Updating Contracts

NEAR accounts separate their logic (contract's code) from their state (storage), allowing the code to be changed.

Contract's can be updated in two ways:

1. Through tools
2. such asNEAR CLI
3. ornear-api-js
4. (if you hold the account'sfull access key
5.).
6. Programmatically
7. , by implementing a method thattakes the new code and deploys it
8. .

Updating Through Tools

Simply re-deploy another contract using your preferred tool, for example, usingNEAR CLI :

- near-cli
- near-cli-rs

(optional) If you don't have an account, create one

near create-account --useFaucet

Deploy the contract

near deploy

(optional) If you don't have an account, create one

near account create-account sponsor-by-faucet-service somrnd.testnet autogenerate-new-keypair save-to-keychain network-config testnet create

Deploy the contract

near contract deploy use-file without-init-call network-config testnet sign-with-keychain send

Programmatic Update

A smart contract can also update itself by implementing a method that:

1. Takes the new wasm contract as input
2. Creates a Promise to deploy it on itself
3. Rust

contracts/self-updates/base/src/update.rs loading ... See full example on GitHub

How to Invoke Such Method?

- near-cli
- near-cli-rs
- JavaScript

Load the contract's raw bytes

CONTRACT_BYTES=cat ./path/to/wasm.wasm | base64

Call the update_contract method

near call update_contract "CONTRACT_BYTES" --base64 --accountId --gas 300000000000000

Load the contract's raw bytes

CONTRACT_BYTES=cat ./path/to/wasm.wasm | base64

Call the update_contract method

near contract call-function as-transaction update_contract base64-args "CONTRACT_BYTES" prepaid-gas '300 TeraGas' attached-deposit '0 NEAR' sign-as network-config testnet sign-with-keychain send // Load the contract's raw bytes const code = fs . readFileSync ("./path/to/wasm.wasm") ;

// Call the update_contract method await wallet . callMethod ({ contractId : guestBook ,

method :

"update_contract" ,

args : code ,

gas :

"300000000000000" }) ; DAO Factories This is how DAO factoriesupdate their contracts

Migrating the State

Since the account's logic (smart contract) is separated from the account's state (storage),the account's state persists when re-deploying a contract.

Because of this,adding methods ormodifying existing ones will yieldno problems .

However, deploying a contract thatmodifies or removes structures stored in the state will raise an error:Cannot deserialize the contract state , in which case you can choose to:

1. Use a different account
2. Rollback to the previous contract code
3. Add a method to migrate the contract's state

The Migration Method

If you have no option but to migrate the state, then you need to implement a method that:

1. Reads the current state of the contract
2. Applies different functions to transform it into the new state
3. Returns the new state

DAO Update This is how DAOsupdate themselves

Example: Guest Book Migration

Imagine you have a Guest Book where you store messages, and the users can pay for such messages to be "premium". You keep track of the messages and payments using the following state:

- Rust

contracts/basic-updates/base/src/lib.rs loading ... See full example on GitHub

Update Contract

At some point you realize that you could keep track of thepayments inside of thePostedMessage itself, so you change the contract to:

- Rust

contracts/basic-updates/update/src/lib.rs loading ... See full example on GitHub

Incompatible States

If you deploy the update into an initialized account the contract will fail to deserialize the account's state, because:

1. There is an extrapayments
2. vector saved in the state (from the previous contract)
3. The storedPostedMessages
4. are missing thepayment
5. field (as in the previous contract)

Migrating the State

To fix the problem, you need to implement a method that goes through the old state, removes thepayments vector and adds the information to thePostedMessages :

- Rust

contracts/basic-updates/update/src/migrate.rs loading ... See full example on GitHub Notice thatmigrate is actually aninitialization method thatignores the existing state ([#init(ignore_state)]), thus being able to execute and rewrite the state.

tip You can follow a migration step by step in the official migration example Edit this page Last updatedonFeb 9, 2024 bygagdiez Was this page helpful? Yes No

Previous Deploying and Using Next Locking Accounts