Cloud-RA: A Reference Architecture for Cloud Based Information Systems

Jalal Kiswani, Sergiu M. Dascalu, and Frederick C. Harris, Jr
Department of Computer Science and Engineering, University of Nevada, Reno, 1664 N. Virginia Street, Reno, NV, USA
Jalal@nevada.unr.edu, {dascalus, fred.harris}@cse.unr.edu

Keywords: Software Reference Architecture, Cloud Computing, Cloud Applications, Software as a Service, Microservices Architecture

Abstract: Software architecture is an essential phase of the software development process, as it significantly increases the success rate of software projects and enables achieving their quality attributes and goals. However, implementing software architecture is not a straightforward process, and requires specialized expertise and knowledge -in both domain and technology- to achieve its requirements. To overcome this complexity, many tools have been developed to make the architecture process systemic, predictable and repeatable. These tools include architectural styles, architectural patterns, and reference architectures. In fact, these tools encourage sharing of experience and reducing the architecture process cost. In addition, tools such as reference architecture can make non-expert architects and developers start with ready-made architecture templates "as is," or with minimal customization. On the other hand, cloud computing is everywhere, and many applications are developed as cloud applications in what is called Software as a Service delivery model. In this paper, we propose Cloud-RA, a reference architecture for developing cloud-based multi-tenant information systems. In particular, it includes the problem, motivation, and proposed architecture. We hope this proposed work can be the bases for future cloud application reference architectures.

1 INTRODUCTION

Cloud Computing is one of the hottest trends in Information Technology nowadays (Armbrust et al., 2009). It changed how organizations from different domains perform their business (Economist, 2008). Cloud Computing service providers offer services in three different delivery models: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (Mell et al., 2011).

SaaS is based on developing software systems that can be used by different tenants (i.e., customers) at the same time. In fact, designing applications in a way that enables all customers to share the same instance of the deployed application is the preferred approach of developing cloud applications. This approach is preferred because it enables full utilization of cloud resources and more efficient scalability. The approach of having all customers using the same instance is called Single-Instance Multi-Tenant approach (SIMT) (Guo et al., 2007).

Designing and developing SIMT applications is more complicated than traditional monolithic applications for many reasons. In particular, multi-tenant support, billing, and monitoring are not easy to achieve. In addition, and with high-competition in the startups' ecosystem, the time to market and faster response to changes are required to achieve more profits and ensure more business sustainability. For these and many other reasons, a new architectural style was developed: Microservices Architecture (MsA) (Fowler and Lewis, 2018a).

MsA is based on having applications developed as separate loosely-coupled services (i.e., components) that are independently deployable, and able to communicate with other services over lightweight protocols such as HTTP. In addition, every service manages its own data storage; in fact, no direct access to other services data-store is allowed (Fowler and Lewis, 2018a). Also, MsA services are designed around business capabilities (Richardson, 2018). Furthermore, they can be developed in any technology (e.g., Java, Python, C++), and can be added and removed at any time without affecting the whole system functionality (Killalea, 2016).

Designing software applications -and more specifically cloud applications- based on MsA can exploit the full benefits of Cloud Computing of elasticity and

global software engineering (Beecham et al., 2014).

Even though MsA has many advantages, it is not a free lunch. Primary concerns are design and development complexity, long learning curve, and lack of expertise and tools (Richardson, 2018).

On the other hand, Information Systems (IS) is a particular category of software applications. IS enables controlled access to a broad base of shared information, such as library management systems and patient record systems (Sommerville, 2015). In such applications, the relational database is the most commonly used engine. However, due to the vast number of database tables in these applications, having user interface views for managing most of these tables is mandatory, which includes Create, Read, Update, and Delete (CRUD). Developing these functionalities increases the development cost and time, and introduces other risks. However, utilizing a metadata-driven approach for building such application can reduce these risks dramatically. In fact, it can enable higher quality and consistency (Kiswani et al., 2017).

Consequently, developing multi-tenant cloud-based information systems by utilizing MsA is challenging and requires specialized expertise. Thus, having an approach for enabling a more efficient way of designing and developing such applications may reduce the development time and cost, and be beneficial for both researchers and practitioners.

Many tools in the software architecture discipline were developed to make it predictable, systematic, and repeatable. Architectural Styles, Architectural Patterns, and Reference Architectures are examples of these tools (Len Bass, 2012).

Architectural styles and architectural patterns solve partial problems of software systems. Client-Server, Publish-Subscribe, and Layered Architecture are examples of architectural styles. While Model-View-Controller (MVC) and State-Logic-Display (i.e., Three-Tier) are examples of architectural patterns (Taylor et al., 2010).

In the other hand, in the reference architecture approach, a complete architecture is used for specific types of solutions and domains. An example is the Microsoft Industry Reference Architecture for Banking (Microsoft, 2012). Furthermore, lists of common reference architecture are documented and explained in literature (Humberto Cervantes, 2016).

For the preceding reasons, we propose Cloud-RA reference architecture. In particular, Cloud-RA is a template architecture that can be used to build cloud information systems based on MsA. Furthermore, it includes multi-tenant support, billing, and monitoring. Also, it includes cross-cutting services such as security, auditing, and logging. Moreover, it includes a metadata-driven approach that can reduce development time and increase software overall quality.

This paper is organized into 4 sections. This section covers the introduction. Then, it is followed by Section 2, which includes the background and related work. Section 3 includes the proposed work. Finally, Section 4 concludes the paper and identifies several directions of future work.

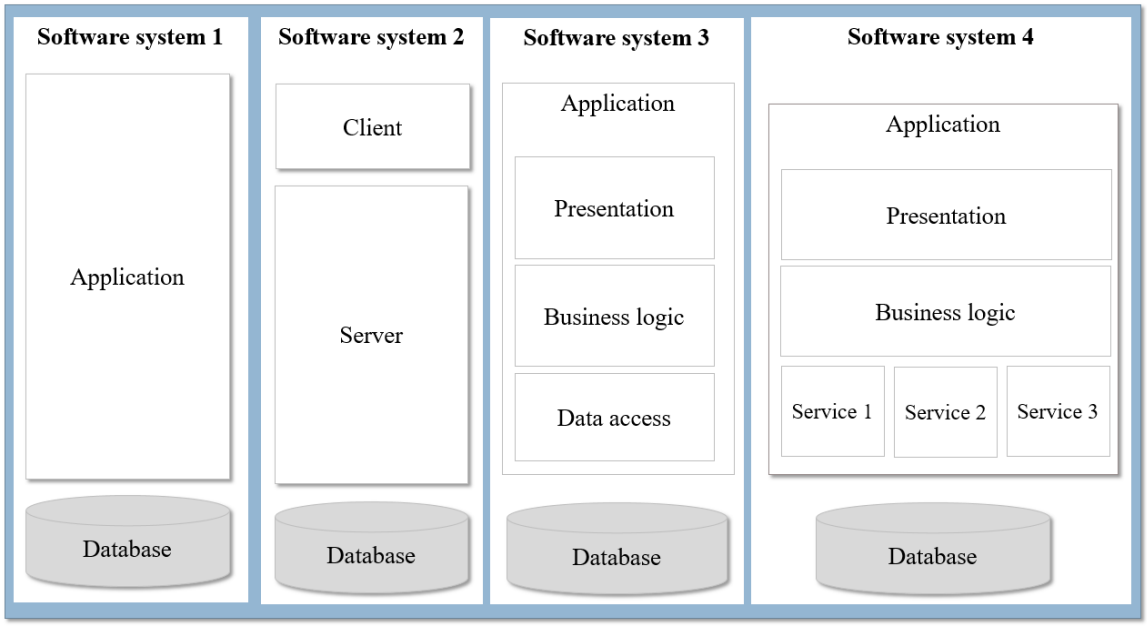

Figure 1: Examples of monolithic software systems

2 BACKGROUND

Organizations from all domains implement software applications with different scales for both internal and external use. Governments implement software systems to enable internal administration management with thousands of users. At the same time, they might implement an online software system to enable smart-government online services for citizens and residents. Another example is that of a scientific community may implement a group-scoped system for scientific Big Data management while enabling visualization features for external entities.

The monolithic approach of software development was the dominant model, in particular, building software applications as a single deployable unit (Fowler and Lewis, 2018a). In the past, this approach was practical since software size was relatively small and consisted of a low number of software components and functionality. In addition, this approach was common due to its convenience and ease of development. Moreover, the systems were only used by a limited number of users, with relatively long-term and stable requirements. Figure 1 shows some examples of monolithic applications.

However, things have changed, with the Internet, smartphones, Big Data, Cloud Computing, and the startups' ecosystem. Moreover, time to market and response to new requirements became more critical. Scalability has reached limits that were never possible and required before, with hundreds of millions of concurrent users accessing the same service at the same time. In addition, Internet of Things (IoT) also affected this wave, which increased the number of devices accessing online services exponentially (Economist, 2008).

With all these factors, the monolithic approach of software applications development may be a severe bottleneck. In fact, it may affect the organization's existence. The high-risks produced by adopting this approach includes a high cost of implementation and scalability, heavyweight testing and deployment processes, and complexity of development. Moreover, factors such as global software engineering (Ebert et al., 2016), and the need to use different technologists in the same project (e.g., Angular for front-end, Python for presentation tier, and Spring-Boot for backend) increased the challenge of selecting the monolithic approach, where the same technology is the main theme for most of the applications. Furthermore, various front-end technologies such as mobile devices, browsers, desktop applications, and IoT devices require lighter communication and full separation between front-end and back-end technologies (Laplante et al., 2008).

Even with the existence of techniques and concepts such as Object Oriented Programming (OOP), design patterns (Gamma, 1995), reusable components development, application frameworks, and software product lines (Linda M. Northrop, 2012), building the modern requirements of software systems is still challenging (Garlan, 2014).

Service Oriented Architecture (SOA) is an approach of software architecture and development driven by the academia and practitioners (Turner et al., 2003). In particular, it is a software constructions model that aims to change how the software systems are built. Its central idea is based on separating an application into smaller self-contained components, which are encapsulated, independently deployable, and communicate over a standard protocol. The common used protocol for SOA is Simple Object Access Protocol (SOAP) based web services. However, SOA did not get high traction because of its heavy-weight nature, in particular, communication, development and configuration complexity, tools and technologies. Moreover, the frequent use of SOA applications was for medium to large-scale information systems that require a relational database as the central persistence mechanism of application data. In fact, a unified instance of the database was used, which limits the flexibility, where developers were constraints in terms of taking design decisions, and created a bottleneck on architectural and database design decisions.

Consequently, a new architectural style has evolved: Microservices Architecture (MsA). In MsA, a software application is built as small, lightweight, reusable and self-deployable components that communicate over a lightweight protocol. In fact, these components can be developed using any technology. Moreover, every service has its business logic, storage engine, and persistent mechanism. Therefore, developers have full decisions control over the design, technology, and implementation of their services. The common used protocol in MsA is the Representational State Transfer (REST). REST is a lightweight version of SOAP web-services. In particular, it works as a layer that operates directly over the HTTP protocol (Fowler and Lewis, 2018a).

The microservices architecture overcomes many issues, and challenges over the monolithic and SOA approaches. However, it requires a particular early architecture and design, mainly for the services and their interfaces. Notably, these decisions may affect the overall success of the project. In fact, arguments about whether to start monolithic or microservices are increasing day after day (Tilkov, 2018) (Fowler and Lewis, 2018b).

Even though MsA can be implemented in traditional on-site applications, its full advantage can directly be gained if implemented in Cloud Applications. Cloud Applications are the applications deployed on Cloud Computing environments such as Infrastructure as a Service (IaaS) or Platform as a Service (PaaS) and are publicly accessible over the Internet by their intended users (Gorelik, 2013). As explained earlier, adopting MsA on the cloud is more efficient than Monolithic or SOA. In addition, implementing the quality attributes of scalability, high availability, auditability, monitorability, and traceability, will achieve the full benefits of Cloud Computing. In fact, including these quality attributes along with MsA in an application is termed as Cloud Native Applications (Balalaie et al., 2016).

3 PROPOSED REFERENCE ARCHITECTURE

This section presents Cloud-RA a proposed reference architecture for developing cloud-based multi-tenant information systems. In particular, it describes the required layers and their components, and the motivation for them.

3.1 Cloud-RA Layers

To achieve higher separation of concerns and modularity, the layered architectural style was used to design the higher level architecture of Cloud-RA. As shown in Figure 2, Cloud-RA consists of 5 layers:

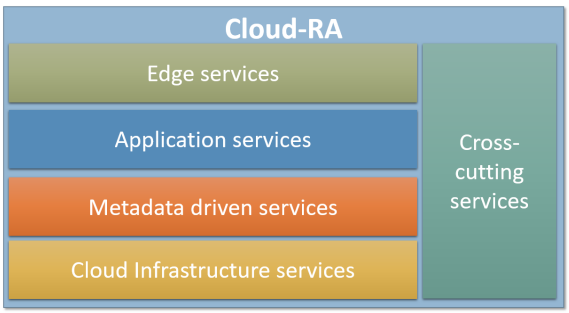

Figure 2: Layers of Cloud-RA

1. **Edge services:** Edge services are the front-line of cloud applications that are exposed to application clients. In fact, no access to the layers below is allowed from outside applications. In modern cloud software systems, client applications can be web browsers, rich client applications, smartphones or smart-device clients. Consequently, enabling universal devices access over a lightweight standard protocol is crucial.
2. **Application services:** This layer includes the application business logic services. These services include the domain logic and the functional requirements.
3. **Metadata services:** This layer contains the services required for dynamic generation of user interface and data access functionality (Kiswani et al., 2017).
4. **Cloud application infrastructure services:** This includes the services required to enable cloud applications features such as multi-tenant support, monitoring, and analytics.
5. **Cross-cutting services:** This layer includes the services that are required for all the other layers, such as configurations, security, and utilities.

3.2 Cloud-RA Microservices

As presented in the previous section, Cloud-RA consists of four horizontal layers and one cross-cutting layer. All the internal components of these layers are based on the Microservices Architecture. In particular, every service is self-deployable and possesses its own data storage. The microservices of Cloud-RA (Figure 3) are:

1. **API Gateway:** It is the core component of the Edge-Services and acts as the front-end proxy that accepts requests from external clients. Its main goal is to direct the requests to the appropriate service instances, where multiple instances of the same service could be projected to achieve the benefit of light-weight horizontal scalability of cloud-applications. In addition, it might include front-end security features.
2. **Multi-tenant:** This microservice handles the multi-tenant support of clients transparently. In particular, it is designed to enable transparent support for multi-tenant, with the illusion of having a separate dedicated deployment for every tenant.
3. **Billing:** Manages the usage and billing for each customer. Examples of billing criteria may be based on calculating the usage based on the number of transactions, the number of users, or data-size.
4. **Monitoring:** This microservice monitors the application activity for the cases of normal load, overload, and attacks. This service may be useful for evaluating the infrastructure needs, detection of security attacks, or identify software design issues and bottlenecks.

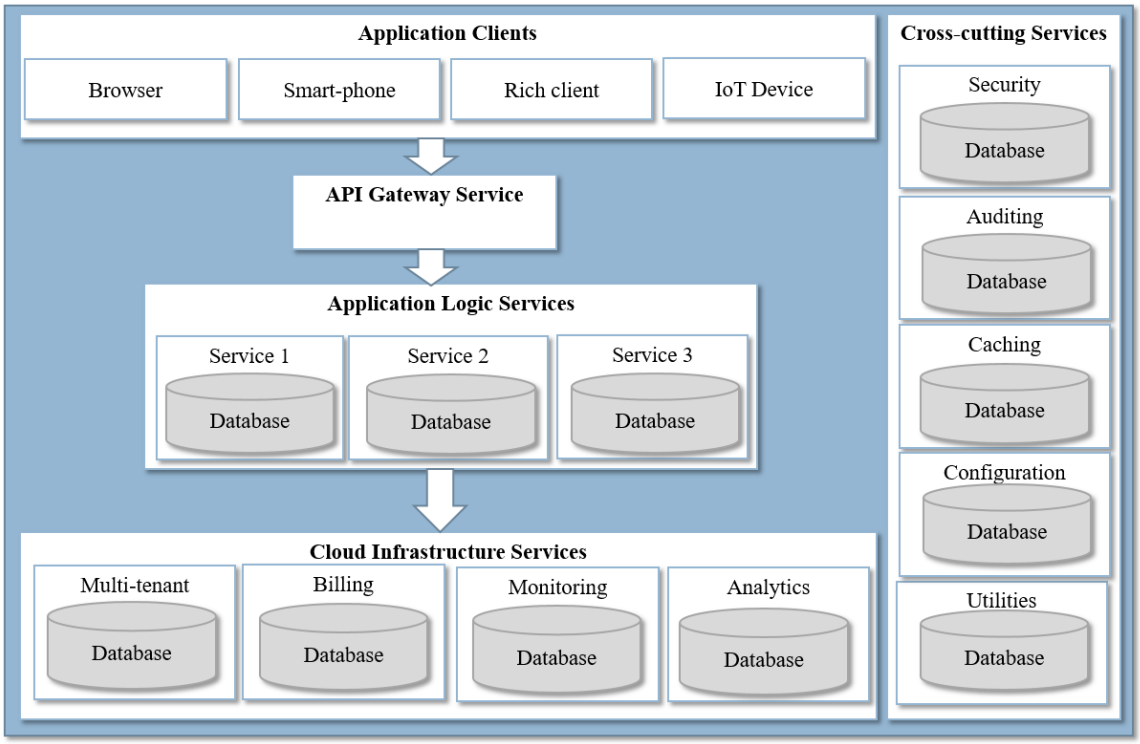

Figure 3: Microservices of Cloud-RA

5. **Analytics:** Analyses the application data dynamically. This can be used for dashboard-like functionality in the application. It may include an internal data warehouse that contains aggregated data collected from other services. This data may be then exposed for further analysis and visualization.
6. **Security:** This microservice manages the security aspects of the applications. It includes the authentication and authorization functionality. In addition, it may include the required encryption, decryption, and hashing implementations.
7. **Auditing:** Provides auditing support for business level transactions. In fact, having automated auditing features is significant in most applications (e.g., financial and governmental) for compliance and internal and external auditing.
8. **Caching:** Acts as a shared in-memory storage across all services for performance reasons. Caching for common required data that have a high percentage of access can dramatically increase system performance and scalability. Also, having a dedicated service for caching management makes it easier to enhance the software design to upgrade from single-host cache to distributed-caching technique.
9. **Configuration:** Provides common configuration for all Cloud-RA components.
10. **Utilities:** Common utilities that can be called from any service, such as getting server date, time, and their formats.

4 CONCLUSIONS

The work presented in this paper is a proposed reference architecture for cloud-based information systems. This could be the base architecture that could be modified according to the specific software project requirements.

However, since the microservices architectural style is relatively new, and the cloud computing trends are still evolving, there will be a need for new reference architectures to cover the new features and requirements.

In addition, reference architectures for other categories of cloud-based applications are essential. Examples of such categories are Big Data and IoT applications.

ACKNOWLEDGMENT

This material is based upon work supported by the National Science Foundation under grant number IIA-1301726. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R. H., Konwinski, A., Lee, G., Patterson, D. A., Rabkin, A., Stoica, I., et al. (2009). Above the clouds: A berkeley view of cloud computing. Technical report, Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Balalaie, A., Heydarnoori, A., and Jamshidi, P. (2016). Microservices architecture enables devops: Migration to a cloud-native architecture. *IEEE Software*, 33(3):42–52.

Beecham, S., O'Leary, P., Baker, S., Richardson, I., and Noll, J. (2014). Making software engineering research relevant. *Computer*, 47(4):80–83.

Ebert, C., Kuhrmann, M., and Prikladnicki, R. (2016). Global software engineering: Evolution and trends. In *Global Software Engineering (ICGSE), 2016 IEEE 11th International Conference on*, pages 144–153. IEEE.

Economist (2008). Let it rise: A special report on corporate IT. Special report, The Economist.

Fowler, M. and Lewis, J. (2018a). Microservices: a definition of this new architectural term. retreived jan 2018, from https://martinfowler.com/articles/microservices.html.

Fowler, M. and Lewis, J. (2018b). Microservices resource guide. retreived jan 2018, from https://martinfowler.com/microservices/.

Gamma, E. (1995). *Design patterns: elements of reusable object-oriented software*. Pearson Education India.

Garlan, D. (2014). Software architecture: a travelogue. In *Proceedings of the on Future of Software Engineering*, pages 29–39. ACM.

Gorelik, E. (2013). *Cloud computing models*. PhD thesis, Massachusetts Institute of Technology.

Guo, C. J., Sun, W., Huang, Y., Wang, Z. H., and Gao, B. (2007). mework for native multi-tenancy application development and management. In *e-commerce Technology and the 4th IEEE International Conference on Enterprise Computing, e-commerce, and E-Services, 2007. CEC/EEE 2007. The 9th IEEE International Conference on*, pages 551–558. IEEE.

Humberto Cervantes, . R. K. (2016). *Designing Software Architectures: A Practical Approach (SEI Series in Software Engineering)*. Addison-Wesley Professional; 1st edition.

Killalea, T. (2016). The hidden dividends of microservices. *Communications of the ACM*, 59(8):42–45.

Kiswani, J., Muhanna, M., and Qusef, A. (2017). Using metadata in optimizing the design and development of enterprise information systems. In *Information and Communication Systems (ICICS), 2017 8th International Conference on*, pages 188–193. IEEE.

Laplante, P. A., Zhang, J., and Voas, J. (2008). What's in a Name? Distinguishing between SaaS and SOA. *It Professional*, 10(3).

Len Bass, Paul Clements, . R. K. (2012). *Software Architecture in Practice (3rd Edition)*. Addison-Wesley Professional.

Linda M. Northrop, Paul C. Clements Contributor Reed Little, J. M. L. O. F. B. J. K. B. G. C. S. G. C. P. D. L. G. J. R. W. K. J. (2012). A Framework for Software Product Line Practice, Version 5.0. Software Engineering Institute, Carnegie Mellon University.

Mell, P., Grance, T., et al. (2011). The NIST definition of cloud computing.

Microsoft (2012). Microsoft Industry Reference Architecture for Banking (MIRA-B).

Richardson, C. (2018). Microservices are not a silver bullet. retreived jan 2018, from https://microservices.io.

Sommerville (2015). *Software Engineering (10th Edition)*. AddisonWesley.

Taylor, R. N., Medvidovic, N., and Dashofy, E. M. (2010). *Software Architecture: Foundations, Theory, and Practice*. Wiley.

Tilkov, S. (2018). Dont start with a monolith. retreived jan 2018, from https://martinfowler.com/articles/dont-start-monolith.html.

Turner, M., Budgen, D., and Brereton, P. (2003). Turning software into a service. *Computer*, 36(10):38–44.