# Downloading and Compiling a Contract

In this section, we will download the code for a sample contract and compile it into a wasm binary executable.

If you haven't already, please review theenvironment setup instructions first and either configure the Node.js REPL or the wasmd Go CLI before you proceed.

## Compiling and Testing the Contract Code

Let's download the repository in which we keepcw-contracts and compile the existing code for a simple name service contract that mimics a name service marketplace.

First, clone the repo and try to build the wasm bundle:

# Download the repository

git clone https://github.com/InterWasm/cw-contracts cd cw-contracts git checkout main cd contracts/nameservice

# compile the wasm contract with stable toolchain

rustup default stable cargo wasm The compilation should output the filetarget/wasm32-unknown-unknown/release/cw_nameservice.wasm . With a quickls -lh you can see that the file size is around 1.8 MB. This is a release build, but not stripped of all the unneeded code. To produce a much smaller version, you can run the following command which tells the compiler to strip the unused parts of the code out:

# RUSTFLAGS

'-C link-arg=-s' cargo wasm This produces a file that is about 165kB in size. We either use the command above or utilize another Rust optimizer, the use of which will be covered in theoptimized compilation section , to produce the smallest final wasm binary before it is uploaded to the blockchain.

## Unit Tests

Let's try running the unit tests:

# RUST_BACKTRACE

1 cargo unit-test After some compilation steps, you should see:

running 15 tests test tests::tests::proper_init_no_fees ... ok test tests::tests::fails_on_register_insufficient_fees ... ok test coin_helpers::test::assert_sent_sufficient_coin_works ... ok test tests::tests::fails_on_register_wrong_fee_denom ... ok test tests::tests::fails_on_register_already_taken_name ... ok test tests::tests::fails_on_transfer_from_nonowner ... ok test tests::tests::fails_on_transfer_insufficient_fees ... ok test tests::tests::fails_on_transfer_non_existent ... ok test tests::tests::proper_init_with_fees ... ok test tests::tests::register_available_name_and_query_works ... ok test tests::tests::register_available_name_fails_with_invalid_name ... ok test tests::tests::returns_empty_on_query_unregistered_name ... ok test tests::tests::register_available_name_and_query_works_with_fees ... ok test tests::tests::transfer_works ... ok test tests::tests::transfer_works_with_fees ... ok

test result: ok. 15 passed; 0 failed; 0 ignored; 0 measured; 0 filtered out; RUST_BACKTRACE=1 will provide you with full stack traces on any error, which is super useful. This only works for unit tests (which test native rust code, not the compiled wasm). Also, if you want to know wherecargo wasm andcargo unit-test come from, they are just aliases defined in the file.cargo/config located in the project directory. Take a look at the file contents to understand the cargo flags better.

## Optimized Compilation

To reduce gas costs, the binary size should be as small as possible. This will result in a less costly deployment, and lower fees on every interaction. Luckily, there is tooling to help with this. You canoptimize production code using therust-optimizer .rust-optimizer produces reproducible builds of CosmWasm smart contracts. This means third parties can verify that the contract is actually the claimed code.

info You will needDocker installed in order to runrust-optimizer . Navigate to the project root and run the following command:

docker run --rm -v " ( pwd ) " :/code \ --mount type = volume,source = "(basename " ( pwd ) " ) _cache",target = /code/target \ --mount type = volume,source = registry_cache,target = /usr/local/cargo/registry \ cosmwasm/rust-optimizer:0.12.11 On Windows, you can use the following command instead

docker run --rm -v {pwd}:/code --mount type=volume,source="("(Split-Path -Path pwd -Leaf)")_cache",target=/code/target --mount type=volume,source=registry_cache,target=/usr/local/cargo/registry ` cosmwasm/rust-optimizer:0.12.11 The binary will be under the folderartifacts and its size will be138 kB . Previous Setting up Environment Next Deployment and Interaction