private key — only then would the DApp relay the transaction to the blockchain.
- Otherwise, the nodes wouldn’t accept the transaction.
- This “signing” of transactions is where Metamask typically comes in.



Courtesy : Preeti - DApp

DApp - Storage on the Blockchain

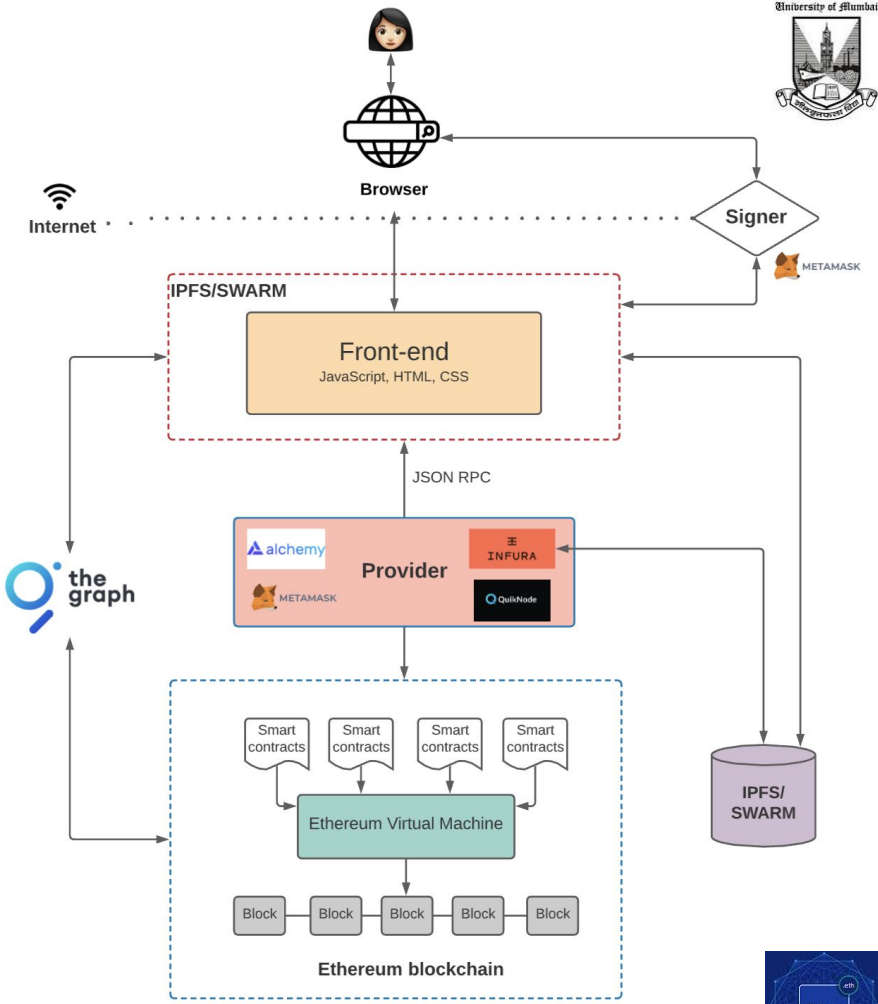
- Storing everything on the blockchain gets really expensive.
- Users to pay extra for using your DApp every time their transaction requires adding a new state is not the best user experience.
- One way to combat this is to **use a decentralized off-chain storage solution**, like IPFS or Swarm.
- IPFS : Distributed file system for storing and accessing data.
  - distributes and stores the data in a peer-to-peer network.
  - Has an incentive layer known as “Filecoin.” This layer incentivizes nodes around the world to store and retrieve this data.
  - IPFS providers
    - **Infura** (which provides you with an IPFS node) or
    - Pinata (“pin” your files to IPFS and take the IPFS hash and store that on the blockchain).
- **Swarm** : A decentralized storage network,
  - **Difference :** Swarm’s incentive system is built-in and enforced through smart contracts on the Ethereum blockchain for storing and retrieving data.



Courtesy : Preeti - DApp

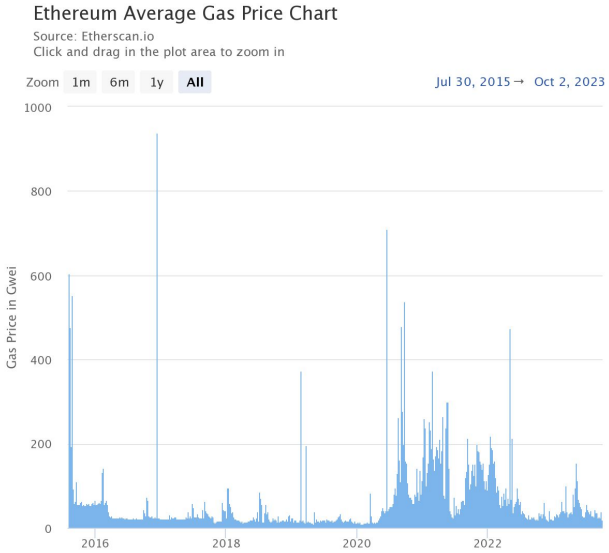
DApp Querying the Blockchain

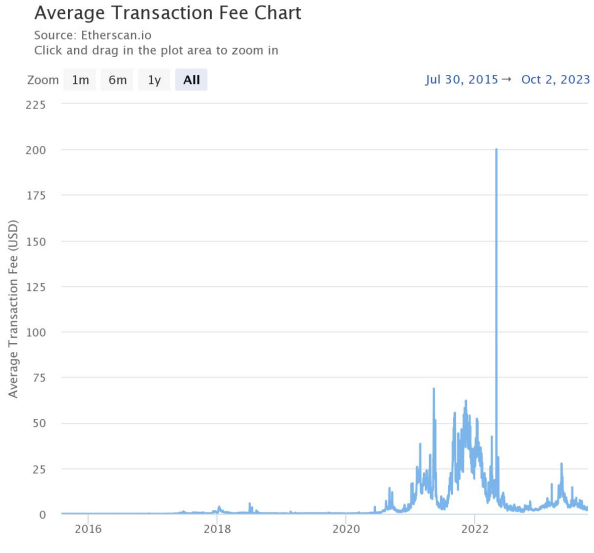
1. **Smart Contract Events :**
   - Use the Web3.js library to query and listen for smart contract events.
   - Here, we need to listen to specific events and specify a callback every time the event is fired.
   - using callbacks to handle various UI logic gets very complex
   - Issue: when you deploy a smart contract and later realize you need an event emitted that you didn't originally include.
2. **The Graph**
   - An off-chain indexing solution that makes it easier to query data on the Ethereum blockchain.
   - Allows to define which smart contracts to index, which events and function calls to listen to, and how to transform incoming events into entities that the frontend logic can consume.
   - It uses **GraphQL** as a query language,
   - By indexing blockchain data, The Graph lets us query on-chain data in our application logic with low latency

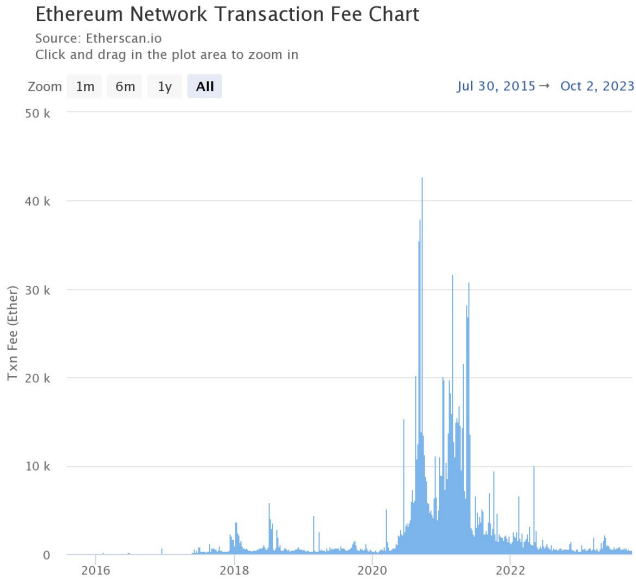


Courtesy : Preeti - DApp

Dapp Scalability







Note : Building a DApp on Ethereum with high gas fees and full blocks leads to a very bad UX.

Courtesy : Etherscan.io

Dapp Scalability

- to handle a growing user base and increasing transaction volume while maintaining a smooth and cost-effective user experience.
- Some strategies and approaches to address Dapp scalability:
  1. **Layer 2 Solutions: :** Protocols built on top of existing blockchains that enable faster and more scalable transactions. Examples include:
     a. **Sidechains:** These are separate blockchains that can communicate with the main blockchain. They can handle transactions more quickly and with lower fees.
     b. **State Channels:** State channels allow users to conduct off-chain transactions, reducing the burden on the main blockchain and improving speed and cost-effectiveness.
     c. **Plasma:** Plasma chains are a framework for creating child chains that can process a high volume of transactions and periodically commit data to the main blockchain.
  2. **Sharding:** A technique where the blockchain network is divided into smaller, interconnected chains or "shards." Each shard can process its transactions independently, significantly increasing the network's throughput.
  3. **Optimistic Rollups:** A type of Layer 2 solution
     a. Allows most transactions to be processed off-chain, with the main blockchain only used for dispute resolution.
     b. Greatly reduces congestion and fees.

Courtesy : Etherscan.io

Dapp Scalability

4. **Blockchain Upgrades: through protocol upgrades.**
   a. For example, Ethereum is transitioning from a proof-of-work (PoW) to a proof-of-stake (PoS) consensus mechanism with Ethereum 2.0, which is expected to increase transaction capacity.
5. **State Management:** is crucial for Dapp scalability.
   a. Minimize on-chain storage and use techniques like Merkle trees to represent complex data structures more efficiently.
6. **Gas Optimization:**
   a. Minimize the use of gas (transaction fees) on the blockchain by optimizing smart contract code and transaction execution. This can reduce costs for users and make the Dapp more accessible.
7. **Caching and Load Balancing:**
   a. Implement caching mechanisms and load balancing to distribute traffic and reduce the load on individual nodes. This can improve response times and overall performance.
8. **Off-Chain Computation:**
   a. Off-load computationally intensive tasks to off-chain servers or services, while still maintaining the security and trustlessness of critical operations on the blockchain.

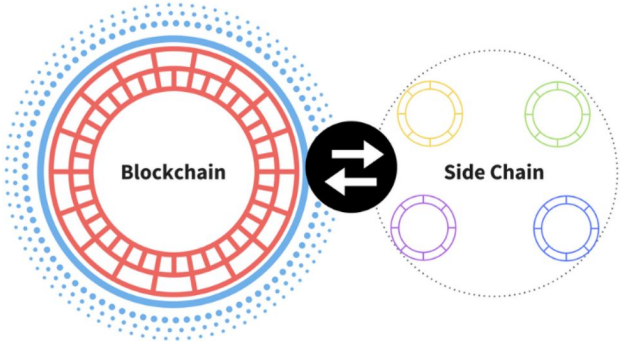
Courtesy : Etherscan.io

Dapp Scalability

9. **Token Design:**
   a. Carefully design the tokenomics of your Dapp. Ensure that the token has utility within the Dapp and mechanisms for handling increased demand without causing congestion on the blockchain.
10. **Governance and Community Engagement:**
   a. Involve the Dapp's community and token holders in governance decisions related to scalability upgrades. This can lead to more consensus-driven and efficient scaling solutions.
11. **Monitoring and Testing:**
   a. Continuously monitor the Dapp's performance and scalability. Conduct stress tests to identify bottlenecks and areas for improvement.
12. **Hybrid Solutions:**
   a. Consider a hybrid approach, combining on-chain and off-chain solutions to strike a balance between security and scalability.

Dapp Scalability

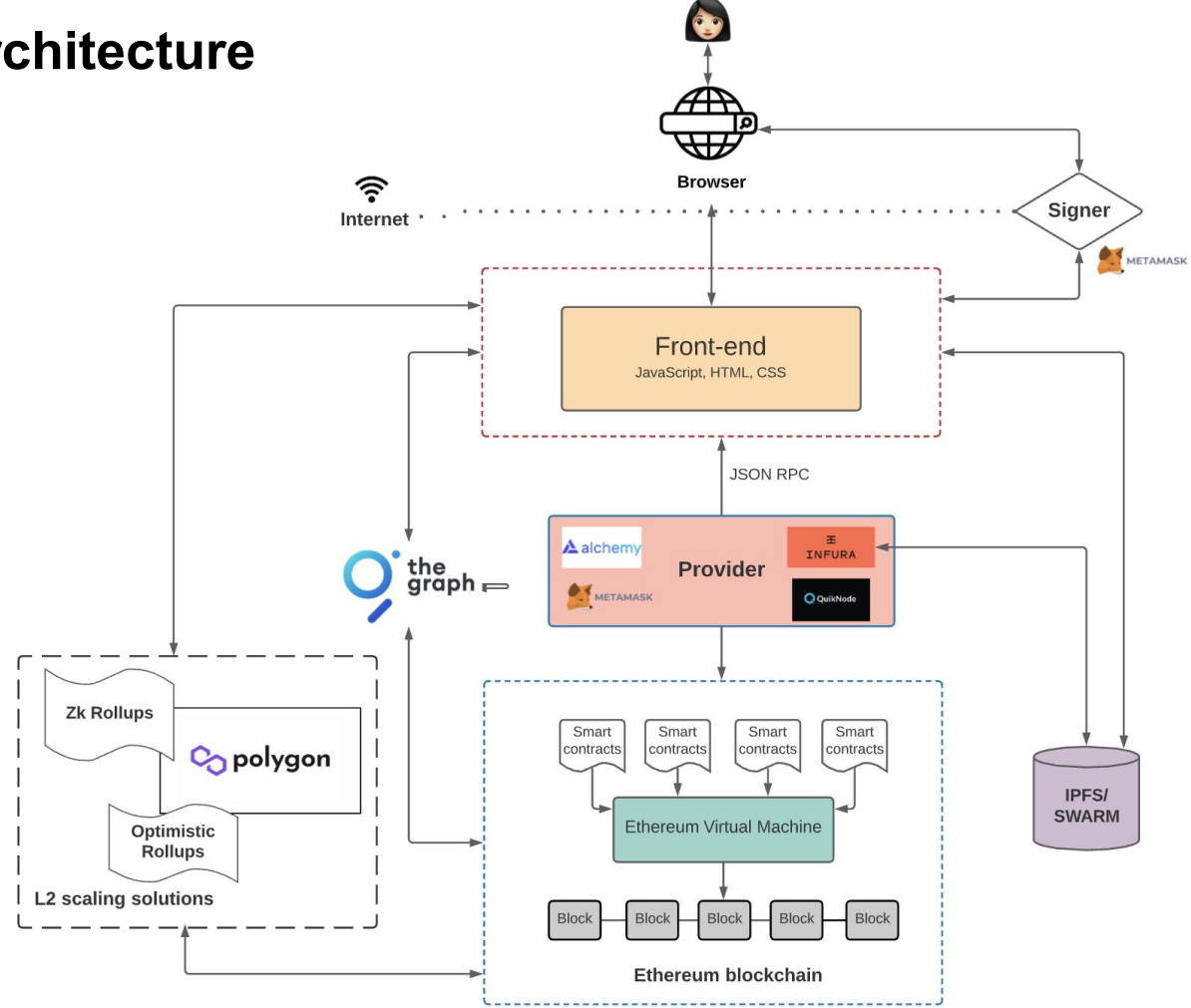
- L2 solutions do transaction execution off-chain, with only the transaction data stored on-chain.
- Helps us scale the blockchain because **we don't have to execute every single transaction on-chain.**
- This also **makes transactions faster and cheaper** — and they can still communicate with the main Ethereum blockchain when necessary.
- Popular Scalability Solution : Polygon, an L2 scaling solution.
  - Polygon has "sidechains" that process and execute transactions.
  - A sidechain is a secondary blockchain that interfaces with the main chain. Every so often, the sidechain submits an aggregation of its recent blocks back to the primary chain.
- Other L2 solutions are Optimistic Rollups and zkRollups.
  - Transactions are **batched off-chain using a "rollup" smart contract and then periodically commit these transactions to the main chain.**



Overall DApp Architecture

Addressing

- Storage,
- Querying
- Scalability



Courtesy : Preeti - DApp

DApp Testing

- Essential to ensure their reliability, security, and functionality.
- **Test Environment**:
  1. Testnet:
     a. Helps to deploy and test DApp without using real assets.
     b. Use testnets to test your smart contracts and transactions.
  2. Private Blockchain:
     a. Helps to set up a private blockchain for more control over the testing environment.
     b. useful for development and early testing stages.

DApp Testing

Types of DApp Testing:

1. Functional Testing:
   - Involves testing the primary functionality of DApp, including interactions with smart contracts, sending transactions, and verifying that the DApp behaves as expected.
2. Security Testing:
   - DApps often deal with valuable assets, so security is critical.
   - Involves identifying vulnerabilities such as reentrancy attacks, front-running, and other potential threats.
3. Performance Testing:
   - To ensure that DApp can handle high user loads,
   - Test for scalability and measure transaction speed and throughput.
4. User Interface (UI) Testing:
   - To ensure that it's user-friendly and responsive.
   - Check for design flaws, responsiveness on various devices, and overall user experience.
5. Interoperability Testing:
   - If the DApp interacts with other DApps or services, then test their interactions to ensure seamless integration.

DApp Testing

- **Testing Tools**:
  1. Truffle:
     - a popular development and testing framework for Ethereum DApps.
     - provides tools for writing and running tests for smart contracts.
  2. Web3.js and Ethers.js:
     - Use these JavaScript libraries to interact with DApp in a testing environment.
     - Helps to simulate interactions with smart contracts.
  3. Hardhat: A development and testing framework that includes built-in features for testing Ethereum-based DApps.
  4. TestRPC/Ganache:
     - Provide a local Ethereum blockchain for testing purposes.
     - Use to deploy smart contracts and run tests.
  5. Mythril and Manticore:
     - Security analysis tools for smart contracts.
     - Helps to identify vulnerabilities in DApp's smart contracts.

DApp Testing

- **Smart Contract Testing:**
  - Unit Testing:
    - Test individual functions and methods within your smart contracts to ensure they work as expected.
  - Integration Testing:
    - Test the **interactions between smart contracts**
    - Ensure they function correctly when used together.
  - Gas Consumption Testing:
    - Verify that smart contracts are not consuming excessive gas, which can lead to higher transaction costs.
- **Security Audits**:
  - Performed by professionals who specialize in blockchain security.
  - Helps identify and mitigate potential vulnerabilities.

DApp Testing

- **User Testing:**
  - Involve real users or a focus group to test DApp's user experience and gather feedback.
  - This can uncover usability issues and help make improvements.
- **Documentation and Reporting:**
  - Maintain comprehensive documentation of the testing process, including test cases, test results, and any issues found.
  - Reporting helps in tracking and fixing problems.
- **Continuous Testing:**
  - Testing is an ongoing process.
  - Regularly test the DApp, especially when updates or changes are made.
  - Automated testing can help streamline this process.
- **Bug Bounty Programs:**
  - Consider running a bug bounty program to incentivize security researchers to find and report vulnerabilities in the DApp.

Introduction to Web3 JS

- JavaScript library that provides a way to interact with the Ethereum blockchain,
- Enables developers to build decentralized applications (DApps) and smart contracts.
- It serves as a **bridge between web application and the Ethereum network**,
- Allows to read data from the blockchain, send transactions, and interact with smart contracts.
- **Web3.js Features:**
  - Ethereum Interaction:
    - Web3.js allows you to interact with the Ethereum blockchain by sending transactions, reading data, and subscribing to events.
  - Smart Contract Interaction:
    - It provides a simple and powerful way to interact with Ethereum smart contracts, enabling you to deploy, call, and manage contracts.
  - Account Management:
    - You can create, manage, and sign transactions with Ethereum accounts using Web3.js.
  - Event Handling:
    - Web3.js can listen for and respond to events emitted by smart contracts,
    - enabling you to build applications that react to on-chain events.

Introduction to Web3 JS

- **Installation:**
  - via npm or include it via a script tag in your HTML.

    npm install web3
- **Initialization**:
  - To use Web3.js, initialize a connection to an Ethereum node.
  - You can connect to a local node (e.g., Ganache) or a remote Ethereum node (e.g., Infura).

    const Web3 = require('web3');

    const web3 = new Web3('https://mainnet.infura.io/v3/YOUR_INFURA_PROJECT_ID');
- **Security Considerations**:
  - When working with Web3.js, ensure that private keys are handled securely

Examples of Web3.js Usage:

- Checking Network Information:

```
web3.eth.net.getId().then(console.log); // Get the network ID (1 for Mainnet)
```

- Sending Ether:

```
web3.eth.sendTransaction({ to: '0xReceiverAddress', value: web3.utils.toWei('0.1', 'ether') });
```

- Smart Contract Interaction:

```
const contract = new web3.eth.Contract(abi, contractAddress);
contract.methods.someFunction().call().then(console.log);
```

- Listening for Events:

```
contract.events.MyEvent()
  .on('data', (event) => console.log('Event data:', event))
  .on('error', console.error);
```

DApp & Web3JS

Connecting to the Blockchain and Smart Contract

1. **Web3 JS - Enables client side App to talk to BLockchain**
2. **Metamask - enables browser to talk to Blockchain**
3. **Ganache - Local DEvelopment Blockchain**
4. **Remix IDE - Smart Contract IDE**

Demo : Build a Dapp in 20 Minutes (Gregory @ DApp University)