Privacy enhanced smart-contract based solution using verifiable multi ledger model

Harihara Vinayakaram Natarajan[1], Prachi Gupta[2], Annant Vijay Kushwaha[3], and Kunal Sunil Kasodekar[4]

Wipro Limited, Bangalore

[1] *harihara.natarajan@wipro.com,* [2] *prachi.gupta4@wipro.com,* [3] *annant.kushwaha@wipro.com,* [4]*kunal.kasodekar@wipro.com*

Abstract

In Oil and Gas Enterprises hydraulic fracturing is used to extract Oil which produces wastewater as a by-product. The wastewater generated is disposed of at the oil well by the truck drivers and this constitutes one trip. These enterprises operate in remote areas with sporadic network connectivity. Hence there may be a delay in communication between the stakeholders resulting in delayed payment for the truck driver. In our previous paper[1], we have proposed a solution that automates the above process using IoT devices and blockchain (Ethereum). All the events in this process were made as an entry onto the blockchain.

Blockchain though promotes a trustless system where every other stakeholder can trust and validate transactions contained in blocks in a distributed manner, but it suffers privacy limitations as the transactions containing sensitive data is visible to all. To solve this issue, we have moved to a centralized immutable ledger that is cryptographically verifiable. It inculcates a trust but verify model in the system so the stakeholders can keep sensitive data that can be audited at their end. The business logic has been implemented in smart contracts written in DAML, which executes the trip auto settlement and invoice generation on receiving the events. This concept of a centralized database with a smart contract can be extended for other use cases where data from multiple parties are involved and requires verification.

The data stored on the Ledger can be used among different stakeholders for procuring insights to enhance the system. However, this data exchange can lead to privacy violations hence we propose a differential privacy-based feature for data sharing. We are handling some real-world scenarios like dropped events, trip cancellations, and truck breakdowns, etc. We have introduced prediction and scheduler entities to reduce waiting time by pre-allocating trips to the truck drivers based on their preferences to promote the gig economy.

1. Introduction

The last few years have seen a massive surge in cloud-based enterprise applications. Cloud computing provides enterprises the ability to focus on solution design and workflow by off- loading infrastructure and deployment concerns to cloud-based services. This has resulted in a significant rise of FaaS and IaaS based cloud services. Microsoft Azure is a leading cloud computing platform preferred by such enterprises for their numerous solutions. To accommodate these many solutions, groups within the organization are stuck with a myriad of RG's and resources without fine-grained control. This results in budget overruns and ultimately revenue loss. To top it off as mentioned above Azures budget API has a long- standing bug that prevents effective budget control. To solve these issues we had created a naive solution as follows:

In today's era, many enterprises operate on a coopetition-based model for mutual growth and market sustainability. However, while collaborating in this model the stakeholders face trust-based issues like data-verification, data-sharing, and other privacy concerns. We have picked up the use case for the Oil and gas enterprises that elucidates how this model can be applied for similar use cases where multiple stakeholders are involved and transparency in the system must be achieved.

In oil and gas enterprises, toxic water gets generated during the hydraulic fracking method of oil drilling. It is a big concern for these enterprises to monitor the disposal of wastewater generated at the disposal sites. These enterprises can face legal consequences if wastewater is not disposed of properly. They operate in the low network connectivity area and involve three main stakeholders - Fleet Owner, Truck Drivers, and Oil field owner. Currently, they follow the conventional method of payment and handling end to end disposal that forms a single trip. In our earlier POC[1], we have proposed a solution that automates the manual process using IoT devices and Ethereum based smart contracts to orchestrate the entire process. Every event in the life cycle of the trip i.e. from the event of wastewater generated to wastewater being dumped in the dumping well is an entry onto the blockchain.

However, there were certain limitations in using Ethereum[2] like all the transactions for a trip are transparent to all the stakeholders who are part of the network causing privacy concerns. Apart from that, querying Ethereum's smart contract caused latency in the system. We have ported our earlier decentralized solution to a centralized model where the transactions for a trip are stored on Immudb[3] that can be audited and verified in case of disputes thereby inculcating trust. Immudb is an immutable database that provides cryptographic proof and verification of data. It is often compared with QLDB (Quantum Ledger Database) [4] and offers better performance. It solves the data privacy and trust issue. To implement the business logic and handle the automation, DAML's smart contracts[5] are used to automate the payment and invoice generation, if still there is a dispute regarding the payment it can be verified from Immudb with proof.

This paper also illustrates the use of differential privacy[6] for anonymizing the data while sharing data among the different stakeholders. These stakeholders can cooperate to create a mutual advantage by sharing access to insights gleaned from collaborative data. Differential privacy can be obtained by adding randomized noise to an aggregate query result to protect individual data without significant changes to the result.

Some more entities like prediction and scheduler are introduced in this paper where the pre- diction entity will predict the time in advance when the tank gets full and scheduler entity will pre-assign the trip for the predicted data considering parameters like location, distance, and time preference of the truck drivers. Hence it promotes the gig economy as it gives liberty to truck drivers to work in their preferred time and location. The process will also not be halted as the truck driver will reach the oilfield when the tank is going to be fill. The identification mechanism to verify stakeholders and services is proposed to be deployed using Sovrin[7] Framework. We have also handled some outstrip cases, if the truck driver accepts the trip but cancels later, booking of the new truck if the truck driver doesn't arrive at the time at the oilfield, handle truck break- down i.e. booking a new truck if the truck stop working in a trip. The events generated in a trip are stored locally in the mobile/IoTs devices of truck drivers, tank operators. On availability of the network, it will give the events to their respective owner to automatically handle the trip and invoice generation using contracts.

2. Solution Architecture

The "Single Responsibility Principle" is one of the widely adopted and reliable principles for soft- ware design. It implies that all the things that change for the same reason should be combined in one unit. Following the same principle, our solution architecture adopts a microservices style pattern where each of the services has been decomposed based on business capabilities. The ser- vices are loosely coupled and each of them has its database. When it comes to a business scenario like this where events change asynchronously, event-driven architecture came out to be a workable solution. Each of the services publishes its event and other services consume it. The intercommunication between services takes place by the means of a pub-sub system using NATS.io[8]. The major challenge we faced was to implement eventual consistency between services. The events are stored in event logs in Immudb and be audited to validate the states.

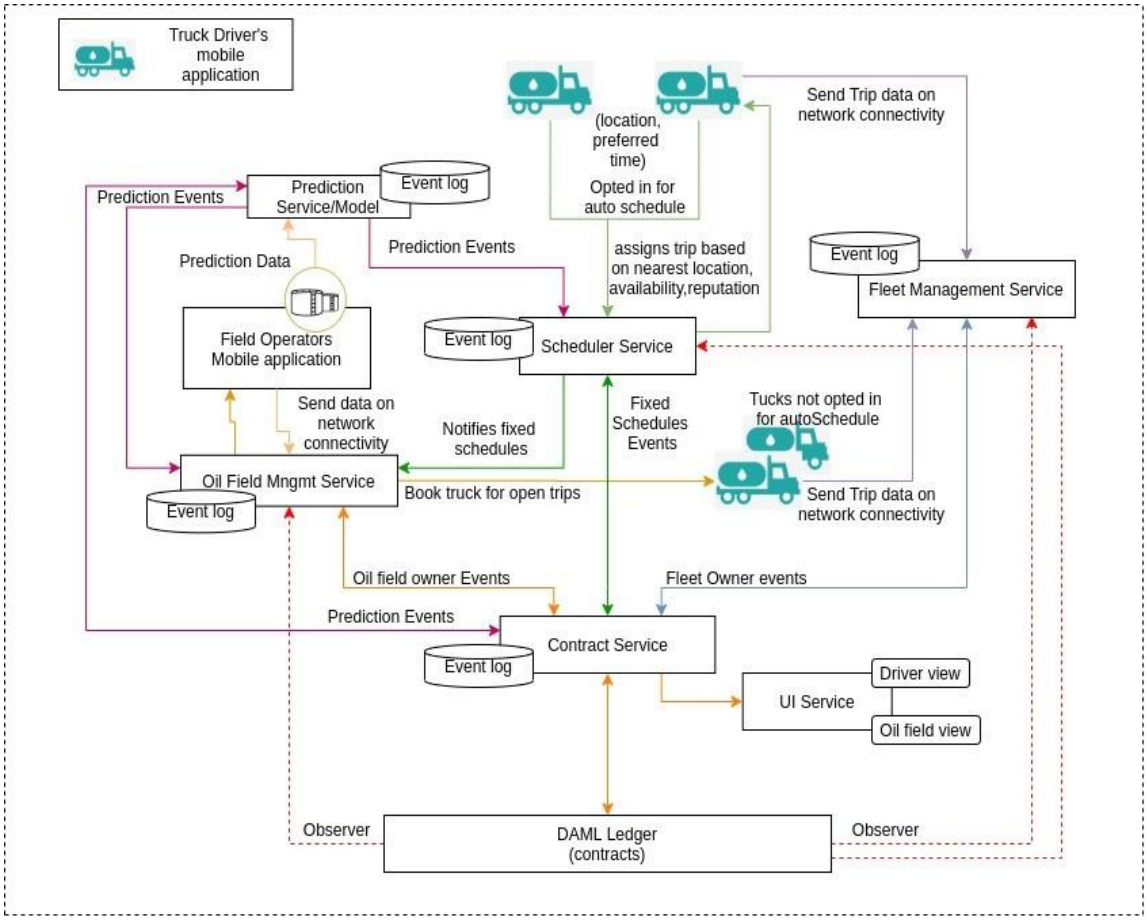


Figure 1: Solution Architecture

2.1 Micro Services Components

The above diagram shows the modular architecture of the system. The individuals interact with the system using their front-end/mobile application which communicates to other services. It comprises of following services:

1. **Prediction Mode/Service:** This service implements AI (Artificial intelligence) model as service to predict the time when a tank is going to be filled. It takes various parameters for a tank in the dataset to output prediction and the measured sensor data is used as feedback mechanism to improve its accuracy. We have explored various models that can be deployed in the current scenario, detailed elaboration, and pros, and cons of each model is out of context in this paper.

2. **Scheduler Service:** This service provides the functionality for truck drivers to opt-in for auto-scheduling based on their work preferences. Scheduler service follows an uberized model to schedule trips and allot it to truck drivers considering their preferences, nearby location constraints, availability, the reputation of truck drivers. All the scheduled trips are emitted at the end of the day and the trips which do not match the involved parameters will fall into open trips that need to be booked by tank operators.

3. **Oil field trip Management Service:** This service consumes the

prediction data events and scheduler events and books the trips which could not be scheduled. This service also man- ages all the trips and consolidates all the events received on network connectivity from individual tank operators' application for a specific trip. It maintains its event logs in Immudb that can be verified.

4. **Fleet Owner Trip Management Service:** This service onboards all the truck drivers under its fleet. This manages all the trips and consolidates all the events received on network connectivity from individual truck drivers' application for the specific trip. It maintains its event logs in Immudb that can be verified.

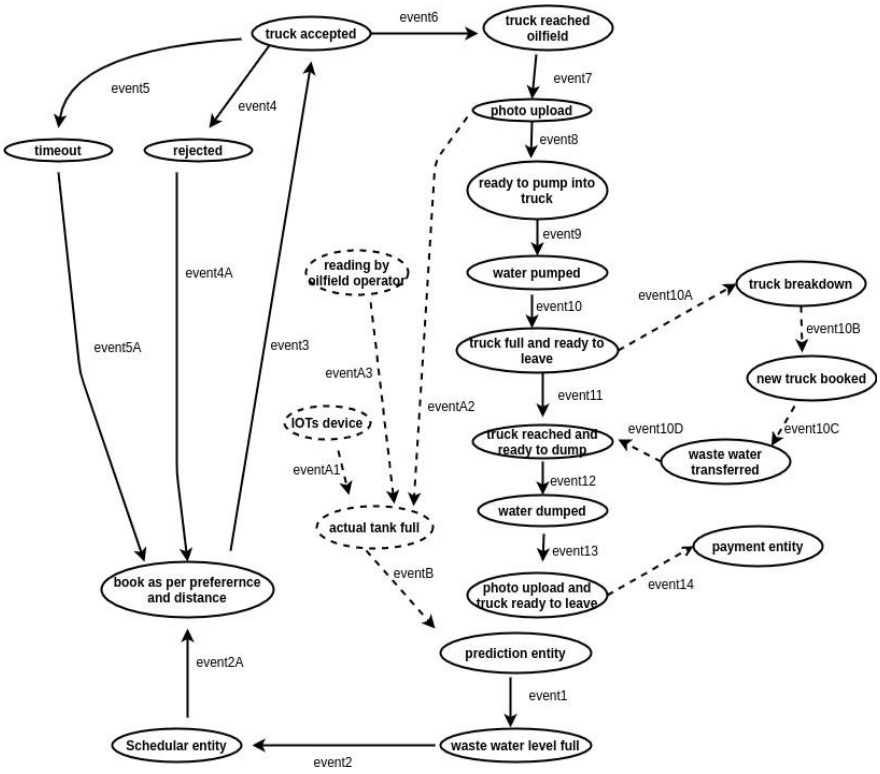
5. **Contract Service:** Contract service implements the "trust but verify" model for the automation of trip settlement and invoice generation for the stakeholders. It consumes the events of other services and create contracts for scheduled and open trips. The contracts are implemented in DAML ledger that allows fine-grained permission to invoke, update, delegate the contract updates, and read-only events for observers. It further keeps track of the events received including the worst possibility of trip cancellation and timeouts and takes actions accordingly to achieve reconciliation based on the events received from the involved stake- holders. Owing to sporadic network connectivity, some of the events might get delayed or dropped. This service tracks those trips and waits for 2-7 days to receive events to settle the trip.

6. **Pub-Sub messaging system:** We have implemented event-driven architecture where the events published by one service get consumed by one or multiple services. For pub-sub system, we are using NATS.io that supports in-memory and persistent storage in a disk. It can also be used as recovery support with queue and can recover the lost data by replaying the events from start.

7. **Identification system:** The identification mechanism is proposed to be deployed using Sovrin framework. Each of the entities and services can issue their own DIDs that can be verified by others in distributed manner. We propose to use Hyperledger Indy to implement self-sovereign identity for all the individual truck drivers and authenticate services.

3. Workflow

Figure 2: Solution Workflow



(a) The diagram below shows the event flow for the various business tasks taking place during the disposal of wastewater. All the services are onboarded onto the system. The identities of truck drivers will be issued and verified using Sovrin while onboarding. The truck driver has an option to give consent and their preference of time, distance, etc. to the scheduler entity. The process starts with the prediction entity that will predict the tank fulfillment time in advance for the next day. This entity will use the sensor measured data as a feedback loop to improvise the model. After receiving the prediction, the scheduler entity will pre-schedule the trip in advance for the truck drivers as per their preference, who have already opt-in for the auto-scheduling with the scheduler entity. Truck drivers can reject a trip within a limited time duration otherwise their reputation will be compromised. The scheduler will try to reschedule the trip considering the parameters, if it does not find any suitable match, these trips will go into open trips that will be booked by the oilfield owner. Contract service on listening to these scheduled trips will create a contract with the involved entities.

(b) On other hand, the oil field owner on receiving the prediction and scheduled trip will book the truck driver for those trips that were not scheduled by the scheduler entity and will update the contract service through an event so that it can create a contract with the involved entity. Once the trip has been allotted to truck drivers, the trips start and there is a possibility that truck drivers can cancel the trip after allotment, cannot reach the oil field at the time, and can get a breakdown in the further trip process. All these events are handled carefully in the implementation. All the events emitted by the front-end application are stored locally and are received by the oil field owner and fleet owner on network connectivity. Oil field owner and fleet

owner services emit these received events at the instant they are received it. These entities will also store these events in the Immudb ledger as key-value pair, where the data stored can be audited by any entity if any dispute arises.

(c) Contract service consumes these events to update the corresponding trip contract to achieve settlement automatically once it receives all the relevant events. In case of timeout or truck driver cancel a trip, the contracts created earlier will reduce the reputation of the truck driver automatically. If the contracts receive the breakdown events, it will generate the invoice for both the involved drivers as per the events received. In the worst-case scenario when events do not arrive even after the given period of 2-7 days, the events stored in the Immudb cryptographically verified database of individual services like oil field trip management service, fleet owner trip management service are audited for trip parameters. When this database will be queried for any entry then it will provide cryptographic proof for that entry and it can be easily verified if data exists. After auditing proof, reconciliation is achieved among the parties and an invoice is generated at the instant.

4. Challenges and Discussion

Moving from a blockchain-based solution to a centralized ledger model presented the challenge of implementing the identification mechanism. The most common approach to secure and authenticate web services is PKI (Public Key Infrastructure)[9]. PKI infrastructure provides a robust infrastructure to secure the system, but it relies on trusted CA (Certificate Authorities). These trusted third parties manage and create identifiers and public keys for the services. Over the years there have been many events of duplicate certificates issued, reuse of revoked certificates, and attacks that were hard to detect as it took a long time. To solve such problems, there has been an open-source solution like Trillian[10] that follows the principle of trust but verifies the model. However, we were looking for a more robust decentralized framework. DPKI (decentralized PKI)[9] that uses a PGP encryption algorithm is an alternate approach to solve the issue. It provides principal owners to issue unique Domain Identifiers (DIDs) and have control over their identities. Sovrin framework makes it feasible for any service to issue its unique Decentralized Identifier that can be used to verify its identity.

Another major change was the way we have implemented business logic in smart contracts. In the previous solution, Solidity is the domain-specific language for smart contracts. The state changes of contract in Ethereum occur in the accounts whereas DAML is platform agnostic and operates with a UTXO model i.e. after each transaction the contract is archived, and a new child contract is created in its place with updated properties. Moreover, DAML allows fine-grained permission for the stakeholders to have read and write access to the states defined in the contracts as observers and signatories. While Solidity offered on-chain logic of automation i.e. a contract can be revoked by the other contract deployed in the same network. On the other hand, DAML offers DAML triggers that is in the preliminary stages of development and off-chain automation is being implemented in the service business requirements. DAML can also be plugged into any supported blockchain framework considering the possibility of switching back to a blockchain- based solution as per the requirement. With many blockchains moving to use Web Assembly as it offers the ideal runtime for smart contracts instead of EVMs, we also explored other options like Substrate ink

domain-specific smart contract language. Ink is written in Rust that is highly type-safe and secure. DAML currently uses a JVM runtime environment, it can be compiled to web assembly. However, the support for compilation is in the initial stages.

The proposed solution has solved many of the worst possible scenarios of rejection and time- outs by handling the appropriate events. Any changes in the scenarios may lead to redesigning of the event architecture. The on-chain automation may make it feasible to solve the problem, but these are currently handled in the off-chain automation logic.

5. Conclusion

In a nutshell, the current solution deploys an end-to-end tracking mechanism on a centralized im- mutable ledger. It automates the payment using DAML's smart contracts. Since DAML is nascent and some of the features like on-ledger automation using DAML triggers are in the developing stage, so we have implemented off-ledger automation in its place. Data stored for the separate stakeholders can be used to bring some useful insights to make the system better without com- promising the privacy of the users using differential privacy. The solution can be extended to the places where there is a need for the automation of a process, a centralized verifiable database to foster trust among stakeholders, and need for a mechanism to share the data for the mutual growth of the system without putting the privacy of users at risk.

References

[1] AV Kushwaha, HV Natarajan, K Jayakumaran, P Gupta, Raksha, SK Karki. (2019). "IoT Based Blockchain Solution To Endorse Positive Human Behaviour", ISRDC@IITB, Third workshop on blockchain technologies and its applications.

[2] Ethereum Whitepaper. (2021). Retrieved from https://ethereum.org/en/whitepaper

[3] Immudb Introduction. (2021). Retrieved from https://docs.immudb.io/master/

[4] What is Amazon QLDB? (2021). Retrieved from https://docs.aws.amazon.com/qldb/

[5] Ameer Rosic. DAML - An open-source ecosystem for building smart contract based distributed applications. Retrieved from https://blockgeeks.com/guides/daml-an-open-source- ecosystem-for-building-smart-contract-based-distributed-applications/

[6] Uber Privacy & Security. (2017). Uber Releases Open Source Project for Differential Privacy. Retrieved from https://medium.com/uber-security-privacy/differential-privacy-open-source-7892c82c42b6

[7] How Sovrin Works. (2016). Retrieved from https://sovrin.org/library/how-sovrin-works/

[8] Ginger-Collison. (2019). Publish-Subscribe. Retrieved from https://docs.nats.io/nats-concepts/pubsub

[9] Christopher Allen, Arthur Brock, Vitalik Buterin, Jon Callas, Duke Dorje, Christian Lundkvist, ... Harlan T Wood. Decentralized Public Key Infrastructure. (2015). Retrieved from http://www.weboftrust.info/downloads/dpki.pdf

[10] Verifiable Data Structures. (2021). Retrieved from https://github.com/google/trillian