Factory

A factory is a smart contract that stores a compiled contract on itself, and automatizes deploying it into sub-accounts.

We have two factory examples:

1. Token Factory
2. : A factory that createsfungible tokens
3. contracts.
4. A Generic Factory
5. : A factory that createsdonation contracts
6. , but allows to change the contract it deploys.

info In this page we will focus on the Donation factory, to learn more about the token factory visit its repository.

Generic Factory

TheGeneric Factory presents a contract factory that:

1. Creates sub-accounts of itself and deploys its contract on them (create_factory_subaccount_and_deploy
2.).
3. Can change the stored contract using theupdate_stored_contract
4. method.
5. Rust
6. deploy.rs
7. update.rs

contract/src/deploy.rs loading ... See full example on GitHub contract/src/manager.rs loading ... See full example on GitHub

Quickstart

1. Make sure you have installedrust
2. .
3. Install theNEAR CLI

Build and Deploy the Factory

You can automatically compile and deploy the contract in the NEAR testnet by running:

./deploy.sh Once finished, check theneardev/dev-account file to find the address in which the contract was deployed:

cat ./neardev/dev-account

e.g. dev-1659899566943-21539992274727

Deploy the Stored Contract Into a Sub-Account

create_factory_subaccount_and_deploy will create a sub-account of the factory and deploy the stored contract on it.

near call create_factory_subaccount_and_deploy '{ "name": "sub", "beneficiary": ""}' --deposit 1.24 --accountId --gas 300000000000000 This will create thesub. , which will have adonation contract deployed on it:

near view sub. get_beneficiary

expected response is:

Update the Stored Contract

update_stored_contract enables to change the compiled contract that the factory stores.

The method is interesting because it has no declared parameters, and yet it takes an input: the new contract to store as a stream of bytes.

To use it, we need to transform the contract we want to store into itsbase64 representation, and pass the result as input to the method:

Use near-cli to update stored contract

export BYTES=cat ./src/to/new-contract/contract.wasm | base64 near call update_stored_contract "BYTES" --base64 --accountId --gas 30000000000000 This works because the arguments of a call can be either aJSON object or aString Buffer

Factories - Concepts & Limitations

Factories are an interesting concept, here we further explain some of their implementation aspects, as well as their limitations.

Automatically Creating Accounts

NEAR accounts can only create sub-accounts of itself, therefore, thefactory can only create and deploy contracts on its own sub-accounts.

This means that the factory:

1. Can
2. createsub.factory.testnet
3. and deploy a contract on it.
4. Cannot
5. create sub-accounts of thepredecessor
6. .
7. Can
8. create new accounts (e.g.account.testnet
9.), butcannot
10. deploy contracts on them.

It is important to remember that, whilefactory.testnet can createsub.factory.testnet , it has no control over it after its creation.

The Update Method

Theupdate_stored_contracts has a very short implementation:

[private]

pub

fn

update_stored_contract (& mut

self)

{ self . code =

env :: input ( ) . expect ("Error: No input") . to_vec ( ) ; } On first sight it looks like the method takes no input parameters, but we can see that its only line of code reads fromenv::input() . What is happening here is thatupdate_stored_contract bypasses the step ofdeserializing the input .

You could implementupdate_stored_contract(&mut self, new_code: Vec) , which takes the compiled code to store as aVec, but that would trigger the contract to:

1. Deserialize thenew_code
2. variable from the input.
3. Sanitize it, making sure it is correctly built.

When dealing with big streams of input data (as is the compiledwasm file to be stored), this process of deserializing/checking the input ends upconsuming the whole GAS for the transaction. Edit this page Last updatedonJan 31, 2024 bygagdiez Was this page helpful? Yes No

Previous Coin Flip Next Complex Cross Contract Call