

A Catalogue of Analysis Operations for API Pricing Plans

Rafael Fresno-Aranda^[0000-0001-8480-5014], Pablo Fernandez^[0000-0002-8763-0819], and Antonio Ruiz-Cortés^[0000-0001-9827-1834]

SCORE Lab, I3US Institute, Universidad de Sevilla, España
{rfresno, pablofm, aruiz}@us.es

Abstract. In recent years, the usage of public APIs offered by third party providers has fostered the development of the API Economy. Companies use APIs as business assets and they bill consumers for their consumption. The pricing of an API usually includes multiple plans with different limitations and features, from which consumers may choose. In this scenario, consumers who wish to use the API may ask some questions about, for example, which plans to get or what they can do with a maximum budget. Answering these questions manually may be tedious and error-prone, so their automation would be useful.

Previous research about analysis operations for limitation-aware microservices architectures led us to the need to get a better understanding of operations for an API pricing. In this paper, we propose an initial catalogue of 11 analysis operations for usual pricing specifications. While it is not an exhaustive set, we intend to show the need to have a catalogue of operations to pave the way for further questions and automated tools.

Keywords: Catalogue · API · Pricing · Automation · Analysis.

1 Introduction

Over the last years, more and more companies have embraced the concept of *API Economy* [1]. In this scenario, companies provide their functionality and information through public APIs, which developers can integrate within their own applications.

Among the different types of APIs, the most popular are RESTful APIs [7]. The increasing popularity of RESTful APIs has fostered the creation of various formal specification languages to describe these APIs in a standardised way, from which the OpenAPI Specification (OAS) [2] has become the de-facto standard. This way, developers can leverage the information available and create automated tools that can, for example, generate test suites [17] or manage security [8].

The base OAS language does not include information about pricings, so, as of today, each API provider describes its pricings in different, non-standardised ways. This makes it difficult to develop automated tools that can analyse a pricing. There is an ongoing effort to create an extension of OAS, named SLA4OAI

[5], that would allow providers to formally specify the details of their pricings. Similar to the base OAS, this extension would foster the creation of a tool ecosystem that reads the information about a pricing to perform various automated tasks.

An example of a task that could be automated is providing answers to different questions, that a developer who wishes to use a public API may ask. For example, *What is the best plan for my needs?*, or *How many requests could I send with a maximum budget?*. Answering these questions manually may be tedious, time-consuming, and error-prone, especially if the pricing is complex and contains multiple plans and limitations.

Our experience providing answers to questions related to the capacity of microservice architectures that consume external APIs [10] has led us to the need to deepen our understanding of the different questions that can be asked about the pricing of an API. In this paper, we propose a catalogue of eleven analysis operations for usual pricing specifications. Analogous analysis operations have been defined in the context of the automated analysis of feature models [6], service level agreements (SLAs) [20,19,18,14], and OAS [16].

We may remark that it is not our intention to propose an exhaustive set of analysis operations, but to show that it makes sense to have a catalogue of questions for API pricings. To the best of our knowledge, there is no existing work that analyses the questions that can be asked about API pricings, so this preliminary work can be considered pioneering.

The rest of the paper is structured as follows: Section 2 provides an overview of the concept of pricing and its elements; Section 3 lists and classifies a set of analysis operations; Section 4 describes various works related to the topic of this paper; and Section 5 provides conclusions and details about future work.

2 Elements of a Pricing

While pricings from different providers may differ, most of them share a similar structure with multiple common elements. Each pricing comprises one or various *plans*, from which API consumers may chose. Some providers allow consumers to choose multiple plans simultaneously, or even select multiple instances of the same plan. Nonetheless, they usually limit the number of free plans to 1 to prevent consumers from potentially avoiding limitations by getting an infinite number of free plans. If a consumer does use multiple subscriptions, it is their responsibility to send requests using all plans. Each plan contains the following elements:

Price. The price of a plan indicates how much money the consumer has to pay the provider in order to use said plan. Most plans in real-world APIs have a periodic subscription model, where the consumer must pay periodically, e.g. \$50/month. In most cases, this period is monthly, but some providers also offer cheaper options for longer subscriptions, such as yearly. Some plans follow a pay-as-you-go model, where the consumer pays based on their actual usage of the

API, e.g. \$0.1/request. Some providers allow consumers to negotiate the price based on their specific needs.

Capacity limitations. Also known as *limitations*, they define how many resources of the API the consumer may use per period of time, e.g. 100 requests/day. Resources may be domain independent (such as requests or credits) or domain dependent (such as emails or weather forecasts). Nonetheless, many domain dependent resources can be mapped to a certain number of requests, e.g. sending an email may involve sending one request. Limitations are usually divided into two groups:

- **Rates** are usually defined over shorter period of times, such as seconds. They define the *speed* at which the consumer may send requests. They are commonly used to prevent the consumer from sending short bursts of requests that may affect the performance of the API. Many pricings have the same rates for all of their plans.
- **Quotas** are defined over longer period of times, such as days or months. These limitations represent how much of the API the consumer may use. Commonly, quotas are the defining factor of a plan, with cheaper plans having lower quotas and viceversa. Some plans may include **overage costs**, which are additional costs billed when the consumer exceeds the quota, e.g. \$0.1 per additional request.

Features. Some pricings may include access to extra features in more expensive plans, e.g. an email API might include additional email templates or the ability to add a signature. In some cases, these features are related to SLA terms, such as access to phone support, email support or guaranteed uptime and response time.

Figure 1 shows an example of a pricing. This pricing corresponds to the SendGrid API, an email service by Twilio. More specifically, the pricing is a simplified version offered at RapidAPI [4], which is an API hub where developers can manage subscriptions to various APIs. The original pricing does not include overage costs or features, so we have extended it using the overage costs listed at [3] and a feature that is common in other public APIs.

Basic	Pro	Ultra	Mega
\$0/month	\$9.95/month	\$79.95/month	\$199.95/month
10 req/s	10 req/s	10 req/s	50 req/s
50 emails/day	40000 emails/month	100000 emails/month	300000 emails/month
Ovg: \$0.001/email	Ovg: \$0.001/email	Ovg: \$0.00085/email	Ovg: \$0.00005/email
No support	No support	24/7 phone support	24/7 phone support

Fig. 1. Extended RapidAPI SendGrid pricing. Each plan has a price, rate, quota, overage cost and, in some cases, a feature.



Some pricings define capacity limitations in terms of *credits*. Each plan has a maximum number of credits per month, and each operation (e.g. each endpoint) costs a specific number of credits. This way, providers can fine-tune the cost of each individual operation. Moreover, a single operation may have different costs depending, for example, on its parameters.

Besides a pricing, APIs usually have certain terms of use and SLAs, including compensations and penalties for both clients and providers. These elements are out of the scope of this paper, which focuses on pricings. However, a more detailed analysis of APIs should include all of their elements.

3 Analysis Operations for Pricing Plans

The main goal of this paper is to show that it makes sense and it is necessary to have a catalogue of analysis operations for an API pricing. By doing so, we aim to build the foundations for a tooling ecosystem that leverages the information of these operations and provides useful results.

In this paper, we focus on consumer-oriented operations and we aim to solve questions related to the selection of plans from a pricing. This way, a consumer may make a decision based on their needs, ensuring that the selected plans fit their specific criteria and budget.

In [10], we devised a catalogue of analysis operations for LAMAs (Limitation-Aware Microservices Architectures). A LAMA is a microservices architecture that consumes external APIs with pricings. In the paper, we identified three main variables from which most analysis operations can be created: **number of requests**, **cost** and **time**. By fixing two variables and optimising the third one, we obtain three basic analysis operations. From them, any number of derivative operations can be devised. These three variables serve as a starting point for our catalogue of analysis operations for a single pricing. Nevertheless, we introduce new elements that were not discussed in the aforementioned paper, such as features.

3.1 Classification of Analysis Operations

In [10], we grouped the set of analysis operations by their optimised variable. In this paper, we now group the operations by one of the fixed variables. This way, we better reflect the goals of these operations. Usually, consumers already have a fixed subscription, a fixed set of features that they want, or a specific number of requests that they need to send to the API. Given this scenario, they then ask questions that involve optimising one of the three main variables. Therefore, the classification that we present in this paper is as follows:

- **Given a specific set of subscriptions.** In this scenario, the consumer is already subscribed to the API. They may have multiple subscriptions to the same plan, and/or simultaneous subscriptions to different plans.

- **Given a maximum budget.** In this case, the consumer is not yet subscribed to the API, and has a maximum budget that they are willing to spend. Based on their needs, they will need to choose between the available plans.
- **Given a desired capacity.** The consumer has specific capacity needs, usually based on the needs of their own customers. This capacity is translated into a minimum number of requests that they need to send to the API.
- **Given a desired set of features.** In this scenario, the consumer wants a specific series of features, which may or may not be available for all plans.
- **Given a desired pricing.** This case corresponds to Software as a Service (SaaS) scenarios, where the API consumer is also offering its own pricing to their customers. The offered pricing needs to be aware of the external API pricing.

These categories are not mutually exclusive, and one analysis operation may be classified into more than one category. Nonetheless, we consider that one of the categories is always the “most appropriate” for a specific operation.

3.2 Catalogue of Analysis Operations

Following the classification in the previous subsection, we now enumerate the different analysis operations under each category. Operations will be identified as O_i , where i is the operation number. This makes it easier to refer to specific operations. For each operation, we include some examples using the extended RapidAPI SendGrid pricing that was presented in Figure 1. We will consider that sending one email requires sending one request.

Given a Specific Set of Subscriptions Analysis operations classified under this category are as follows:

O_1 . **How much time do I need to send N requests?** With their current subscriptions, the consumer wants to know how long it will take to send some requests, taking into account the capacity limitations of the subscribed plans.

Example: let us assume that we want to know how long it will take to send 50000 emails and we have one subscription to plan Pro. In this scenario, we would need 1 month and 1000 seconds to send all emails. If we were to upgrade our subscription to plan Ultra, we would only need 5000 seconds. Furthermore, if we are subscribed to Pro and are willing to use overage costs, then we could also send all emails in 5000 seconds. This would come with an additional cost, but it would be less than upgrading to Ultra.

O_2 . **Can I send N_1 requests per second (RPS) and N_2 requests per day (RPD)?** If the answer is negative, a useful auxiliary operation is **Why not?**. This would help the consumer pinpoint the specific rate or quota that is acting as a bottleneck.

Example: we have one subscription to plan Pro and want to send 10 RPS and 2000 RPD. This would not be possible, as 2000 RPD is equivalent to 60000

requests per month (RPM), which exceeds the quota of Pro. We would either need to upgrade to Ultra or use overage costs.

O_3 . What is the maximum speed at which I can send requests, and for how long? The rates of the subscribed plans will define the maximum speed at which the consumer may send short bursts of requests, while the quotas will determine how many bursts can be sent.

Example: we have one subscription to plan Pro. Therefore, the maximum speed at which we can send requests is 10 RPS, which is the rate of Pro. At this speed, we would consume the entire quota in 4000 seconds. We would then need to wait until next month for the quota to be reset. If we are willing to use overage costs, we could extend this time.

O_4 . What is the maximum time that I will need to wait after consuming all quotas? If the consumer uses all of their available quotas and there are no overage costs (or the consumer does not want to use them), they will need to wait until the quotas are reset. Some APIs actually return the waiting time when a consumer runs out of requests, but only for a single subscription. If the consumer has multiple subscriptions, this information might not be as useful.

Example: let us take the example in O_3 as a starting point. At full speed, we would consume the quota of Pro in 4000 seconds. After that, we would need to wait until next month for the quota to be reset. If we consumed the quota during the first 4000 seconds (which is a bit more than 1 hour) of a 30-day month, we would need to wait 29 days and a bit less than 23 hours for the quota to be reset.

Given a Maximum Budget For this category, all operations O_1 through O_4 still apply. However the first step before solving these operations is determining which subscriptions to get for the given budget. This leads to an additional operation:

O_5 . What is the optimal set of subscriptions to get in a specific scenario? Depending on the circumstances and the consumer needs (e.g. the scenarios in the previous 4 operations), the optimal set will vary. Nonetheless, this set must always take the maximum budget into account.

Example: let us assume that we want to send 50000 RPM and have a maximum budget of \$100. At a first glance, we could simply get one subscription to Ultra, which is \$79.95 and has a quota of 100000 RPM. Nonetheless, we could potentially get many Basic subscriptions for \$0, enough to allow our desired 50000 RPM. However, let us assume that SendGrid limits Basic subscriptions to 1 per client. In that case, we could get 2 subscriptions to Pro, which would be \$19.8 and have a total quota of 80000 RPM.

Given a Desired Capacity Under this category, the analysis operations are the following:

O_6 . What is the cheapest set of subscriptions that meets the desired capacity requirements? This operation would return the set of subscriptions with the lowest total cost that allows the consumer to fulfill their needs.

Example: assuming that we want a capacity of 50000 RPM, the example in O_5 would still apply. The difference is that there is no maximum budget in this operation.

O_7 . **What percentage of the desired capacity is met with a specific set of subscriptions?** If the consumer already has a set of subscriptions or a budget constraint, they may want to know how much of the desired capacity is actually able to be fulfilled.

Example: we want a capacity of 50000 RPM, and, for any specific reason, we want to use one subscription to Pro, which has a quota of 40000 RPM. In this scenario, we are meeting 80% of the desired capacity.

Given a Desired Set of Features This category contains the following operations:

O_8 . **What is the cheapest set of subscriptions that meets the desired feature requirements?** Similar to O_6 , this operation would return the optimal set of subscriptions that meets the consumer needs, but focusing on features instead of capacity.

Example: let us assume that we want to have phone support from SendGrid. Even if we just wanted to send 1000 RPM, neither the Basic or Pro plans include phone support. Therefore, the cheapest option in this case is one subscription to the Ultra plan.

O_9 . **What amount of desired features is met with a specific set of subscriptions?** Similar to O_7 , the consumer may want to know how many of the desired features can be fulfilled with a given set of subscriptions or a budget constraint.

Example: if we want to have phone support and we are subscribed to the Pro plan, then we are meeting 0 of the desired features.

Given a Desired Pricing In a similar way to the previous two categories, this category includes two analysis operations:

O_{10} . **What is the cheapest set of subscriptions that meets the desired pricing requirements?** Similar to O_6 and O_8 , the consumer wants to know the cheapest subscriptions to get in order to meet the needs of their own customers.

Example: let us assume that we want to offer a pricing with 10 RPS and 50000 RPM. In this scenario, the cheapest option would be to get two subscriptions to Pro. However, if we want to offer this pricing to multiple simultaneous customers, we would need to multiply the limitations by the number of customers, e.g. for 10 customers we would need 100 RPS and 500000 RPM. This would increase the required cost to subscribe to the SendGrid API.

O_{11} . **What percentage of the desired pricing is met with a specific set of subscriptions?** Again, similar to O_7 and O_9 , the consumer already has a set of subscriptions or a budget constraints and may want to know how much of their needs can be fulfilled.

Example: if we want a pricing with 10 RPS and 50000 RPM and we only get one subscription to Pro, then we are meeting 100% of the desired rate but only 80% of the quota. These percentages could be aggregated into one, depending on the consumer needs and preferences.

4 Related Work

The concept of a catalogue of analysis operations is not new. The creation of a catalogue paves the way for future research on automated tools that provide answers to the operations. There have been multiple proposals of catalogues for various elements related to service-oriented computing. Nonetheless, to the best of our knowledge, there are no existing catalogues for API pricings.

Our previous work on the topic of automated capacity analysis of LAMAs includes a basic version of a catalogue. In [10], we presented three analysis operations for LAMAs, from which any number of derivative operations can be created. In the same paper and in [11], we presented Smart LAMA, a tool to automate these operations. Given that a LAMA is a microservices architecture that consumes various external APIs, the analysis of a single pricing can be represented as a LAMA with one service and one external API. We have used Smart LAMA to solve some of the analysis operations presented in this paper; however, the tool lacks support for certain elements such as features. Furthermore, the transformation of a LAMA into a constraint satisfaction and optimisation problem (CSOP) presented in the paper is quite complex for the specific case of a single pricing and does not scale well. Nonetheless, we believe that the catalogue for a LAMA and the catalogue for a pricing are related and share some operations.

While we have focused on consumer-oriented operations, it is worth noting that API providers also benefit from their own operations, the main one being *Is my pricing valid?*. The validity of a pricing is defined in [9] as the absence of validity conflicts based on a series of validity criteria, so that the provider can ensure the correctness of its pricing. The same paper also introduces a tool that automatically analyses the validity of a pricing.

In the field of RESTful API pricings, the work by Gamez-Díaz et al. [13,12] includes an analysis of API offerings in the industry and a list of all elements commonly found in real-world API pricings. These papers serve as a foundation for our catalogue, as the analysis operations revolve around these elements. The same authors also introduced the SLA4OAI extension for OAS [5], although it has not been formally published yet.

In [21], Molino-Peña et al. propose a catalogue of operations for terms of use in customer agreements (CA). The authors divide a CA into three sections: terms of use, pricing and SLA. They only focus on terms of use as they found little research in the literature about them. Therefore, they do not include any operation for the other two sections, including pricing.

In [15], García et al. present a model for pricing and billing information in the context of CA. However, this model does not account for multiple plans,

different limitations (such as rates and quotas) or features. Therefore it is not suitable for API pricings, which include the aforementioned elements.

5 Conclusions and Future Work

In this paper, we have presented a first iteration of a catalogue of 11 consumer-oriented analysis operations for API pricings, along with a classification based on different scenarios. This catalogue will be extended with new operations in the future, and we aim to make it as exhaustive as possible. The creation of a catalogue paves the way for automated tools that provide answers to these operations. This way, consumers who use external APIs can make decisions based on their specific scenarios and needs.

Besides extending the catalogue, we plan to develop a tool that takes an SLA4OAI specification file as input and automatically answers the operations in the catalogue. Our approach would follow the one in [10], where we transformed a LAMA into a CSOP. In this case, we would transform the elements of a pricing into a CSOP, and create the corresponding constraints for each analysis operation. The declarative nature of a CSOP makes it easier to further extend the catalogue by simply adding new constraints and variables as needed. The next step would be creating an interface so that the consumer can easily run the various operations. Following the recent popularity of artificial intelligence, we could program a chatbot that takes a query in natural language and translates it into a specific operation. It would then run the operation and return the answer in a easily readable format for the consumer.

Acknowledgements This work has been partially supported by the following grants: PID2021-126227NB-C21, PID2021-126227NB-C22, TED2021-131023B-C21, TED2021-131023B-C22 and PDC2022-133521-I00 which are funded by MCIN/AEI/10.13039/501100011033 and “ERDF a way of making Europe”; grants PYC20 RE 084 US, P18-FR-2895, US-1264651 and US-1381595 which are funded by Junta de Andalucía/ERDF,UE; and grant FPU19/00666 which is funded by MCIN/AEI/10.13039/501100011033 and by “ESF Investing in your future”. The authors would also like to thanks the reviewers for their valuable comments.

References

1. How to build an API economy for your enterprise - Capgemini, <https://www.capgemini.com/2020/12/how-to-build-an-api-economy-for-your-enterprise/>
2. OpenAPI Specification 3.1.0, <https://spec.openapis.org/oas/v3.1.0>
3. SendGrid API Overview, <https://rapidapi.com/blog/sendgrid-api-overview/>
4. SendGrid API: Pricing & Cost, <https://rapidapi.com/sendgrid/api/sendgrid/pricing>
5. SLA4OAI Research Specification, <https://github.com/isa-group/SLA4OAI-ResearchSpecification>



6. Benavides, D., Segura, S., Ruiz-Cortés, A.: Automated analysis of feature models 20 years later: A literature review. *Information systems* **35**(6), 615–636 (2010)
7. Fielding, R.T.: *Architectural styles and the design of network-based software architectures*. University of California, Irvine (2000)
8. Fresno, R.: Easy security management over microservices architectures based on OpenAPI Specification. In: Proc. XV Jornadas de Ingeniería de Ciencia e Ingeniería de Servicios . SISTEDES (2019), <http://hdl.handle.net/11705/JCIS/2019/020>
9. Fresno-Aranda, R., Fernández, P., Ruiz-Cortés, A.: SLA4OAI-Analyzer: automated validation of RESTful API pricing plans. In: Proc. 23rd International Conference on Web Engineering (In press). Springer (2023)
10. Fresno-Aranda, R., Fernández, P., Durán, A., Ruiz-Cortés, A.: Semi-automated capacity analysis of limitation-aware microservices architectures. In: Proc. 19th International Conference on the Economics of Grids, Clouds, Systems, and Services. pp. 75–88. Springer (2023)
11. Fresno-Aranda, R., Fernández, P., Ruiz-Cortés, A.: Smart LAMA API: Automated Capacity Analysis of Limitation-Aware Microservices Architectures. In: Proc. XVII Jornadas de Ingeniería de Ciencia e Ingeniería de Servicios (JCIS). SISTEDES (2022), <http://hdl.handle.net/11705/JCIS/2022/032>
12. Gamez-Diaz, A., Fernandez, P., Ruiz-Cortes, A.: An analysis of RESTful APIs offerings in the industry. In: 15th International Conference on Service-Oriented Computing (ICSOC). pp. 589–604. Springer (2017)
13. Gamez-Diaz, A., Fernandez, P., Ruiz-Cortés, A., Molina, P.J., Kolekar, N., Bhogill, P., Mohaan, M., Méndez, F.: The role of limitations and SLAs in the API industry. In: Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE). pp. 1006–1014 (2019)
14. García, J.M., Fernández, P., Pedrinaci, C., Resinas, M., Cardoso, J., Ruiz-Cortés, A.: Modeling service level agreements with linked usdl agreement. *IEEE Transactions on Services Computing* **10**(1), 52–65 (2016)
15. García, J.M., Martín-Díaz, O., Fernandez, P., Müller, C., Ruiz-Cortés, A.: A flexible billing life cycle for cloud services using augmented customer agreements. *IEEE Access* **9**, 44374–44389 (2021)
16. Martín-Lopez, A., Segura, S., Müller, C., Ruiz-Cortés, A.: Specification and automated analysis of inter-parameter dependencies in web APIs. *IEEE Transactions on Services Computing* **15**(4), 2342–2355 (2021)
17. Martín-Lopez, A., Segura, S., Ruiz-Cortés, A.: RESTest: automated black-box testing of RESTful web APIs. In: Proc. 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA). pp. 682–685 (2021)
18. Müller, C., Gutierrez, A.M., Fernandez, P., Martín-Díaz, O., Resinas, M., Ruiz-Cortés, A.: Automated validation of compensable slas. *IEEE Transactions on Services Computing* **14**(5), 1306–1319 (2021)
19. Müller, C., Oriol, M., Franch, X., Marco, J., Resinas, M., Ruiz-Cortés, A., Rodriguez, M.: Comprehensive explanation of sla violations at runtime. *IEEE Transactions on Services Computing* **7**(2), 168–183 (2013)
20. Müller, C., Resinas, M., Ruiz-Cortés, A.: Automated analysis of conflicts in ws-agreement. *IEEE Transactions on Services Computing* **7**(4), 530–544 (2013)
21. Peña, E.M., Garcia, J.M., Cortés, A.R.: Operaciones de Análisis sobre los Términos de Uso en Customer Agreements. In: Proc. XVII Jornadas de Ingeniería de Ciencia e Ingeniería de Servicios. SISTEDES (2022), <http://hdl.handle.net/11705/JCIS/2022/041>