

# CosmWasm + WasmKit

This guide will explore an overview of lifecycle of a simple CosmWasm contract using WasmKit. You can check out more detailed WasmKit usage [here](#).

WasmKit is a development framework specifically designed for building CosmWasm contracts. The primary goal of the project is to simplify, streamline, and enhance the process of developing CosmWasm contracts.

## WasmKit's approach to abstraction

WasmKit's approach separates the running instances (networks, contracts, accounts) from the core logic (deploy steps, testing logic) to make contract interaction possible with any CosmWasm enabled chain with any account with just a slight change in `wasmkit.config.js` if need be.

WasmKit configuration file `wasmkit.config.js` maps blockchain networks (neutron mainnet, neutron testnet or localnet), accounts (contract admin account, contract deploy account, testing account) and even contracts (their `.wasm` file, schema files) with just their names and only use given names while writing interaction script or testing logic. This is powerful mechanism as it removes the need to store and keep track of data values for these 3 entities, for example, there is no need to store `codeId` after contract deployment, `contractAddress` after instantiation or to store RPC URL or `chainId` for mainnet, testnets.

## Contract development cycle

1. Setup contracts repo
2. : A project repository to hold rust contracts, deployment scripts, tests and configuration files in one place. Can possibly have docs and/or front-end source too.
3. Implement contract logic, compile and debug
4. : Define contract's storage, messages and contract methods for given messages. Compile contracts to generate `.wasm` files as contract binary and contract messages `schema.json` files.
5. Write unit, integration tests
6. : Unit tests can be written in contract's rust source and chain interaction integration tests in `typescript`.
7. Compress `.wasm` binary, deploy to network, instantiate contract
8. : Deploy to network by using a simple deploy script written in `typescript`. Contract instantiation can also be done within the `typescript` script.
9. Query or Execute contract
10. : Contract queries and execute calls can be simply implemented in `typescript` using `thetypescript_schema/`
11. clients generated.
12. Write front-end and scripts to interact with the contract
13. : The `typescript_schema/`
14. clients can be used to write front-end interaction with the contracts.

## Prerequisites

The minimum packages/requirements are as follows:

- Nodev16+
- Yarnv1.22+
- or Npmv6.0+
- Connection to a Neutron node

## Quick start

### Setup rust environment

WasmKit requires a Rust environment installed on local machine to work properly. A working instance of `rustup` and `rustc` should be enough to get started.

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

```
sh rustup --version rustc --version info
```

 Alternatively, you can install rust by following their [documentation](#).

### Installation

```
yarn global add @kubiklabs/wasmkit or
```

npm

install -g @kubiklabs/wasmkit Installation from source:

git clone https://github.com/kubiklabs/wasmkit.git cd wasmkit &&

yarn

install

&&

yarn build cd packages/wasmkit &&

yarn

link chmod +x HOME /.yarn/bin/wasmkit

## Usage

note Depending on your system and how you install, the wasmkit commands could work with npx prefix or without. Try both to be sure on how it works for you.

### Initialize a project

npx wasmkit init < project-name

This will create a directory inside the current directory with boiler-plate code.

- Thecontracts/
- directory has all the rust files for the contract logic.
- scripts/
- directory contains.ts
- scripts that users can write according to the use case, a sample script has been added to give some understanding of how a user script should look like.
- test/
- directory contains.ts
- scripts to run integration tests for the deployed contracts.

### Compile the project

Begin by typing:

cd

< project-name

npx wasmkit --help note Make sure you're inside project's root directory when running compile command. The compile command generates compiled.wasm files in artifacts/contracts/ , schema.json files in artifacts/schema/ and typescript clients in artifacts/typescript\_schema/ .

- Compile all the contracts in the project:

npx wasmkit compile \* Compile only one contracts or a subset of all contracts in the project:

npx wasmkit compile < contract-source-dir

npx wasmkit compile contracts/first\_contract/

## compile only contract with name 'first\_contract'

npx wasmkit compile contracts/first\_contract/ contracts/second\_contract/

## compile only these 2 contracts

- Skip schema generation while compiling:

npx wasmkit compile --skip-schema

## Listing Tasks

To see the tasks (commands) that are available, run wasmkit in project's directory.

`npm run wasmkit` This will display the list of built-in tasks. This is your starting point to find out what tasks are available to run.

## Cleanup Artifacts

- Clear artifacts data:

`npm run wasmkit clean` This will remove the artifacts/ directory completely.

- Clean artifacts for only one contract:

`npm run wasmkit clean < contract-name`

## Start local network

A local instance of neutron network can be created if a following config is specified in `wasmlite.config.js` file:

localnetworks :

```
{
// specify localnetwork docker image, ports and environment variables neutron :
{ docker_image :
"neutron-node" , rpc_port :
26657 , rest_port :
1317 , flags :
```

`[ "RUN_BACKGROUND=0" ] , } , }` , You can see there's a `neutron-node` used as a docker image. For such a setup you'll need to [Build a Neutron node image](#) . Once done, a neutron localnet using above config can be started by doing:

`npm run wasmkit localnet-start neutron` The RPC URL for this localnetwork will be `http://localhost:26657` and REST URL will be `http://localhost:1317` . This can be verified by doing:

`npm run wasmkit node-info --network localnet`

## Running user scripts

User scripts are a way to define the flow of interacting with contracts on some network in the form of a script. These scripts can be used to deploy a contract, query or execute a contract. A sample script `scripts/sample-script.ts` is available in the boilerplate.

`npm run wasmkit run scripts/ < script-name`

`wasmlite.config.js` has list of networks defined and comes by default with 3 networks defined, namely, `testnet` , `localnet` and `mainnet` . To specify which network to the script (or test) for, simply use the `--network` flag similar as given below:

`npm run wasmkit run scripts/ < script-name`

`--network localnet`

## Run tests

To run all the tests in `test/` directory:

`npm run wasmkit test` To run a specific test or a subset of tests, just pass the path of test files:

`npm run wasmkit test test/ < test-name`

For example:

`npm run wasmkit test test/sample-test.ts test/another-test.ts`

# Initiate wasmkit playground

Wasmkit playground is an auto-generated minimal front-end for interacting with deployed contracts. Since wasmkit keeps track of the deployed contract addresses on multiple networks, it can generate the playground front-end using these addresses and the typescript schema generated from the contracts. This playground can be useful when quickly interacting with contracts either for debugging purposes or it can be shared as part of contract documentation for anyone to try out contracts interaction without having to setup environment or installing any cli.

To initiate wasmkit playground, make sure you are in project directory and contracts are compiled and instantiated using compile and run commands of wasmkit respectively.

```
cd
```

```
< project-name
```

```
    npx wasmkit playground This command will clone a react application to interact with deployed contracts. User
    can also modify its theme and logo using config file.
```

Locally run the playground front-end using:

```
cd playground/ yarn start
```

## Configuration guide

If you examine thewasmkit.config.js file, you will find testnet/localnet accounts and various networks, allowing users to customize fee, account, and network information according to their requirements.

```
// You can specify your own testnet/mainnet/localnet accounts // testnet account can be funded using faucets const
testnet_accounts =
```

```
[ { name :
```

```
'account_0' , address :
```

```
'neutron1e...dp5' , mnemonic :
```

```
'omit ... text' } , { name :
```

```
'account_1' , address :
```

```
'neutron1n...z8h' , mnemonic :
```

```
'student ... bicycle' } ] ;
```

```
const localnet_accounts =
```

```
[ { name :
```

```
'account_0' , address :
```

```
'neutron3f...sf2' , mnemonic :
```

```
'clip ... choose' } ] ;
```

```
const mainnet_accounts =
```

```
[ { name :
```

```
'account_0' , address :
```

```
'neutron3f...sf2' , mnemonic :
```

```
'clip ... choose' } ] ;
```

```
// You can specify other networks similarly, // just need to know RPC URL and chainID // custom fee can also be added here
// for detailed example: https://github.com/kubiklabs/wasmkit/blob/master/packages/wasmkit/sample-project/wasmkit.config.js const networks =
```

```
{ localnet :
```

```
{ endpoint :
```

```

'http://localhost:26657/' , chainId :

'test-1' , accounts : localnet_accounts , } , testnet :

{ endpoint :

'https://rpc-palvus.pion-1.ntrn.tech/' , chainId :

'pion-1' , accounts : testnet_accounts , } , mainnet :

{ endpoint :

'https://rpc-kralum.neutron-1.neutron.org' , chainId :

'neutron-1' , accounts : mainnet_accounts , } , } ;

module . exports

=

{ networks :

{ default : networks . testnet , testnet : networks . testnet , localnet : networks . localnet , } , localnetworks :

{

// specify localnetwork docker image, ports and environment variables neutron :

{ docker_image :

"neutron-node" , rpc_port :

26657 , rest_port :

1317 , flags :

[ "RUN_BACKGROUND=0" ] , } , } , mocha :

{ timeout :

60000 } , rust :

{ version :

"1.63.0" , } } ; Previous CosmWasm + Remix IDE Next CosmWasm + ICA

```