Solution Architecture

Version 1.1

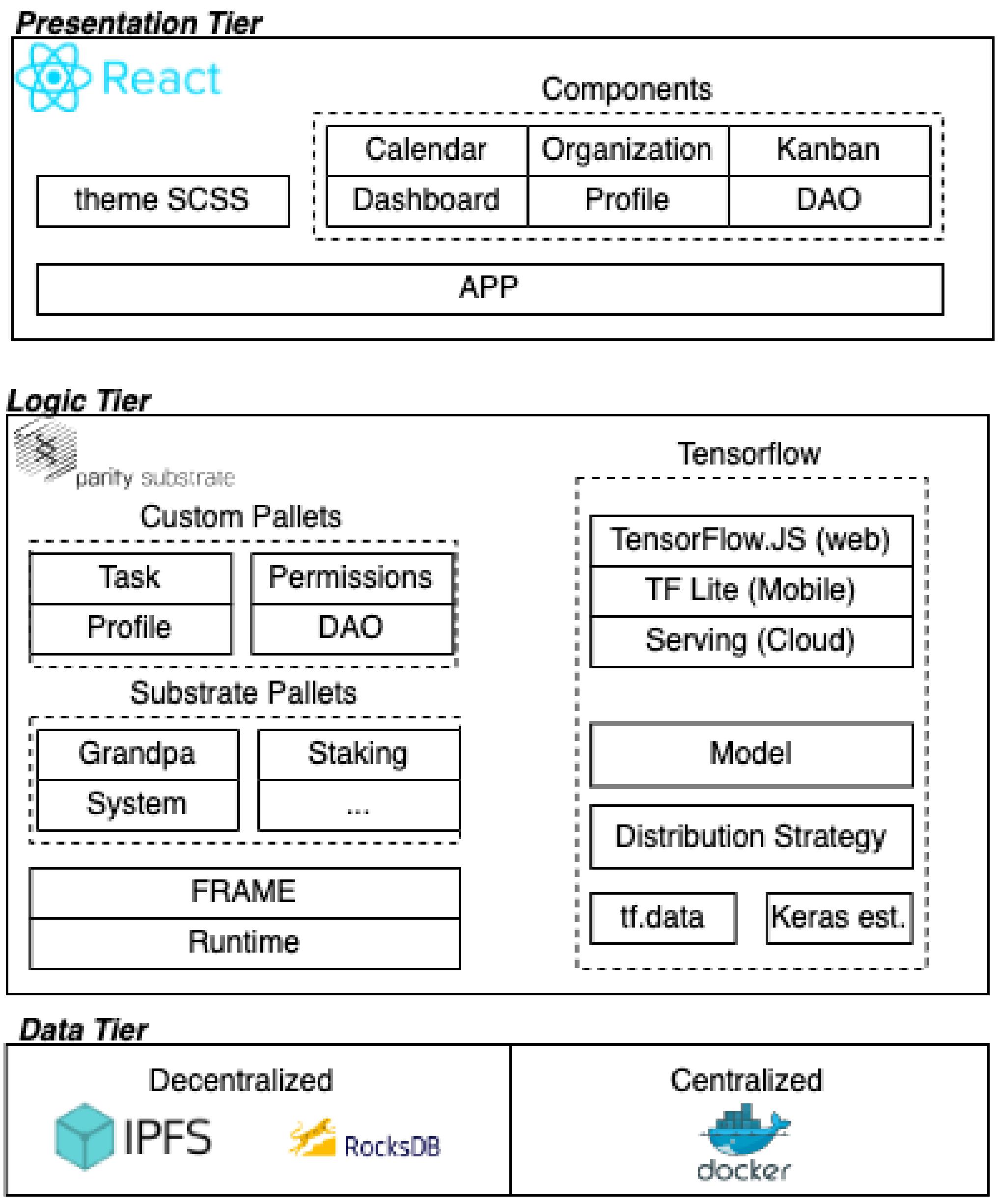
20/04/2022

UNIVERSΛL.

Revision History

Name	Revision comment	Date
Igor Stojanov	Initial draft	28/02/2022
Igor Stojanov	Completed version 1.0	25/03/2022
Igor Stojanov	Replace ActionML with Tensorflow	12/04/2022

Overall Architecture



Substrate

Background

Substrate was created by pioneers and veterans of the blockchain industry who set out to overcome the limitations of previous-generation blockchain networks. It was **born from the vision** that developers shouldn't have to recreate fundamentals when building and optimizing a blockchain. As such it provides the needed tooling to **quickly create decentralized applications**.

Main Features of Substrate

- **Flexible** - a fully modular blockchain framework that unleashes developers instead of forcing them to work within the confines of others' design decisions.
    - **Forkless**: Substrate-based nodes take a different approach that enables automatic upgrades, no user intervention is required.
    - **Fast**: Transaction latency can be alleviated through configurable block times, flexible transaction queues, and/or horizontal scaling. Transaction fees are configurable even to the point of feeless transactions.
- **Open** - uses familiar open protocols such as libp2p and JSON-RPC while letting the developer decide how much they want to customize their architecture.
    - **Tooling**: Great tooling enables faster development, deployment, and debugging. Not only does Substrate have comprehensive, high-quality tooling, it also enables developers to use tooling developed by others, since everything is based on the same underlying framework.
    - **Community**: Substrate has a large, active, and extremely helpful builder community. Many of the pallets have been created by the community itself.
- **Interoperable** - Developers are often forced to make tradeoffs between compatibility, security, and efficiency to interact with data that exists off-chain and cross-chain. This has led to the creation of bridges, oracles, and other interoperability protocols, all with their own limitations.
    - **Compatible**: Substrate-based blockchain networks have the choice of either operating as a solo chain, a solo chain with a bridge, or integrating as a parachain. Integrating as a parachain enables independent Substrate-based blockchains to gain interoperability with the other independent blockchains. The secret sauce of parachain interoperability lies in XCMP (Cross-Chain Message Passing).
    - **Secure**: Substrate chains can inherit security from Substrate-based relay chains like Polkadot or Kusama. As a result, even a small blockchain network can gain billions of dollars in security guarantees.
    - **Efficient**: Because of Substrate's modularity, gas is completely optional, and the introduction of off-chain features greatly reduces computation and storage costs.
- **Future-proof** - Blockchain technology is evolving at a rapid pace and has spurred innovation in other areas such as zero-knowledge, consensus mechanisms, cryptographic libraries, and much more. It's challenging

enough to keep up with technology let alone integrate it. Substrate enables your blockchain to assimilate new technology as it comes along.

- **Upgradable**: With any structure, the foundation is often the most difficult part to modify, and thus upgrade. For this reason, Substrate's base has intentionally been built on an extremely simple and unchanging foundation using the widely-accepted open protocol WebAssembly.
- **Composable**: As a core design principle, composability enables developers to build a blockchain comprised of components specific to their needs on a battle-tested framework.
- **Adaptable**: An ever-growing number of pallets are available, created by both Parity Technologies and the community. These can be combined in many combinations to fit the needs of the desired use case.

Our Addition to Substrate

Substrate is built using **FRAME** which is an extensible framework that composes pallets to create the initial blockchain infrastructure. The **Framework for Runtime Aggregation of Modularized Entities** (FRAME) is a set of modules and support libraries that simplify runtime development. In Substrate, these modules are called **Pallets**, each hosting domain-specific logic to include in a chain's runtime.

UniversalDOT application incorporates many of these Pallets within its runtime. Reusing these pallets enables faster development time, shared security, and communication with other parachains.

At the highest level of abstraction, Substrate has been used within the **UniversalDOT application** to enable the **decentralization** and **trust** features. Many of the composable pallets have been used as-is and we benefit from the added security of the Polkadot and Kusama Ecosystem.

Few layers of custom business logic are developed by the UniversalDot Foundation which enables the creation of **profiles, tasks, and DAOs**. This functionality is developed through pallets which is a composite unit of work within the Substrate Framework.

Profile Pallet Overview

The Profile Pallet creates a user profile per *AccountID*.

The Profile is used to enrich the AccountID information with user-specific metadata such as personal interests, name, reputation, etc.

Task Pallet Overview

The Task Pallet creates a way for users to interact with one another.

Two types of users can interact with tasks. We call them **Initiators** and **Volunteers**.

- **Initiators** are people who have the permission to create and remove Tasks.
- **Volunteers** are people who have the permission to start and complete Tasks.

Anybody can become an Initiator or Volunteer.

DAO Pallet Overview

Organizes **people** with a **common vision** to work on projects. This module works as an extension to the Task module since it enables the creation of large projects which collect many tasks.

A visionary user can **propose a vision** for the future.

Within the vision, a specified **roadmap** is created that is broken down into **tasks**. Thus a DAO is a collection of tasks that are undertaken by people that believe in the vision of the founder.

Users support a vision by signing a vision document. **Signing a vision** document enables users to be added to a DAO where they will be able to create/fulfill tasks in support of the overall vision.

For completion of tasks, users are awarded tokens and enjoy increased reputation.

React

Overview

React (also known as React.js or ReactJS) is a **free** and **open-source front-end JavaScript** library for building user interfaces based on UI components. It is maintained by Meta (formerly Facebook) and a community of individual developers and companies. React can be used as a base in the development of **single-page** or mobile applications. However, React is only concerned with state management and rendering that state to the DOM, so creating React applications usually requires the use of additional libraries for routing, as well as certain client-side functionality.

Integration with Substrate

Communication with Substrate Nodes happens via **Polkadot JS API**. The API provides application developers the ability to **query** a node and interact with the Polkadot or Substrate chains using Javascript.

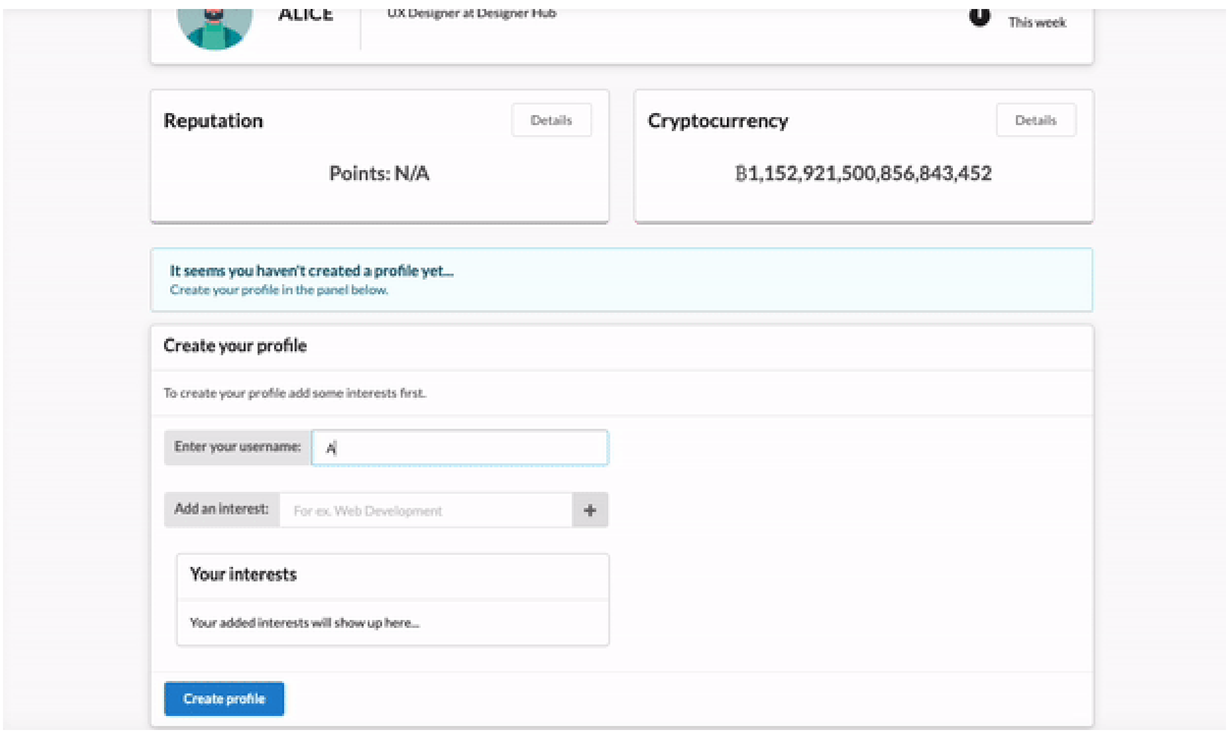
The API is split up into a number of internal packages -

- **@polkadot/api** The API library, providing both Promise and RxJS Observable-based interfaces. This is the main user-facing entry point.
- **@polkadot/api**-derive Derived results that are injected into the API, allowing for combinations of various query results (only used internally and exposed on the Api instances via api.derive.*)
- **@polkadot/metadata** Base extrinsic, storage and constant injectors for injection
- **@polkadot/rpc-core** Wrapper around all JSON-RPC methods exposed by a Polkadot network client
- **@polkadot/rpc-provider** Providers for connecting to nodes, including WebSockets and Http

Type definitions for interfaces as exposed by Polkadot & Substrate clients -

- **@polkadot/types** Codecs for all Polkadot and Substrate primitives

Current Design



ALICE

UX Designer at Designer Hub

This week

Reputation

Details

Points: N/A

Cryptocurrency

Details

Ƀ1,152,921,500,856,843,452

It seems you haven't created a profile yet...
Create your profile in the panel below.

Create your profile

To create your profile add some interests first.

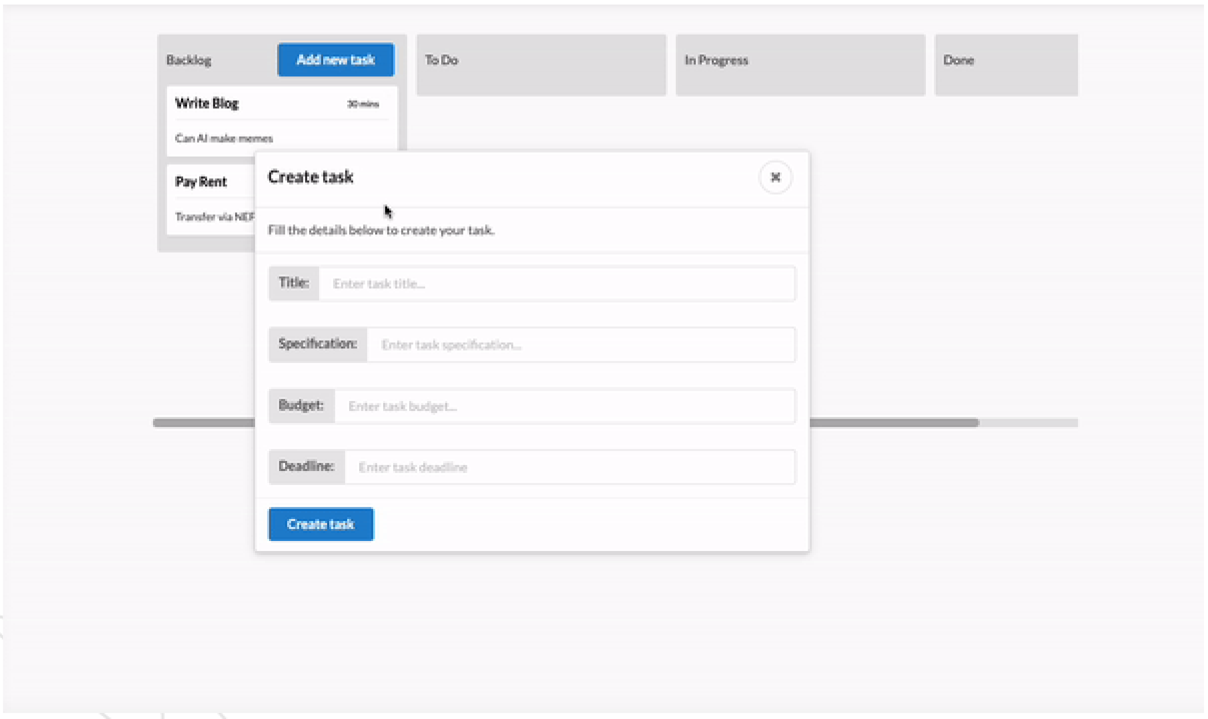
Enter your username: A

Add an interest: For ex. Web Development

Your interests

Your added interests will show up here...

Create profile



Backlog

Add new task

Write Blog 30 mins

Can AI make memes

Pay Rent

Transfer via NEF

To Do

In Progress

Done

Create task

Fill the details below to create your task.

Title: Enter task title...

Specification: Enter task specification...

Budget: Enter task budget...

Deadline: Enter task deadline

Create task

Tensorflow

Overview

TensorFlow Serving is a flexible, high-performance serving system for machine learning models, designed for production environments. TensorFlow Serving makes it easy to deploy new algorithms and experiments, while keeping the same server architecture and APIs. TensorFlow Serving provides out of the box integration with TensorFlow models, but can be easily extended to serve other types of models.

System Architecture

TensorFlow Servables

These are the central rudimentary units in TensorFlow Serving. TensorFlow Servables are the objects that clients use to perform the computation.

The size of a servable is flexible. A single servable might include anything from a lookup table to a single model to a tuple of inference models. Servables can be of any type and interface, enabling flexibility and future improvements such as:

- Streaming results
- Experimental APIs
- Asynchronous modes of operation.

Servable Versions

TensorFlow Serving can handle one or more versions of a servable, over the lifetime of a single server instance. This opens the door for fresh algorithm configurations, weights, and other data to be loaded over time. They also enable more than one version of a servable to be loaded concurrently, supporting gradual roll-out and experimentation. At serving time, clients may request either the latest version or a specific version id, for a particular model.

Servable Streams

A sequence of versions of a servable sorted by increasing version numbers.

TensorFlow API

TensorFlow Models

A Serving represents a model as one or more servables. A machine-learned model may include one or more algorithms (including learned weights) and lookup or embedding tables. A servable can also serve as a fraction of a model, for example, a large lookup table can be served as many instances.

TensorFlow Loaders

Loaders manage a servable's life cycle. The Loader API enables common infrastructure independent from specific learning algorithms, data or product use-cases involved. Specifically, Loaders standardize the APIs for loading and unloading a servable.

Sources in TensorFlow Architecture

Sources are in simple terms, modules that find and provide servables. Each Source provides zero or more servable streams. For each servable stream, a Source supplies one Loader instance for each version it makes available to be loaded.

TensorFlow Managers

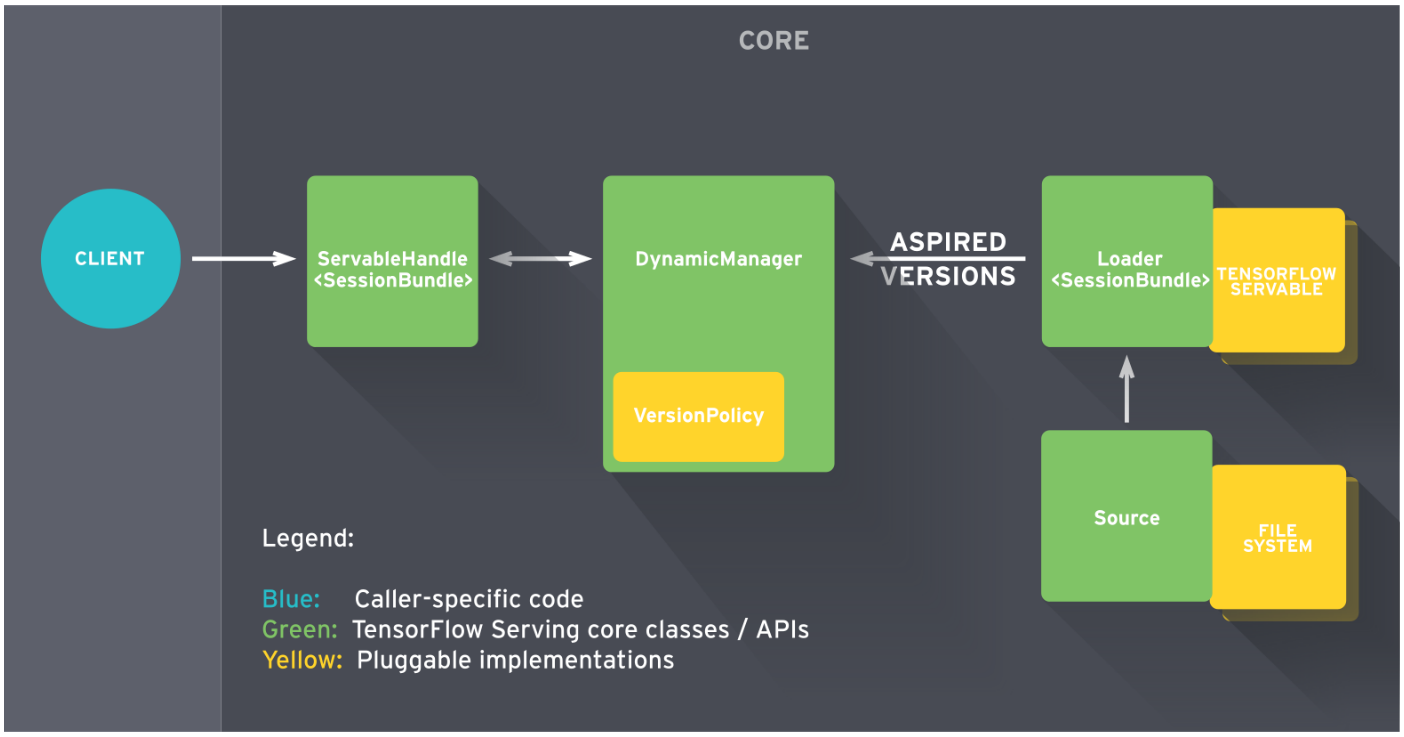
Tensorflow Managers handle the full lifecycle of Servables, including:

- Loading Servables
- Serving Servables
- Unloading Servables

Managers listen to sources and track all versions. The Manager tries to fulfil Sources' requests but, may refuse to load an aspired version. Managers may also postpone an "unload". For example, a Manager may wait to unload until a newer version finishes loading, based on a policy to guarantee that at least one version is loaded at all times. For example, GetServableHandle(), for clients to access loaded servable instances.

TensorFlow Core

This manages (via standard TensorFlow Serving APIs) the following aspects of servables:


CORE
CLIENT
ServableHandle <SessionBundle>
DynamicManager
VersionPolicy
ASPIRED VERSIONS
Loader <SessionBundle>
TENSORFLOW SERVABLE
Source
FILE SYSTEM
Legend:
Blue: Caller-specific code
Green: TensorFlow Serving core classes / APIs
Yellow: Pluggable implementations

TensorFlow Architecture: TensorFlow Core

- Lifecycle
- Metrics
- TensorFlow Serving Core treats servables and loaders as opaque objects.

Tensorflow Technical Architecture

- Sources create Loaders for Servable Versions, then Loaders are sent as Aspired Versions to the Manager, which loads and serves them to client requests.
- The Loader contains whatever metadata it needs to load the Servable.
- The Source uses a callback to notify the manager of the Aspired Version.
- The manager applies the configured Version Policy to determine the next action to take.
- If the manager determines that it's safe, it gives the Loader the required resources and tells the Loader to load the new version.
- Clients ask the manager for the Servable, either specifying a version explicitly or just requesting the latest version. The manager returns a handle for the Servable. The Dynamic Manager applies the Version Policy and decides to load the new version.
- The Dynamic Manager tells the Loader that there is enough memory. The Loader instantiates the TensorFlow graph with the new weights.

- A client requests a handle to the latest version of the model, and the Dynamic Manager returns a handle to the new version of the Servable.

TensorFlow Loaders

These Loaders are the extension point for adding algorithm and data backends. TensorFlow is one such algorithm backend. For example, you would implement a new Loader in order to load, provide access to, and unload an instance of a new type of servable machine learning model.

Batcher in TensorFlow Architecture

Batching of multiple requests into a single request can significantly reduce the cost of performing inference, especially in the presence of hardware accelerators such as GPUs. TensorFlow Serving includes a request batching widget that let clients easily batch their type-specific inferences across requests into batch requests that algorithm systems can more efficiently process.

IPFS

Overview

IPFS is a **peer-to-peer (p2p)** storage network. Content is accessible through peers located anywhere in the world who might relay information, store it, or do both. IPFS knows how to find what you ask for using its content address rather than its location.

There are **three fundamental principles** to understanding IPFS:

- Unique identification via content addressing
- Content linking via directed acyclic graphs (DAGs)
- Content discovery via distributed hash tables (DHTs)

These three principles build upon each other to enable the IPFS ecosystem.

IPFS is a **natural fit for blockchain** use cases. The common state of the chain is distributed on-chain among participants, and specific data is stored on IPFS. Thanks to content addressing, the blockchain only needs to store the IPFS multi hash, and users are sure to fetch correct data from any of their peers. This architecture is becoming the de facto standard for blockchain applications.

Off-chain - IPFS

offchain::ipfs allows you to account for your data transactions and DHT status in the blockchain. These on-chain insights can serve as a foundation for incentivized data storage and replication. This means no separate executable: both blockchain and distributed storage are together in one.

By including the Rust implementation IPFS in the native **Substrate runtime**, and by allowing pass-through wasm calls via **Substrate's Off-chain Workers**, we enable a powerful and familiar subset of the IPFS APIs, including

- ipfs add - Write data to IPFS
- ipfs cat - Read data from IPFS
- ipfs dht findpeer - Discover peers
- ipfs dht findprovs - Discover content
- ipfs swarm connect/disconnect - Swarm with other IPFS peers
- ipfs pin add / rm - Pin and unpin content

UNIVERSΛL.

References

Substrate: https://substrate.io/technology/

React: https://reactjs.org/

Polkadot API: https://github.com/polkadot-js/api

Polkadot API docs: https://polkadot.js.org/docs/api

Tensorflow: https://www.tensorflow.org/tfx/serving/architecture

IPFS: https://github.com/iridium-labs/substrate/tree/offchain_ipfs_v3

IPFS: User Manual: https://rs-ipfs.github.io/offchain-ipfs-manual/introduction.html