Getting started

In this tutorial we'll get a testnet account, use NEAR CLI to add a key to our computer's file system, and set up the basic skeleton of a Rust smart contract.

Getting a testnet account

VisitNEAR Wallet for testnet and register for a free account. For the purposes of this tutorial, you may skip the option to add two-factor authentication if you wish.

What just happened? When you created your NEAR testnet account, a private key was created and placed into your browser's local storage. You may inspect this using developer tools and see it.

Creating a new key on your computer

We'll want to use a command-line interface (CLI) tool to deploy a contract, but at the moment the private key only exists in the browser. Next we'lladd a new key to the testnet account and have this stored locally on our computer as a JSON file. (Yes, you can have multiple keys on your NEAR account, which is quite powerful!)

Let's install NEAR CLI. (Please ensure youhave NodeJS

12.)

npm install -g near-cli

You may now run:

near

to see various commands, which are coveredin detail here .

We'll start by "logging in" with this command:

near login

This will bring you to NEAR Wallet again where you can confirm the creation of afull-access key. We'll get to full-access and function-call access keys later, just know that for powerful actions like "deploy" we'll need a full-access key. Follow the instructions from the login command to create a key on your hard drive. This will be located in your operating system's home directory in a folder called.near-credentials .

How was a key added? When you typednear login , NEAR CLI generated a key pair: a private and public key. It kept the private key tucked away in a JSON file and sent the public key as a URL parameter to NEAR Wallet. The URL is long and contains other info instructing NEAR Wallet to "add a full access key" to the account. Our browser's local storage had a key (created when the account was made) that is able to do several things, including adding another key. It took the public key from the URL parameter, used it as an argument, and voilà: the testnet account has an additional key! You can see the keys associated with your account by running the following command, replacingfriend.testnet with your account name:

near keys friend.testnet

Setting up Rust

You may have found theonline Rust Book , which is a great resource for getting started with Rust. However, there are key items that are different when it comes to blockchain development. Namely, that smart contracts aretechnically libraries and not binaries , but for now just know that we won't be using some commands commonly found in the Rust Book.

We won't be using cargo run during smart contract development. Instead, we'll be iterating on our smart contract by building it and running tests.

Install Rust usingrustup

Please see the directions from theRustup site . For OS X or Unix, you may use:

curl --proto '=https' --tlsv1.2 -sSfhttps://sh.rustup.rs | sh

(Taken from theRust installation guide)

Add Wasm toolchain

Smart contracts compile to WebAssembly (Wasm) so we'll add the toolchain for Rust.

rustup target add wasm32-unknown-unknown

(More info ontargets and this toolchain here .)

Start writing Rust

There's a basic repository that's helpful to clone or downloadlocated here .

The first thing we'll do is modify the manifest file atCargo.toml :

[package] - name = "rust-template" + name = "my-crossword" version = "0.1.0" - authors = ["Near Inc hello@near.org"] + authors = ["NEAR Friend friend@example.com"] edition = "2018" By changing thename here, we'll be changing the compiled Wasm file's name after running the build script. (build.sh for OS X and Linux,build.bat for Windows.) After running the build script, we can expect to find our compiled Wasm smart contract inres/my_crossword.wasm .

Now let's look at our main file, insrc/lib.rs :

contract/src/lib.rs loading ... See full example on GitHub As you can see, this is a stub that's ready to be filled in. Let's pause and point out a few items:

- Note thenear_bindgen macro
- is above the struct and the impl
- Here the main struct is calledContract
- , while in other examples it might beCounter
- or something else. This is purely stylistic, but you may learn more from the link in the previous bullet.
- You may notice the word "Borsh" and wonder what that means. This is a binary serializer. Eventually, we'll want to save data as ones and zeroes to validators' hard drives, and do it efficiently. We use Borsh for this, as is explainedon this website
- .

Next, let's modify this contract little by little…Edit this page Last updatedonJan 19, 2024 byDamián Parrino Was this page helpful? Yes No

Previous Crossword Game Overview Next Add basic code, create a subaccount, and call methods