AQG4SD: Automated GraphQL Query Generation for Cloud Service Discovery and Selection

Yann Ramusat[1,*], Vincent Tran[1], Sébastien Nguyen[1] and Zakaria Maamar[2]

[1]Devoteam Research, 1 Rue Galvani, 91300 Massy, France
[1]University of Doha for Science and Technology, Doha, State of Qatar

Abstract

This paper demonstrates AQG4SD (*Automated Query Generation for Service Discovery*), an interactive tool that automates the discovery and selection of cloud services from user requirements defined in natural language. Leveraging Large Language Models (LLMs) and automated GraphQL query generation, the tool outperforms existing solutions by dynamically binding real-time public public cloud pricing APIs. AQG4SD has the capacity of translating user requirements into actionable GraphQL queries, even when confronted to APIs not natively optimized for complex selection criteria. Preliminary experimental results are promising demonstrating the robustness of AQG4SD along with its suitability for an efficient and automated cloud resource management.

Keywords

Cloud Service, Discovery, Selection, Generative AI, GraphQL, LLM.

1. Introduction

Cloud computing is ubiquitous in the modern development and deployment of applications, providing highly accessible, on-demand, and elastic services to companies willing to trade capital expenditures for operational expenditures by mitigating upfront costs [1]. FinOps strategies[1], which focus on optimizing cloud costs through judicious service discovery and selection, can significantly benefit from academic research outcomes in service discovery and selection. These outcomes identify services based on their functional capacities (e.g., service configurations, RAM and vCPUs) and subsequently select the most appropriate ones based on *Quality of Service* (QoS) criteria and other non-functional requirements, among which price is often cited as the most critical factor.

An analysis of existing solutions reveals two primary paradigms: ontology-based and assistant-based systems. The former offer the distinct advantage of consolidating diverse cloud service catalogs (e.g., Azure, AWS, and GCP) into a common, semantically rich format [2, 3, 4]. This is achieved by utilizing a robust set of concepts, including entities, attributes, and their inter-relations, typically formalized through semantic Web languages such as *Resource Description Framework* (RDF) and *Web Ontology Language* (OWL). Similarly, the latter typically rely on centralized service description catalogs, which are either constructed through Web crawling and *Natural Language Processing* (NLP) reasoning, as explored in [5], or populated by Web search engines leveraging an underlying ontology, as demonstrated in [6]. Both paradigms then support the description of user requirements in natural language and provide semantic similarity-based matching against their respective constructed service catalogs. However, their primary limitation stems from their inherent inability to address the real-time synchronization of dynamic cloud service catalogs within the ontology, as services are continuously evolving through updates or the emergence of new offerings.

To address the limitations above, we present AQG4SD (*Automated GraphQL Query Generation for Cloud Service Discovery and Selection*), a novel interactive tool designed to automate the discovery and selection of cloud services based on user requirements defined in natural language. The tool advocates

Cooperative Information Systems - Early Research Achievement and Demos, 2025
*Corresponding author.
yann.ramusat@devoteam.com (Y. Ramusat); vincent.tran@devoteam.com (V. Tran); sebastien.duckhang.nguyen@devoteam.com (S. Nguyen); zakaria.maamar@udst.edu.qa (Z. Maamar)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

[1]https://www.finops.org/framework.

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-05-05

CEUR-WS.org/Vol-4203/paper06.pdf

for automated GraphQL query generation by binding publicly available APIs to enable a dynamic provisioning of up-to-date contextual information such as cloud providers' service offerings and pricing to *Large Language Models* (LLMs).

The remainder of this paper is organized as follows. Section 2 is an overview of foundational concepts. Section 3 presents the internal architecture of AQG4SD. Section 4 outlines the demonstration scenario. Section 5 discusses the tool's maturity and showcases preliminary performance assessments. Finally, Section 6 concludes the paper and outlines future work.

2. Foundations

GraphQL is an open-source[2] query language for APIs and a runtime for fulfilling data requests, enabling clients to precisely specify the data they need. Companies exposing services and data via a GraphQL endpoint define their own *schema*, which precisely describes the available information and how it can be queried or modified through operations, e.g., *queries* for data retrieval and *mutations* for data modification.

Infracost is a widely used tool for FinOps optimization[3]. Based on a given Terraform configuration, it estimates deployment costs and can even trigger configured alerts if predefined thresholds are exceeded. Underlying this tool is a GraphQL API called *Cloud Pricing API*[4] that provides real-time cost data for cloud services across major platforms, e.g., AWS, Azure, and GCP.

Cloud service discovery identifies necessary services from available catalogs based on specified functional (e.g., RAM) and non-functional user requirements (e.g., hourly costs). Following discovery, cloud service selection ranks and considers those services that satisfy some QoS criteria and/or optimization functions.

Infracost API exposes comprehensive functional details about cloud service offerings, such as vCPU count and RAM capacity. However, its GraphQL endpoint does not natively support complex ranking-based queries which are essential for service discovery and selection [2, 4]. This lack of support worsens as it fails to filter services based on minimum requirements (e.g., at least 8GB of RAM) or to provide native ranking options. This requires external emulation of advanced filtering (e.g., `min`, `max`, and `limit`) and sorting operators

3. The Tool in a Nutshell

AQG4SD integrates a suite of advanced strategies for efficient text-to-GraphQL generation. Notably, several of these strategies are inspired by and adapted from the literature in text-to-SQL generation [7]. Specifically, they encompass advanced prompt engineering techniques, including structured prompt design, in-context learning, and schema injection [8]. In addition, AQG4SD supports multi-query orchestration, a critical feature for effectively querying APIs. The capability to query a GraphQL API endpoint is exposed through a dedicated mechanism, leveraging a *Model Context Protocol* (MCP) server.

AQG4SD's overall architecture encompasses the following key components, designed for modularity and efficient interaction:

- **Front-end Interface:** Built using Streamlit Python library, it allows users to engage in a conversational manner, outlining their cloud service requirements.
- **MCP Server:** Exposing specialized tools, such as 'process_user_query', which executes the generated GraphQL queries against public APIs and performs the necessary result aggregation.
- **Orchestrator (MCP Client):** Implementing the tool's core business logic, it retrieves the above-mentioned tools from the MCP server prior to initializing the LLM, so that the GraphQL queries are generated. To this end, the orchestrator injects the relevant schema as context into the LLM

[2] https://graphql.org.
[3] https://www.finops.org/members/infracost.
[4] https://www.infracost.io/docs/supported_resources/cloud_pricing_api/

and provides examples for in-context learning to the LLM as well. The orchestration logic itself is implemented using LangGraph.

4. Demonstration

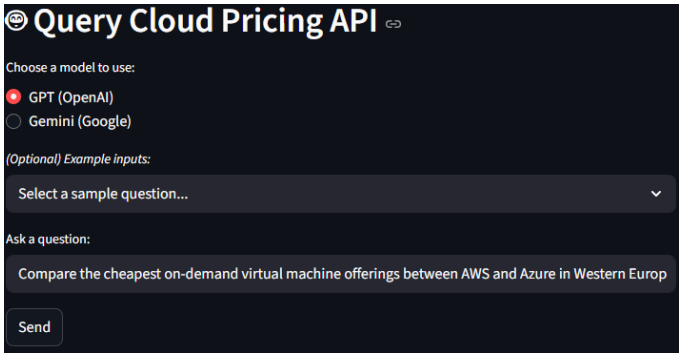
Our demonstration guides attendees through the interactive capabilities of AQG4SD, showcasing its ability to automate cloud service discovery and selection from natural language input. We also illustrate how the tool addresses the complexity of real-world API interactions and satisfy non-functional requirements.

4.1. Optimizing Application Infrastructure Selection Scenario

The scenario refers to a use-case scenario involving a user seeking to deploy an application's infrastructure, such as a GitLab CE instance linked with Keycloak for identity management. It starts with the user formulating their infrastructure needs in natural language, such as "Compare the cheapest on-demand virtual machine offerings between AWS and Azure in Western Europe, with a minimum of 8GB RAM and 4 vCPUs." in the dedicated front-end interface of AQG4SD, showcased in Figure 1a. The following interactive steps will be rolled out:

1. **Automated GraphQL Query Generation:** The tool processes the natural language input, leveraging LLMs and advanced prompt engineering, to automatically generate the corresponding GraphQL queries. These queries are designed to retrieve relevant service offerings and pricing information from public APIs.
2. **Real-time API Interaction and Emulation:** The tool executes the generated GraphQL queries against real-time public APIs, such as Infracost's API. A key aspect demonstrated here is the ability to emulate advanced filtering (e.g., `min`, `max`, and `limit`) and ordering operators, which are not inherently supported by the underlying GraphQL API.
3. **Result Inspection and Analysis:** The obtained results, including service configurations, pricing details, and a ranking based on user-defined criteria, are presented to the user. Attendees will be able to inspect the generated GraphQL queries and the intermediate API responses for further transparency and verification, as shown in Figure 1b.

This scenario highlights AQG4SD's capacity to handle complex functional and non-functional constraints, including those inspired by test cases from [3, 4], which often utilize query languages like SPARQL and DL-lite.



(a) User input via front-end.

(b) Inspection of a generated GraphQL query.

Figure 1: Interactive use of AQG4SD.

4.2. Interactive Experience for Attendees

The demonstration provides attendees with a direct opportunity for hands-on interaction. They will formulate their own custom natural language queries and observe the tool's real-time response. This

direct engagement should offer first-hand insights into AQG4SD's capabilities, its adaptability across various service discovery and selection use cases, and its current operational limitations, fostering a comprehensive understanding of its potential.

5. Maturity and Performance Assessment

As a research prototype, AQG4SD has achieved a robust level of maturity, offering versatile deployment options. It can be seamlessly used either through an interactive GUI for direct user engagement or programmatically via an API endpoint. This API, which exposes its capabilities via an MCP (Model Context Protocol) server, facilitates its seamless composition within automated workflows and integration into advanced agentic systems.

For practical application and evaluation, AQG4SD requires users to provide API keys for both a generative model (OpenAI or Gemini) and the underlying cloud pricing service (Infracost). It is noteworthy that Infracost's free plan offers sufficient functionality for comprehensive testing. To ensure full transparency and facilitate external research, the tool's source code is made publicly available on GitHub[5]. Additionally, a comprehensive 4-minute video preview, showcasing the usage and features of the entire tool, is accessible online[6].

5.1. Evaluation

We conducted preliminary assessments of the tool's efficiency in generating correct GraphQL queries from user inputs, focusing on foundational capabilities. Our investigation addresses the Infracost API's limitations, which miss fail to provide advanced filtering (e.g., `min`, `max`, `limit` and `order by`) and often require one to six orchestrated GraphQL queries per request.

To understand the contribution of our optimization strategies, we performed an ablation study, summarized in Table 1 comparing the *accuracy* of query generation (defined as the match of generated queries to ground-truth queries) with and without schema injection and in-context learning. For a baseline, we also compared the generated GraphQL queries to those human-crafted, parameterized GraphQL queries – where the LLM only has to fill parameters, referred to as "Prepared queries" in the table. While accurate for known cases, this baseline limits adaptability, as developers must pre-define all query structures, hindering responses to novel user requests. All accuracy values presented are averaged over five independent runs to mitigate the inherent non-determinism of Large Language Models.

Table 1
Accuracy results for GraphQL query generation strategies for five distinct queries (Q1–Q5).

Generation Strategy	Q1	Q2	Q3	Q4	Q5	Mean Accuracy
Schema, In-context	1.0	1.0	1.0	0.6	0.4	0.8
Schema only	0	0.2	0.0	0.0	0.2	0.08
In-context only	1.0	0.8	0.8	0.8	0.2	0.72
Prepared queries	1.0	1.0	0.0	1.0	1.0	0.8

As shown in Table 1, AQG4SD achieved an overall mean accuracy of 0.8, a performance competitive with the 0.8 accuracy yielded by the "Prepared queries" baseline. This substantiates our system's capacity to rival human-crafted query generation while preserving its inherent generality and adaptability to novel queries. Notably, the results for the third query (Q3) critically highlight a limitation of the ready-to-use queries. This specific case demonstrates when a ready-to-use query template fails to precisely capture a user's requirements. This underscores that, without AQG4SD's adaptability, it becomes quite impossible to handle unanticipated user queries.

[5] https://github.com/yannramusat/AQG4SD
[6] https://drive.google.com/file/d/1-Vt-_SUwLfaMLz8aYh00ykIXQyk6YlA8/view?usp=sharing

6. Conclusion and Future Work

AQG4SD establishes significant relevance for automated cloud service discovery and selection by automating the translation of natural language user-requirements into actionable GraphQL queries and leveraging real-time public APIs; this directly addresses critical limitations of existing works, which often struggle with catalog's ongoing changes and semantic contextualization. Crucially, our work proves that a robust and efficient solution for cloud service discovery and selection can be achieved even when interfacing with public pricing APIs that are inherently not optimized for complex selection queries. AQG4SD effectively mitigates these API limitations by emulating advanced filtering and ordering functionalities within its own logic, notably leveraging multi-query orchestration and aggregation.

We plan to conduct further comprehensive experiments, including a thorough ablation study and an evaluation of our model's performance on established benchmarks and cloud offering datasets such as [9]. These future investigations will focus on incorporating and assessing advanced techniques such as automated validation and feedback loops for iterative refinement. Furthermore, we will investigate the implementation of semantic caching for GraphQL query results, aiming to significantly reduce latency and API load.

Finally, the overarching long-term goal of our research at Devoteam is to realize a robust solution for automating cloud architecture generation, directly translating informal business needs into deployable infrastructure. Consequently, AQG4SD will seamlessly be integrated into a comprehensive end-to-end solution for automated architecture generation, aiming to further optimize cloud service consumption through precise and optimal dimensioning.

Acknowledgments & Declaration on Generative AI

The authors would like to thank Gael Kamdem de Teyou and Lydia Arezki for their valuable inputs during the early conceptualization phase of the current project, particularly for pointing out the potential of structured query generation.

During the preparation of this work, the authors used Google Gemini in order to check grammar and spelling, as well as to paraphrase and reword certain sections. After using this service, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] P. M. Mell, T. Grance, SP 800-145. The NIST Definition of Cloud Computing, Technical Report, National Institute of Standards & Technology, 2011.

[2] M. Parhi, B. Pattanayak, M. Patra, An ontology-based cloud infrastructure service discovery and selection system, International Journal of Grid and Utility Computing 9 (2018) 108.

[3] M. Zhang, R. Ranjan, A. Haller, D. Georgakopoulos, M. Menzel, S. Nepal, An ontology-based system for cloud infrastructure services' discovery, in: CollaborateCom, 2012, pp. 524–530.

[4] M. Rekik, K. Boukadi, H. Ben-Abdallah, Cloud description ontology for service discovery and selection, in: ICSOFT, volume 1, 2015, pp. 1–11.

[5] H. Gabsi, R. Drira, H. H. B. Ghezala, Cloud services discovery assistant for business process development, in: Evaluation of Novel Approaches to Software Engineering, 2021, pp. 51–80.

[6] H. Taekgyeong, K. Sim, An ontology-enhanced cloud service discovery system, in: International MultiConference of Engineers and Computer Scientists (IMECS), volume 1, 2010.

[7] C. Shorten, C. Pierse, T. B. Smith, K. D'Oosterlinck, T. Celik, E. Cardenas, L. Monigatti, M. S. Hasan, E. Schmuhl, D. Williams, A. Kesiraju, B. van Luijt, Querying databases with function calling, 2025. arXiv:2502.00032.

[8] B. Ganesan, S. Ghosh, N. Gupta, M. Kesarwani, S. Mehta, R. Sindhgatta, Llm-powered graphql generator for data retrieval, in: IJCAI '24, 2024.

[9] H. Gabsi, R. Drira, H. Hajjami Ben Ghezale, Cloud services registry, 2019. URL: https://data.mendeley.com/datasets/7cy9zb9wtp/2.