International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 5, ISSN No. 2455-2143, Pages 30-36
Published Online September 2020 in IJEAST (http://www.ijeast.com)

AN ARCHITECTURAL MODEL FOR SLA NEGOTIATION BETWEEN SAAS AND CUSTOMERS

Ajayi Oluwabukola
Department of Computer Science
Babcock University, Ilishan-Remo, Ogun State, Nigeria

Ajayi Adebowale
Department of Computer Science
Babcock University, Ilishan-Remo, Ogun State, Nigeria

Abstract— **Software as a Service (SaaS) is a cloud computing model where consumers use Cloud applications without controlling the hardware, operating system and network devices of the environment providing the application. In order to enlarge their customer base, SaaS providers need to attract customers with special requirements and for this a novel negotiation framework was proposed to establish service level agreements (SLAs) with these special QoS requirements.**

This paper presents an architectural model for SLA negotiation between SaaS and Customers and describes as SaaS providers want to enlarge market share, they need to provide more flexibility in terms of services to cater to variations associated with an individual customer. This is generally done by a negotiation process between customers and service providers. However, while undertaking this negotiation process, the service provider needs to take into consideration not only what they can provide to customers but also the competition with other SaaS providers. Thus, the new negotiation frameworks proposed are needed for the SaaS provider that considers dynamism in Cloud environment with time and market factors to make the best possible decisions for negotiation. The proposed negotiation framework can be used for the SaaS provider and the SaaS broker model

Keywords— **Cloud Computing, Software as a Service (SaaS), Service Level Agreement (SLA), Negotiation.**

I. INTRODUCTION

Cloud computing provides huge computing services to the business for improving the organizational growth. Basic requirement needed for this technology is Internet but provides higher capability when compared to the Internet.

Software as a Service (SaaS) is a cloud computing model that the consumer uses Cloud application but does not control the hardware, operating system and network devices of the environment providing the application. This layer includes the software applications, such as social computing applications and enterprise applications, which is deployed by PaaS providers renting resources from IaaS providers

Yeo and Buyya (2006) highlighted that customer satisfaction is an important success factor to excel in the service industry and the best way to ensure the Quality of Service (QoS) is to define a legal contract which is a Service Level Agreement (SLA), between a service provider and a consumer (Buco, et al., 2004) to measure the CSL.

SLAs can be traced back to 1980s in telecommunication companies where telecommunication companies include SLA within the terms of their contracts with customers to define the level(s) of service being sold to them in plain language terms.

A Service Level Agreement (SLA) is a formal, negotiated document that defines (or attempts to define) in quantitative (and perhaps qualitative) terms the service being offered to a Customer . An alternative definition going a bit away from the pure process-oriented Information Technology Infrastructure Library (ITIL) as: “A Service Level Agreement (SLA) is a formal negotiated agreement between two parties. It is a contract that exists between the Service Provider (SP) and the Customer.

II. LITERATURE REVIEW

Glen & Alfonso (2006) presented a unified QoS ontology applicable to QoS-based Web services selection, QoS monitoring, and QoS adaptation. However, they did not consider the enforcement of other service application types.

Mike, Stephen & Alfonso (2007) discussed dynamic service provisioning using GRIA (a Service Oriented Architecture framework) SLA. The authors explored how web service management using SLA and dynamic service provisioning can maximise resource utilization while fulfilling the QoS commitments to the existing customers. In their approach, they proposed two possible policy enforcement strategies for handling SLA violation: i) prevention before violation and ii) reaction after violation. The prevention strategy was based on prediction of possible future violations, which can be obtained by monitoring predefined prevention thresholds. These prevention thresholds have to be defined on per SLA basis. With dynamic provisioning, when the prevention threshold is exceeded, a new service instance is

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 5, ISSN No. 2455-2143, Pages 30-36
Published Online September 2020 in IJEAST (http://www.ijeast.com)

started so that new requests are redirected to the new instance to ensure their SLA. The reaction strategy is only acceptable if the violation does not result in complete service failure. The service provider allows the violation of an SLA in order to enforce others. In such cases, it specifies priority for different SLAs based on business impact. Moreover, they did not detail how the low-level metric are monitored and mapped to high-level SLAs to enforce the application SLA objectives at runtime.

Congwu, Lei & Jun (2007) discussed Aspect Oriented Programming (AOP) based trustable SLA compliance monitoring for web services. The authors proposed a novel trustable mechanism to monitor and evaluate SLA compliance based on the Aspect Oriented Programming paradigm. In their approach, authoritative monitoring features were supplied by a trustable SLA manager and by focusing the aspects into susceptible service runtime, provider can accurately monitor and report their service status. However, their approach targets only web services.

Bastian, Lutz (2007) discussed autonomous SLA management using a proxy-like approach. They implemented an architecture that can be exploited to define SLA contracts. The architecture allows autonomous management of such contracts, once service providers and customers explicitly provide the requirements for the contracts. Based on the architecture, they outlined some guidelines on how such a system can be setup and reused. Their strategy was based on WSAgreement. Moreover, their approach is limited to Web services and did not consider other applications types.

Henar & Loannis (2009) discussed the main approach of the EU project BREIN (2015) to develop a framework that extends the characteristics of computational Grids by driving their usage inside new target areas in the business domain for advanced SLA management. BREIN applies SLA management to Grids, whereas this work target SLA management in Clouds.

Stefano, Vittorio, Fabio, Michele & Elisa (2010) proposed QoS-aware Clouds. In their approach they discussed the design and evaluation of a middleware architecture that enables SLA-driven dynamic configurations to respond effectively to the QoS requirements of the Cloud customer applications. The proposed architecture was proactive, it uses continuous monitoring and dynamic resource allocation to enforce the agreed SLA objectives for the customer applications. However, they did not consider optimal monitoring interval for efficient monitoring and enforcement of SLA objectives.

Kornel, Jakub, Renata & Jacek (2010) presented the application of the ESB architecture for distributed monitoring of the SLA requirements. The authors identified some issues affecting efficient SLA enforcement processes such as different technologies for the evaluation of the SLA documents, complex deployment processes, and scalability issues. Their SLA enforcement strategy was based on the continuous monitoring of the system to identify violation situations. But they did not address the issues of individually enforcing customer SLAs for applications executing on the same host.

Lee et al. (2010) propose profit-driven SLA based scheduling algorithms in Clouds to maximize the profit for service providers. The application model used in this work can be classified as SaaS and PaaS. The service types supported by their algorithm are dependent services, which mean one sub-service can not start until the pre-required services complete. However, their work does not support multiple providers and full simulation configuration is not available.

III. METHODLOGY

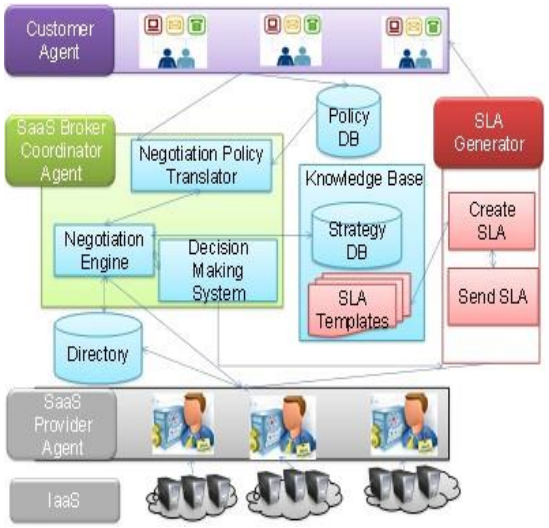

Fig 1: Negotiation Framework High Level Architecture (Brandic, Musicand Dustdar, 2009**)**

The main components in the negotiation framework as shown in figure 1 are: Customer Agent (CA), Broker Coordinator Agent (BCA), Provider Agent (PA), IaaS Provider, SLA Generator, Directory, Policy Database (PD), and Knowledge Base (KB).

Customer Agent: Represents a customer that submits requests for software services and registers their QoS requirements into PD.

Broker Coordinator Agent: Represents the broker by receiving customer requests and negotiates with providers to achieve business objectives. It will include **Negotiation Policy Translator (NPT), Negotiation Engine (NE),** and **Decision Making System (DMS)**.

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 5, ISSN No. 2455-2143, Pages 30-36
Published Online September 2020 in IJEAST (http://www.ijeast.com)

Negotiation Policy Translator: Maps customer's QoS parameters to provider level parameters. **Negotiation Engine:** Includes workflows which use negotiation strategies during the negotiation process. **Decision Making System:** Uses decision making heuristics to update the negotiation status.

Provider Agent: Represents the provider. PA could include the thirdparty monitoring system to update the provider's dynamic information.

The SLA Generator: When the negotiation has been successfully completed, the SLA Generator creates an SLA between the customer and the provider using templates retrieved from the KB. The template includes specified Service Level Objectives (SLOs) according to the QoS.

The Directory: The repository stores the providers' registered service information.

The Policy DB: The repository stores QoS terms that both providers and customers understand.

The Knowledge Base: The repository stores negotiation strategies and SLA templates.

The focus of this research is on two main components: the NE, by proposing strategies considering both time and market, and the DMS, by proposing heuristics for different objectives.

This paper considered three entities: consumers, SaaS brokers and SaaS providers. Each consumer *c* submits a service request to the SaaS broker, who leases software services from SaaS providers.

The **customer** *c* requests services with the following attributes:

- Budget B_C: the maximum price a customer can afford.
- Software service set SR_b: the service editions.
- The service start time t_{ss}: the latest service available time for a customer *c*.
- The contract length indicates the period of service usage *conLength*, so that customer *c* must be able to use software service within the contract term.
- The service refresh time *tr*: time it takes a query operation to be executed in a software service.
- The service process time t_p: the maximum time for a consumer *c* to wait for completing a transaction.
- The service availability *avai:* the minimum availability that the customer requires.
- The expected discount percentage for budget σ: the percentage a customer can save from their actual budget.
- The preference level of each QoS parameter γ: the absolute importance level which varies (0, 1).

The broker receives the customer request and calculates the expected budget, expected refresh time, process time, and availability. These expected values are the **best** values that the broker expects to provide to the customer and they will be proposed to providers in the quote request process. If providers cannot fulfil these expected values, the broker will adjust the expected value up to the customer requested value during the negotiation process. The broker always seeks to secure the expected value from provider.

Each **provider** offers the same or different types of services. The provider can host or lease infrastructure services from 3rd party IaaS providers.

A. Negotiation Objectives

In sophisticated markets, the negotiation objective is not only price but also other elements such as quality, reliability of supply, or the creation of long-term relationships. Multiple objectives were considered including cost, refresh time, process time and availability. The main objectives for a customer, a SaaS broker and a provider are:

Customer: minimize price and guaranteed QoS within expected timeline.

SaaS Broker: maximize profit from the margin between the customer's budget and the providers' negotiated price.

SaaS Provider: maximize profit by accepting as many requests as possible to enlarge market share.

B. Negotiation Policy Specification

The negotiation policy specifications are used to specify QoS parameters, which are to be negotiated and the acceptable range of them to reach the mutual agreement.

1) QoS Model

Different terms used by different participants' is one of the critical challenges in SLA negotiation. For this framework, a QoS model is used to provide shared knowledge about QoS attributes among negotiating participants. The QoS model defines a set of QoS dimensions. Each QoS dimension represents a specific quality aspect of a service, such as refresh time, availability, and price. In this QoS model, a quality dimension is defined using: a title, a category, a name, a description, and a metric. The QoS model is shared among service consumers and service providers, therefore, they have a common understanding on the QoS attributes about how they are defined, how they are measured, and so on. For the QoS dimensions the following was considered– price, refresh time, process time and availability. These dimensions are the ones that are mostly used and they are domain- independent.

Before negotiation, both participants specify the rule of QoS parameter in a policy specification. The policy usually refers to a high-level description of goals to be achieved and actions to be taken in different situations.

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 5, ISSN No. 2455-2143, Pages 30-36
Published Online September 2020 in IJEAST (http://www.ijeast.com)

C. Negotiation Protocol

The negotiation protocol refers to a set of rules, steps or sequences during the negotiation process, aiming at SLA establishment. It covers the negotiation states (e.g. propose offer, accept/reject offer, and terminate negotiation). It is common to characterize negotiations by their settings: bilateral, one-to-many, or many-to-many. This work focuses on the one-to-many bargaining setting, where three types of agents are considered (CA, BCA and PA). A BCA negotiates with many PAs in a bilateral fashion.

During the negotiation process, the negotiation status is updated using negotiation states described in Table 1.

States	**Description**
Propose	The agent propose initial or counter offer to the opponent agent.
Reject	The agent does not accept the offer proposed by the opponent agent.
Accept	The agent accepts the offer proposed by the opponent agent.
Failure	System failure, trigger renegotiation.
Terminate	Negotiation is terminated due to timeout or no mutual agreement.

Table 1: The Negotiation States and Description Summary

The sequential negotiation process for this framework is described as follows and depicted in Figure 2:

Phase 1: *CA submits requests*: CA requests services on behalf of the customer to the Broker.
Phase 2: The BCA requests initial proposals from all providers, who are registered in the Directory. The values sent from BCA to PAs are expected values.

Phase 3: *PAs propose initial offer:* All PAs propose initial offers based on their current capabilities and availability to fulfil BCA's requirements.

Phase 4: *Negotiation Process with PAs:*

a. if there are providers who can fulfil all requirements, then the BCA selects the best vendor.
b. If there is no provider that can fulfil all requirements, then the BCA starts the negotiation process with PAs.

*Step 1***:** BCA selects the best initial offer from all offers that are proposed by all providers according to the objective.

*Step 2***:** BCA adjusts its initial offer according to the offer selected in **Step 1** to generate new counter offer and propose it to all providers.

*Step 3***:** A PA evaluates BCA's counter proposal.

Step 4: If the counter offer proposed by BCA cannot be accepted, PA proposes a counter offer.

*Step 5***:** Terminate negotiation. There are three termination conditions: First, when negotiation deadline expires. Second, when the offer is mutual agreed by both the CA and the PA. Third, when BCA is not able to accept any counter offer proposed by all providers within the negotiation deadline.

Phase 5: *SLA Generation:* Initiate SLA creator to generate SLA for customer and provider respectively using SLA templates stored in KB.

Phase 6: *Send SLA to all participants:* The generated SLA will be sent to the customer and provider respectively by the SLA creator.

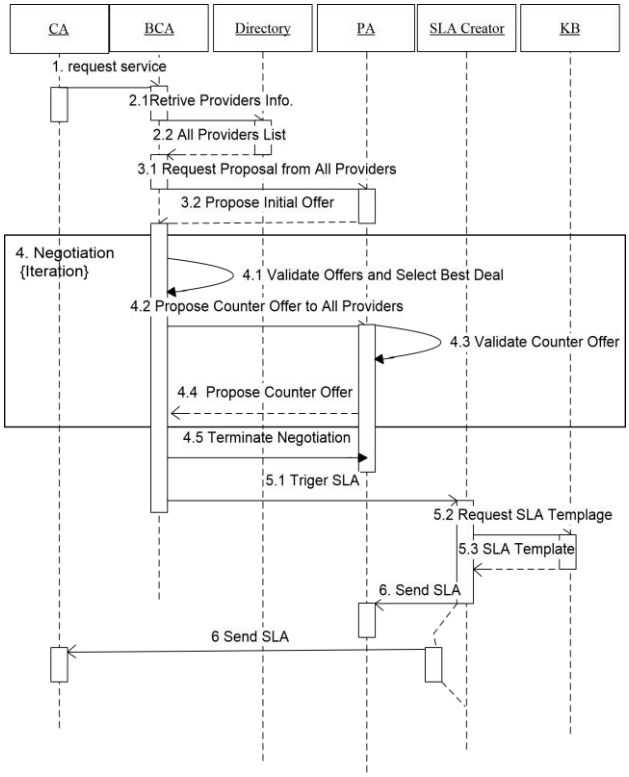

Fig 2. The Interaction between Components during Negotiation Process

D. Decision Making System

In the negotiation process, the action that a participant performs is determined by a decision making system. In the decision making system, three main questions need to be answered: 1) how to evaluate the offer; 2) what actions to take: accept, reject or generate counter offer; and 3) how to generate counter offer? Therefore the negotiation heuristics to answer them is designed from the broker and provider's perspectives.

Broker

International Journal of Engineering Applied Sciences and Technology, 2020
Vol. 5, Issue 5, ISSN No. 2455-2143, Pages 30-36
Published Online September 2020 in IJEAST (http://www.ijeast.com)

After **BCA** requests quotes from all PAs, each PA proposes an initial offer to the BCA, which selects the best offer and makes a decision. If the decision is to propose a counter offer, then the new counter offer will be proposed to all PAs. The best offer is selected based on different objectives. To consider cost-benefit objectives:

Minimum cost: selects the offer with the lowest price first and then the highest cumulative CSL for all QoS.

Maximize CSL: selects the offer with the highest cumulative CSL for all QoS first and then the lowest price.

Table 2: The Mincost Heuristic

Conditions	**Within BCA's expB**	**Exceed BCA's expB**
All QoS parameters are Satisfied	If deadline condition is urgent, agree. Otherwise decrease expB.	If expB is less than actual budget, then increase expB. Otherwise reject.
Not all QoS are satisfied	Satisfy all parameters and reduce expB.	Satisfy all parameters by negotiating on minimal (not desired) values.

Table 3: The *Maxcsl* Heuristic

Conditions	**Within BCA's expB**	**Exceed BCA's expB**
all QoS parameters are satisfied	If deadline condition is urgent, agree. Otherwise decreases the least preference parameter to decrease expB.	Decreases the value of parameters, which are better than expected to decrease price.
Not all QoS are Satisfied	Satisfy all parameters increases expB.	Increases expB.

After selecting the best offer, the broker needs to decide how to deal with the selected best offer. One of three actions can be adopted: accept, reject or generate counter offer according to negotiation heuristics. Therefore for these, two broker negotiation heuristics (*mincost* heuristic and *maxcsl* heuristic) to decide which action to take according to different objectives were designed.

In these two heuristics (Table 2 and 3), cost and other Issue values are calculated using negotiation strategy functions, where the most desired and the minimal acceptable values for each issue are considered for the broker. In both decision making heuristics, two criteria is used to evaluate the offer:

1. whether offer is within BCA's expected budget: whether the service price offered by provider *price*$_p$ is less than the broker's expected budget *expB*, and
2. whether all QoS parameters are satisfied

The above two criteria generate four combined conditions. For each condition, the decision making heuristics guide the broker to make different decisions on which Issue requires adjustment. There are two factors that require consideration when making adjustments. Firstly, trade-off between cost and QoS parameters depends on the objective. Secondly, when the broker must concede on QoS parameters, it always adjusts the least preferred parameter. After the broker decides which Issue to adjust, the new value of the Issue is calculated. The time complexity of these heuristics is *O(CPI)* depending on the number of customers (*C*), the number of providers (*P*) and the number of Issues (*I*).

b. Providers

Table 4: Provider's Decision Making Heuristic

Conditions	**Within BCA's expB**	**Exceed BCA's expB**
All QoS parameters are satisfied	If deadline condition is urgent, agree. Otherwise decrease the expB.	If expB is less than actual budget, increase expB. Otherwise decrease the QoS value.
Not all Satisfied	Satisfy all parameters and increase price.	Increase price.

The provider's objective is to maximize profit by accepting as many requests as possible. Therefore, the provider does not reject requests but continues to negotiate with each broker until negotiations have ended. Table 4 shows the provider's decision making heuristic.

E. Experimental Methodology

A prototype of the framework considering both time and market factors using real data shared by cloud provider CA Technologies was implemented. CA Technologies offers a number of enterprise software solutions to customers delivered as SaaS. The data provided included the response, refresh and processing times of an enterprise solution hosted on VMs, as measured by the quality assurance team. Availability data was collected from CloudHarmony benchmarking system (2019), which provides real data from Cloud providers. These data were collected over 4 days including weekdays, weekends.

Availability: Varies from 98.654% (Colosseum) to 100% (Amazon EC2) as derived from Cloud Harmony.

Process Time: The mean 5.243 (± 2.043) s.
Refresh Time: The mean 1.581 (± 1.383) s.

Cost: Cost is considered similar to Windows VMs from 3rd party IaaS providers, which varies from $0.34 per hour (VCloud Express) to $0.46 per hour (Amazon EC2).

IV. RESULTS

This section presents the performance results obtained from an extensive set of experiments done by comparing the proposed heuristics with the most recently proposed heuristic which is the baseline (Zukernine and Martin, 2011). The performance of each proposed heuristic depends on three factors: time, cost and market constraints. Therefore, to analyse how these heuristic can achieve customer, broker and provider's objectives, the following experimental scenarios were considered:

Impact of negotiation deadline (time factor): The impact of 4 sets of negotiation timeframes from the customer's perspective was observed; number 1 to 4 was used to represent the variation from 'very crucial' to 'very trivial'.

Impact of broker expected margin (cost factor): The impact of 4 sets of initial broker expected margins (varying from 20% to 50% over budget), were observed.

Impact of market factor: The impact of 4 sets of market factors (varying the ratio in relation to the number of providers and customers from less than 10%, 30%, 70%, and more than 90%), were observed. Numbers 1 to 4 were used to represent each set.

V. CONCLUSION

Once a request is accepted by the SaaS provider, there is a possibility for customers to change their existing requirements (such as add more accounts or upgrade service package). Thus, SaaS is expected to be scaled up and out dynamically according to the customers' QoS requirements.

As SaaS providers want to enlarge market share, they need to provide more flexibility in terms of services to cater to variations associated with an individual customer. This is generally done by a negotiation process between customers and service providers. However, while undertaking this negotiation process, the service provider needs to take into consideration not only what they can provide to customers but also the competition with other SaaS providers. Thus, the new negotiation frameworks proposed are needed for the SaaS provider that considers dynamism in Cloud environment with time and market factors to make the best possible decisions for negotiation. The proposed negotiation framework can be used for the SaaS provider and the SaaS broker model.

VI. REFERENCE

[1] Yeo, C. S., and Buyya, R. (2006). A Taxonomy of Market-based Resource Management Systems for Utility-driven Cluster Computing. Software: Practice and Experience (SPE), 36 (13), (pp.1381-1419).

[2] Buco, J., Chang, N., Luan, Z., Ward, C., Wolf, L., and Yu, S. (2004). Utility Computing SLA Management based upon Business Objectives. IBM Systems Journal, 43(1), (pp. 159- 178).

[3] Glen D., and Alfonso S. (2006). Towards unified QoS/SLA ontologies. In IEEE Services Computing Workshops, 2006. SCW '06 pages 169 –174.

[4] Congwu C., Lei L., and Jun W. (2007). Aop based trustable SLA compliance monitoring for web services. In Seventh International Conference on Quality Software, 2007. QSIC '07, pages 225 –230.

[5] Bastian K., and Lutz S. (2007). Towards autonomous sla management using a proxylike approach. Multiagent Grid Syst., 3(3):313–325

[6] Henar M., and Ioannis K. (2009). BREIN: Business objective driven reliable and intelligent grids for real business. International Journal of Interoperability in Business Information Systems, 3(1):39–42.

[7] Stefano F., Vittorio G., Fabio P., Michele P., and Elisa T. (2010). Qos-aware clouds. In 2010 IEEE 3rd International Conference on Cloud Computing, pages 321–328.

[8] Service Level Agreement in the Data Centre. (April 2002). Retrieved March 2020, from Sun Microsystems: http://www.sun.com/blueprints.

[9] Zukernine, F., and Martin, P. (2011). An Adaptive and Intelligent SLA Negotiation System for Web Services. IEEE Transactions of Service Computing, 4(1), (pp. 31-43).

[10] Kornel S., Jakub S., Renata S., and Jacek K. (2010). Application of the esb architecture for distributed monitoring of the sla requirements. In Proceedings of the 2010 Ninth International Symposium on Parallel and Distributed Computing, pages 203–210.

[11] Lee C., Wang, C., Zomaya Y., and Zhou B. (2010). Profit-driven Service Request Scheduling in Clouds. Proceedings of the International Symposium on Cluster Computing and the Grid (CCGRID). Melbourne, Australia.

[12] Brandic, I., Music, D., and Dustdar, S. (2009). Service Mediation and Negotiation Bootstrapping as First Achievements towards Self-adaptable Grid and Cloud Services. In Grids and Service-Oriented Architectures for Service Level Agreements. P. Wieder, R. Yahyapour, and W. Ziegler (eds.), Springer, New York, USA.

[13] CloudHarmony benchmarking system (2019), Retrieved on 06 December 2019: http://www.cloudharmony.com.

[14] Amazon web services, retrieved Nov 2019 from AWS: http://aws.amazon.com.