Hello Contract

NEAR accounts can host programs known as smart contracts. Smart contracts canstore data , andexpose methods so other users and contracts interact with them.

In this quickstart tutorial, we will guide you in creating your first smart contract in the NEARtestnet that stores and retrieves a greeting.

Prerequisites

- JavaScript
- Rust

Before starting, make sure you have the following installed:

1. Node.js
2. , to use our scaffolding tool.
3. NEAR CLI
4. , to deploy and interact with the contract.

Easy Install * NEAR-CLI: * Installnear-cli * tools using * npm i -g near-cli Before starting, make sure you have the following installed:

1. NEAR CLI-RS
2. , to deploy and interact with the contract.
3. cargo-near
4. , to easily create testnet accounts.
5. Rust
6. , to create Rust contracts.
7. Node.js
8. (Optional), to install the tools.

Easy Install * NEAR-CLI-RS: * Install bothnear-cli-rs * andcargo-near * tools using * # Using node * npm i -g near-cli-rs cargo-near * # Using macOS, Linux, WSL * curl --proto '=https' --tlsv1.2 -LsSf https://github.com/near/near-cli-rs/releases/latest/download/near-cli-rs-installer.sh | sh * curl --proto '=https' --tlsv1.2 -LsSf https://github.com/near/cargo-near/releases/latest/download/cargo-near-installer.sh | sh Testnet Account There is no need to have atestnet account to follow this tutorial.

However, if you want to create one, you can do so througha wallet , and use it from thenear-cli by invokingnear login .

Creating the Contract

- JavaScript
- Rust

Create a smart contract by running ourcreate-near-app scaffolding tool and following the interactive menu.

npx create-near-app@latest create-near-app in action

The resulting folder structure will change slightly depending on the chosen language. Here is the general structure you can expect to see:

├── sandbox-ts # sanbox testing │ ├── src │ ├── └── main.ava.ts │ ├── ava.config.cjs │ └── package.json ├── src │ └── contract.ts # contract's code ├── package.json # package manager ├── README.md └── tsconfig.json Create a smart contract by running ourcreate-near-app scaffolding tool and following the interactive menu.

cargo near new create-near-app in action

The resulting folder structure will change slightly depending on the chosen language. Here is the general structure you can expect to see:

├── src │ └── lib.rs # contract's code ├── test │ └── test_basics.rs # testing code ├── Cargo.toml # package manager ├── README.md └── rust-toolchain.toml

The Contract

Your new smart contract stores agreeting: string attribute in their state, and exposes two methods to interact with it

(set_greeting ,get_greeting).

- JavaScript
- Rust

contract-ts/src/contract.ts loading ... See full example on GitHub contract-rs/src/lib.rs loading ... See full example on GitHub
There are 3 important things to notice:

1. Theget_greeting
2. method is aview
3. method, meaning it only reads from the contract and can be called for free by anyone.
4. By default, the contract is initialized with thegreeting
5. attribute set to"Hello"
6. .
7. Theset_greeting
8. method is achange
9. method, meaning it modifies the contract's state and requires a user to sign a transaction in order to be executed.

Build and Test

Building and testing the contract is as simple as running two commands.

- JavaScript
- Rust

npm run build npm run test

Expected:

returns the default greeting

changes the greeting

cargo build cargo test

Expected:

Passed gets default greeting

Passed changes the greeting

Failing tests? If the tests are failing, make sure that you are usingnode v16 and thetoolchain v1.69 inrust . You can always use

- nvm use 16
- to switch tonode v16
- rustup default 1.68
- to switch totoolchain v1.69 In the background, these commands are calling the build tools for each language and invoking theSandbox tests from thesandbox-ts/rs directory.

Sandbox Testing the contracts within a Sandbox allows you to understand how the contract will behave once deployed to the network while having total control over the testing environment.

Create a Testnet Account

Now that we know the contract is passing the tests, let's create a testnet account in which to deploy the contract.

While there are different ways tocreate accounts in NEAR, in this quickstart we will use thecargo-near tool to create a new randomnamed account .

- JavaScript
- Rust

Create a new testnet account

Replace with a custom name

near create-account --useFaucet Example Result

> near create-account lovely-event.testnet --useFaucet

Console response

New account "lovely-event.testnet" created successfully. # Response

Create a new testnet account with a random name

cargo-near near create-dev-account use-random-account-id autogenerate-new-keypair save-to-legacy-keychain network-config testnet create

Create a new testnet account

Replace with a custom name

cargo-near near create-dev-account use-specific-account-id lovely-event.testnet autogenerate-new-keypair save-to-keychain network-config testnet create Example Result

If you want to create account with a random name

> cargo-near near create-dev-account use-random-account-id autogenerate-new-keypair save-to-legacy-keychain network-config testnet create

New account "lovely-event.testnet" created successfully. # Response

If you want to create account with a custom name

> cargo-near near create-dev-account use-specific-account-id lovely-event.testnet autogenerate-new-keypair save-to-keychain network-config testnet create

New account "lovely-event.testnet" created successfully. # Response tip Here we are creating a random account since we do not care about the account's name. Remember that you can create a named account through any wallet (i.e.MyNearWallet) and then use it from thenear-cli by invokingnear login .

Deploy the Contract

Having our account created, we can now deploy the contract into it:

- JavaScript
- Rust

near deploy build/release/hello.wasm near contract deploy use-file ./target/wasm32-unknown-unknown/release/contract_rs.wasm without-init-call network-config testnet sign-with-keychain send Congrats ! your contract now lives in the NEAR testnet network.

Interacting with the Contract

To interact with your deployed smart contract, you can call its methods using thenear-cli ornear-cli-rs tools.

Get Greeting

Theget_greeting method is aview method, meaning it only reads from the contract's state, and can thus be called forfree .

- near-cli
- near-cli-rs

  near view get_greeting

"Hello" # Response

  near contract call-function as-read-only get_greeting json-args {} network-config testnet now

"Hello" # Response

Set Greeting

Theset_greeting method is achange method, meaning it modifies the contract's state, and thus requires a user to sign a transaction in order to be executed.

- near-cli
- near-cli-rs

  near call set_greeting '{"greeting": "Hola"}' --accountId

Log: Saving greeting "Hola" # Response In this case we are asking the account that stores the contract to call its own contract's method (--accountId).

  near contract call-function as-transaction set_greeting json-args '{"greeting": "Hola"}' prepaid-gas '100.0 Tgas' attached-deposit '0 NEAR' sign-as network-config testnet sign-with-keychain send

Log: Saving greeting "Hola" # Response In this case, we are asking the account that stores the contract to call its own contract's method (sign-as).

Moving Forward

That's it for the quickstart tutorial. You have now seen a fully functional contract with a minimal user interface and testing.

Go ahead and check otherexamples or proceed straight to theDevelop section to know how to write your own contract.

If you have any questions, do not hesitate to join us orDiscord . We regularly host Office Hours, in which you can join our voice channel and ask questions.

Happy coding! Edit this page Last updatedonMar 6, 2024 bymatiasbenary Was this page helpful? Yes No

Previous What is a Contract? Next Modules, Types & Structs