Cloud Service Billing and Service Level Agreement Monitoring based on Blockchain

Nils Neidhardt[1], Carsten Köhler[1] and Markus Nüttgens[1]

Abstract: In cloud computing, a customer sources parts or even his complete IT infrastructure out to a cloud service provider. This often results in a loss of control: Due to a lack of transparency, the verification of the billing process, as well as the provider's adherence to service level agreements (SLAs) can be difficult to track for the customer – which can diminish his trust in the service provider. As a solution therefore we propose a blockchain- and smart contract based concept, which implements the SLA monitoring-, as well as cloud billing services in a decentralized, transparent manner, thus reducing the need for the customer's trust in the provider. Hereby, tokens are exchanged between the customer, the provider, as well as external SLA monitoring services in order to timely document customer- and provider actions.

Keywords: Blockchain, Smart Contract, Cloud Service.

1 Introduction

1.1 Problem description

The outsourcing of IT infrastructure to a cloud service provider often results in a loss of control for a customer: This is because he loses visibility into his infrastructure and therefore depends on the information that he is given to by the provider. As a result, significant trust of the customer in the service provider is necessary: Hereby, the consumer relies on the correctness of the invoices that he receives, as well as on the fact that the service level agreements were uphold. If the customer suspects inconsistent or faulty behavior from the provider, his only option is typically limited to questioning them on an individual basis.

1.2 Motivation and prior work

Park [PHC13] and Sekar [SM11], who have recognized the aforementioned problem as well suggested a solution through a neutral "verifier", which monitors and controls transactions between the customer and the provider and settles in the event of a dispute. Zou, who has discussed this approach as part of her dissertation however argues that the verifier would require just as much trust and hence form the bottleneck of the process [Zo16]. As a possible alternative therefore, she suggests the blockchain technology: Due

[1] Hamburg Research Center for Information Systems, University of Hamburg, firstname.lastname@uni-hamburg.de

to its distributed structure and consensus mechanism, it allows participants to exchange transactions without the need to trust each other or a central party. Hence, a blockchain based solution may be able to address the trust problem between cloud service provider and consumer and as such forms the basis of our concept.

1.3 Research questions

The following research questions are proposed in this paper:

RQ1: Can blockchains improve the transparency of the billing process and SLA compliance of cloud service providers, and therefore reduce the need for a customer to trust the provider?

RQ2: How can the efficiency of the billing process improved via smart contracts?

RQ3: What are potential challenges for the use of blockchains in the context of cloud service provisioning?

2 Concept description

2.1 Blockchain selection process

In order to identify the most suitable blockchain platform for our purpose, we used an Analytical Hierarchy Process [Sa15] based on the following selection criteria: Metacoin capability (7), smart contract [Sz97] support (6), smart oracle support (5), size of the technical community (4), a private, public or consortium BC (6), transaction costs (4), performance (2). As a result of the selection process, we chose Ethereum [Bu14]: The main reasons were its smart contracting capabilities, which would allow us to issue custom tokens, as well as to query external data sources via so called smart oracles (see 2.3). Further, the size of the technical community and therefore the maturity of existing development tools played a decisive role.

2.2 Service initialisation and usage tracking

As a precondition for our concept, both the service provider, as well as the customer need to have an Ethereum wallet. Thereafter, the customer can book various services from the provider, such as hosting or backups. For each of those services, the provider then sends custom Service-Coins to the customer's wallet. Those coins could be different for each service in order to track them individually. The customer then sends the service coins back to the provider on a per-use basis – e.g. when requesting a backup. The used coins will then be billed for accordingly in the billing process (2.4), e.g. at the end of each month. This allows a transparent, granular invoicing to the customer.

At the same time, the costs of using the Ethereum network, or more specifically the Gas price, have to be considered. Gas has to be paid for each transaction - the relationship between GAS and Ether is ETHER = STARTGAS * GASPRICE. STARTGAS is the amount of GAS that is required to perform a transaction, with the standard value being 21000. The GASPRICE is currently 4 GigaWei. For example, a normal transaction would cost 84,000 GigaWei or 0.000084 Ether. Converted in Euro it is 0.045 € (1 ETH = 540 EUR[2]). We take this price and calculate the costs for a billing period. Table 1 shows the cost calculation for the service usage and the quantity of various services of the customers. We assume that the service is used on 22 days a month as this is the average of monthly workdays.

Table 1: Transaction cost prognosis for service usage concept

No of services	Usage (22 d)	Cost per customer	100 customers
1	22	0,99 EUR	99,00 EUR
5	110	4,95 EUR	495,00 EUR

2.3 SLA monitoring

For the verification of the provider's adherence to the uptime SLA, we developed a smart contract. For each service offered by the provider, it hereby maintains a list of customers. Consequently, the Ethereum address of a new customer is added to the appropriate list after both parties signed the SLA. They further agree on a service checking interval that should be used by the smart contract, e.g. every 5 minutes. The customer will then receive SLA-coins whenever the smart contract determines that the service is unavailable.

At the end of a month, the SLA coins are counted and if the amount is higher than a certain threshold, the SLA is violated.

A main challenge for the smart contract is then to detect whether a monitored service was unavailable. Since a blockchain like Ethereum is naturally a closed system, external information (as in this case the service availability) has to be transferred into the blockchain so it could subsequently be processed by our smart contract.

This is typically achieved via so-called Oracles: These are external services, which actively push external data into the Ethereum blockchain. A potential shortcoming of Oracles is the fact that they have to be trusted, which could undermine the benefits of using a blockchain in the first place.

As a remediation we therefore used the "Oraclize" [Or16] oracle service, which enables verifiable honesty. This is accomplished via the integration of TLSnotary [TL14], a service which can intercept a TLS-connection and therefore attest that a certain server has sent certain data at a specific time. Furthermore, Oraclize can utilise multiple independent data sources and e.g. return the median value of their outputs to the blockchain, which mitigates the risk of having a corrupt Oracle.

[2] The price information is from coinmarketcap.com accessed on 1.5.2018

A potential alternative to Oraclize could be the service “Reality Keys”, which offers a mixed form of purely automatic and human driven Oracles [Re17].

The overall workflow of the SLA monitoring process is illustrated in Figure1: Hereby, the Oracle sends service availability information into the blockchain, which is then read by the SLA-monitoring smart contract. If a service is unavailable, the contract then sends SLA coins to a list of customers, who could then proof whether their promised SLA levels were uphold. Eventually, SLA coins could then be used by the billing smart contract to reimburse customers in case of an SLA breach.

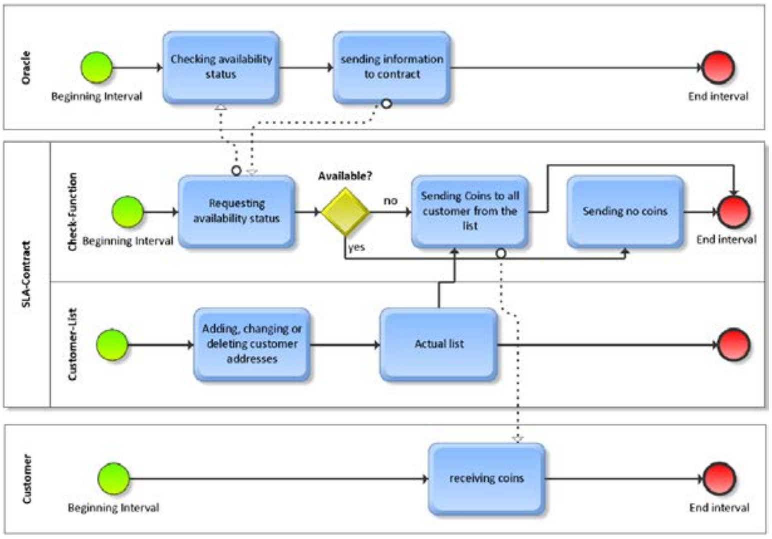

Figure 1: SLA Monitoring Process

2.4 Service billing

The service billing smart contract bills a customer based on the service coins that he used. Therefore, two options exist:

The customer could pay in fiat currency, which means that a price per service coin is agreed upon in advance.

Alternatively he could pay in Ether: This could either be done after receiving the bill, or by locking up Ether in advance, which would automatically be used by the smart contract. In this scenario the smart contract would act as a decentralised escrow party.

The price per SLA token could either be agreed upon in Ether, or in fiat currency. In the latter case, the smart contract would have to query the exchange rate from Ether to fiat from an oracle service.

3 Conclusion and further outlook

Our smart contract based concept and prototype implementation thereof has shown promise to improve the transparency of the billing process, as well as the SLA compliance of cloud service providers.

At the same time, several challenges still have to be overcome:

Managing cryptocurrency wallets and private keys adds complexity, which should be made as opaque as possible to the customer. Determining service prices in Ether may also be impractical due to the high volatility of the currency. This also makes the cost of transacting on the Ethereum blockchain difficult to predict, which poses a risk for the economic viability of the solution. This could be mitigated by choosing a different blockchain with smart contracting capabilities and lower transaction costs.

Lastly, the open nature of the blockchain may allow competitors to derive information about the business relationships of the cloud service provider, which could be addressed via the addition of recent cryptographic techniques such as ZK-SNARKs [Be14] .

Bibliography

[Be14] Ben-Sasson, E. et al.: Succinct Non-Interactive Zero Knowledge for a von Neumann Architecture. In USENIX Security Symposium, 2014.

[Bu14] Buterin, V.: A next-generation smart contract and decentralized application platform.

[Or16] Oraclize: Understanding oracles. https://blog.oraclize.it/understanding-oracles-99055c9c9f7b, accessed 23 Sep 2017.

[PHC13] Park, K.-w.; Han, J.; Chung, J.: THEMIS: A Mutually Verifiable Billing System for Cloud Computing Environment. In IEEE Transactions on Service Computing, 300-313, 2013.

[Re17] RealityKeys. https://www.realitykeys.com, accessed 16 Mar 2018.

[Sa15] Saaty, T. L.: How to make a decision: The analytic hierarch process 48. European journal of operational research, pp. 112-125, 2015.

[SM11] Sekar, v.; Maniatis, P.: Verifiable Resource Accounting for Cloud Computing Service. In Chicago, 2011.

[Sz97] Szabo, N.: Formalizing and securing relationships on public networks, 1997.

[TL14] TLSNotary. https://tlsnotary.org, accessed 16 Mar 2018.

[Zo16] Zou, J.: Accountability in Cloud Services, 2016.