

Integration tests

There is a set of integration tests which cover main Neutron features. If you develop a smart contract for Neutron you can add some tests into this set to make sure everything works as expected.

- [Installation](#)
- [Running the tests](#)
- [Environment variables](#)
- [Creating your own tests](#)

Installation

- `git clone git@github.com:neutron-org/neutron-integration-tests.git`
- `git clone -b v2.0.3 git@github.com:neutron-org/neutron.git`
- `git clone git@github.com:neutron-org/neutron-query-relayer.git`
- `git clone -b v14.0.0 git@github.com:cosmos/gaia.git`
- `cd neutron-integration-tests`
- `*make -C setup build-all`
- `yarn`
- Make sure you have docker installed and docker daemon running

*Only for the first run, to build hermes ibc relayer and gaiad containers

Running the tests

`yarn test # all tests` `yarn test:simple # basic tests` `yarn test:interchaintx # interchain txs test` `yarn test:interchain_tx_query_plain # interchain tx query test` `yarn test:interchain_tx_query_resubmit # interchain tx query test #2` `yarn test:interchain_kv_query # interchain kv query test`

Environment variables you can redefine

APP_DIR - applications directory where Neutron, Gaia and Neutron query relayer are located
NEUTRON_DENOM - neutron network denom
COSMOS_DENOM - gaia (cosmoshub) network denom
IBC_ATOM_DENOM — denom of a native token which is used as a fake IBC transferred ATOM
IBC_USDC_DENOM — denom of a native token which is used as a fake IBC transferred USDC
CONTRACTS_PATH - path to contracts that will be used in tests
NEUTRON_ADDRESS_PREFIX - address prefix for neutron controller network
COSMOS_ADDRESS_PREFIX - address prefix for gaia (cosmoshub) host network
NODE1_URL - url to the first node
NODE1_WS_URL - url to websocket of the first node
NODE2_URL - url to the second node
NODE2_WS_URL - url to websocket of the second node
BLOCKS_COUNT_BEFORE_START - how many blocks we wait before start first test
NO_DOCKER - do not start cosmopark for tests
NO_REBUILD - skip containers rebuilding

Config

src/config.json

Creating your own tests

Creating your contract

To create a new contract you can refer to [Neutron Cosmwasm SDK Repo](#) to have an idea how to use Neutron SDK.

Updating artifacts

You'll need to update artifacts in ./contracts folder in case you have created a new contract. Place your contract(s) into ./contracts/artifacts folder. Let's say you have the contract with namemy_contract.wasm

Your first test

Create a file named new_one.test.ts in ./src/testcases/parallel with following code:

```
import
```

```
{
```

```
TestStateLocalCosmosTestNet
```

```

}

from
'./common_localcosmosnet' ; import
{ NEUTRON_DENOM , CosmosWrapper , WalletWrapper , }

from
'../../helpers/cosmos' ;

describe ( 'Neutron / My test' ,
( )
=>
{ let
testState :
TestStateLocalCosmosTestNet ; let
neutronChain :
CosmosWrapper ; let
neutronAccount :
WalletWrapper ; let
codeId : string ; let
contractAddress : string ;
beforeAll ( async
( )
=>
{ testState =
new
TestStateLocalCosmosTestNet ( ) ; await testState . init ( ) ;

```

neutronChain

```

new
CosmosWrapper ( testState . sdk1 , testState . blockWaiter1 , NEUTRON_DENOM , ) ; neutronAccount =
new
WalletWrapper ( neutronChain , testState . wallets . qaNeutron . genQaWal1 , ) ; } ) ;
test ( 'store contract' ,
async
( )
=>
{ codeId =
await neutronAccount . storeWasm ( 'my_contract.wasm' ) ; expect ( codeId ) . toBeGreaterThan ( 0 ) ; } ) ; test ( 'instantiate'
,
async

```

()

=>

```
{ const res =
```

```
await neutronAccount . instantiateContract ( codeId , '{}' , 'my_contract' , ) ; contractAddress = res [ 0 ] . _contract_address ;  
expect ( contractAddress ) . toStartWith ( 'neutron' ) ; } ) ; test ( 'execute contract' ,
```

```
async
```

()

=>

```
{ const res =
```

```
await neutronAccount . executeContract ( contractAddress , JSON . stringify ( { my_method :
```

```
{ //we assume you have this method in the contract foo :
```

```
'bar' , } , } ) , ) ; expect ( res . code ) . toEqual ( 0 ) ; } ) ; } ) ; Warning: Usesrc/testcases/run_in_band folder for your test if it  
cannot be run in parallel! This is usually the case if test mutates some global chain state that other tests use directly or  
indirectly. Then updatepackage.json in the root folder. Like this
```

```
... "test:new_one": "jest --runInBand -b src/testcases/parallel/new_one", ... Now you can run your test:
```

```
yarn test:new_one Previous CosmWasm + ICQ Next Overview
```