Cosmos Hub Proposal: CosmWasm

May 8, 2020 / v0.3.0
By Ethan Frey (Confio)

Summary

Validator-reviewed smart contract capability for the Cosmos Hub
Value: Enable staking derivates, "Rented Security", and dynamic IBC for a competitive Cosmos Hub
Milestone One Value: Governance control over complete lifecycle of smart contracts on the Hub, including upgrades and freezing. Working example of staking derivatives.
Timeline: 2-4 months
Funding amount: First Milestone: 25k Atoms

Background

CosmWasm is a Cosmos SDK module that enables Wasm smart contracts to be securely executed and interact with many of the native Go modules. CosmWasm is in a v0.7 alpha-release and has been deployed on 3 public testnets to date with no reported security bugs. It is compatible with any application based on Cosmos SDK v0.38.

We propose to prepare a version of CosmWasm with some feature enhancements for a future IBC-enabled version of the Cosmos Hub. This proposal asks for support in developing a feature set that would be desirable for the Cosmos Hub.

Applicants

Confio, founded by Ethan Frey, is applying for this grant. Ethan/Confio is the creator of CosmWasm.com and works along with Simon Warta on maintaining and developing the smart contract platform. Ethan has been involved in the Cosmos ecosystem since 2016 (building ABCI apps on Tendermint 0.6) and worked full-time for AiB Jan 2017 - Feb 2018, building out light client proofs and the first IBC specification, along with the original Cosmos SDK. Ethan and Simon worked together at IOV and built out a custom Golang ABCI framework along with a full-fledged TypeScript client SDK. You can see some work here:

- github.com/CosmWasm

- github.com/confio
- github.com/ethanfrey
- github.com/webmaster128

Purpose

IBC is the future of the Cosmos Hub, and CosmWasm is the key to enabling the Cosmos Hub to remain the center of development and innovation in this networked ecosystem. It is key to the two largest value propositions on top of IBC/ICS20 - Dynamic IBC and "Rented Security". Failure to adopt these will allow a more competitive hub with more features to assume a central role in a future IBC ecosystem.

Providing security for other zones may be one of the largest post-IBC value propositions for Cosmos Hub validators. How this would be implemented has been an open discussion for the last year. Recently, the key architect of this, Zaki (Iqlusion), described CosmWasm as the key to rolling this out:

Zaki (Iqlusion): *"For ATOMs to be financial viable, ATOMs must secure more than just hub consensus. The Hub must be able to punish faults that occur in other protocols.*

I believe that running a narrow set of CosmWasm contracts on the hub is the key to getting this done. Specifically, CosmWasm should be integrated into IBC, staking and slashing modules allowing staked ATOMs to opt into new slashing conditions."

The other key to remaining a central hub is enabling many zones to both securely and flexibly communicate with each other. Since a hub must speak the language of every zone's protocol that it re-transmits, the hub will be limited to just ICS20 or slow evolution if there is not an approach to enable Dynamic IBC in the Cosmos Hub.

Gavin (Figment Networks): *"I'm curious to know if Agoric's hub/zone(?) will compete with the Cosmos Hub, since dIBC will make compatibility easier to connect with new zones (instead of requiring protocol upgrades for each existing zone). Could CosmWasm enable the Hub to connect new zones without requiring protocol upgrades?"*

Proposed Features

Limit Code Upload (and Contract Instantiation) to require governance approval.

This will only allow a limited set of approved contracts to run on the hub, to provide the dynamic capabilities described above, while still allowing the validators to control any functionality upgrades to the hub via on-chain governance. This is a much quicker and safer approach than releasing a new Go binary for each update, but still allows governance to select which features are desirable to run on the Hub.

Demonstrate Staking Derivatives using Contract
Whether or not the hub chooses to enable this, it is a highly requested feature by many. Demonstrating a contract that can be governance-enabled to dynamically add such functionality will provide an easy way for people to test such contracts on other networks, and governance the possibility to adopt it later, if it proves successful.

Allow Permissioned Contract Upgrades/Freezes
The contract owner (in this case, the governance board) should be able to update contracts as needed to make bugfixes or feature enhancements. It should also have a mechanism to orderly shut down a contract (eg. if it is a DEX, freeze all activity and release all funds locked in open orders to the respective owners). This will allow the governance a similar lifecycle to work with contracts as it has for native modules.

Integrate IBC Into CosmWasm
There is a detailed funding proposal awaiting approval from ICF. In the case this is not funded, we would request support from the Cosmos Hub governance to perform this non-trivial work. Potentially there will be multiple funding sources for this.

Demonstrate using Dynamic IBC to route multi-hop packets
This depends on the IBC implementation above, and allows us to dynamically add routing potential for new, dynamic IBC protocols. By writing a router as a Web Assembly contract, we will not only provide an example/template for others to build on, but uncover if there are any small gaps that need to be filled in.

Demonstrate Simple "Rented Security" using Contract
This also depends on the IBC implementation above. Provide a contract that implements a relatively simple (not game-theoretical secure) version of "Rented Security", to demonstrate technical feasibility and provide a template on which more advanced contracts can be developed.

Write Contracts in AssemblyScript or Golang
Currently we only support writing CosmWasm contracts in Rust. Many languages can compile

down to Wasm, but you need to implement a lot of low-level code around the import and export calls, as well as memory management to get it to work. We provide higher-level ergonomic bindings in the Rust language, but even that is a barrier to entry. We have researched providing contract support in both AssemblyScript (TypeScript subset) as well as TinyGo (subset of Go with little-to-no reflection). Both compile efficiently into Wasm and supporting either language is feasible (but far from trivial), lowering the barrier to entry (and increase number of reviewers) for CosmWasm contracts. We are happy to expand the language support for wirting CosmWasm contracts when there is a clear need and sponsor.

Roadmap

Rather than a huge ask for all work, we propose a few smaller, targeted grants with clear deliverables and timelines. Successful delivery is a prerequisite for future grants. Thus, we can request a relatively small grant for the first milestone, and the experience upon delivering that will inform the decision to fund future milestones.

This proposal only concerns itself with the first milestone, which should be valuable by itself, but also the foundation to build more complex features for the Cosmos Hub. Based on feedback and interest from the Cosmos Hub governance, we could prioritize various features listed above, or add other features, for future milestones. Adding more languages for contract development, or additional features build on top of IBC could be prioritized after viewing the results of the first milestone.

First Milestone

We would propose for the first step to build the foundations that are independent of IBC: *Limit Code Upload to require governance approval, Allow Permissioned Contract Upgrades/Freezes*, as well as a simple example on top of it, *Demonstrate Staking Derivatives using Contract*. This will be demoed in *wasmd* (a fork of Gaia), showing how CosmWasm can quickly provide a solution to a long-standing debate on the Cosmos Hub, and let validators get some first hand experience in testnets.

Concretely, we will update https://github.com/CosmWasm/wasmd to add support for contract migation. We will add a second binary (*govd)* that works like the existing *wasmd* permissionless engine, but requires governance permission to upload new code (approve new functionality), instantiate of a given contract, and also upgrade or freeze any contract . Eg. governance can approve a generic ERC20-like token contract, and then decide if anyone can create a token, or each token created needs a further governance vote. They can also "orderly" liquidate a staking derivative contract and return funds to the owners, if they decide to disable such functionality. This functionality will be based on CosmosSDK 0.38 unless a stable 0.39 is released before the testnet is launched (in which case, we will upgrade to 0.39 after the end of this deliverable).

Beyond providing such support in the code, we will launch a testnet running this binary to give Cosmos Hub validators some practice with such a system. For this testnet, we will airdrop significant staking tokens to every Cosmos Hub address that votes on this proposal (yes or no), which signals their interest in the idea. This allows interested parties involved in the Cosmos Hub to have the voting power on this network. We will also offer a faucet with some small amount of funds for other people who want to try to use the system to execute such contracts.

This entire proposal, from writing the code to launching the testnet will be completed in 2-4 months.

Contract Lifecycle

There have been some questions as to how a contract lifecycle would work. This description here is for the proposal of *govd*, where a governance vote is able to make each decision. In *wasmd,* anyone can create contracts, and no one can modify it. Code for both would live in the same repo and you can mix-and-match which permissions belong to governance when setting up app.go and compiling the blockchain. This allows the hub to easily open up some functionality to be permissionless later on.

- *Upload Code*: Here you upload Wasm bytecode, which provides potential functionality to the hub (eg ability to run a DEX), but doesn't actually provide any useable instances. At this phase, the contract code should undergo an audit (formal or informal) and the Cosmos validators should debate if this general functionality is a useful addition to the Hub. Upon a successful vote, this bytecode will be installed on the hub, awaiting activation.
- *Instantiate Contract*: Once the bytecode is uploaded (eg. an ERC20-like template), you need to make an instance to be able to use it. This will provide a token symbol and decimals, initial distribution, and a unique contract address. This stage requires a further governance vote, so they can decide if the requested use case makes sense. After agreeing to add ERC20-like functionality, we may also control who can issue such new tokens.
- *Upgrading Contracts:* If a bug is discovered in the code, or a feature upgrade is desired, we will first Upload some new code with the requested fixes (see above), and then can

vote to migrate existing contracts from the older version to a newer version. During an upgrade, we can run a state migration to change any data as needed. Upgrading must be voted on for each contract, and requires agreement from governance. We will allow a single vote to upgrade many contracts at once, but still allow more granular control than upgrade every contract using a given code id.

- *Disabling Contracts*: If the hub decides a given contract is counter-productive to the goals of the Cosmos Hub, they should be able to disable it in an orderly way. This is an exceptional vote for governance, but will be possible if a majority agree. However, we need to do this in an orderly way not to destroy funds. For example, if there were a DEX on the hub, we can upgrade it to another contract that only has one supported message type "withdraw funds" and this lets users withdraw the funds they have locked in open trades. This would allow the Hub to freeze a DEX, but return all funds to users. You could also imagine a "brick" contract that deletes all data and returns error on any message. A migration to such a "brick" contract would be the same as destroying the contract.

Concrete example of contract lifetimes:

- A project wrote a simple batch-trading orderbook as a CosmWasm contract and proposed to add this to the Hub. After some discussion and audit, the Hub approves it. This Wasm code is now available under Code ID 3 on the blockchain.
- There are 3 proposals to add individual orderbooks for ATOM-PETH, ATOM-EEUR and EEUR-PETH. Each are approved and open up their own orderbook for trades.
- There is concern that the trades are taking up too much traffic on the hub due to arbitrage, and they decide to add a variable "Tobin Tax" on each trade that goes to the community fund. This requires a code upgrade.
- Developers modify the original contract with this new functionality and propose it to a vote. It is approved and now available as Code ID 4.
- At the same time, hearing some complaints, they prepare a liquidation contract that would freeze all trades and let users pull out their tokens tied up in open orders. This is also approved and available as Code ID 5.
- Hub governance decides that they only want trades that involve ATOM on the hub and decide to shut down the EEUR-PETH orderbook, by "upgrading" the contract to use Code ID 5. At this point all users may withdraw their funds, but no more trades may happen.
- In a separate governance vote, hub governance decides to implement the "Tobin Tax" on the ATOM-PETH and ATOM-EEUR. They upgrade those two orderbooks to use Code ID

4.

- In this final state, we have "improved" the experience of trading of the ATOM pairs and shut down the other trades, without anyone losing access to funds, or requiring a hard fork.

You can see the potential here for fine grained control of functionality on the hub, without requiring downtime or forking. All changes need to go through hub governance to ensure decisions are made after proper reflection.

Payment

In order to provide some control and oversight on the delivery of this contract, we propose that the payment is sent to a multisig account controlled by respected actors on the Cosmos Hub, Zaki Manian (Iqlusion), Martin Dyring (e-money), and Gregory Landua (Regen Network). The particular address of this account is *cosmos1y3x7q772u8s25c5zve949fhanrhvmtnu484l8z*.

They will release payment to Confio's account according to successful completion of milestones. Failure to complete successful delivery within 4 months leads to a forfeit of any remaining funding. The payment will be issued as follows: 10k atoms at contract start, 7.5k atoms upon code completion and demo of the functionality to the multisig holders, and the final 7.5k after successful demonstration of contract deployment on the testnet, with contract approved by real ATOM holders, and integrating reasonable feedback.

Relationships and Disclosure

Confio is a LLC registered in Estonia, solely owned by Ethan Frey. We have received an ICF grant to take CosmWasm from a Hackatom project to a functioning application with gas metering, security guarantees, and tooling. This grant ended early February 2020, and we have a pending grant proposal for IBC integration into CosmWasm. We also have a contract with Terra to develop out some features for our 0.8 release allowing tight integration of CosmWasm contracts to the staking module, as well as their custom modules (swap, oracle, and treasury).

Confio is now focused solely on developing CosmWasm, and is pursuing other contracts from projects that benefit from the development, in order to fund new features.