Article 1: Develop Smart Contracts Using Rust

For the development of smart contracts in NEAR blockchain, Rust programming language is recommended. Rust's emphasis on performance and safety are the most important reasons for this choice. Further, Rust does not have any garbage collector which makes sure that no indeterministic factor comes into account which is very important for smart-contract development.

For setting up basic Skelton and installing related tools, please refer to the URL, https://www.near-sdk.io/

NEAR ecosystem provides all the decorators for serialization-deserialization, near-bindgen, etc., and the libraries to convert the code written in RUST language into a format compatible for deployment to the NEAR blockchain. For the structure of the contract, please refer to the link, https://www.near-sdk.io/#!

Further, the NEAR guides provide an interface that can be followed to code a smart contract in the right way. There are different method types: public methods, private methods, payable methods, etc. Mutability is a very important part of the rust language. It provides the much-needed safety for contract writing. To understand this further, please refer to the link, https://www.near-sdk.io/contract-interface/public-methods

An important feature of the NEAR blockchain is its asynchronous nature. This can be tricky for developers who are new to NEAR blockchain. The NEAR guides have specifically explained how promises and cross-contact calls work in the NEAR blockchain. Since it is not mandatory that NEAR transactions are executed within one block, proper care needs to be taken care while calling those cross-contract calls which may fail. Basically, if some asynchronous call is failing then the structure of all the accounts should be reset.

Further, the NEAR guides provide the best-practices document, https://www.near-sdk.io/best-practices. It is one of the most important documents to understand the contract structure as well as to follow the best standard to avoid some well-known pitfalls.

Article 2: Architecture and Basic Concepts

The structure of a NEAR node consists roughly of a blockchain layer and a runtime layer. These layers are designed to be independent of each other: In theory, the blockchain layer can support runtimes that process transactions differently, have a different virtual machine (e.g. RISC-V), and has different fees. On the other hand, the runtime is oblivious to where the transactions are coming from; it is not aware whether the blockchain it runs on is sharded, what consensus it uses, and whether it runs as part of a blockchain at all.

The blockchain layer and the runtime layer share the following components and invariants:

Transactions and Receipts

Transactions and receipts are fundamental concepts in the Near Protocol. Transactions represent actions requested by the blockchain user, e.g. sending assets, creating accounts, executing a method, etc. Receipts are an internal structure; a receipt is a message that is used inside a message-passing system.

Transactions are created outside the NEAR Protocol node, by the user who sends them via RPC or network communication. Receipts are created by the runtime from transactions or as the result of processing other receipts.

Blockchain layerS cannot create or process transactions and receipts, it can only manipulate them by passing them around and feeding them to a runtime.

Account-Based System

Similar to Ethereum, Near Protocol is an account-based system. This means that each blockchain user is roughly associated with one or several accounts (there are exceptions though when users share an account and are separated through the access keys).

The runtime is essentially a complex set of rules on what to do with accounts based on the information from the transactions and the receipts. It is therefore deeply aware of the concept of account.

The blockchain layer however is mostly aware of the accounts through the trie (see below) and the validators (see below). Outside these two it does not operate on the accounts directly.

Assume every account belongs to its own shard

Every account in the NEAR protocol belongs to some shard. All the information related to this account also belongs to the same shard. The information includes:

- Balance
- Locked balance (for staking)
- Code of the contract
- Key-value storage of the contract
- All Access Keys

Runtime assumes it's the only information that is available for the contract execution. While other accounts may belong to the same shards, the runtime never uses or provides them during contract execution. It is assumed that every account belongs to its own shard and there is no reason to intentionally try to collocate accounts.

Trie

NEAR Protocol is a stateful blockchain — there is a state associated with each account and the user actions performed through transactions mutate that state. The state then is stored as a trie, and both the blockchain layer and the runtime layer are aware of this technical detail.

The blockchain layer manipulates the trie directly. It partitions the trie between the shards to distribute the load. It synchronizes the trie between the nodes, and eventually, it is responsible for maintaining the consistency of the trie between the nodes through its consensus mechanism and other game-theoretic methods.

The runtime layer is also aware that the storage that it uses to perform the operations on is a trie. In general, it does not have to know this technical detail and in theory, we could have abstracted out the trie as generic key-value storage. However, we allow some trie-specific operations that we expose to the smart contract developers so that they utilize Near Protocol to its maximum efficiency.

Tokens and gas

Even though tokens are a fundamental concept of the blockchain, it is neatly encapsulated inside the runtime layer together with the gas, fees, and rewards.

The only way the blockchain layer is aware of the tokens and the gas is through the computation of the exchange rate and the inflation which is based strictly on the block production mechanics.

Validators

Both the blockchain layer and the runtime layer are aware of a special group of participants who are responsible for maintaining the integrity of the Near Protocol. These participants are associated with the accounts and are rewarded accordingly. The reward part is what the runtime layer is aware of, while everything around the orchestration of the validators is inside the blockchain layer.

Blockchain Layer Concepts

Interestingly, the following concepts are for the blockchain layer only and the runtime layer is not aware of them:

- Sharding — the runtime layer does not know that it is being used in a sharded blockchain, e.g. it does not know that the trie it works on is only a part of the overall blockchain state;
- Blocks or chunks — the runtime does not know that the receipts that it processes constitute a chunk and that the output receipts will be used in other chunks. From the runtime perspective it consumes and outputs batches of transactions and receipts;

- Consensus — the runtime does not know how the consistency of the state is maintained;
- Communication — the runtime doesn't know anything about the current network topology. The receipt has only a receiver_id (a recipient account), but knows nothing about the destination shard, so it's a responsibility of a blockchain layer to route a particular receipt.

Runtime Layer Concepts

- Fees and rewards — fees and rewards are neatly encapsulated in the runtime layer. The blockchain layer, however, has an indirect knowledge of them through the computation of the tokens-to-gas exchange rate and the inflation.

To dive deeper into the tech which underpins NEAR Protocol, check out the White Paper here. If you’re a developer and would like to learn how to build on NEAR, check out the technology section of the Wiki, or head to our technical documentation.

Article 3: Use-Cases

Decentralized Finance (DeFi)

An open alternative to the current financial system that transcends borders.

Learn more about DeFi

Non-fungible Tokens (NFTs)

A way to represent anything as a unique asset and manage its ownership.

Learn more about NFTs

Decentralized Autonomous Orgs (DAOs)

A new way to organize, fund, and empower communities that is democratic by default.

Learn more about DAOs