Fadroma

Made with

at Hack.bg

[

Check me on GitHub

](https://github.com/hackbg/fadroma)

Intro

Hello Secret Network developers and community!

Over the past year we've been working on Fadroma alongside the projects that we've been building on Secret Network, so we can ease our development workflow and strenghten some reusable components across projects.

It is now time we share it officially with the community, secretly hoping it will help more people and we'll get feedback and contributions so we can further develop and improve the performance and features of Fadroma.

Introducing Fadroma - Industrial-strength components and workflows for Cosmos smart contract development and deployment.

Small text note: We currently only support Secret Network but plan to extend support to other CosmWasm based blockchains.

User stories

How this product solves a problem or improves people's experience:

- As a smart contract developer, I need to focus on the business logic of what I'm writing. For that, I need to iterate on a tight feedback loop. Therefore, I use Fadroma because it lets me avoid platform specific boilerplate code and automates the repetitive workflow steps by providing essential smart contract building blocks which allows me to maintain high development velocity and deliver a reliable product.
- As a Web 2.0 developer learning blockchain development, I have an expectation of an easy onboarding onto a well-thought-out development environment that consists of dev tools that provide me with useful feedback, and reliable middlewares that demonstrate modern best practices. Therefore, I use Fadroma. Because it lets me transfer my skillset to the new ecosystem more easily and provides a state-of-the-art development experience.

Value proposition

To continually improve the viability of the blockchain paradigm as a foundation for the decentralized software ecosystem of tomorrow, the tooling used for developing smart contract-based products needs to be qualitatively better than what is currently being used for mainstream Web development.

We'd like for Fadroma to provide a streamlined workflow for rapid development of Cosmos-based products, guiding developers' attention towards extensive high-level validation of smart contracts' behavior, and preventing them from getting bogged down in platform details.

Eventually, we'd like for Fadroma to become the frontend for an extensive library of pre-validated templates for smart contracts that can be customized and deployed with little to no coding, similar to what OpenZeppelin provides for Ethereum, or CosmWasm Plus provides for Cosmos.

Strategy and roadmap

Currently, Fadroma is being developed as an in-house framework facilitating theSienna Network project. In order to be able to bring it to a larger audience, development hours must be dedicated to improving test coverage and documentation, stabilizing the APIs, and ensuring compatibility with a wide range of use cases and underlying platforms.

The feature that we most look forward to building is the Fadroma GUI

. Normally, the different steps of a development workflow are accessed manually through a variety of GUIs and CLIs, which incurs a cost in terms of mental context switching. We'd like to put all that in an embeddable HTML5-based control panel that lets the developer do all this from a common control center.

Where we stand

Among other features, we currently provide:

- Fadroma Prelude

: Reexport of platform libraries, with added types for frequently used building blocks: * ContractLink

, a data type that contains a contract address and code hash

- Humanize

and Canonize

- traits to convert structures that contain HumanAddr

to their CanonicalAddr

equivalent and vice versa

- ViewingKey

and Permit

viewing key and permit with custom permissions

- Uint256
- 256-bit equivalents of the CosmWasm numeric types, etc.
- ContractLink

, a data type that contains a contract address and code hash

- Humanize

and Canonize

- traits to convert structures that contain HumanAddr

to their CanonicalAddr

equivalent and vice versa

- ViewingKey

and Permit

viewing key and permit with custom permissions

- Uint256
- 256-bit equivalents of the CosmWasm numeric types, etc.
- Fadroma Derive

: A derive macro that generates boilerplate code like message definitions and dispatch init/handle/query functions, as well as wasm32 entry points. * The macro allows to compose contracts out of separate subsystems.

- We use the macro to build Sienna Lend and will include documentation for it very soon.
- The macro allows to compose contracts out of separate subsystems.
- We use the macro to build Sienna Lend and will include documentation for it very soon.
- Fadroma Components

: Implementations of frequently used smart contract functionality. * These are written using the derive macro and can be included with very little setup by using Fadroma Derive.

- Fadroma Admin

: TX sender-based admin auth

- Fadroma VK

: viewing key-based auth

- Fadroma Permit

: query permit-based auth

- Fadroma Killswitch

: pause or permanently disable contracts

- Fadroma Token

: A SNIP-20 implementation forked from thereference implementation and rebuilt as a trait. * This allows individual token contract functions to be overriden without having to copy the entire codebase.

- Furthermore, it enables token functionality to be added to service contracts, obviating the need for a separate contract to represent the service token.
- This allows individual token contract functions to be overriden without having to copy the entire codebase.
- Furthermore, it enables token functionality to be added to service contracts, obviating the need for a separate contract to represent the service token.
- These are written using the derive macro and can be included with very little setup by using Fadroma Derive.
- Fadroma Admin

: TX sender-based admin auth

- Fadroma VK

: viewing key-based auth

- Fadroma Permit

: query permit-based auth

- Fadroma Killswitch

: pause or permanently disable contracts

- Fadroma Token

: A SNIP-20 implementation forked from thereference implementation and rebuilt as a trait. * This allows individual token contract functions to be overriden without having to copy the entire codebase.

- Furthermore, it enables token functionality to be added to service contracts, obviating the need for a separate contract to represent the service token.
- This allows individual token contract functions to be overriden without having to copy the entire codebase.
- Furthermore, it enables token functionality to be added to service contracts, obviating the need for a separate contract to represent the service token.
- Fadroma Ensemble

: Fast integration testing of multiple smart contracts in Rust by mocking the CosmWasm API. * This is very useful for testing application logic when it is distributed among several contracts.and having them run instantaneously during development.

- This is very useful for testing application logic when it is distributed among several contracts.and having them run instantaneously during development.
- Fadroma Ops

: Node.js-based operations framework for managing the smart contract lifecycle. Provides a rich vocabulary of classes for modeling your smart contract deployment workflow. * Fadroma Build

: Compile smart contracts from the working tree or a past point in Git history.

- Fadroma Localnet

: Run a localnet for development,

- Fadroma Receipts

: Keep track of uploads and instantiations. It is possible to update an existing deployment by specfying its id from the receipts and adding additional instances or executing messages all from code.

- Fadroma Schema

: Generate TypeScript definitions from the JSON Schema exported by contracts.

- Fadroma Bundle

: Execute multiple messages in a single transaction, or generate an unsigned transaction for manual multi-signing.

- Fadroma Build

: Compile smart contracts from the working tree or a past point in Git history.

- Fadroma Localnet

: Run a localnet for development,

- Fadroma Receipts

: Keep track of uploads and instantiations. It is possible to update an existing deployment by specfying its id from the receipts and adding additional instances or executing messages all from code.

- Fadroma Schema

: Generate TypeScript definitions from the JSON Schema exported by contracts.

- Fadroma Bundle

: Execute multiple messages in a single transaction, or generate an unsigned transaction for manual multi-signing.

What we want to achieve (short-term roadmap)

- 100% test coverage.
- Docker Compose integration

for a portable development environment based on familiar tools.

- Fadroma CLI

. Command-line entrypoint for creating new projects and running default and custom management commands.

- (init) Start a project
- (add) Add contracts to it
- (compile) Compile them

- (init) Start a project
- (add) Add contracts to it
- (compile) Compile them

- Fadroma GUI

.

- Visualize dependencies between contracts.
- Automatically generate microfrontends from smart contract schema, to let you easily construct management dashboards and product UIs.
- Generate and browse documentation of your project and its dependencies.
- Explore blockchain activity.
- Visualize dependencies between contracts.

- Automatically generate microfrontends from smart contract schema, to let you easily construct management dashboards and product UIs.
- Generate and browse documentation of your project and its dependencies.
- Explore blockchain activity.
- Advanced testing tools

for your smart contracts:

- WASM-FFI integration

for testing production-ready smart contract builds.

- BDD

support to allow non-programmers to specify test cases using a Cucumber-like language and reusable test steps generated from the contract schema.

- Fuzzing

, property-based testing

, eventually formal validation

.

- View test results

and coverage reports

in a friendly Web UI.

- WASM-FFI integration

for testing production-ready smart contract builds.

- BDD

support to allow non-programmers to specify test cases using a Cucumber-like language and reusable test steps generated from the contract schema.

- Fuzzing

, property-based testing

, eventually formal validation

.

- View test results

and coverage reports

in a friendly Web UI.

- See our full roadmap on GitHub

Risks

Fadroma is intended to become a common foundation for the community to build on. The main existential risk for the project is that it fails to capture a large enough audience to become a standard component of Cosmos developers' workflows in the way Truffle and Hardhat are for the Ethereum ecosystem.

Core team members

Check the list of core members and active maintainers here.

How to get Fadroma

[Fadroma 22.01

](https://github.com/hackbg/fadroma/tree/22.01) is currently available as an unstable Git repository that we include into our projects as a [Git submodule

](https://git-scm.com/book/en/v2/Git-Tools-Submodules) (although [Git subtree

](https://www.atlassian.com/git/tutorials/git-subtree) might be less hassle, so feel free to try that too.)

You can do this today with:

git submodule add -b 22.01 https://github.com/hackbg/fadroma fadroma

or

git subtree add --prefix fadroma https://github.com/hackbg/fadroma 22.01

Note:

When consuming Fadroma as a submodule or subtree, there are some special steps that you need to take:

- To use the Rust

components of Fadroma, add them to your Cargo workspace

and reference them by path in your contracts' Cargo.toml

.

- To use the TypeScript

components of Fadroma, you need: 1. A workspace-aware package manager, such asPNPM or Yarn.

1. A Node.js wrapper which compiles TypeScript on demand, such asGanesha or esno/esmo.
2. A workspace-aware package manager, such as PNPM or Yarn.
3. A Node.js wrapper which compiles TypeScript on demand, such asGanesha or esno/esmo.

Wen stable?1

22.01

is a calendar version

, reflecting the rapid flux of changes to Fadroma over the past year that madesemantic versioning

impractical. We are now preparing for a stable release

, which will be available as a regular pre-packaged library on Cargo and NPM under a standard semantic version.

Taking into account that we tagged an internal v1.0.0

early on, we are currently deliberating whether to tag our first stable as v23.0.0

or v100.0.0

, so as not to introduce ambiguities when migrating our Fadroma dependencies back to standard semantic versioning. What do you think?

How to help

- First and foremost, we would really love to get more feedback. Take Fadroma for a test drive and report your observations in our issue tracker!
- We'd like to get more collaborators aboard as well - checkcontributing guidelines and feel welcome to submit your pull requests!
- P.S. Secret Foundation if you think all of the above makes sense and is in line with your agenda - a grant might speed things up

Shh