Multicloud API binding generation from documentation

Michał J. Gajda[1], Vitor Vitali Barrozzi[1], and Gabriel Araujo[1]

Migamake Pte Ltd migamake@migamake.com https://www.migamake.com

1 Introduction

We propose a solution for low-level integration cloud service integration among different providers without additional support from the cloud API vendor.

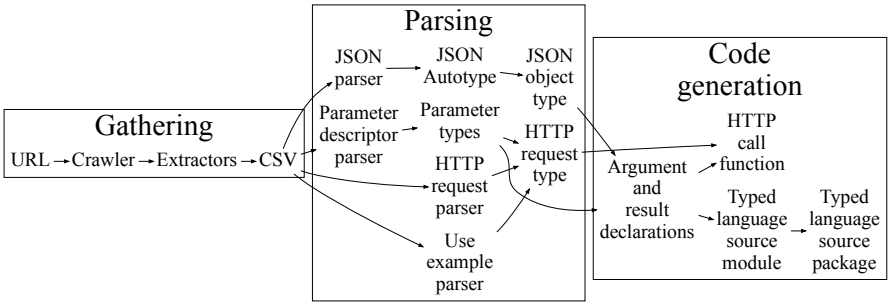
Multicloud service orchestration allows leveraging the full power of specialized, scalable best-of-a-kind services in the cloud. Its advent coincides with the so-called *no-code* and *low-code* solutions that enable rapid prototyping of cloud applications by small teams of startup companies. While wonderful in theory, but in practice SDKs usually address only a few programming languages, descriptions are untyped and incomplete, API description languages like Swagger/OpenAPI[10] are rarely used. That means that initial implementation is often an incoherent mix of incompatible scripts, deployment is only partly automated, and the source code uses a variety of platforms with varying level of automation.

An attempt to integrate these stumbles upon the barrier of cloud APIs documented in many different ways, usually *ad-hoc* and without particular rigour. The API interface bindings often contain tacit assumptions, untyped JSON or text-based bindings.

We solve this significant and essential interoperability problem by automatically parsing API documentation, and then generating API bindings in any chosen programming language[1]. The other efforts to support cloud API bindings by code generation are limited to single API and single language [1, 2, 4], or require a significant effort of handwriting Swagger/OpenAPI declarations for entire API[13], if the vendor did not generate them.

[1] We currently generate Haskell[5] code, but developing generation to any other typed language is offered as a paid service upon request.

2 Implementation



Our solution (*hURL*), is a *data pipeline*, divided into three different stages: (I) data gathering from the web pages (II) parsing and analysis (III) code generation.

At the initial stage of *data gathering*, we gain a complete description of each cloud API call with Scrapy library[11] using Python. It allows us to download HTML pages with Chrome, then use XPath[3], CSS[12], or JQ[6] selectors to extract data in a structured manner. As part of a *data analysis* stage implemented in Haskell[5], we parse many possible formats: (a) HTTP request path description with variables (b) cURL command options (3) extracts from tables of parameter descriptions. The parameter descriptions are tagged with the *cloud call convention*[2]: (I) as part of an HTTP request path (II) URL-encoded query parameter (3) part of a request body as JSON or plain text (4) HTTP header or cookie.

As a last state, we *generate code* in a typed programming language, we first make a reference data structure that describes functions and types in the target language. This part is language-independent except for function generating the identifiers themselves. The second stage of binding generation is code generation itself; we use techniques previously described in [7–9]. Following the best current practice, we also attach links to the original documentation website, which allows user to cross-reference the information with the original API documentation.

3 Conclusion

We implemented the retargetable code generator for cloud API bindings that presents the following benefits: (1) provide a binding for thousands of API calls within months; (2) language retargeting with little effort; (3) the systematic approach allows easy scaling to a number APIs; (4) removes a dependency on the cloud API provider support; (5) it significantly reduces maintained code base as compared with handwritten cloud API bindings. We offer to generate cloud API bindings for other programming languages and other cloud APIs as a paid service.

[2] Cloud API calls differ from binary function calls in this respect, and a different argument passing convention may be assigned for each parameter.

Bibliography

[1] Amazonka – A comprehensive Amazon Web Services SDK for Haskell: 2013. *https://github.com/brendanhay/gogol*.
[2] Boto 3 - The AWS SDK for Python, release 1.12.9: 2020. *https://github.com/boto/boto3*.
[3] Clark, J. and (eds.), S.D. 1999. *XML path language (XPath) version 1.0*. W3C.
[4] Gogol – A comprehensive Google Services SDK for Haskell: 2015. *https://github.com/brendanhay/gogol*.
[5] Haskell 2010 Language Report: *https://www.haskell.org/definition/haskell2010.pdf*.
[6] jq is a lightweight and flexible command-line JSON processor: 2012. *https://stedolan.github.io/jq/manual/*.
[7] JSON autotype: Presentation for Haskell.SG: 2015. *https://engineers.sg/video/json-autotype-1-0-haskell-sg--429*.
[8] Michal J. Gajda, D.K. 2020. Fast XML/HTML tools for Haskell: XML Type-lift and improved Xeno. Manuscript under review.
[9] Michal J. Gajda, V.V.B. Do not give us a bad name.
[10] OpenAPI 3.0.2 Specification: 2018. *https://swagger.io/docs/specification/about/*.
[11] Scrapy: *https://scrapy.org/*.
[12] Selectors Level 3: 2011. *https://www.w3.org/TR/2011/REC-css3-selectors-20110929/*.
[13] SwaggerHub: *https://swagger.io/tools/swaggerhub/*.