Compile and Deploy

In this example, we will compile, upload, and instantiate our first smart contract using SecretCLI and LocalSecret. Now that you've set up your LocalSecret development environment, we are going to learn how to compile, upload, and instantiate a smart contract using SecretCLI in your LocalSecret testnet environment.

For a step-by-step Secret Network SecretCLI video tutorial,follow along here . Otherwise, continue reading! We will be working with a basiccounter contract , which allows users to increment a counter variable by 1 and also reset the counter. If you've never worked with smart contracts written in Rust before that is perfectly fine. By the end of this tutorial you will know how to upload and instantiate a Secret Network smart contract in your terminal using SecretCLI.

Generate your new counter contract

The first thing we need to do is clone the counter contract from theSecret Network github repo . Secret Network developed this counter contract template so that developers have a simple structure to work with when developing new smart contracts, but we're going to use the contract exactly as it is for learning purposes.

Go to the folder in which you want to save your counter smart contract and run:

```

Copy cargo generate --git https://github.com/scrtlabs/secret-template.git --name my-counter-contract

```

When you run the above code, it will name your contract folder directory "my-counter-contract". But you can change the name by altering the text that follows the--name flag. Start by opening themy-counter-contract project folder in your text editor. If you navigate tomy-counter-contract/src you will seecontract.rs, msg.rs, lib.rs, and state.rs —these are the files that make up our counter smart contract. If you've never worked with a Rust smart contract before, perhaps take some time to familiarize yourself with the code, although in this tutorial we will not be going into detail discussing the contract logic.

Next, configure SecretCLI to work with LocalSecret by running the following code in your text editor terminal (this is assuming you already have an instance of LocalSecret running in docker, which we learned in the previous section)

```

Copy secretcliconfignodehttp://localhost:26657 secretcliconfigchain-idsecretdev-1 secretcliconfigkeyring-backendtest secretcliconfigoutputjson

```

Did you know? By default, SecretCLI tries to connect to a locally running network

Compile the Code

Since we are not making any changes to the contract code, we are going to compile it exactly as it is. To compile the code, runmake build in your terminal. This will take our Rust code and build a Web Assembly file that we can deploy to Secret Network. Basically, it just takes the smart contract that we've written and translates the code into a language that the blockchain can understand.

Linux/WSL/MacOS Windows Secret IDE ```

Copy make build

Copy env:RUSTFLAGS='-C link-arg=-s' cargo build --release --target wasm32-unknown-unknown cp ./target/wasm32-unknown-unknown/release/*.wasm ./contract.wasm

``` Runmake build from the terminal, or just GUI it up - This will create acontract.wasm andcontract.wasm.gz file in the root directory.

While we could upload this contract wasm file to the blockchain exactly as it is, instead we are going to follow best practices andoptimize the wasm file. This just means we are going to reduce the size of the file so that it costs less gas to upload, which is critical when you eventually upload contracts to mainnet. Make sure you have an instance of LocalSecret running and then run the following code:

Optimize compiled wasm

```

Copy dockerrun--rm-v"(pwd)":/contract\ --mounttype=volume,source="(basename"(pwd)")_cache",target=/code/target\ --mounttype=volume,source=registry_cache,target=/usr/local/cargo/registry\ enigmampc/secret-contract-optimizer

```

You should now have an optimizedcontract.wasm.gz file in your root directory, which is ready to be uploaded to the blockchain! Also note that the optimizer should have removed thecontract.wasm file from your root directory

Storing the Contract

Now that we have a working contract and an optimized wasm file, we can upload it to the blockchain and see it in action. This is calledstoring the contract code . We are using a local testnet environment, but the same commands apply no matter which network you want to use - local, public testnet, or mainnet.

In order to store the contract code, we first must create a wallet address in order to pay for transactions such as uploading and executing the contract.

Creating a Wallet

To interact with the blockchain, we first need to initialize a wallet. From the terminal run:

secretcli keys add myWallet

You can name your wallet whatever you'd like, for this tutorial I chose to name my wallet "myWallet" You should now have access to a wallet with a unique name, address, and mnemonic, which you can use to upload the contract to the blockchain:

The wallet currently has zero funds, which you query by running this secretcli command (be sure to use your wallet address in place of mine)

```

Copy secretcli query bank balances "secret16u7w28vp68qmldffuc89am4f02045zlfsjht90"

```

To fund the wallet so that it can execute transactions, run:

```

Copy curl http://localhost:5000/faucet?address=secret16u7w28vp68qmldffuc89am4f02045zlfsjht90

```

Your wallet address should now have 1000000000 uscrt

Upload the contract

Finally, we can upload our contract:

```

Copy secretcli tx compute store contract.wasm.gz --gas 5000000 --from--chain-id secretdev-1

```

--from refers to which account (or wallet) is sending the transaction. Update to your wallet address.

--gas 5000000 refers to the cost of the transaction we are sending. Gas is the unit of cost which we measure how expensive a transaction is.

--chain-id refers to which chain we are uploading to, which in this case is LocalSecret! To verify whether storing the code has been successful, we can use SecretCLI to query the chain:

```

Copy secretcli query compute list-code

```

Which should give an output similar to the following:

```

Copy [{ "code_id":1, "creator":"secret16u7w28vp68qmldffuc89am4f02045zlfsjht90",
"code_hash":"2658699cea6112052a342d16fd57ac4411cdf1c05cdac3deceba8de0f6ce026d" }]

```
```

Instantiating the Contract

In the previous step we stored the contract code on the blockchain. To actually use it, we need to instantiate a new instance of it.

```

Copy secretcli tx compute instantiate 1 '{"count": 1}' --from--label counterContract -y

```

- instantiate 1
- is the code_id that you created in the previous section
- {"count": 1}
- **** is the instantiation message. Here we instantiate a starting count of 1, but you can make it anyi32 you want
- --from
- **** is your wallet address
- --label
- is a mandatory field that gives the contract a unique meaningful identifier * Let's check that the instantiate command worked:

```

Copy secretcliquerycomputelist-contract-by-code1

```

Now we will see the address of our deployed contract

```

Copy [ { "contract_address":"secret18vd8fpwxzck93qlwghaj6arh4p7c5n8978vsyg", "code_id":1, "creator":"secret16u7w28vp68qmldffuc89am4f02045zlfsjht90", "label":"counterContract" } ]

```

Congratulations! You just finished compiling and deploying your first contract! Now it's time to see it in action!

Last updated1 month ago On this page *Generate your new counter contract * Compile the Code * Storing the Contract * Instantiating the Contract

Was this helpful? Edit on GitHub Export as PDF