Institute of Architecture of Application Systems

Non-Functional Data Layer Patterns for Cloud Applications

Steve Strauch, Vasilios Andrikopoulos, Uwe Breitenbücher,
Oliver Kopp, Frank Leymann
Institute of Architecture of Application Systems,
University of Stuttgart, Germany
{firstname.lastname}@iaas.uni-stuttgart.de

BibTeX:

```
@inproceedings{StrauchABKL2012,
  author    = {Steve Strauch, Vasilios Andrikopoulos, Uwe Breitenbuecher,
               Oliver Kopp, and Frank Leymann},
  title     = {Non-Functional Data Layer Patterns for Cloud Applications},
  booktitle = {Proceedings of the 4th IEEE International Conference on Cloud
               Computing Technology and Science, CloudCom 2012,
               3-6 December 2012, Taipei, Taiwan},
  year      = {2012},
  pages     = {601-605},
  publisher = {IEEE Computer Society}
}
```


Universität Stuttgart
Germany

2012 IEEE 4th International Conference on Cloud Computing Technology and Science

Non-Functional Data Layer Patterns for Cloud Applications

Steve Strauch, Vasilios Andrikopoulos, Uwe Breitenbuecher, Oliver Kopp, Frank Leymann
Institute of Architecture of Application Systems (IAAS),
University of Stuttgart, Stuttgart, Germany
{*firstname.lastname*}*@iaas.uni-stuttgart.de*

***Abstract*—Cloud services allow for hosting applications partially or completely in the Cloud by migrating their components and data. Especially with respect to data migration, a series of functional and non-functional challenges like data confidentiality arise when considering private and public Cloud data stores. In this paper we identify some of these challenges and propose a set of reusable solutions focusing on the non-functional aspects, organized together as a set of Cloud Data Patterns.**

***Keywords*-Data layer; Cloud applications; Data migration; Cloud Data Patterns; Cloud data stores**

I. Introduction

Cloud computing has become increasingly popular with the industry due to the clear advantage of reducing capital expenditure and transforming it into operational costs [1]. To take advantage of Cloud computing, an existing application may be moved to the Cloud or designed from the beginning to use Cloud technologies. Applications are typically built using a three layer architecture model consisting of a presentation layer, a business logic layer, and a data layer [2]. The presentation layer describes the application-users interactions, the business layer realizes the business logic and the data layer is responsible for application data storage. The data layer is in turn subdivided into the Data Access Layer (DAL) and the Database Layer (DBL). The DAL encapsulates the data access functionality, while the DBL is responsible for data persistence and data manipulation. Figure 1 visualizes the positioning of the various layers.

Each application layer can be hosted using different Cloud deployment models. Possible Cloud deployment models, also shown in Figure 1, are: Private, Public, Community, and Hybrid Cloud [3]. Figure 1 shows the various possibilities for distributing an application using the different Cloud types. The "traditional" application not using any Cloud technology is shown on the left of the figure. In this context, "on-premise" denotes that the Cloud infrastructure is hosted inside the company and "off-premise" denotes that it is hosted outside of the company.

In this work we focus on the lower two layers of Figure 1, the DAL and DBL layers of the application. Application data is typically moved to the Cloud because of e. g., Cloud bursting, data analysis or backup and archiving. Using Cloud technology leads to challenges such as incompatibilities with the database layer previously used or the accidental disclosing of critical data by e. g., moving them to a Public Cloud. Incompatibilities in the DBL may refer to inconsistencies between the functionality of an existing traditional DBL, and the functionality and characteristics of an equivalent DBL provided in the Cloud. For instance, the Google App Engine Datastore [4] is incompatible with Oracle Corporation MySQL, version 5.1 [5], because the Google Query Language [6] supports only a subset of the functionality provided by SQL, e. g., joins are not supported. An application relying on such functionalities cannot therefore have its data moved to the Cloud without deep changes to its implementation. It has to be noted here that, for the purposes of this work, we assume that the decision to migrate the data layer to the Cloud has already been made based on criteria such as cost, effort etc. [7], [8].

The contribution of this paper is the identification of such challenges and the description of a set of *Cloud Data Patterns* as the best practices to deal with them. As defined in [9], a Cloud Data Pattern describes a *reusable and implementation technology-independent solution for a challenge related to the data layer of an application in the Cloud for a specific context*. For this purpose, in the following we present an initial catalog of Cloud Data Patterns dealing with non-functional challenges of having the application data layer realized in the Cloud. We focus on two aspects: enabling data store scalability and ensuring data confidentiality. The presented list of patterns is a result of our collaboration with industry partners and research projects. We do not claim that the list of patterns is complete and we plan to expand it in the future.

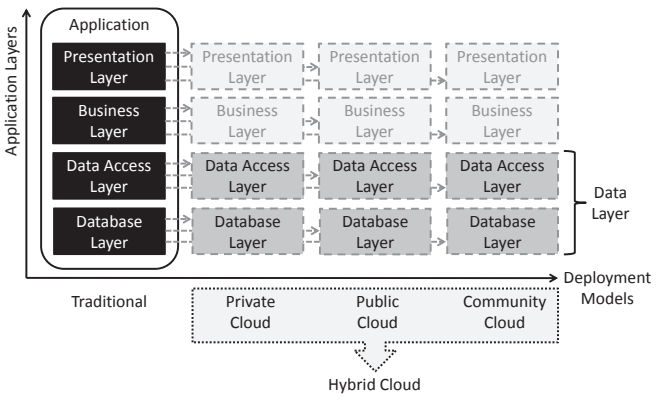

Figure 1. Overview of Cloud Deployment Models and Application Layers

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

The presentation of the patterns uses the format defined by Hohpe and Woolf [10], consisting of the description of a *context* where the pattern is applicable; the *challenge* posed; external or internal *forces* that impose constraints that make the problem difficult to solve; a proposed *solution* for the challenge; detailed technical issues (as *sidebars*); the *results* of applying the proposed solution in the defined context; an *example* of use and other patterns to be considered (*next*). A representative *icon* and a graphical *sketch* of the pattern are also provided.

The remainder of this paper presents our proposed set of non-functional Cloud Data Patterns as a set of best practices. Section II focuses on patterns for addressing challenges related to the elasticity of data stores. Section III summarizes a set of patterns we defined in [9] for the purpose of preserving data confidentiality. A presentation of related work is contained in Section IV; conclusions and future work in Section V.

II. Scalability Patterns

These patterns focus on providing solutions for ensuring an acceptable Quality of Service (QoS) level by means of scalability in case of increasing data read or data write load. When considering the data, rather than the database system, there are two scaling options available: vertical and horizontal data scaling. Vertical data scaling can be achieved by moving the data to a more powerful database system, which offers better performance, advanced functionalities, or both. Horizontal data scaling is based on partitioning the data according to functional groups [11]. Examples for functional groups are European customers and American customers. Each functional group may be itself distributed among different database systems to increase search speed. This method is also called *sharding* [12]. In this section we present two patterns, expanding the work of Brian Adler [13]. More specifically:

A. Local Database Proxy

The *Local Database Proxy* enables read scalability by requiring a master/multiple slave model and forwarding read requests to any read replica.

Context: A Cloud data store does not inherently support horizontal scalability for data reads. When the data read load of the application permanently increases, e. g., due to increased user acceptance and usage, a mechanism for horizontally scaling read requests is required. Additionally, the business logic of processing user requests is also moved to the Cloud.

Challenge: How can a Cloud data store not supporting horizontal data read scalability provide that functionality?

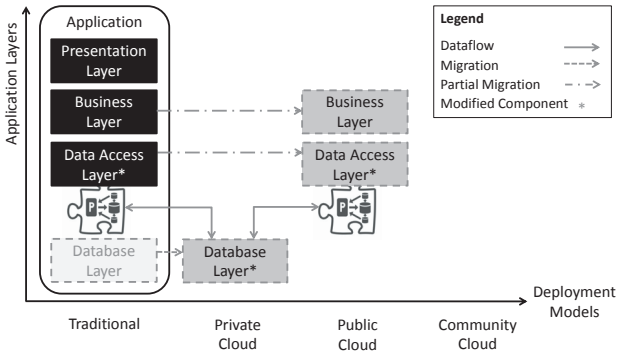

Figure 2. Sketch of Local Database Proxy

Forces: In case there was one centralized database proxy, which might suffer an outage, there would be no possibility for the business layer to access the database layer. This might be overcome by using at least two centralized database proxies to avoid the single point of failure vulnerability. The use of local database proxies ensures that only some application servers in the business layer will be affected by an outage of the local proxy.

Solution: The Cloud data store is configured using a single master/multiple slave model, denoted by "Database Layer*" in Figure 2. "Database Layer*" represents both the master and the multiple slaves. The master handles data writes and the slaves are used as replicas serving read requests only. In case the application has to deal with stale data, the replication of data may be lazy. A proxy component is locally added below each data access layer. All requests from each data layer are routed through the respective proxy. The proxy routes data read requests to any slave and write requests to the master.

Sidebars: The data access layer has to be configured to use the respective local database proxy. An appropriate replication strategy has to be applied at the Cloud data store to ensure a proper balance between consistency and availability [14]. This technique is not recommended for applications with high update frequency where different parts of the application read and write data concurrently.

Results: The read load of the Cloud data store is distributed among several slave data stores used as read replicas. When the read load increases, additional slaves may be added. The master is still a potential bottleneck. This can be overcome by using a watchdog solution for the master. In case of an outage of the master is detected, one of the slave databases can become the new master.

Example: A public transport company offering bus routes hosts its data store infrastructure in a Private Cloud using MySQL on Eucalyptus Machine Images (EMI) using Eucalyptus [15]. The front-end application is reading the available routes and the back-end application is updating the route information with live info. The public Cloud hosts

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

the business layer, while the bus route management is still hosted on-premise. The application is rewritten to use the local database proxy to access the data. The data is moved to the private Cloud. One write master and several read slaves are installed to enable coping with high read loads.

Next: In case both data read and writes should scale the "Local Sharding-Based Router" pattern has to be considered.

B. Local Sharding-Based Router

The *Local Sharding-Based Router* enables read and write scalability by requiring the independent splitting and distributing of data into functional groups and forwarding read and write requests to the corresponding shard.

Context: A Cloud data store does not inherently support horizontal scalability for data reads and writes. When the data read load of the application permanently increases, e. g., due to increased user usage, a mechanism for horizontally scaling read requests to the data store is required. Furthermore, a permanent high data update rate of the application requires also horizontally scaling of data writes. The business logic of processing user requests is moved to the Cloud.

Challenge: How can a Cloud data store not supporting horizontal data read and write scalability provide that functionality?

Forces: Considering the local database proxy pattern, the first solution to choose might be a configuration of several slave databases used as read replicas together with several master databases used as write replicas. As each data write to any master has to be replicated to all other master databases, leading to additional data writes, this approach is not efficient for horizontal scaling data writes. In case there was one centralized sharding-based router, which might suffer an outage, there would be no possibility for the business layer to access the database layer. This might be overcome by using at least two centralized sharding-based routers to avoid single point of failure. If the additional sharding-based routers are configured as fail-over, the capacity of the active sharding-based router might be too small. If the additional sharding-based routers are configured as independent units, each of them is still a single point of failure. The usage of a local sharding-based router ensures that only some application servers in the business layer will be affected by an outage.

Solution: The data to be stored in the Cloud are split horizontally. This means that tables with many rows are split into several data stores. Each data store is assigned a distinct number of rows of the original table. This technique is called "sharding" [12]. A local sharding-based router is added locally below each data access layer (Figure 3). All requests from each data layer are routed through the respective sharding-based router. The local sharding-based router forwards data read and write requests to the appropriate Cloud data store.

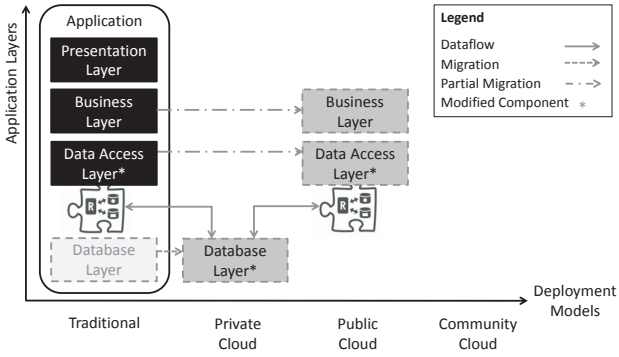

Figure 3. Sketch of Local Sharding-Based Router

Sidebars: The data to be split have to be independent from each other. A business logic requiring many joins involving multiple tables of different stores leads to an increased processing time. The pattern is not applicable in case the data cannot be split independently.

Results: The data are categorized and independently split among multiple Cloud data stores. Thus, horizontal read and write scalability is enabled.

Example: An online store hosts its data store infrastructure in a Private Cloud using MySQL on Eucalyptus Machine Images (EMIs) using Eucalyptus. The database contains the available products and the number of items in stock. In case an item is bought, the number of items in stock is reduced. New items put in the stock increase this number. As the online store offers many products of different categories, the product data is split using the respective category to multiple MySQL databases. The front-end application is hosted in the public Cloud. The data requests are routed through a local sharding-based router, which selects the appropriate data source. The back-end application is still hosted traditionally and is modified to use a local sharding-based router.

Next: In case only data reads have to be distributed in order to scale, the "Local Database Proxy" pattern has to be considered.

III. Confidentiality Patterns

In our previous work [9], we presented Cloud Data Patterns for confidentiality. They deal with data to be kept private and secure, commonly referred to as "critical data". In the following, we summarize these patterns.

Critical data can be categorized into different confidentiality levels. As data may be categorized using different categorizations (or not categorized at all), the confidentiality level has to be harmonized. The *Confidentiality Level Data Aggregator* provides one confidentiality level for data from different sources with potentially different confidential categorizations on different scales.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

The *Confidentiality Level Data Splitter* splits data according to predefined privacy levels. This is required when an application writes data to multiple data stores with different confidentiality levels.

The *Filter of Critical Data* ensures that no confidential data are disclosed to the public. The filter enforces that no data leave the private Cloud by filtering out critical data.

The *Pseudonymizer of Critical Data* implements pseudonymization. Pseudonymization is a technique to provide a masked version of the data to the public while keeping the relation to the non-masked data in private [16]. This enables processing of non-masked data in the private environment when required.

The *Anonymizer of Critical Data* implements anonymization [16]. Anonymization is a technique to provide a reduced version of the critical data to the public while ensuring that it is impossible to relate the reduced version to the critical data.

IV. Related Work

Pattern languages defining reusable solutions for recurring challenges in architecture have been first proposed by Christopher Alexander [17]. A series of well-established patterns have been previously identified concerning, e. g., software engineering [18], enterprise integration [10] and application architecture [19]. Such general works do not consider building or migrating the database layer in the Cloud. Nevertheless, we reuse the pattern format defined by Hohpe and Woolf [10] for describing our Cloud Data Patterns.

Petcu [20] proposes Cloud usage patterns for Cloud-based applications based on existing use cases. Fehling et al. [21] and Pallmann [22] provide high-level architectural patterns to design, build, and manage applications using Cloud services. None of these works discusses patterns for building and/or moving the data layer to the Cloud. Adler [13] provides contributions regarding best practices for scalable applications in the Cloud. In this paper we reuse some of the results presented in [13] to form the non-functional patterns presented in Section II.

ARISTA Networks, Inc. [23] provides seven patterns for Cloud computing of which only one (the Cloud Storage pattern) deals with data in the Cloud. Nock [24] provides patterns for data access in enterprise applications, without however treating Cloud data stores in the same manner as we do.

Schumacher et al. [25] present reusable solutions for securing applications, but do not deal with data pseudonymization, data anonymization, and data filtering. Hafiz [26] presents a privacy design pattern catalog consisting of nine patterns achieving anonymity by mixing data with data from other sources instead of providing a general pseudonymization, anonymization, or filtering pattern. Creese et al. [27] consider design patterns for data protection of Cloud services. Romanosky et al. [28] describe privacy patterns applicable for online interactions. Schumacher [29] introduces an approach for mining security patterns from security standards and presents two patterns for anonymity and privacy. These works do not consider building a data layer in the Cloud or migrating an existing one there; some of the mechanisms identified however (e. g., pseudonymization) are reused in the Cloud Data Patterns we propose.

Finally, Schuemmer [30] presents patterns filtering personal information to establish boundaries for interactions between users utilizing collaborative systems. Our patterns are more general in the sense that they are not limited to filtering of personal data.

V. Conclusions and Future Work

This work presented a set of reusable solutions to face the challenges of moving the data layer to the Cloud or designing an application using a data store in the Cloud, focusing on the non-functional aspect. The challenges and proposed solutions were organized as a non-exhaustive catalog of Cloud Data Patterns that was the result of our collaboration with industry partners and research projects. Patterns for elasticity and confidentiality issues in particular were discussed.

The presentation of the patterns did not go into technical details. For instance, scalability and single points of failure in the realization of the patterns has not been treated appropriately. A possible scalability mechanism and a countermeasure to single point of failure is to implement each pattern using a hot-pool of pattern realizations in the Cloud. A hot-pool consists of multiple instances of the realization component and a watchdog. For example, implementing the Local Sharding-Based Router (Section II) as a single component may result in a bottleneck, or even to a complete failure of the data access/database layer connection.

In addition, Cloud data stores can be considered as appliances where a fixed set of functionality is provided [13]. Each data store is geared towards a specific application domain, and therefore does not come with all possible features. Furthermore, the offered functionalities may be configurable but not extensible. For this purpose, in our current work we also discuss *functional* patterns. Such issues and their possible solutions will be investigated as part of a larger scale evaluation of our patterns using an industrial case study. Toward this direction, we also plan to formalize a general composition method of Cloud Data Patterns and

978-1-4673-4510-1/12/$31.00 ©2012 IEEE

expand our catalog with identified patterns presented here with more patterns.

ACKNOWLEDGMENTS

The research leading to these results has partially received funding from the 4CaaSt project part of the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 258862 and the BMWi-project Cloud-Cycle (01MD11023). We thank Tobias Unger for his valuable input.

REFERENCES

[1] M. Armbrust *et al.*, "Above the Clouds: A Berkeley View of Cloud Computing," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2009-28, 2009.

[2] J. Dunkel *et al.*, *Systemarchitekturen fuer Verteilte Anwendungen: Client-Server, Multi-Tier, SOA, Event Driven Architectures, P2P, Grid, Web 2.0*. Hanser Verlag, 2008.

[3] P. Mell and T. Grance, "Cloud Computing Definition," *National Institute of Standards and Technology*, 2009.

[4] Google, Inc., "Google App Engine Datastore," 2011. [Online]. Available: https://developers.google.com/appengine/docs/python/datastore/

[5] Oracle Corporation, "MySQL," 2011, http://www.mysql.com.

[6] Google, Inc., "Google App Engine GQL Reference," 2011, https://code.google.com/intl/en/appengine/docs/python/datastore/gqlreference.html.

[7] B. C. Tak, B. Urgaonkar, and A. Sivasubramaniam, "To move or not to move: the economics of cloud computing," in *Proceedings of HotCloud'11*. Berkeley, CA, USA: USENIX Association, 2011.

[8] M. Menzel and R. Ranjan, "Cloudgenius: decision support for web server cloud migration," in *Proceedings of WWW '12*. New York, NY, USA: ACM, 2012.

[9] S. Strauch, U. Breitenbücher, O. Kopp, F. Leymann, and T. Unger, "Cloud Data Patterns for Confidentiality," in *Proceedings of CLOSER'12*. SciTePress, 2012.

[10] G. Hohpe and B. Woolf, *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2003.

[11] K. Küspert and J. Nowitzky, "Partitionierung von Datenbanktabellen," *Informatik-Spektrum*, vol. 22, pp. 146–147, 1999.

[12] J. Zawodny and D. Balling, *High Performance MySQL: Optimization, Backups, Replication, Load-balancing, and More*. O'Reilly & Associates, Inc. Sebastopol, CA, USA, 2004.

[13] B. Adler, "Building Scalable Applications In the Cloud: Reference Architecture & Best Practices, RightScale Inc." 2011, http://www.rightscale.com/info_center/white-papers/building-scalable-applications-in-the-cloud.php.

[14] D. Pritchett, "BASE: An ACID Alternative," *Queue*, vol. 6, no. 3, pp. 48–55, 2008.

[15] Eucalyptus Systems, Inc., "Eucalyptus," 2011. [Online]. Available: http://www.eucalyptus.com

[16] Federal Ministry of Justice, "German Federal Data Protection Law," 1990. [Online]. Available: http://www.gesetze-im-internet.de/bdsg_1990/

[17] C. Alexander *et al.*, *A Pattern Language. Towns, Buildings, Construction*. Oxford University Press, 1977.

[18] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman, 1994.

[19] M. Fowler *et al.*, *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional, November 2002.

[20] D. Petcu, "Identifying Cloud Computing Usage Patterns," in *Proceedings of IEEE CLUSTER WORKSHOPS'10*. IEEE, October 2010.

[21] C. Fehling *et al.*, "An Architectural Pattern Language of Cloud-based Applications," in *Proceedings of PLoP'11*. ACM, 2011.

[22] D. Pallmann, "Windows Azure Design Patterns," 2011. [Online]. Available: http://neudesic.blob.core.windows.net/azuredesignpatterns/index.html

[23] ARISTA Networks, Inc., "Cloud Networking: Design Patterns for Cloud-Centric Application Environments," 2009, http://www.aristanetworks.com/smt/user/upload/File/downloads/CloudCentricDesignPatterns.pdf.

[24] C. Nock, *Data Access Patterns: Database Interactions in Object Oriented Applications*. Prentice Hall Professional Technical Reference, 2008.

[25] M. Schumacher *et al.*, *Security Patterns: Integrating Security and Systems Engineering*. Wiley, 2006.

[26] M. Hafiz, "A Collection of Privacy Design Patterns," in *Proceedings of PLoP'06*, New York, NY, USA, 2006.

[27] S. Creese *et al.*, "Data Protection-Aware Design for Cloud Services," in *Proceedings of CloudCom'09*, 2009.

[28] S. Romanosky *et al.*, "Privacy Patterns for Online Interactions," in *Proceedings of PLoP'06*. ACM, 2006.

[29] M. Schumacher, "Security Patterns and Security Standards," in *Proceedings of EuroPLoP'02*, 2002.

[30] T. Schuemmer, "The Public Privacy–Patterns for Filtering Personal Information in Collaborative Systems," in *Proceedings of CHI'04*, 2004.

All links were last followed on October 5, 2012.

978-1-4673-4510-1/12/$31.00 ©2012 IEEE