Migrating Application Data to the Cloud using Cloud Data Patterns

Steve Strauch, Vasilios Andrikopoulos, Thomas Bachmann and Frank Leymann
Institute of Architecture of Application Systems, University of Stuttgart, Stuttgart, Germany

Keywords: Application Data Migration, Cloud Data Patterns, Cloud Migration Scenarios, Application Refactoring.

Abstract: Taking advantage of the capabilities offered by Cloud computing requires either an application to be built specifically for it, or for existing applications to be migrated to it. In this work we focus on the latter case, and in particular on migrating the application data. Migrating data to the Cloud creates a series of technical, architectural and legal challenges that the State of the Art attempts to address. We organize such efforts into a set of migration scenarios and connect them with a list of reusable solutions for the application data migration in the form of patterns. From there we define an application data migration methodology and we demonstrate how it can be used in practice.

1 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 (Armbrust et al., 2009). To take advantage of Cloud computing, an existing application may be moved to the Cloud (Cloud-enabling it) or designed from the beginning to use Cloud technologies (Cloud-native application). Applications are typically built using a three layer architecture pattern consisting of a presentation layer, a business logic layer, and a data layer (Fowler et al., 2002). 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 (Mell and Grance, 2009). 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 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. The migration of the Data Layer to the Cloud includes two main steps to be considered: migration of the DBL to the Cloud, and adaptation of the DAL to enable Cloud data access. This separation of concerns allows for taking into consideration existing applications for migration that could potentially keep their business logic (partially) on-premises and use more than one Cloud data store provider at the same time. It has also 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. (Tak

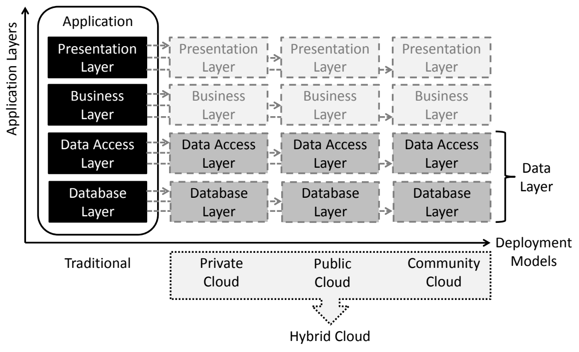

Figure 1: Overview of Cloud Deployment Models and Application Layers.

Strauch S., Andrikopoulos V., Bachmann T. and Leymann F..
Migrating Application Data to the Cloud using Cloud Data Patterns.
DOI: 10.5220/0004376300360046
In *Proceedings of the 3rd International Conference on Cloud Computing and Services Science* (CLOSER-2013), pages 36-46
ISBN: 978-989-8565-52-5
Copyright © 2013 SCITEPRESS (Science and Technology Publications, Lda.)

et al., 2011; Menzel and Ranjan, 2012; Andrikopoulos et al., 2012).

As previously discussed (Strauch et al., 2012b), using Cloud technology leads to challenges such as incompatibilities with the database layer previously used, or even accidental disclosing of critical data by e. g., moving them to a Public Cloud. For this purpose, in (Strauch et al., 2012b), a set of *Cloud Data Patterns* is identified addressing these challenges using the format defined by Hohpe and Woolf (Hohpe and Woolf, 2003). 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.

The contribution of this work is focused on:

1. analyzing the State of the Art in migrating applications, and in particular application data to the Cloud, and organizing these efforts into a set distinct migration scenarios with particular characteristics,
2. mapping the identified Cloud Data Patterns with the migration scenarios as best practices in order to deal with each of the scenarios,
3. providing an application data migration methodology based on the scenario/pattern mapping, and demonstrating its applicability by means of an illustrative case.

The remainder of this paper is organized as follows: Section 2 discusses a motivating scenario that will be used throughout the paper. Section 3 presents the existing work on migrating the application and application data and Section 4 organizes it into migration scenarios. Section 5 summarizes the Cloud Data Patterns that were identified in (Strauch et al., 2012b) and (Strauch et al., 2012a) and highlights their key points. Section 6 then discusses the mapping between scenarios and patterns, and the application data migration methodology that is based on this mapping. The motivating scenario from Section 2 is refactored in order to demonstrate our proposal in practice. Finally, Section 7 concludes the paper and discusses future work.

2 MOTIVATING SCENARIO

For purposes of an illustrative example let us consider the case of a Health Insurance Company (HIC) in Germany. As a result of the increase of the numbers of its clients, the company stores their data in two data centers in different parts of Germany. The data centers, covering geographical regions A and B, respectively, form a Private Cloud data hosting solution. This Private Cloud acts as a uniform access point and offers a unified view of the data to the various applications used by the employees of the company. The company is required to provide access to an External Auditing Company (EAC) to audit the financial transactions processed by the company. EAC executes a series of predefined complex queries on the financial transactions data at irregular intervals and reports back to HIC and the responsible authorities their findings. HIC however is also obliged by law to protect the personal data privacy and confidentiality of the medical record of its clients. For this purpose, the company takes special care to anonymize the results of the queries executed by the auditor in order to ensure that no client information is accidentally exposed.

Providing EAC with direct access to the database of HIC raises a series of concerns about ensuring the security of the company-internal data, and the performance of the company systems, as an indirect result of the unpredictable additional load imposed by the complex queries executed by the auditor. As a solution to these issues, it is proposed to use a Public Cloud data hosting solution provider and migrate a consistent replica of the financial transaction records to the Public Cloud, stripped (to the extent that it is possible) of any personal data. The auditing company would then be able to retrieve the necessary information without burdening the company systems. Such a migration to the Cloud however, even if only partial, requires addressing different kinds of challenges: confidentiality-related (ensuring that it is impossible to recreate the medical records and other personal information of the company clients using the data in the Public Cloud), functionality-related (providing both all the necessary data and the querying mechanisms for EAC to operate as required), and non-functional (ensuring that the partial migration does not encumber in any way the performance of HIC's systems). In the following we discuss how Cloud Data Migration Scenarios and Patterns can be used in conjunction to address such issues.

3 BACKGROUND

Data migration can either be seen as part of the migration of the whole application, or as the migration of only the Data Layer. For this purpose, in the following we investigate not only the State of the Art on use cases for application migration to the Cloud, including which types of applications are suitable for migration to the Cloud, but also use cases for database layer migration only. Some of the cases include also migration of application from the Cloud back to a traditional on-premises model. Based on this analysis in Section 4 we derive specific Cloud Data Migration Scenarios.

Application Migration. Amazon proposes a phase-driven approach for Cloud application migration, which includes one phase focusing on data migration (Varia, 2010). The data migration is done in two steps: selection of the Amazon AWS service, and migration of the data. Furthermore, Amazon provides recommendations regarding which of their data and storage services best fit for storing a specific type of data, e.g., Amazon Relational Database Service (Amazon RDS)[1]. Apart from that, Amazon describes three application migration use cases (Varia, 2010):

- Marketing and collaboration Web sites.
- Digital asset management solution using batch processing pipelines.
- Claims processing systems using back-end processing workflows.

Microsoft identifies the following eight types of applications to be considered for migration to the Cloud (Microsoft Azure, 2012):

1. SaaS applications
2. Highly-scalable Web sites
3. Enterprise applications
4. Business intelligence and data warehouse applications
5. Social or customer-oriented applications
6. Social (online) games
7. Mobile applications
8. High performance or parallel computing applications

Microsoft also provides a step-by-step migration guideline for migrating local MS SQL Server databases to Azure SQL Database Service mainly consisting of two phases: database schema migration supported by a wizard, and migration of the data itself based on command-line tools (Lee et al., 2009).

Furthermore, Cunningham specifies the following four general scenarios when an application is suitable for migration to the Cloud (Cunningham, 2010):

1. The application is used only in predefined periods.
2. The rapid increase in the need for resources cannot be compensated by buying new hardware.
3. The application load can be anticipated, e.g., for seasonal businesses, allowing to optimize resource utilization.
4. In case of unanticipated load, the load increases without prior indication.

Laszewski et al. (Laszewski and Nauduri, 2011) identify five scenarios for migration of a legacy application to the Cloud: *rewrite/re-architect* the application, *changing the platform*, automated *conversion* using suitable tools, *emulation*, e.g., through Service-Oriented Architecture enablement and Web services, and *data modernization*. Data modernization entails the migration of the database to the Cloud and it is therefore directly relevant for this work. Finally, Armbrust et al. identify the following types of applications as drivers for Cloud computing (Armbrust et al., 2009):

- Mobile interactive applications
- Parallel batch processing
- Business analytics as a special case of batch processing
- Extension of computationally-intensive desktop applications

Data Migration. With respect to the migration of the database layer in particular, Microsoft identifies the following migration scenarios where only the database layer is migrated to the Cloud and the other layers are kept hosted traditionally (Lee et al., 2009):

1. Web applications.
2. Applications only used by departments or smaller working groups within a company.
3. Data hubs, where the data is mirrored, e.g., on the laptops of employees, and regularly synchronized with the data store in the Cloud.

Especially for the first two scenarios, the performance issues and complexity due to the potential latency challenges after the migration of the database layer to the Cloud have to be considered.

Informatica[2] is an SaaS provider for secure data migration, data replication and data synchronization into the Cloud. The user configures the synchronization and migration service via a Web interface and the data migration is done using locally installed secure agents and encryption.

Different solutions and approaches can therefore be used for migrating application data to the Cloud. In the following section we organize them into distinct migration scenarios with particular characteristics.

[1]http://aws.amazon.com/rds/

[2]http://www.informatica.com

Table 1: Cloud Data Migration Scenarios Overview.

Migration Scenario	**Migration Direction (Traditional or Cloud)**
Database Layer Outsourcing	Traditional $\longrightarrow$ Cloud
Using Highly-Scalable Data Stores	Traditional/Cloud $\longrightarrow$ Cloud
Geographical Replication	Traditional/Cloud $\longrightarrow$ Cloud
Sharding	Traditional/Cloud $\longrightarrow$ Cloud
Cloud Bursting	Traditional/Cloud $\longleftrightarrow$ Cloud
Working on Data Copy	Traditional $\longrightarrow$ Cloud
Data Synchronization	Traditional $\longleftrightarrow$ Cloud
Backup	Traditional/Cloud $\longrightarrow$ Cloud
Archiving	Traditional/Cloud $\longrightarrow$ Cloud
Data Import from the Cloud	Cloud $\longrightarrow$ Traditional/Cloud

4 CLOUD DATA MIGRATION SCENARIOS

Table 1 summarizes the Cloud Data Migration Scenarios we derived from the State of the Art as discussed in the previous section. The table also identifies the direction of data migration (from/to the traditional to/from any Cloud-based deployment model in Fig. 1). The migration scenarios *Backup, Archiving, and Data Import from the Cloud* can be applied independently from the migration of an application to the Cloud. All other scenarios have been derived from the types of applications and use cases for application migration to the Cloud.

Database Layer Outsourcing is the complete or partial migration of the local, traditionally hosted database layer to the Cloud without changing the type of the data store, e.g., relational, NoSQL, or Binary Large Object (BLOB) data store. An example of a complete Database Layer Outsourcing of a relational data store is the migration of a local MySQL database to Amazon RDS with MySQL configuration.

The migration scenario *Using Highly-Scalable Data Stores* assumes that before migration the data are hosted in a non highly-scalable relational data store, which might be hosted traditionally, or in a Cloud environment. The data in this scenario are migrated to a NoSQL or BLOB data store. Two examples taken from the industry are the media content delivery company Netflix (Anand, 2010) and the Web information company Alexa (Amazon.com, Inc., 2011). Both of them migrated their data from a relational data store to a NoSQL (AmazonSimpleDB[3]) and BLOB data store (Amazon S3[4]) in the Cloud for the purpose of achieving better scalability.

For latency critical applications, the data are moved as near as possible either to the user, or to the processing logic in order to reduce data access latency. The *Geographical Replication* migration scenario considers replicating the database layer for static data, as in the case of content delivery networks, or for data changing dynamically using, e.g., read replicas or read-write replicas. This migration scenario implies that the different replicas have to be kept consistent by realizing synchronization mechanisms. An example for replication of static data is Amazon CloudFront[5] and an example for replication of data changing dynamically is Microsoft Azure SQL used with its Data Sync functionality (Redkar and Guidici, 2011).

In contrast to the previous scenario, *Sharding* distributes the data into disjoint groups without requiring synchronization between the different data stores (shards) (Pritchett, 2008). The distribution can be either done geographically or based on the functional grouping of the data. Guidelines for the realization of a solution considering sharding are available for Microsoft Azure SQL[6] and Amazon RDS[7].

Cloud Bursting is the temporary outsourcing of the database layer to the Cloud in order to use additional resources for off-loading of peak loads, because the available on-premise resources are not suf-

[3]http://aws.amazon.com/simpledb/

[4]http://aws.amazon.com/s3/

[5]http://aws.amazon.com/cloudfront/

[6]http://social.technet.microsoft.com/wiki/contents/articles/1926.how-to-shard-with-windows-azure-sql-database.aspx

[7]http://aws.amazon.com/articles/0040302286264415

ficient. Typical cases where Cloud bursting is applied are seasonal businesses like Christmas shopping outlets. A realization of a solution based on this scenario can be implemented using Microsoft Azure SQL used with its Data Sync functionality (Redkar and Guidici, 2011).

The migration scenario *Working on Data Copy* requires the creation of a complete or partial copy of the database layer in the Cloud for the purpose of avoiding additional load on the production system by, e.g., running complex data analysis. As NoSQL data stores are highly scalable and thus are able to handle additional load in parallel to the production process in the general case the source and target data store for this migration scenario are relational data stores. An example of this migration scenario are data warehouse applications operating on non-live data.

Data Synchronization enables a set of users to work in parallel and temporarily off-line while sharing relatively up-to-date data without latency for the data access. The database layer is copied into the Cloud and a synchronization mechanism is realized. Each user works on a local replica of the database layer which is regularly synchronized with the database layer in the Cloud. A detailed example for Data Synchronization is provided in (Lee et al., 2009). The sales staff are working on local copies of customers data and lists of company product prices on their laptops. The data are regularly synchronized in the background when the employees of the company are connected to the Web.

In order to fulfill compliance regulations such as SOX (United States Congress, 2002) it is required to keep business data and store them over a fixed period of time. These data provide a snapshot at a specific point in time and serve as a save point to return to, e.g., in case of data loss. Especially when using Cloud providers it is recommended to regularly backup your data as you have no direct control over the data when stored on the infrastructure of the Cloud provider (Badger et al., 2012). *Backup* therefore constitutes a particular case of data migration to the Cloud, occurring at predefined intervals. An example backup solution in the Cloud is provided by Datacastle[8]. *Archiving* also creates a complete copy of the database layer to the Cloud, but serving a different purpose than Backup. The usage of archived data is not foreseen in case of errors or data loss, but for analysis and as evidence in court cases (Consultative Committee for Space Data Systems, 2002). Amazon provides the Cloud archiving service Glacier [9] for this purpose.

[8]http://www.datacastlecorp.com

[9]http://aws.amazon.com/glacier/

Several Cloud services are offering access to data via an API. Thus, the database layer of the Cloud service can be partially or completely imported to the local data center in order to minimize the latency for data access during processing. Examples for Cloud services enabling the migration scenario *Data Import from the Cloud* are governments providing their data in a machine readable format like Open Data[10] and services publishing data such as Twitter [11] in order to enable social media monitoring.

5 CLOUD DATA PATTERNS

In order to provide support when migrating application data to the Cloud there is a clear need for describing reusable solutions to overcome the recurring challenges such as incompatibilities on an abstract and technology independent level as patterns. For this purpose, in previous work (Strauch et al., 2012a; Strauch et al., 2012b) an initial list of reusable and technology independent solutions for the identified challenges was proposed in the form of *Cloud Data Patterns*. 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* (Strauch et al., 2012b).

These patterns have been identified as part of the work in various EU research projects, and especially during the collaboration with industry partners. The identified patterns are also based on literature research focusing on available reports from companies that already migrated their application data to the Cloud and adapted their application accordingly, e.g., by Netflix (Anand, 2010). Additionally, guidelines and best practices on how to design and build applications in the Cloud, e.g., for enabling scalability (Adler, 2011), are also taken into consideration. The patterns are furthermore based on requirements and challenges concerning adaptation of applications for the Cloud environment (Andrikopoulos et al., 2012) and management of these applications in the Cloud (Conway and Curry, 2012).

So far, three categories of Cloud Data Patterns have been identified:

1. *Functional Patterns*
2. *Non-functional Patterns*
3. *Confidentiality Patterns*

Confidentiality patterns can be considered a subcategory of the non-functional patterns; they are treated

[10]http://www.opendata-network.org

[11]http://dev.twitter.com/docs/streaming-api/methods

Table 2: Cloud Data Patterns Summary.

Category	Name	Icon	Challenge
Functional	Data Store Functionality Extension	🧩	How can a Cloud data store provide a missing functionality?
	Emulator of Stored Procedures	🧩	How can a Cloud data store not supporting stored procedures provide such functionality?
Non-functional	Local Database Proxy	🧩	How can a Cloud data store not supporting horizontal data read scalability provide that functionality?
	Local Sharding-Based Router	🧩	How can a Cloud data store not supporting horizontal data read and write scalability provide that functionality?
Confidentiality	Confidentiality Level Data Aggregator	🧩	How can data of different confidentiality levels from different data sources be aggregated to one common confidentiality level?
	Confidentiality Level Data Splitter	🧩	How can data of one common confidentiality level be categorized and split into separate data parts belonging to different confidentiality levels?
	Filter of Critical Data	🧩	How can data-access rights be kept when moving the Database Layer into the private Cloud and a part of the Business Layer and a part of the data access layer into the public Cloud?
	Pseudonymizer of Critical Data	🧩	How can a private Cloud data store ensure passing critical data in pseudonymized form to the public Cloud?
	Anonymizer of Critical Data	🧩	How can a private Cloud data store ensure passing critical data only in anonymized form to the public Cloud?

separately however due to their importance to the Data Layer. Table 2 provides an overview of the identified and described Cloud Data Patterns so far.

Functional Cloud Data Patterns like *Data Store Functionality Extension* and *Emulator of Stored Procedures* provide reusable solutions for challenges related to offered functionality by Cloud data stores and services. In case the type of data store changes during the migration, e.g., from RDBMS to NoSQL, or BLOB store, it might be not sufficient to emulate or add additional functionality by using Functional Cloud Data Patterns. For example, there may be no equivalent database schema, the consistency model may change from strict to eventual consistency (Vogels, 2009), and ACID transactions may not be supported.

Non-Functional Cloud Data Patterns focus on providing solutions for ensuring an acceptable Quality of Service (QoS) level by means of scalability in case of increasing data read of data write load. There are two options for this purpose: vertical and horizontal data scaling (Pritchett, 2008; Zawodny and Balling, 2004). Elasticity with respect to data reads is normally achieved by data replication (Buretta, 1997) using read replicas with a master/slave configuration. This is because when write replicas (several master databases) are used, the performance might decrease depending on the consistency model (strict or eventual consistency (Vogels, 2009)).

Confidentiality Cloud Data Patterns provide solutions for avoiding disclosure of confidential data (Table 2). Confidentiality includes security and privacy. With respect to confidentiality, we consider the data to be kept secure and private as critical data such as business secrets of companies, personal data, and health care data, for instance. The personal data and account information of the customers as part of the billing data of HIC, for example, have to be pseudonymized or anonymized before migrating the billing data to the Public Cloud. In this case, the actual people the data is about (HIC customers), and the owner and user of the data (HIC and EAC, respectively) are clearly distinguished. The migration of anonymized or pseudonymized data to the Cloud therefore does not effect the management of the identity of users of Cloud data hosting solutions, e.g., in large-scale Public Cloud environments. The interested reader is referred to (Strauch et al., 2012a; Strauch et al., 2012b) for an in-depth discussion on these patterns.

6 PATTERN-BASED APPLICATION REFACTORING

In this section we first introduce the mapping of Cloud Data Migration Scenarios to Cloud Data Patterns (Section 6.1). Afterwards we propose a step-by-step methodology for the migration of the data layer to the Cloud covering all migration scenarios and using the patterns (Section 6.2). Finally, we apply the methodology to the application introduced in the motivating scenario focusing on application architecture refactoring using the patterns (Section 6.3).

6.1 Mapping Scenarios to Patterns

Table 3 provides an overview on how Cloud Data Migration Scenarios map to Cloud Data Patterns. In order to avoid repetition, in the following we discuss the mapping of the migration scenario *Database Layer Outsourcing* to the various patterns in detail and for the other scenarios we focus on explaining the differences compared to the mapping of this migration scenario.

All patterns are applicable for the scenario *Database Layer Outsourcing* depending on the specific conditions of the required solution. As no change of the data store type takes place, but there might be a change of the product, potential incompatibilities, e.g. regarding realization of data types, or missing functionalities used before, like stored procedures, have to be added or emulated using the functional patterns. In case the DBL is completely moved to the Cloud, a solution to reduce the latency for data access is to host read replicas locally and to use a *Local Database Proxy* to forward data reads to these replicas instead of querying the DBL in the Cloud for every read request. In order to scale both data writes and reads, a *Sharding-Based Router* can be used to enable sharding in case it is not supported by the Cloud data store(s) chosen for migration.

When the database layer is migrated to the public Cloud, it has to be decided which critical data will not be moved to the Cloud, e.g., as business secrets. The patterns *Confidentiality Level Data Aggregator* and *Confidentiality Level Data Splitter* enable the differentiation and harmonization of the confidentiality level of different data sets from potentially different domains and different data sources. The other confidentiality patterns enable filtering of critical data that have to stay locally in case the database layer is only partially migrated to the Cloud, and removing or masking the critical data by anonymization or pseudonymization before moving them to the public Cloud.

As in the migration scenario *Using Highly-Scalable Data Stores* the data in the DBL can also be only partially migrated, the non-functional patterns might be applicable as in the fist migration scenario. Thus, there are no differences in the pattern mapping compared to the migration scenario *Database Layer Outsourcing*. The usage of the *Sharding-Based Router* is not typical for the migration scenario *Geographical Replication* since sharding the data disjointly distributes them, instead of replicating them. When combining replication and sharding however, e.g., by sharding some write replicas that are more often accessed, this pattern is also applicable.

A non-typical use of the *Local Database Proxy* pattern for the scenario *Sharding* can be applied in case, e.g., some shards that are more often accessed via read are replicated for read scalability. The *Local Sharding-based Router* might be even used in the scenario *Sharding* in order to keep the sharding in the database layer transparent for the business layer to avoid adaptations to the business logic. In the *Working on Data Copy* scenario, data analysis is performed on the complete or partial copy of the database layer, instead of the actual data. As such, the non-functional patterns are not applicable for this scenario.

None of the *Cloud Data Patterns* are applicable in the *Backup* scenario, because the purpose of this scenario is to restore the latest, saved state of the system as fast as possible in case an error or outage occurs. No further processing on the backup data takes place. Thus, in such a time critical situation even the usage of the *Pseudonymizer of Critical Data* is to much overhead that delays restoring the latest status. The *Pseudonymizer of Critical Data* however can be used in the scenario *Archiving* to save even critical data, e.g. personal data, in the public Cloud. The relation of the masked data to the original data has to be stored separately and safely in order to avoid revealing and to be able to revert the masking of the data before the archived data can be used. All other

Table 3: Scenario/Pattern Mapping.

	Functional		**Non-functional**		**Confidentiality**				
	Data Store Functionality Extension	Emulator of Stored Procedures	Local Database Proxy	Local Sharding-based Router	Confidentiality Level Data Aggegrator	Confidentiality Level Data Splitter	Filter of Critical Data	Pseudonymizer of Critical Data	Anonymizer of Critical Data
Database Layer Outsourcing	✓	✓	✓	✓	✓	✓	✓	✓	✓
Using Highly-Scalable Data Stores	✓	✓	✓	✓	✓	✓	✓	✓	✓
Geographical Replication	✓	✓	✓	(✓)	✓	✓	✓	✓	✓
Sharding	✓	✓	(✓)	(✓)	✓	✓	✓	✓	✓
Cloud Bursting	✓	✓	✓	✓	✓	✓	✓	✓	✓
Working on Data Copy	✓	✓	–	–	✓	✓	✓	✓	✓
Data Synchronization	✓	✓	✓	✓	✓	✓	✓	✓	✓
Backup	–	–	–	–	–	–	–	–	–
Archiving	–	–	–	–	–	–	–	✓	–
Data Import from the Cloud	–	–	–	–	(✓)	(✓)	(✓)	(✓)	(✓)

Legend: ✓: applicable, (✓) : might be applicable, – : not applicable

patterns are not applicable to scenario *Archiving*, as there will be no further processing on the data after it has been archived.

As the user has no impact on the API offered by the Cloud service to import data, the functional and non-functional patterns are not relevant for the migration scenario *Data Import from the Cloud*. The confidentiality patterns might be applicable depending on the goal of using the imported data. Thus, the data might be filtered, anonymized, pseudonymized, categorized or harmonized based on pre-defined confidentiality levels before they are imported.

6.2 Cloud Data Migration

In this section we propose a methodology for migration of the database layer to the Cloud and adaptation of the data access and business layer accordingly. The methodology entails the following steps:

1. Identify relevant Cloud Data Migration Scenario.
2. Describe desired Cloud data store.
3. Select suitable Cloud Data Store.
4. Identify the applicable patterns to solve incompatibilities and to refactor the application.
5. Refactor the application by adapting the database layer and upper architectural layers while implementing the identified patterns.
6. Migrate the data to the selected Cloud Data Store.

First of all the *Cloud Data Migration Scenario* has to be identified. Therefore, the specific requirements of the solution needed have to be mapped to the characteristics of the migration scenarios defined in Section 4. Afterwards, the functional and non-functional requirements such as data store type and read/write throughput of the desired Cloud data store have to be defined. By mapping this set of requirements to the specifications provided by the Cloud data store providers, a specific Cloud data store can be selected. This step can be automated, e.g., by querying a Cloud data store repository containing the concrete properties of available Cloud data stores.

In order to identify the patterns to be used to solve incompatibilities and refactor the application, the non-functional and functional requirements of the database layer used before migration have to be specified. The comparison and mapping of these require-

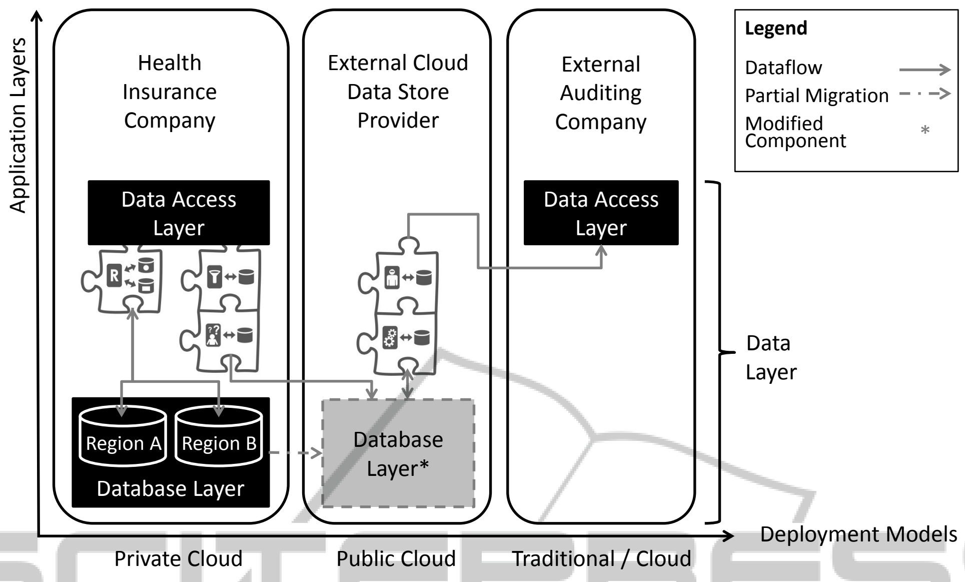

Figure 2: Refactoring of motivating scenario using Cloud Data Patterns (Data Layer).

ments of the database layer previously used, to the defined requirements for the desired Cloud data store lead to the identification of potential conflicts and incompatibilities. The mapping of the identified migration scenario to the corresponding patterns (Section 6.1) should be used for orientation and limiting the number of patterns to be considered for application refactoring. Based on the characteristics of the Cloud Data Patterns, and on the specific conditions when they can be applied (Section 5), the patterns to solve the incompatibilities and conflicts have to be chosen.

After the patterns to be applied have been identified, the actual application refactoring can take place by implementing the patterns and thus adapting the database layer and upper architectural layers accordingly. This is because the adaptation of the database layer (and/or data access layer) by using Cloud Data Patterns may require adaptations of the business layer as well. For example the usage of confidentiality patterns for filtering, anonymization, or pseudonymization creates a different view on the data for the business layer by retrieving the application data in an aggregated format in comparison to the original database layer. Finally, the data themselves have to be migrated to the selected Cloud data store, including the necessary DB schema creation. Existing tools and services like the ones discussed in Section 3 can be used for this purpose.

6.3 Refactoring the Motivating Scenario

In the following, we discuss how the functional, non-functional, and confidentiality patterns can be used in practice to realize the application based on the motivating scenario introduced in Section 2 and refactor the application accordingly. As discussed in Section 2, HIC has to provide access to EAC to audit the financial transactions processed by the company while avoiding additional load on the system. This essentially means that only the data on financial transactions have to be replicated. Furthermore, EAC can do no changes to the actual application data and therefore no synchronization is required. These specific conditions on the required solution lead to the choice of the migration scenario *Geographical Replication* in the first step of the methodology (Section 6.2). Thus, the database layer of the HIC is partially replicated in the public Cloud by focusing on the financial transactions only.

In the following we focus on identifying the patterns to be used and on the refactoring of the application based on the requirements described in the motivating scenario. According to the mapping of the scenario *Geographical Replication* to *Cloud Data Patterns* (Table 3) all patterns are potentially applicable to this scenario. Figure 2 provides an overview of the refactored application and realization of the scenario using Cloud Data Patterns.

More specifically, in order to provide horizontal scalability for reads and writes to the data of the clients and their transactions, the Local Sharding-Based Router pattern is used. The data is separated between the two data centers according to the geographical location. The Google App Engine Datastore is chosen for replicating the data on financial transactions, configured accordingly. The client information and their corresponding medical records are critical data to be kept only in the company's private Cloud. Financial transactions information will appear both in the private and in the public Cloud. In order to keep both the data stored in the private data store and the one outsourced to the public Cloud consistent, data updates and inserts should be done in parallel to both the private and the public part of the database layer.

However, only a part of the financial data is necessary for the auditing (e. g., client names can be removed) and the remaining can be pseudonymized before moving to the public Cloud (e. g., bank account numbers replaced by serial IDs). A composition of the Filter of Critical Data and the Pseudonymizer of Critical Data is used to fulfill both requirements. The filter is configured so that only data on financial transactions pass it. After passing the filter, the information on financial transactions is pseudonymized before it is stored in the public Cloud. Storage of data on the auditing company side can be done either in a private Cloud or in the traditional manner; this is out of the scope of this discussion.

Due to the challenges identified in the motivating scenario regarding the data access of the auditing company, the query results must not contain any relations concerning clients and their corresponding medical records or personal data. Therefore, this information has to be completely deleted before passing the query results to the auditing company (instead of simply obfuscating this relation by using pseudonymization). In addition, the queries to be executed by the auditor have to be agreed upon in advance. For these purposes, the realization uses a composition of the Anonymizer of Critical Data and the Emulator of Stored Procedures patterns. The emulator is used to predefine and restrict the data queries allowed to be executed by the auditing company. The Anonymizer of Critical Data additionally ensures the removal of any critical information from the results of the queries.

A combination of functional, non-functional, and confidentiality patterns can therefore be used in tandem by the proposed methodology to address the requirements of the use case posed by the motivating scenario.

7 CONCLUSIONS AND FUTURE WORK

In the previous sections we discussed the need for supporting the migration of application data to the Cloud. A number of technical, architectural and legal issues related to this effort were identified by means of a motivating scenario. The State of the Art in migrating applications and application data to the Cloud was then analyzed for best practices and associated challenges, and it was organized into ten distinct data migration scenarios. These scenarios cover different aspects of data migration to and from the Cloud and include, among others, outsourcing the database layer, replication of data to reduce latency, Cloud bursting, and synchronization. A set of Cloud Data Patterns as reusable solutions to common migration-related challenges was then briefly summarized, and a mapping between the identified scenarios and the patterns was presented. An application data migration methodology was then introduced based on this mapping, and the motivating scenario application used in the beginning of the paper was refactored to demonstrate the applicability of our work.

Neither the list of the migration scenarios, nor the one of the patterns is claimed to be complete. Further effort in augmenting and refining both scenarios and patterns is required in the future. Providing tooling support for applying the application data migration methodology discussed in Section 6 could provide significant help in this direction. A decision support system built for this purpose, in the manner of work like (Menzel and Ranjan, 2012) and (Khajeh-Hosseini et al., 2012), could be an important contribution on its own, guiding practictioners and researchers through the migration of their application to the Cloud. Finally, the discussion in this work has to be positioned within a larger framework dealing with the migration of applications to the Cloud, and the adaptations that could result from this migration.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the 4CaaSt project (http://www.4caast.eu) part of the European Union's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 258862.

REFERENCES

Adler, B. (2011). Building Scalable Applications In the Cloud: Reference Architecture & Best Practices, RightScale Inc.

Amazon.com, Inc. (2011). AWS Case Study: Alexa.

Anand, S. (2010). Netflix's Transition to High-Availability Storage Systems.

Andrikopoulos, V., Binz, T., Leymann, F., and Strauch, S. (2012). How to Adapt Applications for the Cloud Environment. *Springer Computing*.

Armbrust, M. et al. (2009). Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley.

Badger, L., Grance, T., R., P.-C., and Voas, J. (2012). Cloud Computing Synopsis and Recommendations - Recommendations of the National Institute of Standards and Technology. NIST Special Publication 800-146.

Buretta, M. (1997). *Data Replication: Tools and Techniques for Managing Distributed Information*. John Wiley & Sons, Inc.

Consultative Committee for Space Data Systems (2002). Reference Model for an Open Archival Information System (OAIS).

Conway, G. and Curry, E. (2012). Managing Cloud Computing: A Life Cycle Approach. In *Proceedings of CLOSER'12*. SciTePress.

Cunningham, S. R. (2010). Windows Azure Application Profile Guidance. Custom E-Commerce (Elasticity Focus) Application Migration Scenario.

Fowler, M. et al. (2002). *Patterns of Enterprise Application Architecture*. Addison-Wesley Professional.

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

Khajeh-Hosseini, A., Greenwood, D., Smith, J. W., and Sommerville, I. (2012). The Cloud Adoption Toolkit: Supporting Cloud Adoption Decisions in the Enterprise. *Software: Practice and Experience*, 42(4):447–465.

Laszewski, T. and Nauduri, P. (2011). *Migrating to the Cloud: Oracle Client / Server Modernization*. Elsevier Science.

Lee, J., Malcolm, G., and Matthews, A. (2009). Overview of Microsoft SQL Azure Database.

Mell, P. and Grance, T. (2009). Cloud Computing Definition. *National Institute of Standards and Technology, Version 15*.

Menzel, M. and Ranjan, R. (2012). CloudGenius: Decision Support for Web Server Cloud Migration. In *Proceedings of WWW '12*, New York, NY, USA. ACM.

Microsoft Azure (2012). Generic Migration Scenarios and Case Studies.

Pritchett, D. (2008). BASE: An ACID Alternative. *Queue*, 6(3):48–55.

Redkar, T. and Guidici, T. (2011). *Windows Azure Platform*. Apress.

Strauch, S., Andrikopoulos, V., Breitenbücher, U., Kopp, O., and Leymann, F. (2012a). Non-Functional Data Layer Patterns for Cloud Applications. In *Proceedings of CloudCom'12*. IEEE Computer Society Press.

Strauch, S., Breitenbücher, U., Kopp, O., Leymann, F., and Unger, T. (2012b). Cloud Data Patterns for Confidentiality. In *Proceedings of CLOSER'12*. SciTePress.

Tak, B. C., Urgaonkar, B., and Sivasubramaniam, A. (2011). To Move or Not to Move: The Economics of Cloud Computing. In *Proceedings of HotCloud'11*, Berkeley, CA, USA. USENIX Association.

United States Congress (2002). Sarbanes-Oxley Act (SOX).

Varia, J. (2010). Migrating your Existing Applications to the AWS Cloud. A Phase-driven Approach to Cloud Migration.

Vogels, W. (2009). Eventually Consistent. *Communications of the ACM*, 52(1):40–44.

Zawodny, J. and Balling, D. (2004). *High Performance MySQL: Optimization, Backups, Replication, Loadbalancing, and More*. O'Reilly & Associates, Inc. Sebastopol, CA, USA.

All links were last followed on February 5, 2013.