Rapid Prototyping

When you change the interface of a contract and re-deploy it, you may see this error:

Cannot deserialize the contract state.

Why does this happen?

When your contract is executed, the NEAR Runtime reads the serialized state from disk and attempts to load it using current contract code. When your code changes but the serialized state stays the same, it can't figure out how to do this.

How can you avoid such errors?

When you're still in the Research & Development phase, building a prototype and deploying it locally or ontestnet , you can just delete all previous contract state when you make a breaking change. See below for a couple ways to do this.

When you're ready to deploy a more stable contract, there are a couple ofproduction strategies that will help you update the contract state without deleting it all. And once your contract graduates from "trusted mode" (when maintainers control aFull Access key) to community-governed mode (no more Full Access keys), you can set up your contract toupgrade itself .

Rapid Prototyping: Delete Everything All The Time

There are two ways to delete all account state:

1. Deploying on a new account each time
2. Deleting & recreating contract account

For both cases, let's consider the following example.

Therust-status-message example contract has the following structure:

- Rust

src/lib.rs loading ... See full example on GitHub Let's say you deploy this contract to testnet, then call it with:

- near-cli
- near-cli-rs

near call [contract] set_status '{"message": "lol"}' --accountId you.testnet near view [contract] get_status '{"account_id": "you.testnet"}' near contract call-function as-transaction [contract] set_status json-args '{"message": "lol"}' prepaid-gas '30 TeraGas' attached-deposit '0 NEAR' sign-as you.testnet network-config testnet sign-with-keychain send

near contract call-function as-read-only [contract] get_status text-args '{"account_id": "you.testnet"}' network-config testnet now This will return the message that you set with the call toset_status , in this case"lol" .

At this point the contract is deployed and has some state.

Now let's say you change the contract to store two kinds of data for each account:

[near_bindgen]

[derive(BorshDeserialize, BorshSerialize)]

pub

struct

StatusMessage

{ taglines :

LookupMap < AccountId ,

String

, bios :

LookupMap < AccountId ,

String

, }

impl

Default

for

StatusMessage

{ fn

default ( )

->

Self

{ Self

{ taglines :

LookupMap :: new (b"r") , bios :

LookupMap :: new (b"b") , } } }

[near_bindgen]

impl

StatusMessage

{ pub

fn

set_tagline (& mut

self , message :

String)

{ let account_id =

env :: signer_account_id ( ) ; self . taglines . insert (& account_id ,

& message) ; }

pub

fn

get_tagline (& self , account_id :

AccountId)

->

Option < String

{ return

self . taglines . get (& account_id) ; }

pub

fn

set_bio (& mut

self , message :

String)

{ let account_id =

env :: signer_account_id ( ) ; self . bios . insert (& account_id ,

& message) ; }

pub

fn

get_bio (& self , account_id :

AccountId)

->

Option < String

{ return

self . bios . get (& account_id) ; } } You build & deploy the contract again, thinking that maybe because the newtaglines LookupMap has the same prefix as the oldrecords LookupMap (the prefix isr , set byLookupMap::new(b"r".to_vec())), the tagline foryou.testnet should be"lol" . But when younear view the contract, you get the "Cannot deserialize" message. What to do?

1. Deploying on a new account each time

When first getting started with a new project, the fastest way to deploy a contract is creating an account anddeploying the contract into it usingNEAR CLI .

- near-cli
- near-cli-rs

near create-account --useFaucet near deploy ./path/to/compiled.wasm near account create-account sponsor-by-faucet-service .testnet autogenerate-new-keypair save-to-keychain network-config testnet create

near contract deploy .testnet use-file without-init-call network-config testnet sign-with-keychain This does a few things:

1. Creates a new testnet account pre-funded with 10N from the faucet
2. Stores the private key for this account in the~/.near-credentials
3. folder
4. Deploys your contract code to this account

2. Deleting & Recreating Contract Account

Another option to start from scratch is to delete the account and recreate it.

- near-cli
- near-cli-rs

Delete sub-account near delete app-name.you.testnet you.testnet Delete sub-account near account delete-account app-name.you.testnet beneficiary you.testnet network-config testnet sign-with-keychain send This sends all funds still on theapp-name.you.testnet account toyou.testnet and deletes the contract that had been deployed to it, including all contract state.

Now you create the sub-account and deploy to it again using the commands above, and it will have empty state like it did the first time you deployed it. Edit this page Last updatedonFeb 9, 2024 bygagdiez Was this page helpful? Yes No

Previous Basic Instructions Next Post Processing Tools