CottonAlley

/ k t.ən æl.i/

Reto Furrer, Sascha Häring, Manuel Metzler

December 04, 2023

Agenda

- Vision
- Internet Computer
- Product
- Architecture
- Components

Section 1

Vision

Idea

- Autonomous marketplace
- Dual-deposit system
- Business logic running on the Internet Computer

Goals

- Tamper-Proof
- Privacy-Focused
- Cloud-Native
- Mobile-Compatible

Section 2

Internet Computer

Overview

Architecture

- Data Centers
- Nodes
- Subnets
- Canisters

IC Protocol

- Peer-to-peer
- Consensus
- Message Routing
- Execution

Canister

Capabilities

- Execute code
- Manage their own state
- Expose endpoints
- Can be called via HTTP

Endpoints

- Update
- Query

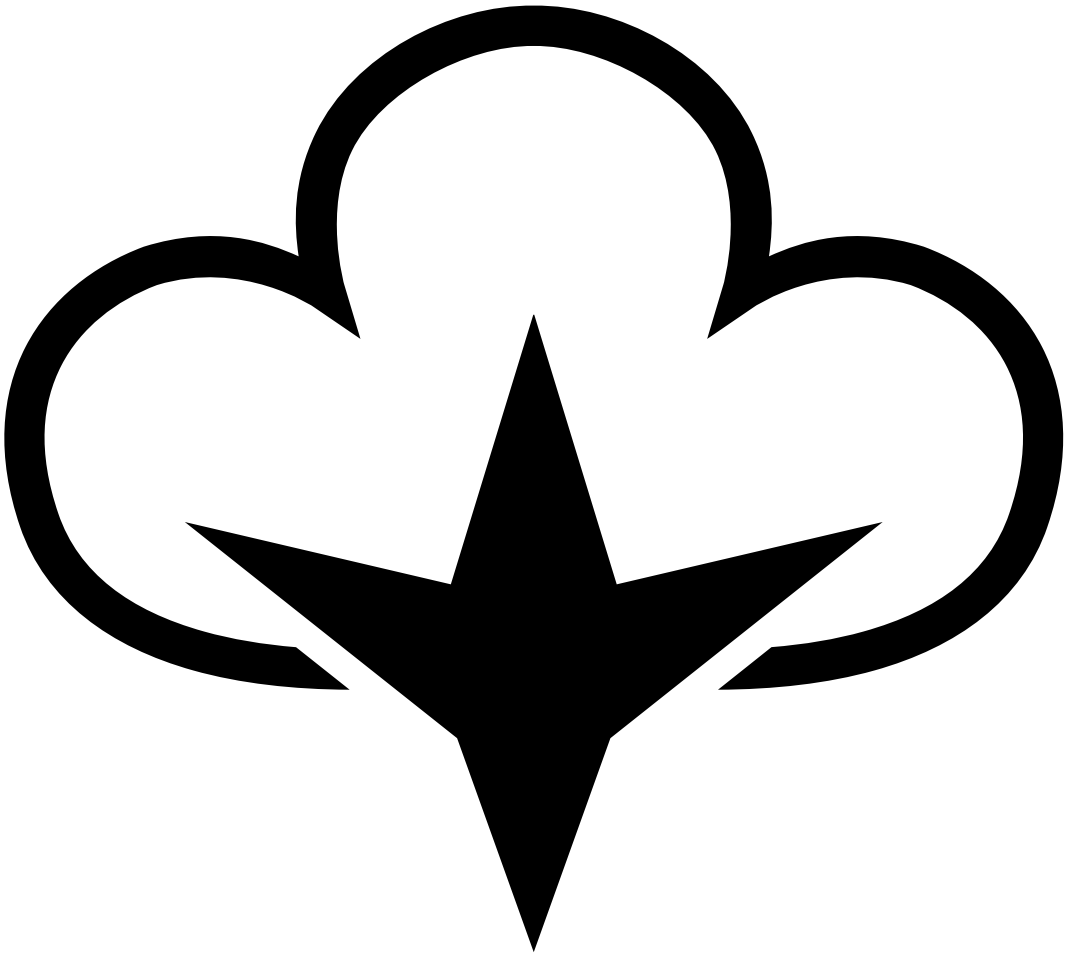
Gas

- Reverse Gas Model
- Cycles (tied to fiat currency)

Section 3

Demo

Demo



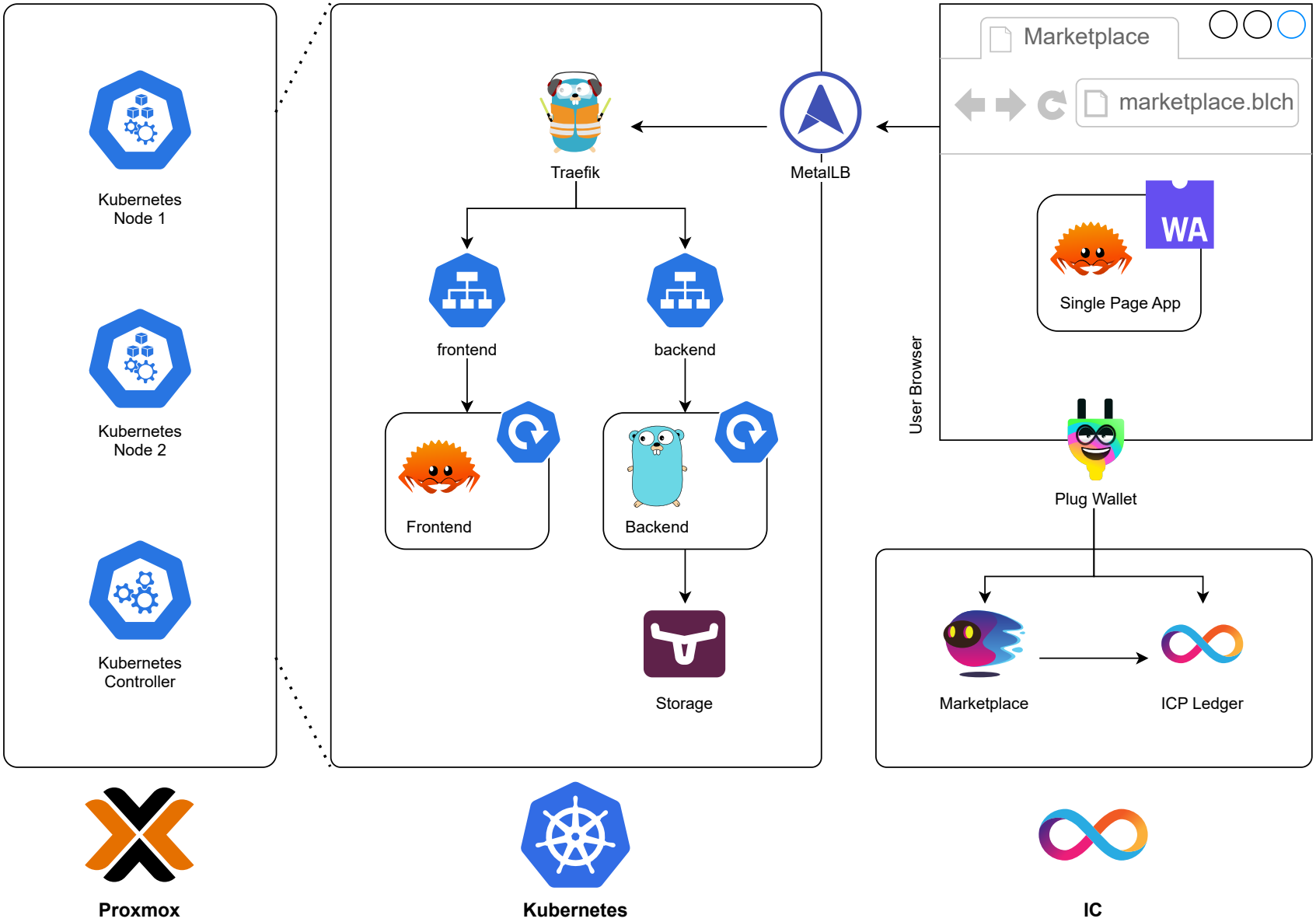
Section 4

Architecture

Concepts

- Fully on-chain business logic
- Stateless backend
- Single page application

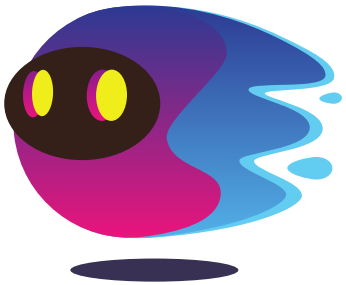
Overview


Kubernetes Node 1
Kubernetes Node 2
Kubernetes Controller
Traefik
MetalLB
frontend
backend
Frontend
Backend
Storage
Marketplace
marketplace.blch
Single Page App
WA
User Browser
Plug Wallet
Marketplace
ICP Ledger
Proxmox
Kubernetes
IC

Section 5

Parts

Canister



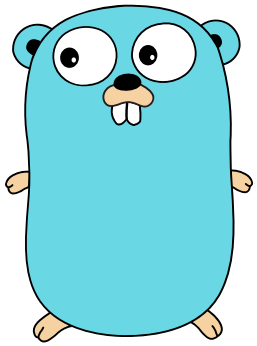
Purpose

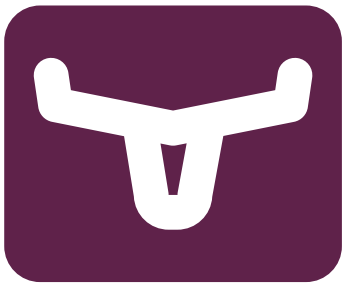
- Manage offers and payments
- Implement dual-deposit
- Verify correct transactions

Details

- Exposes Update and Query endpoints
- Manages it's own state

Backend





Purpose

- Serve canister address
- Save and server offer details

Details

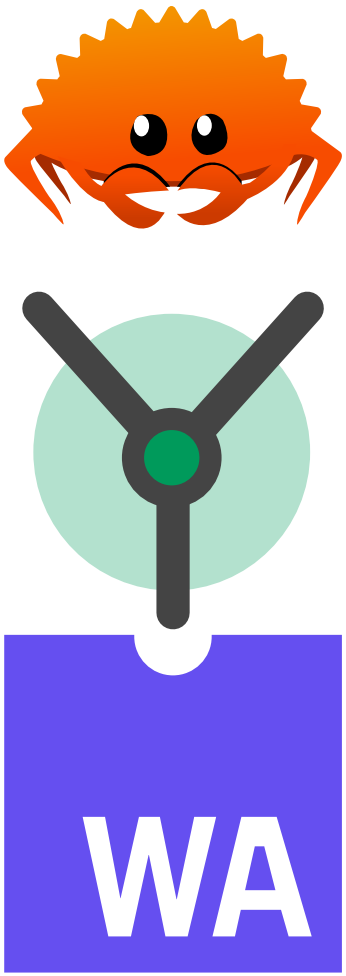
- Exposes RESTful-API
- Longhorn storage backend

Frontend

Building backend services in frontend languages

Building frontend services in

Frontend



Purpose

- Simple UI for the canister
- Buy and create capabilities

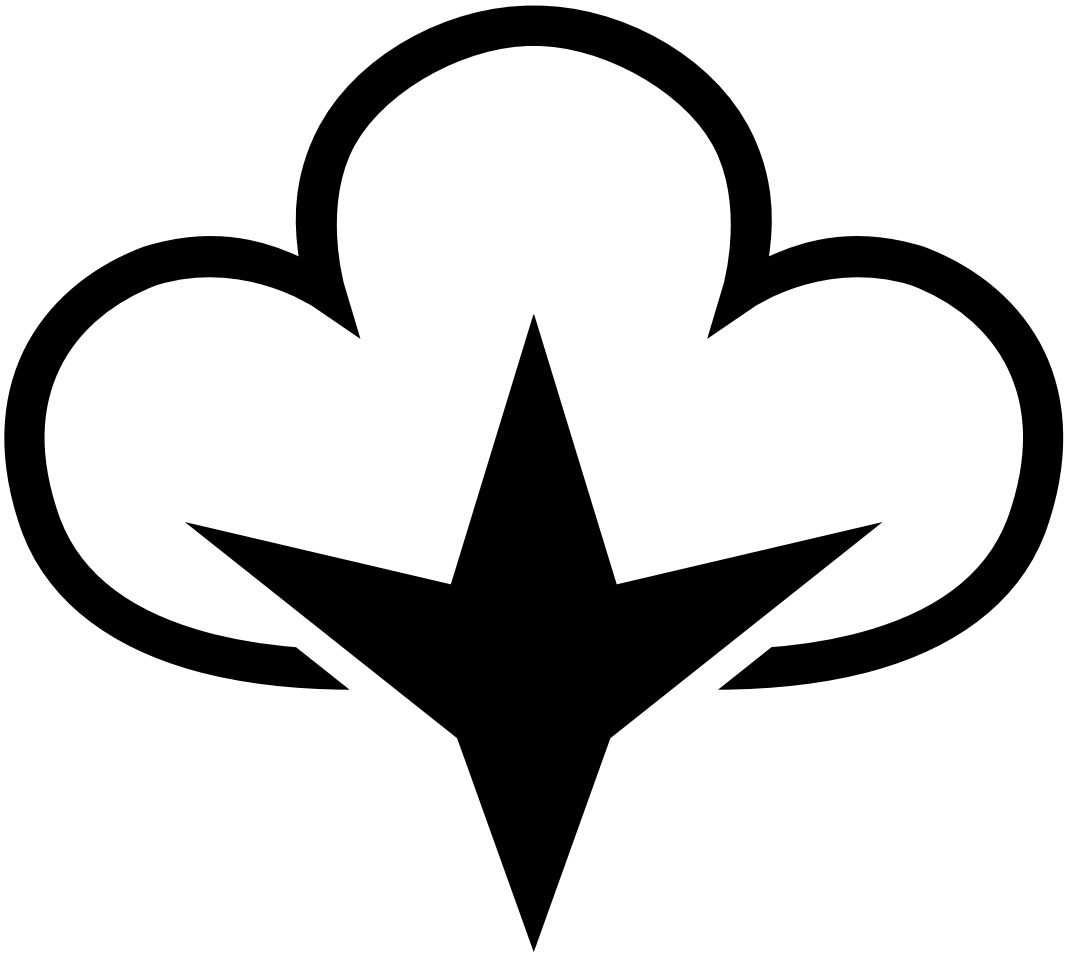
Details

- Connects to Plug
- Interacts with the canister and the ledger
- Calls the backend to get offer details

Section 6

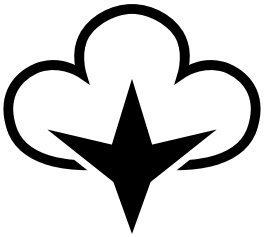
Conclusion

Conclusion



Blockchain Challenge Task

CottonAlley



Reto Furrer, Sascha Häring, Manuel Metzler

November 30, 2023

Contents

1 Overview

This document describes the architecture, infrastructure, and design choices for our Blockchain Challenge Task Project, CottonAlley. The project's goal was to develop an application running on the IC, enabling users to create offers for digital assets such as photos, product keys, or software.
Other key requirements for our platform included:

Tamper-Proof The entire business logic is implemented on the blockchain, reducing possible attempts to manipulate the offers.

Privacy-Focused Users can create offers without publishing unnecessary data, a key requirement for a privacy-friendly environment.

Cloud-Native All non-blockchain components, namely the backend, frontend, and storage, are horizontally scalable.

Mobile-Compatible As mobile phones are frequently used for asset transactions nowadays, the developed frontend should run on both mobile devices and the web.

1.1 Technologies

Given that this project serves as a submission for the Challenge Task, we wanted the technology stack to not only addresses the task at hand but also allows us to learn and experiment with unfamiliar and new technologies and software. With this in mind the following technologies were selected for our project.

Internet Computer [1] is a blockchain that can host and execute decentralized applications. It's goal is to make the internet the primary distributed computing platform. The IC is used for the deployment and execution of the CottonAlley canister.

Motoko [2] is a specialized programming language tailored for the Internet Computer. Designed to abstract unique IC features, Motoko has native support for execution as a canister. It is the language used to develop the canister.

Web Assembly [3] serves as a binary execution format compatible with common web browsers. Languages like Motoko and Rust can compile to WASM. The frontend of CottonAlley is served as a WASM binary for efficient execution.

Rust [4], known for its speed and memory safety, is the primary language used for the frontend. It prioritizes performance, type safety, and concurrency. The CottonAlley fronted is compiled to WASM for efficient execution in web browsers.

Kubernetes [5] is the currently leading container orchestration system. It plays a central role in automating the deployment, scaling, and management of software using pods. For CottonAlley, Kubernetes handles the deployment of both backend and frontend servers.

Go [6] is a statically typed and garbage-collected programming language. It focuses on simplicity and scalability, providing a robust foundation for building scalable systems. It was chosen for developing the backend application.

Although the team was unfamiliar with IC, Motoko, WASM, and Rust, both Kubernetes and Go were already familiar to the team. This intentional decision was made to avoid depending entirely on new technologies.

1.2 Tools and Libraries

In addition to the mentioned technologies, we used various tools for the development and deployment of our final product. The utilized tools include the following.

Ansible [7], a configuration management tool, is used for config management.

Cilium [8], a container networking tool, ensures secure inter-pod communication.

Kubespray [9], dedicated to Kubernetes deployment, configures the cluster.

Longhorn [10], specializing in distributed block storage, acts as the storage provider.

MetalLB [11], a bare-metal load balancer, serves as the ingress target.

OpenTofu [12], an Infrastructure as Code tool, facilitates infrastructure deployment.

Progressive Web App [13] enables web app installation on smartphones.

Proxmox [14], a hypervisor, used to automate Kubernetes cluster deployment.

Traefik [15], a reverse proxy and load balancer, serves as the ingress controller.

Yew [16], a frontend framework, is used for developing the frontend of our application.

1.3 Architecture

Figure 1 illustrates the architecture of CottonAlley. Subsequent to this graphic, the chapters will explain the individual components.

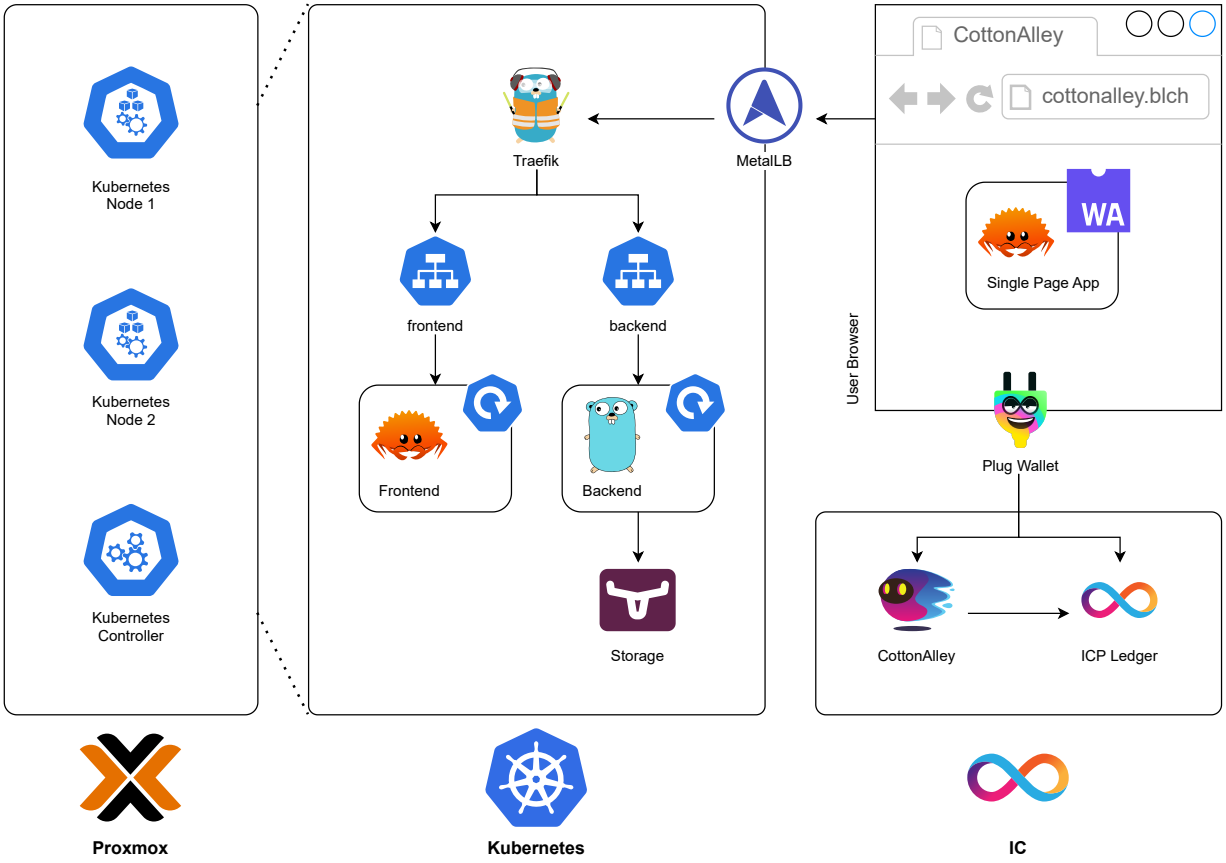


Figure 1: Architecture of CottonAlley

2 Internet Computer

Introduction The Internet Computer is a blockchain designed to host canister smart contracts, eliminating the need for traditional IT infrastructure [17]. Available on the public internet since May 2021, the IC aims to make the internet not only the primary distributed networking platform but also the primary distributed computing platform. Other goals include hosting Web3 services, decentralization, achieving horizontal scalability and optimizing performance. This chapter will provide an overview of the Internet Computer and its functionality.

2.1 Architecture

Layers The architecture consists of four main layers. At the bottommost layer reside all *data centers* that contribute to the hosting of nodes and can each host one or more nodes. These *nodes*, building the second layer, are the machines responsible for running and executing the Internet Computer. Multiple nodes are then grouped into various independent and concurrently running *subnetworks*. Finally, each subnet has the capacity to host and execute thousands of *canister smart contracts* for the IC. The entire system is managed by the Internet Computer Protocol or ICP for short The IC Protocol is also where the name for the utility token comes from.

IC Protocol The Internet Computer Protocol itself also consists of four layers: Peer-to-peer, Consensus, Message Routing, and Execution. [17]

Peer-to-peer The bottommost layer enables communication between all nodes within a subnet. It also transmits artifacts, representing various types of messages such as user input, block messages, or proposals. This layer operates based on IP.

Consensus Responsible for accepting messages and determining their order, the consensus layer ensures the deterministic execution of canisters on every node within a subnet. The implementation of the consensus algorithm guarantees cryptographically assured finality.

Message Routing The message routing component enables the inclusion of messages from a consensus-derived block into the input queues of target canisters. It also handles the routing of messages from canisters output queues to their respective recipients, which may include canisters on different subnets. This process is known as cross-subnet messaging (XNet).

Execution As the topmost layer, the execution layer is responsible for executing canisters. Unlike other blockchains, it can split the execution of a canister into multiple rounds and has the capability to execute code concurrently. Additionally, it has access to an unpredictable and unbiased pseudorandom number generator.

2.2 Canisters

Capabilities Canister smart contracts, or simply canisters, are the counterparts of smart contracts on the Internet Computer (IC). In contrast to smart contracts on other blockchains, canisters have the unique capability to manage their own state. The code of a canister is a WebAssembly module, and its state includes the WASM module's heap and stable memory. However, the maximum available storage for each canister is capped at 96 GiB, with an additional 4 GiB for heap storage. Additionally, canisters have access to a distributed pseudorandom generator per subnet [17].

Endpoints All canisters expose endpoints that can be invoked by other canisters or actors from outside of the IC, such as mobile browsers or apps. These endpoints are asynchronous to ensure non-blocking behavior. There are two different types of endpoints: update and query. [17]

Update Endpoints Update endpoints are functions that can modify the canister's state, and any changes made must be persisted. These endpoints are executed by multiple nodes on the chain, and all nodes must reach the same result for the call to be valid. The number of nodes involved in calculating a call to an update endpoint ranges from about 13 to 34, depending on the subnet. If an error occurs or the canister traps during execution, the state is rolled back to the state after the last successful update call.

Query Endpoints Query endpoints are functions that can also change the state of the canister during execution, but the state reverts as soon as the function completes. All query endpoints are executed by a single node in a subnet, introducing the risk of potential data tampering by that node. To address this risk without compromising query speed, the returned data can be signed. This signature has to be precalculated each time an update call changes the canister's state, allowing query endpoints to be executed by only one node while ensuring consensus-backed results.

Updates Another notable capability of canisters is their ability to update the code or delete the canister [17]. When a canister's code is updated, the contents of the heap memory are deleted, while the stable memory contents persist through updates. Typically, canister updates are managed by the DAO, but the IC also supports updates from a central point or can enforce immutability.

Deployment In order to deploy a canister on the IC's mainnet, there are various possibilities. One common method is to use the dfx command. The dfx command can interact with the IC, allowing actions such as canister creation, deployment, and management. Regardless of the chosen deployment method, the following steps are required in all of them.

1. A new empty canister must be created on the IC with the needed amount of cycles. The canister creation typically costs approximately 1 terracycles.
2. The project needs to be compiled into a WASM module, and a Candid file must be generated for this module.
3. Both the WASM module and the Candid file must be copied into the canister.

Once these steps are completed, the canister is ready to be called, provided there are sufficient cycles within the canister to pay for its execution.

2.3 Tokens and Cycles

Token The IC has its native utility token known as the ICP Token, serving multiple roles within the network. Firstly, ICP is used for staking to enable voting on or proposing governance changes, with participants earning voting rewards. Additionally, ICP can be converted into cycles to pay for various resources, compensate node providers, participate in token swaps of DAOs on the IC, and even for the exchange of goods. [17]

Cycles As mentioned, ICP tokens are convertible to cycles, which act as fuel for computation, bandwidth, and other resources on the IC [17]. This conversion is done to provide a reliable cost prediction, as cycles are tied to fiat currencies (specifically XDR), unlike ICP tokens. The IC follows a "reverse gas model", where developers prepay costs by loading canisters with cycles [17]. This allows users to interact with the dapp without having to pay, as all necessary cycles for computation, storage, and other resources are supplied by the canister.

Rewards ICP tokens play are also used for rewarding node providers and voters [17]. Node providers receive rewards in newly minted ICP, with the amount specified in a fiat currency (XDR). This mechanism ensures fair compensation for node providers based on their real-world expenses.

3 Canister

Introduction The canister is the core component of the CottonAlley. It contains the entire business logic and ensures the integrity of offers. This helps against tampering, as every offer must undergo the consensus process of the IC when created or purchased. A core aspect of the canister is the dual-deposit system, which enables trustless transactions for digital assets between two parties, eliminating the need for intermediaries. This chapter explains both the capabilities of the canister and the dual-deposit system.

3.1 Capabilities

Purpose The canister is designed to enable users to create and purchase offers. He exposes various update endpoints for offer creation and manipulation, as well as query endpoints to retrieve existing offers.

View To obtain an offer, a client can request one of the query endpoints, which returns a list of currently stored offers in the canister. These query endpoints generally offer faster responses compared to the other update endpoints.

Create When creating a new offer the create endpoint can be used. This endpoint is expecting a blockID from the transmit transaction of the ICP Ledger, a SHA256 hash value of the asset to sell, and the Ed22519-PublicKey of the seller. The canister checks the given blockID‘s existence on the ICP Ledger, retrieves the transferred amount to determine the product price, and creates and stores a new offer if all values are correct. Finally the create endpoint returns the ID of the newly created offer on success.

Buy For buying an existing offer, the canister exposes a dedicated endpoint, expecting a blockID for a transmit transaction of the ICP Ledger and the Ed22519-PublicKey of the buyer. The canister checks the request similarly to when creating an offer. After determining the transmitted ICP, it checks whether the amount is twice the price. The offer is then marked as “bought but not completed”.

Asset transmission At this point, the buyer must inform the seller about where to send the asset, and this message must be signed with the private key of the buyer. The seller verifies the buyer‘s identity using the public key provided while calling the buy endpoint. In this way, he can ensure that it is the real buyer from whom he received the message. The buyer then receives the signed asset from the seller. He then checks the signature, and ensures the correct product was sent with the provided hash provided in the offer. The asset transmission takes place via the preferred method by the participants and is therefore not handled by CottonAlley. This allows the flexibility to choose a suitable transmission channel rather then enforcing a specific one.

Refunding Once the transfer is done, and the buyer has checked the asset, he informs the canister about the state by either accepting or rejecting delivery. Upon acceptance, the offer is marked as successfully completed, and both the seller and buyer receive the refund of the asset price, their deposit. In case of rejection, the buyer proves it by sending the signature of the received asset to the canister. Depending on the result of the signature verification executed by the canister, either the seller, the buyer or neither of them get the deposited ICP refunded. This dual-deposit procedure is explained in more detail in the next chapter.

Other endpoints Additionally, the canister offers endpoints not directly required by the buy process. These include endpoints such as retrieving the wallet address, canceling existing offers, or refunding accidentally sent ICPs. The canister also remembers the blockID‘s already used to prevent double spending.

3.2 Dual-Deposit

Source The contract uses a dual-deposit system to secure digital asset transactions without relying on a trusted mediator. This concept was inspired by the paper "Solving the Buyer and Seller's Dilemma", which proposes a dual-deposit protocol for cheat-proof digital asset transactions. [18] However, our implementation differs mostly due to initial challenges in on-chain signature verification. One important factor that must be addressed before publishing the system for productive use, is to prevent potential replay attacks.

Concept The dual-deposit system requires both the seller and buyer to deposit a specific amount for both creating and buying offers. The CottonAlley implementation requires both parties to deposit an amount equivalent to the price of the offered asset. This approach ensures that the deposit aligns proportionally with the asset's price, in order to maintain the effectiveness of the protocol. For example, a deposit of 1 ICP for an asset valued at 1'000 ICP would not be sufficient to be effective.

Offering When creating an offer, the dual-deposit system requires the inclusion of the public key and product hash. The product hash is needed for verifying potential cheating by the seller, buyer, or both, while the public key serves the purpose of enabling the buyer to check the asset signature. Additionally, the canister utilizes the public key to determine any potential cheating party.

Buying Similarly, when buying an existing offer, the buyer must specify their public key. Unlike the seller's public key, the buyer's public key is used solely for the buyer to verify the seller's identity. This allows both parties to establish a communication channel, ensuring each party recognizes their counterpart as the seller or buyer, respectively.

Transferring The seller is required to send the offered asset together with a signature. This allows the buyer to verify the signature and checksum against what was provided during offer creation. There are different possibilities depending on the outcome of this verification process.

Accepting When the buyer accepts the transmission of the assets, regardless of whether the transmission has been successfully fulfilled, everything is considered okay, and both parties receive their deposits back. Additionally, the seller also receives the purchasing price.

Rejecting In the case the buyer rejects the transmission, it must be proven by sending the signature of the asset received from the seller.

Valid signature, invalid product hash In this scenario, the seller cheated by sending an asset they weren't offering. Therefore, only the buyer gets their deposit and the purchase price back, as they didn't receive the promised product and didn't cheat. The seller loses their deposit permanently.

Valid signature, valid product hash Here, the buyer attempted to cheat by falsely reporting fraud by the seller. In this case, the seller gets their deposit and the price back, while the buyer's deposit is kept, resulting in a permanent loss for them.

Invalid signature or garbage string In this situation, the canister cannot determine who attempted to cheat. Therefore, both deposits and the price are kept, resulting in a permanent loss for both parties. Keeping the price is necessary because, if the seller was honest and the buyer attempted to cheat by sending a garbage string, the honest seller would be the only one who loses. This outcome would not be ideal.

Limitations This system prevents cheating without losing the deposit, but it's essential to note that if the buyer is willing to lose their money, they could still cheat by rejecting with a random string. While requiring higher deposits could counteract this, it might discourage system usage if deposits are too high. Additionally, the dual-deposit system cannot safeguard buyers when dealing with sellers offering invaluable assets. In such a scenario, the seller could choose neither to accept nor reject the transmission, leading to a loss for both parties. This situation can serve as an incentive for sellers to provide authentic assets.

4 Services and Deployment

Contents CottonAlley consists of various components, as illustrated in Figure 1. This section delves into the explanation of each service. Initially, the canister's operation and purpose are explained. Following that, the backend service's functionality and role are described. Subsequently, the explanation extends to the frontend service. Lastly, there is a brief exploration of Kubernetes and its role in deploying both the frontend and backend services.

4.1 Canister

Purpose The canister serves as the core business logic of CottonAlley. It includes functions used for offer creation, purchasing, cancellation, and completion. Additionally, it exposes functions crucial for managing and verifying the dual-deposit system. Given its crucial role in the system, the dedicated Section 3 Canister provides an in-depth explanation.

Details The canister itself is developed in Motoko, a high-level programming language designed to abstract the features of the IC. This Motoko code is then compiled into a WebAssembly module and a Candid file. Candid is a common interface description language developed by Dfinity. It is utilized to describe data types and endpoints within the Internet Computer. Finally, both the WASM module and the Candid file are deployed to the IC.

4.2 Backend

Purpose The backend functions as the repository for static offer information. This includes all details of an offer that are not saved in the canister, such as offer images, title, description, and the buyer's contact information. It has the ability to store and retrieve these data for a caller. Additionally, the backend service can also provide the canister ID of the CottonAlley canister. This is required for the frontend service to dynamically call the correct canister without hardcoding it in the source code.

Details Implemented in Go, the service exposes a straightforward RESTful-API for interaction. It is designed to ensure that the service itself retains no state other than the static files meant for serving. The application stores these files on a Longhorn volume. As the service is deployed on Kubernetes, this enables the backend to be auto-scaled and replicated as needed.

4.3 Frontend

Purpose The frontend is implemented as a single-page application. It abstracts both, the functionalities of the canister and the backend, presenting a clean interface for user interaction. Its capabilities include creating new offers, including required payments via the ICP Ledger. Naturally the frontend can also handle purchasing existing offers with corresponding payments. Finally it provides access to view and browse all current and past

offers.

Details The frontend is developed in Rust using the Yew web framework. It connects to the IC through the Plug Wallet, a browser-installed web extension. Plug was prioritized over direct calls to the canister function through the provided IC libraries, in order to issue payments from the frontend code without requiring access to the user's private key. Additionally, the service communicates with the backend service to fetch and store static offer data. The frontend is served by a dedicated web server in the form of a WASM binary.

4.4 Kubernetes

Purpose While Kubernetes is not a specific component of CottonAlley, it plays a important role in deploying both the backend and frontend servers. Kubernetes serves as the hosting environment, ensuring HTTP access to the service endpoints, managing the necessary replicas and auto-scaling of services. It also handles TLS certificate management for services exposed via HTTPS. Additionally, Kubernetes hosts Longhorn, utilized by the backend to store its static offer files.

Details The Kubernetes cluster is installed on Proxmox, with MetalLB serving as the ingress target for optimized network communication in setups not hosted on a cloud provider. Traefik serves as the ingress controller, to efficiently route external traffic to our services. For secure inter-pod communication, the Cilium network plugin is installed and configured, offering a robust solution for managing network connectivity. Longhorn, installed as a Helm release, addresses storage requirements and manages the provisioning of persistent storage volumes. Finally, Kubernetes deployments play a crucial role in hosting our backend and frontend servers.

Setup OpenTofu is used to provision new virtual machines on Proxmox to host the Kubernetes cluster. This is necessary due to budget constraints preventing the use of physically distinct computers. Subsequently, Ansible initiates the setup of the new Kubernetes cluster on the previously created VMs. This is accomplished through Kubespray, an Ansible playbook designed to deploy a production-ready Kubernetes cluster. Finally, using kubectl, all services are deployed onto the cluster. At this point both the backend and frontend services are deployed and can be accessed.

5 Conclusion

Plug Wallet Plug is currently in the early stages of development, providing a less-than-optimal user experience. Moreover, its documentation is not comprehensive and sometimes even inaccurate. This makes the plugin feel somewhat alpha, containing UI bugs and incomplete features that are challenging to use.

Internet Computer In contrast, the IC toolchain is more feature-complete than Plug, despite potentially challenging documentation, especially for readers unfamiliar with the key concepts of the IC. Unfortunately, these essential key concepts are documented at a relatively high level and somewhat sparingly. Other functionalities like package management for Motoko have even less documentation. It's important to note that most of the software used is in a beta stage, still pre-version 1.0.0.

Rust Lastly, Rust is a powerful language but can be challenging to learn and master. The difficulty is even higher when used with the Yew web framework, as it requires writing bindings to abstract a dynamically loosely typed language into an extremely strict and statically typed one.

References

[1] Dfinity. *Internet Computer*. 2023. URL: https://internetcomputer.org/.

[2] Dfinity. *Motoko*. 2023. URL: https://motoko.org/.

[3] Web Assembly Community. *Web Assembly*. 2023. URL: https://webassembly.org/.

[4] Rust Team. *Rust*. 2023. URL: https://www.rust-lang.org/.

[5] The Kubernetes Authors. *Kubernetes*. 2023. URL: https://kubernetes.io/.

[6] The Go Authors. *Go*. 2023. URL: https://go.dev/.

[7] RedHat. *Ansible*. 2023. URL: https://ansible.com/.

[8] Isovalent. *Cilium*. 2023. URL: https://cilium.io/.

[9] The Kubespray Special Interest Group. *Kubespray*. 2023. URL: https://kubespray.io/.

[10] Longhorn Authors. *Longhorn*. 2023. URL: https://longhorn.io/.

[11] The MetalLB Contributors. *MetalLB*. 2023. URL: https://metallb.universe.tf/.

[12] Linux Foundation. *OpenTofu*. 2023. URL: https://opentofu.org/.

[13] World Wide Web Consortium. *Web Application Manifest*. 2023. URL: https://www.w3.org/TR/appmanifest/.

[14] Proxmox Server Solutions GmbH. *Proxmox*. 2023. URL: https://proxmox.com/.

[15] Traefik Labs. *Traefik*. 2023. URL: https://traefik.io/traefik/.

[16] Yew Community. *Yew*. 2023. URL: https://yew.rs/.

[17] DFINITY Team et al. “The internet computer for geeks”. In: *Cryptology ePrint Archive* (2022).

[18] Aditya Asgaonkar and Bhaskar Krishnamachari. “Solving the buyer and seller’s dilemma: A dual-deposit escrow smart contract for provably cheat-proof delivery and payment for a digital good without a trusted mediator”. In: *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE. 2019, pp. 262–267.