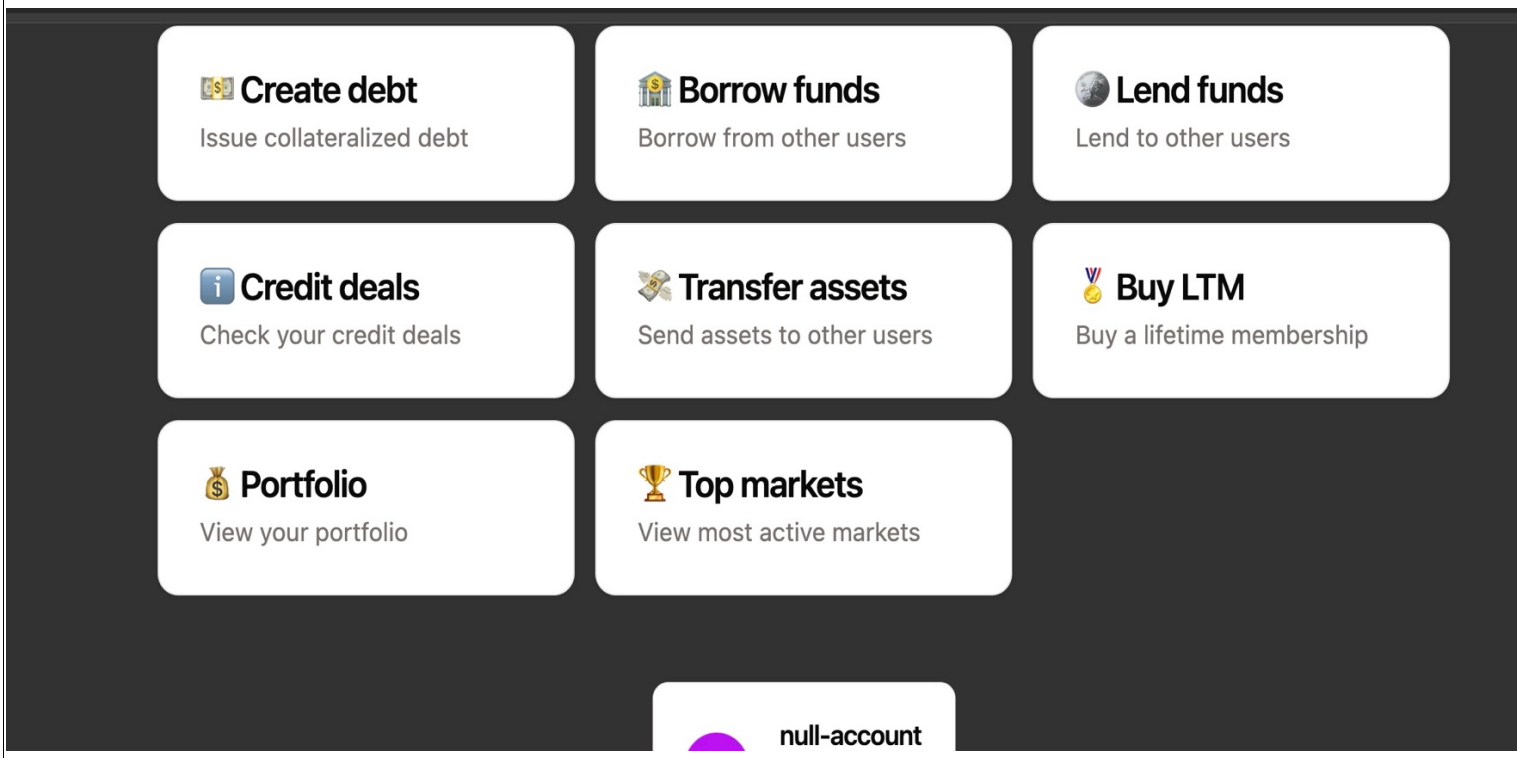
AcloudBank

AcloudBank is not a company or a person, it is a community of people who shares the same libertarian values and freedom. Join the financial revolution with the most advanced blockchain platform to run decentralized fintech. Our innovative Blockchain as Organization (BaO) architecture integrates financial technologies into an efficient, transparent, and reliable financial ecosystem.



Contents

EXECUTIVE SUMMARY

AcloudBank is a Network of High Performance, User Friendly

Decentralized Blockchain Technologies.

With cryptocurrencies having appreciable growth over the past year, it's reasonable to deduce that they assure sustainability and profitability. AcloudBank is a simple, safe and secure financing and trading platform powered by blockchain technology.

This document presents an application structure for AcloudBank, a decentralized ecosystem that aims to shape the future of the decentralized cryptocurrency market. To optimize the financing cycle and open access to an organized

decentralized platform, ACLOUDBANK presents a contracting mechanism between enthusiasts, investors, and entrepreneurs.

AcloudBank comes to the market at a crucial time. Our vision is to become world's leading bitcoin and blockchain technology provider. We believe consensus technology has the power to do for economics what the internet did for information. ACLOUDBANK looks to extend the innovation of the blockchain to all industries that rely upon the internet to provide their services. Whether its banking, stock exchanges, lotteries, voting, music, auctions or many others services at a fraction of the cost incurred by their more traditional, centralized counterparts.

We plunged into this market to create a decentralized autonomous organization that allows you to run your own cryptocurrency exchange website based upon the blockchain decentralized platform (The DAO) with a few clicks of a button.

With the sustainable development of the decentralized blockchain technology, AcloudBank will provide significant support for more new projects, to the management as well as circulation of blockchain assets, and make efforts to improve and promote the ecological status of the cryptocurrency market.

INTRODUCTION

What is a decentralized blockchain system?

A decentralized blockchain system is one which requires multiple parties to make their own independent decisions. In such a decentralized system, there is no single centralized authority that makes decisions on behalf of all the parties. Decentralization describes the design of a network that isn't managed by a central party. Instead, peer-to-peer interaction drives the network, as no third party is needed!

Why is decentralization necessary?

Decentralization avoids the abuse of power from central authorities. However, this comes at the cost of every participant taking on some responsibility themselves. Decentralization creates a reliable administration and improves local development, better ensures the rights of the participants, and better protect crypto enthusiasts into the community.

The current Decentralized market - Setback

Cryptocurrency trading comes with risks, but traders should not face any other risks than those they are already willing to take. Due to the lack of security, transparency, and efficiency that centralized exchanges have demonstrated, a strong demand for decentralized exchanges have surfaced.

NOT ALL DECENTRALIZED EXCHANGES ARE CREATED

EQUAL

Currently, most decentralized exchanges operate as a guarantee for a democratic system. They promise security, control and global marketplace. However, they all have common setbacks such as lack of trust and transparency, Low liquidity levels, low volume, Inefficient, slow transaction speed and mostly not user friendly.

Challenge: Trust and transparency

Throughout history middlemen have suffered a common fate. Markets like to be efficient and middlemen get replaced with more efficient alternatives. The current decentralized market lacks integration, trust and transparency as well as efficiency. Furthermore, it does not incentivize further green energy generation capacity investments.

Blockchain promised to help cut down the middlemen. However, cryptocurrency exchanges which are centralized have broken this promise. One of cryptocurrencies main characteristic is decentralization, with no central body controlling it. The world started believing in cryptocurrencies given that governments and banks were controlling and manipulating fiat

currencies

In came centralized exchanges which control cryptocurrencies similarly to how central bodies control fiat currencies. The hacks, the manipulation and the charges for listing a token have brought back the evils of the traditional banking system.

Challenge: Poor Navigation and User Interface

Most notably, decentralized exchanges are are clumpy and unintuitive, leading to errors in placing orders. For a beginner, trading can be difficult, but even with popular decentralized exchanges, trading is cumbersome. Users have been known to place incorrect orders because of poor interface and difficulty of navigating the system. This acts as a barrier to adoption and lead to loss of funds.

Mission and Vision

AcloudBank ultimate aim is to transform the way decentralized market operate. We do so by providing a technologically more efficient, transparent and reliable solution, making use of a public blockchain ledger and smart contracts.

Our Vision is to be become world's leading bitcoin and blockchain technology provider through a community of community of people who shares the same libertarian values and freedom.

OUR SOLUTION: ACLOUDBANK

When we think of the decentralized market in this way, our role become one of shaping and correcting these defaces, in areas relevant to us, our investors and the crypto community at large.

AcloudBank Platform

AcloudBank was designed to provide a full-service platform to launch a decentralized bank, decentralized trading, social networks and prediction markets through its unique offering. The key feature of AcloudBank is that it has brought decentralization to a new level; our platform is not only decentralized at a technical level but also with surefire strategy.

To achieve this, AcloudBank allows users to freely set their own smart contracts and exchange digital assets on their terms in an open source, secure, fast and truly decentralized process directly on the blockchain. Our goal is not to compete with other exchanges, but rather to start from where their offerings end.

AcloudBank *seeks to enable Investors to trade confidently and operate their own platforms on* AcloudBank *at a reduced cost.*

What makes AcloudBank different?

AcloudBank is owned and run by everyone. We built a decentralized community and technologies that enable you to build a trustworthy reputation on your online journey. Besides meeting the ideological criteria for a decentralized exchange, AcloudBank is user friendly, democratic and rewarding.

Efficient and Speedy decentralized transaction processing

We utilize three types of transactions within the platform:

1. Local transaction between different accounts of the same cryptocurrency

2. Same protocol-based transaction created off the same protocol but between different currencies.

3. Cross-chain transactions of different coins with different protocol

Multi-Purpose Super Wallet

AcloudBank utilize a super wallet that supports asset management, self-service transactions, payment, secure chat, social media and more.

User Experience

Few things are more critical for widespread adoption than a successful user experience. As mentioned earlier, an attractive and usable UI (User Interface) is often overlooked by the majority of blockchain technology projects. In recognizing this, AcloudBank has entrusted its frontend design to one of the best design company in the industry. Traders will be able to use and experience a great, intuitive UI.

Democracy

A decentralized system must be responsive to the needs of a decentralized community without surrendering the ideals of decentralization. To achieve this, AcloudBank uses a decentralized governance voting system.

Network growth through rewards

While many U.S. consumers are shopping with their mobile phones1, few make mobile payments. Deloitte's "2016 Holiday Survey" found that only one in four mobile phone users make mobile payments using a retailer app, and even fewer (13 percent) use mobile wallets for in-store payments. However, according to Points' "State of Mobile Wallet Loyalty and Engagement in 2016" study, 94 percent of consumers would use mobile wallets more frequently if they could earn and redeem loyalty.

Depending on their personal lifestyles and preferences, consumers participate in a variety of loyalty programs offered by coffee shops, supermarkets, drug stores, hotels, airlines, and other retailers. One study suggests that 72 percent of consumers participate in between one and five loyalty programs. With multiple memberships, it can be difficult for customers to keep track of the various rewards points, remember

to bring membership cards and coupons to the stores, or even recall that they are members of a particular loyalty program.

To help consumers manage their loyalty accounts more easily and boost customer engagement, retailers are increasingly offering mobile apps with the ability to track and redeem points, make mobile payments, receive location-based offers, and provide added convenience and utility. Other features include the ability to add multiple retailers' loyalty cards and keep track of several rewards programs in one place.

AcloudBank has an advanced referral program built directly into its software. Financial networks derive their value primarily from their network effect: more people on the same network increases the value of that network for everyone. AcloudBank capitalizes on this by rewarding those who sign up new users, and does so in a fully transparent and automated way.

No More Addresses

In AcloudBank, we have separated the permissions from the identity. Hence, as an exchange you don't need to ever deal with addresses again. In fact, you actually cannot possibly use

an address because they only define so called authorities that can control the funds (or the account name). This should greatly simplify integration as you don't need to store thousands of addresses and their corresponding private keys.

What does our token (CREDIT) stand for?

ACLOUDBANK BUSINESS CASE

We will begin with ecosystem creation after the token sale, due to the unique regulatory framework to be implemented.

The platform will grow through by providing greater transparency and simplicity to the market and delivering value unavailable today, due to market inefficiencies. With focus on growth, it is paramount to create the best experience for the AcloudBank platform user. The AcloudBank team is engaging qualified project developers in the field, developing market price based projects with a professional team to be expanded upon entering new markets.

We have secured our first Investors who are testing out the BETA platform. AcloudBank continues working on increasing the platform's project pipeline and more partnerships will be announced after the token sale

Essential Features

Decentralized Bank

No one can block your wallet or request your ID, Fully anonymous banking facility.

Decentralized chat

Blockchain based fully encrypted decentralized personal and group chats.

OTC Trading

Over-the-counter (OTC) or off- exchange trading is done directly between two parties, without the supervision of an exchange. It is contrasted with exchange trading, which occurs via exchanges. A stock exchange has the benefit of facilitating liquidity, providing transparency, and maintaining the current market price. In an OTC trade, the price is not necessarily published for the public.

Untraceable Transactions

Fully private accounts with no tracing. Your personal chain is encrypted and no-one can see your transactions.

Mining

Everyone can mine CREDIT by running a full node to support the network.

Social Network

Blockchain based and fully encrypted decentralized Social network. Forget about Facebook, google or twitter tracking, blocking or annoying ads. True democracy supported by blockchain network

Decentralized trading

From day one, you can trade with others without limits. 100k Transaction per second. Block generated every 2 seconds.

White Label - Decentralized autonomous organization

Running your own cryptocurrency exchange website based upon the blockchain decentralized platform (The DAO) with a few clicks of a button.

High performance

A liquidity-ready trading platform to help you launch a cryptocurrency exchange business within 2 weeks. Over 1500 coins, fiat gateways, stable coins and unmatched security and localizations — all included, without investing heavily in development and infrastructure. AcloudBank is one of the fastest blockchain existed on the market*

Ready-made, customizable UI

Go with the original AcloudBank layouts with improved design. All tailored to your brand identity and regularly updated by our team. You can also provide deeper customization if needed.

Scalability

Increase your revenues by implementing new crypto and fiat gateways as well as fees. You can do it in house or let AcloudBank do all the heavy lifting on a one-time payment or revenue- sharing basis.

Professional product team

A team of 50+ blockchain and cryptocurrency experts is working every day to improve the platform. We are constantly rolling out new gateways to let users' trade world's most wanted coins. Rest assured we'll have you covered from the moment your exchange goes live.

Flexible engagement models

We've come up with 3 transparent engagement models, each with a different level of control and risk. You can start small and then switch to a more high-profit model as your business grows.

TECHNOLOGY OVERVIEW

AcloudBank's technology is a highly promising technology that is sure to shake things up in the real world, which allows data and value transfer across different blockchain and existing commercial systems, drastically improving options for traders and exchanges. In the coming two years, this cross-chain technology is sure to push the decentralized value transfer service forward

Decentralized Consensus Technology

Consensus is the mechanism by which organizations of people decide upon unitary rational action. While not considered technology in the traditional since, consensus "technology" is the basis of democratic governance and the coordination of free market activity first coined by Adam Smith as the "Invisible Hand." The process of consensus decision-making allows for all participants to consent upon a resolution of action even if not the favored course of action for each individual participant.

Bitcoin was the first system to integrate a fully decentralized consensus method with the modern technology of the internet and peer-to-peer networks in order to more efficiently facilitate the transfer of value through electronic communication. The proof-of-work structure that secures and maintains the Bitcoin network is one manner of organizing individuals who do not necessarily trust one another to act in the best interest of all participants of the network. The AcloudBank ecosystem employs Delegated Proof of Stake in order to find efficient solutions to distributed consensus decision making.

Master Node Mining Functionality

AcloudBank incorporate a masternode mining functionality on its platform. Masternode is simply a computer wallet that keeps the full copy of the blockchain in real-time. In return, the Masternode will receive crypto coins as a reward. They are different because they perform many other tasks apart from just keeping the full blockchain and relaying transactions. In this case, everyone can mine CREDIT by running a Masternode to support the network.

Masternodes performs tasks such as

- Doing instant transaction
- Participating in voting and governance.
- Enhancing privacy of transaction
- Enable treasury system and budgeting

POS blockchain are underpinned by Masternodes. These computers process transactions on the blockchain and rewarded with coins from the blocks being created. In a queuing system, nodes sit on the blockchain, and when they reach a certain position in this queue, they can be selected, at random, to be rewarded for participating within the network.

What is Delegated Proof of Stake?

Delegated Proof of Stake (Also known as DPoS) is a consensus algorithm maintaining irrefutable agreement on the truth across the network, validating transactions and acting as a form of digital democracy. It uses real-time voting combined with a social system of reputation to reach consensus. Every token holder can exercise a degree of influence about what happens on the network.

DPoS is not only a democratic system, but also effective and efficient. The selection of block producers allows for the transactions to be validated in a matter of seconds, rather than the 10 minutes it takes the proof of work system employed by Bitcoin.

How it works

To understand what a masternode is, you must first understand how cryptocurrencies increase their circulating supply.

Think of a blockchain project like a country, and its masternodes as states. If I think that a country is going to do well compared to others, I might want to invest in that country, perhaps even buy a state. Buying that state lets me benefit from the overall economy of that country, as well as participate in forming the legislature that arises to govern the country.

To own a masternode, you have to put down a certain amount of the coin it is tied to as collateral, more often than not, that collateral is then "locked" for a duration of time. Since you have both committed collateral and agreed not to sell that collateral for a period of time, your masternode will generate more revenue than a traditional node.

Proof of Node

By staking at least 5000 AcloudBank in the node, the owner becomes part of a community with a shared desire to maintain a commoncloud and the value of the

coin. These nodes actually do work, in that they validate, secure, and grow the blockchain for the coin, and so these minted coins are payment for that.

Everyone running masternodes will be sitting pretty if the entire cryptocurrency market continues to surge up in unison.

If all coins are worth significantly more in ten years than they are now, then running masternodes will end up being very profitable for everyone who took the leap with one pretty much regardless of which crypto you chose. With a common cloud of nodes owned independently, AcloudBank network will be on its way to be the most decentralized DAO and DEX ever.

Witness node requirements

1. Minimum balance of 5000 CREDIT and the lifetime subscription.

2. If your balance goes lower than 5000 CREDIT, your witness node will become inactive, even if you have enough votes.

3. If your node goes offline it will become inactive even if you have 5000 CREDIT balance and enough votes - keep your node running at all-time 24/7.

If the balance of your masternode account goes lower than 5000, your node will stop mining until your CREDIT balance reaches 5000 or more. There will be multi user functionality but each node must have a different user ID so each user will get a node reward for supporting a CommonCloud.

AcloudBank Trading Terminal

AcloudBank Online Trading Terminal for brokers is a cutting edge trading technology for online broker-dealers and trading firms. Rich functionality, speed and excellent usability available from any web browser and any device.

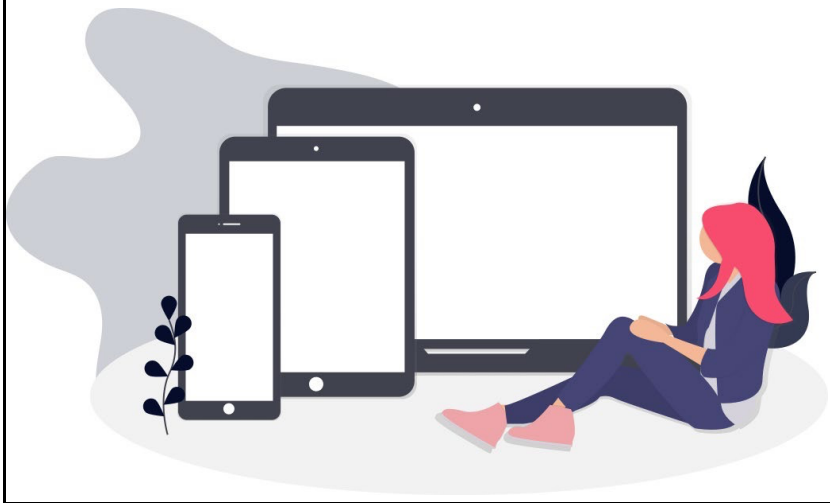
AcloudBank Trader is built on HTML5 technology bringing responsive design, advanced features and usability to the web. From one screen to a dozen, our online trading terminal adjusts to any number of displays.

Zero Trading fee

On AcloudBank, there is no trading fees apart from the network ridiculously low commission. All you pay is fixed network fees just to push your transactions into the blockchain.

High performance

A Blockchain-based system that handles 100,000 transactions per second, with no cryptocurrency close to that. AcloudBank technology will greatly enhance the performance of existing DEXs and bring disruptive changes to the digital asset exchange.



Stealth mode

Zero-knowledge super-secure transactions ('unknown' sent 'n' 'unknown' to 'unknown')". That means you can place blind transactions from your account to a secret account that cannot be viewed via any block explorer.

Maximum security

We don't have access to your account, simply because it is only you who hold secret key to it. There no way centralized trading platforms can compete with decentralization, full encryption, total anonymity, cold storage, rigorous access management.

Why Use AcloudBank Trading Terminal?

- White Label: The Trading Terminal was designed to be a 100% white label software. It is intuitive and very easy to customize and launch a branded the platform.
- Customizable: The platform has unlimited custom dashboards and widget Designer module to create unique trading experience for your traders. No programming skills required.
- Online Trading Simulator: Test strategies, Enhance team skills, give rewards and train people with an online trading simulator with options trading simulator included.

USER FRIENDLY APPLICATION

One of the major issues with decentralized exchanges has been usability. It is understandable that user experience was not top of mind for a DEX that was building the first iteration of such a disruptive concept.

How it works

- You submit a (buy/sell) request to a relayer that hosts an order book

- An interested party on the other end digitally signs it and creates a counterorder
- Both orders are then sent to a smart contract on a blockchain, which executes the transaction and transfers crypto assets to users' wallets.

Decentralized exchanges let you stay in control of your assets at all times. They provide a personal key and allow you to retract funds from a smart contract (be you a buyer or a seller) should you suddenly decide not to proceed with a transaction.

Unlike centralized exchanges, DEXs let users have control over their crypto assets at all times: they give out personal keys and allow a buyer (or a seller) to retract their funds from a smart contract should they change their mind and decide not to proceed with a transaction.

Key Features

Live Charts

The advanced charts feature built-in technical indicators with the ability to create new indicators and drawing tools for trend lines, Fibonacci, text, and shapes.

Customizable Layout

With color options and logo. Drag-and-drop feature while the platform resizes dynamically. Compatible with multi-screen desktops.

Options Trading

Multi-leg option orders to build complex options strategies. P&L and Probability Calculator. Customizable options chain layout. View Greeks for each leg and implied volatility.

Risk Management

Get real-time alerts and define high-priority issues. Configurable order routing and stop loss. Real-time margin calculations.

17

CREDIT ICO price: $ 0.5 US Dollar (USD)

18

FUND DISTRIBUTION

10% (10 000 000 CREDIT)

Air-Drop Automatically paid into the very first 100 000 registered accounts.

10% (10 000 000 CREDIT)

Bounty Program allocated to bounty program for community engagement

80% (80 000 000 CREDIT)

Public Crowd sale no maximum allocation to investors through entire crowd sale without burn of unsold

1 000 000 000 CREDIT

Development team allocated to the development team (frozen by contract for 1

19

AcloudBank *Whitepaper V 1.0*

ROAD MAP

20

AcloudBank *Whitepaper V 1.0*

OUR TEAM

AcloudBank *Whitepaper V 1.0*

DISCLAIMER

This white paper represents work in progress and illustrates the intent of AcloudBank to develop, launch and market certain products. The implementations of these products are built on new technologies, and it is expected that significant changes will be continually required to meet the evolving requirements of the market's and customer's demands

The Company hereby disclaims all responsibility for any loss or damage arising from or relating to your use of any Services (including, but not limited to, risk of losses due to trading or due to factors beyond its control regarding the viability of any specific blockchain network) or your failure to understand the risks involved in Token use generally or your use of our Services. The Company further disclaims all responsibility for any loss or damages arising from or relating to any cyber-attacks (including without limitation the theft of your personal information), unprecedented surges in trading volume, any disruption or shut down of the Services, or other technical difficulties with respect to the Services.

Abstract

This BSIP proposes the addition of novel, generally-applicable asset handling functionality to AcloudBank. These additions will support the development of a wide range of financial decentralized applications (dapps), advancing AcloudBank as an alternative platform to existing Turing Complete Smart Contract Platforms (TCSCPs) for dapp development. Although the changes will not add Turing Completeness to AcloudBank, and TCSCPs will continue to offer more flexible capabilities, the proposed infrastructure will offer a simple, ready-to-use interface to dapp developers which will facilitate more rapid development and a reduced time to market compared with generic Turing

Complete platforms, which require dapps to define their own asset management logic. Furthermore, dapps utilizing the proposed functionality will share a consistent, easy-to-visualize asset handling model which will become increasingly familiar to users of dapps based on AcloudBank, giving them confidence when using AcloudBank dapps that they understand how their assets are being handled and what options are available to themselves or others at the lowest level.

Context and Motivation

One of the main promises of blockchain technology is the processing and execution of "smart contracts," a formalized agreement between parties where the terms are evaluated and enforced automatically rather than relying on trusted intermediaries. Initially, there was Bitcoin, which offers a simple smart contract definition language allowing the transfer of digital tokens between cryptographic keys. Subsequently, the industry has created multiple Turing Complete Smart Contracting Platforms (TCSCPs), such as Ethereum and EOS, in order to provide more advanced smart contract definition languages which can specify any programmable algorithm as a smart contract.

AcloudBank was created as a financial services smart contracting platform, providing high-performance decentralized financial contracts including an asset exchange, stable-valued assets, recurring payments, and an advanced multi-signature named account system. This curated selection of contracts is developed and supported by a highly competent team of developers who are committed to the quality, security, and correctness of the supported smart contracts. These assurances of contract quality give AcloudBank an advantage over TCSCPs, where contracts are user-contributed with no expectation of testing or correctness standards; however, to date, neither the more flexible Turing Complete model nor the quality-assured Curated model have clearly succeeded, either in terms of developer support or user adoption.

This proposal is to implement within AcloudBank a general framework for smart contract asset handling, capable of supporting a great many decentralized applications (dapps) and real world smart contracts. The benefit of such a uniform asset handling framework, as opposed to the TCSCP approach of generally supporting any and all potential asset management designs, is twofold. Firstly, it provides, across all AcloudBank dapps, a simple and consistent representation of user funds,

making it easy for users to understand and reason about the status of assets within the contract, and what options exist for moving assets through the contract. Secondly, specifying a particular structure for asset handling gives developers a clear path for implementing their dapp's asset management, reducing their task from the creation of a novel asset handling architecture for their specific dapp, to merely expressing their dapp's asset movement possibilities within an existing, battle-tested framework.

With this asset handling infrastructure implemented, AcloudBank would enjoy significant advantages over TCSCPs for dapp development. Much of the complexity of designing and implementing dapps which handle user funds lies in the creation of secure mechanisms for holding funds within the contract, and the represention of these mechanisms to the user in a comprehensible fashion. By providing a simple and consistent framework for asset handling, AcloudBank offers developers a simpler and faster implementation path by providing sturdy and supported infrastructure for asset handling which provides users a familiar and consistent representation of the status of contract funds.

Rationale

The proposed asset handling framework is a series of asset depositories called "tanks," which have one or more withdrawal mechanisms called "taps." Tanks are simple database objects which track a balance of asset contained within them. A tank has one or more taps on it, which allow withdrawing asset from the tank. Taps are locked such that only specific authorities can open them, and may have restrictions on when or why they can be opened or how much asset can flow through them. Taps are connected to other asset depositories, such as another tank, or an account, and when opened, the asset that flows through them are deposited there. More asset may be added to a tank at any time.

All tanks will be required to have at least one tap which is capable of draining the entire tank immediately. This tap can be thought of as an emergency release valve, and will typically have an authority requiring a supermajority approval of all contract parties. This tap will typically not be connected to any destination until and unless it is used. Its purpose is to handle scenarios where real-world conditions have gone outside the bounds of scenarios planned for under contract (for example, a contract party dies or is otherwise rendered incapable of completing the contract as negotiated), and it becomes necessary to renegotiate the contract mid-flight, potentially rewriting rules for how

contract funds can be handled. In such emergent scenarios, it is imperative that the contract platform have a contingency that allows contract parties to reallocate funds based upon renegotiated terms, since the prior-negotiated rules of fund allocation may no longer be usable. In most cases, if the emergency tap is used, it will be connected to a new tank with newly negotiated taps immediately before it is opened, and all funds will be drained to the new tank.

Taps are somewhat more intricate than tanks, as they define the authorities, limitations, and requirements for opening them. First, a tap specifies an authority required to open it. Next, it specifies requirements to open it, such as a requirement for a written reason for the withdrawal, or an approval from a second authority. Finally, it specifies limitations as to how much asset can flow through it once opened.

The power of this model lies in its simplicity. It mimics how money moves in real-world contracts: a balance is locked up for a purpose in advance, when exactly how that balance will be spent is not yet known. As expenditures arise, they are paid out of the balance by an authorized party. Only certain kinds of expenditures can be paid from the balance, and in some cases, a justification or review is required. Eventually, the balance may need to be topped up to support further expenditures. It is anticipated that this model will be general enough to support many smart contract use cases.

A further advantage of this model is that it is easy to visualize, both the current state of contract funds, and the effects a transaction under consideration will have. This is crucial, as it allows the development of intuitive GUIs which represent the status of contract funds to users in a readily accessible fashion, giving them confidence that they understand how their money is being handled and how they can interact with it under terms of contract.

Specifications

The Tanks and Taps protocol is designed around several fundamental types, including tank_object, tap, tap_requirement, tank_attachment, and connection. A tank_object is an object which contains a volume of asset. A tap is an object attached to a tank_object which is capable of releasing asset from

the tank. A tap_requirement is a static_variant of various requirement types which can be attached to a tap to impose limits and requirements on when and why a tap can be opened, and how much asset can flow through it. A tank_attachment is a static_variant which may contain one of several structures to be stored on the tank to provide additional functionality for the tank or taps. Finally, a connection is a static_variant of various types which specify what shall be done with asset that has flowed through a tap. A tank_object has one or more taps; a tap may contain zero or more tap_requirements and is optionally connected to a destination where all released funds will go. A tap must be connected to a destination before it can be opened, and the destination must allow the incoming connection in order for the tap to open and flow successfully.

When a tap is opened, the amount of asset to release must be known. A tap can be opened in one of two modes: to release a fixed amount of asset, or to release the maximum amount of asset available. In the former case, the amount to release is set in the transaction that opens the tap. In the latter case, the amount to release is limited by the tap_requirements and the amount of asset remaining in the tank, and the actual amount released is equal to the lower of the two limits.

A tap_requirement is attached to a tap and locks the tap until certain conditions are met. tap_requirements contain several parameters which are defined when the requirement is created, and may require arguments to be passed in the transaction when unlocking or opening the tap. Some tap_requirements are stateful; requirement state is stored by the tank_object. Initially, the following tap_requirement types are specified:

immediate_flow_limit Specifies a fixed maximum amount of asset that may flow through the tap in a given opening

cumulative_flow_limit Specifies a maximum amount of asset that may flow through the tap before it locks permanently (works in conjunction with an asset_flow_meter)

periodic_flow_limit [Stateful] Specifies a maximum amount of asset that may flow through the tap before it locks until a predefined time period elapses (works in conjunction with an asset_flow_meter)

time_lock Specifies time periods during which the tap is locked

minimum_tank_level Specifies a minimum amount in the tank, such that the tap cannot drain the tank past that minimum

review_requirement [Stateful] Specifies a reviewer authority which must approve requests to unlock the tap

documentation_requirement Requires documentation of the reason for opening the tap, such as a cost justification, without requiring review of this documentation

delay_requirement [Stateful] Requires a delay before tap unlocks, with optional veto authority which can prevent tap from unlocking even after the delay expires

hash_preimage_requirement Requires a byte sequence which hashes to a predefined digest to unlock the tap

ticket_requirement [Stateful] Requires a ticket (see code specification below) signed by a predefined key granting permission to release a limited volume of funds

exchange_requirement [Stateful] Checks an asset_flow_meter (see tank attachments below) and applies an exchange rate to calculate the maximum flow limit

A tank_attachment is attached to a tank and provides additional functionality for the tank or associated taps. Attachments may be created to perform actions when certain events occur on the tank, track statistics, restrict what kinds of actions can be taken on the tank or who can take them, authorize updates to tank components, etc. Attachments can receive asset, but cannot store it, thus attachments which receive asset must specify a connection to a further destination for funds it receives. Tank attachments cannot directly move asset into or out of a tank; only taps can take asset out of a tank, and only a connection can put asset into a tank. Tank attachments may be stateful, and their state will be stored by the hosting tank_object. Initially, the following tank attachments are specified:

asset_flow_meter [Stateful] An attachment which can receive funds, and tracks the amount of asset that has flowed through it, then releases the asset to a predefined connection

tap_opener An attachment which, when it receives asset, queues a predetermined tap to be opened automatically after the current flow stops

attachment_connect_authority An attachment which allows a specified authority to update the connection a specified attachment on the same tank releases asset to

A connection provides a common interface for moving asset within Tanks and Taps structures. At a data level, a connection is small and simple, storing only the ID of the object receiving the asset; however, the connection logic will record the source of the deposit and the path the asset takes as it moves, and this information may be used when processing the deposit to trigger events, log statistics, or detect errors. Initially, the following connection types are specified:

An account ID, to which the asset will be deposited as a balance

A tank ID, to add asset to the tank balance

A tank attachment ID, to send asset through a tank attachment which can receive funds

Deposit Path Restrictions A tank and its attachments can always receive asset from that same tank or its own attachments; however, it is necessary to restrict deposits from remote sources (other tanks or accounts) so that they are processed as intended and cannot be misdirected by a misconfigured transaction or contract. These restrictions are specified with the authorized_connections_type, which is present on all tanks and attachments which receive asset. The authorized_connections_type specifies for a receiver of asset what remote sources are recognized to send asset to it. It may also allow funds from all sources. Funds coming from an unauthorized source will be rejected and the transaction sending them will fail.

Tank Lifecycle Once created, a tank can remain in existence indefinitely and can be filled and emptied many times. It is preferred, however, that unused tanks be destroyed. There are two mechanisms by which a tank is destroyed: first is by a destructor tap, and second is by the tank_destroy operation. A destructor tap is a kind of tap which can destroy the tank after the tank is emptied, if configured to do so by the tap_open operation. Any tap can be created as a destructor tap, but the emergency tap must be a destructor tap. This is expected to be the most common method of destroying a tank, as it can be done within normal usage of the tank. Alternatively, a tank can be destroyed by using the tank_destroy operation; however, this operation requires the emergency tap authority and is therefore unlikely to be frequently used in practice.

To incentivize the destruction of tanks that are no longer being used, a deposit of core asset is required to create a new tank. This deposit is held for the lifetime of the tank, and is released when the tank is destroyed. The deposit is returned directly to the account which pays the fee for the operation that destroys the tank; it is not sent through a tap.

Restricted Assets Some assets specify account whitelists or blacklists, to restrict which accounts may transact in that asset. It is desired that these assets be able to utilize the Tanks and Taps infrastructure without opening the possibility of using Tanks and Taps to circumvent the restrictions on asset ownership; therefore, a restricted asset check will be enforced in three instances. First, when asset is released through a connection to an account balance, a restriction check will be performed on that account. Second, every time a tap is opened, the account which paid the fee on the transaction causing the tap to be opened is checked for authorization to use the released asset. Third, when a connection receives asset from an account, that account checked for authorization to handle the asset. If any of these authorization checks fails, the transaction is rejected as invalid. With these checks in place, restricted assets can be used in conjunction with the Tanks and Taps architecture without compromising the efficacy of the asset restrictions.

Pseudocode definitions of these types are provided below:

```
attachment_id_type {
/// ID of the tank hosting the attachment
                    tank_id_type tank_id;
                    /// ID of the attachment on the tank
                    uint16 attachment_id;
                    }
```

```
type connection = static_variant<tank_id_type,
account_id_type, attachment_id_type>;


struct all_sources{};

type authorized_connections_type =
static_variant<flat_set<connection>,
all_sources>;


struct unlimited_flow{};

type tap_flow_limit = static_variant<share_type,
unlimited_flow>;


asset_flow_meter {

state_type {

/// The amount of asset that has flowed through
the meter

share_type metered_amount;

}

/// The type of asset which can flow through this
meter
```

```
                asset_id_type asset_type;

                /// The connection where the metered asset is
                released to

                connection destination;

                /// What remote sources, if any, can deposit to
                this meter

                authorized_connections_type remote_sources;

                /// The authority which may reset the meter; if
                null, only the emergency tap authority is
                accepted

                optional<authority> reset_authority;

                }

                tap_opener {

                /// Index of the tap to open (must be on the same
                tank as the opener)

                uint16 tap_index;

                /// The amount to release

                tap_flow_limit release_amount;

                /// The connection that asset is released to after
                flowing through the opener
```

```
connection destination;

/// What remote sources, if any, can deposit to
this opener

authorized_connections_type remote_sources;

}

attachment_connect_authority {

/// Authority that may reconnect an attachment

authority connect_authority;

/// Attachment which may be reconnected

attachment_id_type attachment_id;

}


type tank_attachment =
static_variant<asset_flow_meter, tap_opener,

attachment_connect_authority>;


immediate_flow_limit { share_type limit; }

cumulative_flow_limit {

struct state_type {
```

```
/// The amount of asset released so far
share_type amount_released;
};
share_type limit;
}
periodic_flow_limit {
state_type {
/// Sequence number of the period during which the last withdrawal took place
uint32 period_num = 0;
/// The amount released during the period
share_type amount_released;
}
/// Duration of periods in seconds
uint32 period_duration_sec;
/// Maximum cumulative amount to release in a given period
share_type limit;
}
```

```
time_lock {

/// If true, the tap is initially locked

bool start_locked;

/// At each of these times, the tap will switch between locked and unlocked --

/// must all be in the future

vector<time_point_sec> lock_unlock_times;

}

minimum_tank_level {

/// Minimum tank balance; tap cannot drain tank below this balance

share_type minimum_level;

}

review_requirement {

/// This type describes a request for withdrawal waiting for review or

/// for redemption

request_type {
```

```
/// Amount requested for release -- reset if
reviewer denies request

optional<tap_flow_limit> request_amount;

/// Optional comment about request, max 150
chars -- reset if reviewer

/// denies request

optional<string> request_comment;

/// Starts false, set to true if request is approved,
and back to false after

/// a release of funds

bool approved;

}

state_type {

/// Number of requests made so far; used to
assign request IDs

uint16 request_counter;

/// Map of request ID to request

flat_map<uint16, request_type>
pending_requests;

}
```

```
                /// Authority which approves or denies requests

                authority reviewer;

                /// Maximum allowed number of pending
                requests; zero means no limit

                index_type request_limit;

                }

                documentation_requirement {

                /* no fields; if this requirement is present,
                evaluator requires a nonempty

                * documentation argument exist, but does no
                further verification upon it

                */

                }

                delay_requirement {

                /// This type describes a request for withdrawal
                waiting for its delay to pass

                request_type {

                /// When the request matures and can be
                consumed

                optional<time_point_sec> delay_period_end;
```

```
/// Amount requested

optional<tap_flow_limit> request_amount;

/// Optional comment about request; max 150
chars

optional<string> request_comment;

}

state_type {

/// Number of requests made so far; used to
assign request IDs

uint16 request_counter;

/// Map of request ID to request

flat_map<uint16, request_type>
pending_requests;

}

/// Authority which can veto request during
review period; if veto occurs,

/// reset state values

optional<authority> veto_authority;

/// Period in seconds after unlock request until
tap unlocks; when tap opens,
```

```
                /// all state values are reset

                uint32 delay_period_sec;

                /// Maximum allowed number of outstanding
                requests; zero means no limit

                uint16 request_limit;

                }

                hash_preimage_requirement {

                type hash_type = static_variant<sha256,
                ripemd160, hash160>;

                /// Specified hash value

                hash_type hash;

                /// Size of the preimage in bytes; a preimage of a
                different size will be rejected

                /// If null, a matching preimage of any size will
                be accepted

                optional<uint16> preimage_size;

                }

                ticket_requirement {

                /// The type of the ticket that must be signed to
                unlock the tap
```

```
struct ticket_type {

/// ID of the tank containing the tap this ticket is for

tank_id_type tank_id;

/// Index of the tap this ticket is for

uint16 tap_index;

/// Maximum asset release authorized by this ticket

tap_flow_limit max_withdrawal;

/// Must be equal to tickets_consumed to be valid

uint16 ticket_number;

}

state_type {

/// Number of tickets that have been used to authorize a release of funds

uint16 tickets_consumed;

}

/// Key that must sign tickets to validate them
```

```
                                public_key_type ticket_signer;

                                }

                                exchange_requirement {

                                /// The maximum release amount will be:

                                /// meter_reading / tick_amount *
                                release_per_tick - amount_released

                                state_type {

                                /// The amount of asset released so far

                                share_type amount_released;

                                }

                                /// The ID of the meter to check

                                tank_attachment_id_type meter_id;

                                /// The amount to release per tick of the meter

                                share_type release_per_tick;

                                /// Amount of metered asset per tick

                                share_type tick_amount;

                                /// Authority which can reset the amount
                                released; if null, only the
/// emergency tap authority is authorized
```

```
optional<authority> reset_authority;

}


type tap_requirement = static_variant<immediate_flow_limit, cumulative_flow_limit,

periodic_flow_limit, time_lock, minimum_tank_level,

review_requirement, documentation_requirement,

delay_requirement, hash_preimage_requirement,

ticket_requirement, exchange_requirement>;


                    tap {

                    /// The connected connection, if present

                    optional<connection> connected_connection;

                    /// The authority to open the tap; if null, anyone can open
                    the tap if they can

                    /// satisfy the requirements

                    optional<authority> open_authority;

                    /// The authority to connect and disconnect the tap. If unset,
                    tap must be connected

                    /// on creation, and the connection cannot be later modified
                    -- emergency tap must
```

```
/// specify a connect_authority

optional<authority> connect_authority;

/// Requirements for opening this tap and releasing asset

vector<tap_requirement> requirements;

/// If true, this tap can be used to destroy the tank when it
empties

bool destructor_tap;

}


tank_object {

/// Amount of asset contained in the tank

asset contained_asset;

/// Taps on this tank. ID 0 must be present, and must not
have any tap_requirements

flat_map<uint16, tap> taps;

/// Counter of taps added; used to assign tap IDs

uint16 tap_counter;

/// Attachments on this tank

flat_map<uint16, tank_attachment> attachments;
```

/// Counter of attachments added; used to assign attachment IDs

uint16 attachment_counter;

/// What remote sources, if any, can deposit to this tank

authorized_connections_type remote_sources;

/// Amount of the deposit paid to create the tank (deposit is always core asset)

share_type deposit_amount;

}

This proposal additionally seeks to add eight operation types to the protocol:

tank_create Creates a new tank, specifying taps, tap requirements, and sink connections

tank_update Updates a tank, adding or removing taps (including requirements and sinks)

tank_delete Deletes a tank (fails if the tank contains asset)

tank_query Pass arguments to tap requirements or tank attachments to perform actions or comply with requirements

tap_open Release funds through a tap, passing arguments as necessary to satisfy tap requirements

tap_connect Connect a tap to a destination

account_fund_connection Move asset from an account to a connection (such as a tank)

connection_fund_account Virtual operation, generated when a connection deposits asset into an account balance

Pseudocode definitions of these operations are provided below:

tank_create {

/// Account that pays fee and deposit

account_id_type fee_payer;

/// Deposit of core asset paid to create the tank

share_type deposit_amount;

/// Type of asset the tank will contain

asset_id_type tank_asset;

/// Taps on the tank -- index 0 must be present and must have no tap_requirements

vector<tap> taps;

```
/// Attachments on the tank
vector<tank_attachment> attachments;
/// Sources that are authorized to deposit to the tank
authorized_connections_type authorized_sources;
}
tank_update {
/// Account that pays fee
account_id_type fee_payer;
/// Authority to update tank: must match tank->taps[0].open_authority
authority update_authority;
/// ID of tank to update
tank_id_type tank_to_update;
/// Change in deposit amount on tank; credited or debited to fee payer
share_type deposit_delta;

/// IDs of taps to remove
flat_set<uint16> taps_to_remove;
```

```
/// Map of ID-to-new-value for taps to replace

/// Note that state data for all requirements of replaced taps
will be deleted

flat_map<uint16, tap> taps_to_replace;

/// New taps to add

vector<tap> taps_to_add;



/// IDs of attachments to remove

flat_set<uint16> attachments to remove;

/// Map of ID-to-new-value for attachments to replace

/// Note that state data for replaced attachments will be
deleted

flat_map<uint16, tank_attachment>
attachments_to_replace;

/// New attachments to add

vector<tank_attachment> attachments to add;



/// Set this field to replace the tank's deposit source
authorizations
```

```
optional<authorized_connections_type>
new_authorized_sources;

}

tank_delete {

/// Account that pays fee

account_id_type fee_payer;

/// Amount of deposit reclaimed for deleting tank

share_type deposit_amount;

/// Authority to delete tank: must match tank-
>taps[0].open_authority

authority delete_authority;

/// ID of tank to delete; tank must be empty of asset to
destroy

tank_id_type tank_to_delete;

}

tank_query {

/// Account that pays the fee

account_id_type fee_payer;

/// Authorities required to perform the queries
```

```
flat_set<authority> required_authorities;


/// ID of the tank to be queried

tank_id_type tank_id;

/// Contents of the queries to run (format is implementation-
defined)

vector<tank_query_type> queries;

}

tap_open {

/// Account that pays the fee

account_id_type fee_payer;

/// Authorities required to satisfy the tap's
requirements/open authority

flat_set<authority> required_authorities;

/// Any queries which should be run prior to opening the tap

/// (format is implementation-defined)

vector<tank_query_type> queries;


/// ID of the tank with the tap to be opened
```

```
tank_id_type tank_id;

/// Index of the tap to open

uint16 tap_index;

/// Amount to release from the tap

tap_flow_limit release_amount;

/// Total number of taps opened by this transaction, i.e. due to tap_openers

uint16 taps_to_open;

/// If emptying tank via a destructor tap, the deposit is returned to fee_payer

/// Specify amount of deposit here to enable tank destruction

optional<share_type> claimed_deposit;

}

tap_connect {

/// Account that pays the fee

account_id_type fee_payer;

/// Authority to connect the tap: must match

/// tank_id->taps[tap_index].connect_authority
```

```
authority connect_authority;

/// ID of the tank holding the tap to be opened

tank_id_type tank_id;

/// Index of the tap to be connected

uint16 tap_index;

/// Connection to connect the tap to; if null, tap will be
disconnected

optional<connection> new_connection;

/// Clear the tap connect authority after connecting the tap;
if true,

/// new_connection must be specified

bool clear_connect_authority;

}

account_fund_connection {

/// Account that provides funds, and pays fee; must be
authorized to handle asset

account_id_type funding_account;

/// Destination for the funds

connection funding_destination;
```

```
/// Amount that the account is moving through the connection

asset funding_amount;

}

connection_fund_account {

/// Account receiving funds

account_id_type recipient;

/// Amount the recipient received

asset amount_received;

/// The path of connections the asset took to arrive at the account, including the origin

vector<tnt::connection> asset_path;

}
```

Additionally, these new committee parameters will be defined:

max_deposit_path_length The maximum number of connections in a deposit path

max_taps_to_open The maximum number of taps a single transaction can open

tank_deposit_amount The amount of CREDIT required for a tank deposit Further parameters may be defined to control the amounts of tank deposits based on the details of the tank.

Tank Queries and Arguments

Many of the tap requirements and tank attachments support modes of user interaction which may or may not result in the immediate movement of asset. Those interactions which do cause movement of asset are performed using the tap_open operation, and those which do not are performed using the tank_query operation. Both of these operations will accept arguments specifying what actions should be taken on which requirements/attachments.

Initially, the following tank attachment queries will be supported:

asset_flow_meter

Reset the meter to zero (No arguments)

attachment_connect_authority

Reconnect the specified attachment (Accepts a connection argument)

The following tap requirement queries will be supported:

review_requirement

Create a request to open the tap (Accepts a request_type argument and an optional string reason)

Resolve a request to open the tap (Accepts a request ID, a boolean decision, and an optional string reason)

Cancel a request (Accepts a request ID and an optional string reason)

Consume an approved request and open the tap (Accepts a request ID)

documentation_requirement

Document the reason for the release of asset (Accepts a string reason)

delay_requirement

Create a request to open the tap (Accepts a request_type argument)

Veto a request to open the tap (Accepts a request ID and an optional string reason)

Cancel a request (Accepts a request ID and an optional string reason)

Consume a matured request and open the tap (Accepts a request ID)

hash_preimage_requirement

Reveal the hash preimage and open the tap (Accepts a data buffer)

ticket_requirement

Redeem a ticket and open the tap (Accepts a ticket_type argument)

exchange_requirement

Reset the amount released (No arguments; meter is required to be at zero when processing)

Please note that while this specification attempts to provide a sufficient level of technical detail to convey the essence of Tanks and Taps, some detail has been elided for brevity, and the final implementation may diverge from the specification in order to improve the correctness, stability, efficiency, maintainability, or functionality of the Tanks and Taps framework.

Discussion and Summary for Shareholders

The proposed modifications to the AcloudBank protocol will add eight new operation types, one new database object, and several new committee-controlled chain parameters. These modifications add new features, and do not modify or restrict any existing features, and therefore all currently supported use cases should be unaffected. In return, however, the new features will provide a powerful platform for financial dapp and smart contract development, and this platform can be augmented to become even more powerful in the future, requiring only minimal changes to support additional use cases. The proposed changes, in conjunction with future updates, will make AcloudBank increasingly suitable as a financial dapp platform, notably offering dapp developers simplified development and a

reduced time to market as compared with Turing Complete Smart Contract Platform alternatives which offer greater flexibility at the price of increased complexity.

Copyright

This document is created for the betterment of humanity and is hereby placed into the public domain.