Cloud Deployment Patterns: Migrating a Database Driven Application to the Cloud using Design Patterns

A.A. Adewojo, J.M. Bass and K.Hui, I.K. Allison

Abstract— Cloud computing provides scalable and reliable computing services that can be beneficial to software organizations that intend to migrate their existing or new applications to the cloud. However, migration is potentially complex, so cloud computing deployment patterns are proposed to support the migration process. This research compares the format, structure and notations of previous object oriented design patterns with a recent cloud computing design pattern. Firstly, the gaps in cloud computing design patterns catalogue are identified. Secondly, we present a template for creating pattern catalogue for cloud deployment patterns. This template was derived from a widely accepted and most highly cited design pattern catalogue and we applied this template to the shared component pattern, a variant of multi-tenancy pattern. Finally, we demonstrated the shared component's pattern validity by applying it to the data model of a database driven desktop application that was migrated to the cloud. The result shows that: (i) there is an improvement in the structure and clarity of the shared component pattern catalogue; and (ii) Information conveyed to software developers is enhanced.

Index Terms— SaaS; Cloud Computing; Deployment Patterns; Cloud Migration

I. Introduction

Global software engineering researchers and practitioners have rightly focused on distributed models of software development [1] [2] [3]. We argue here that the development of software services for worldwide deployment presents a significant challenge for practitioners of global software engineering.

Cloud services centralize deployment, maintenance and evolution but cater for a worldwide user audience [4]. Cloud services are classified into three namely: Software as a Service (SaaS), Platform as a Service (PaaS) and Infrastructure as a Service (IaaS) [5] [6]. SaaS releases application to customers as a service [7], these services are accessed via the internet and consumers are charged for only the quantity of software, its functionalities and time used [5]. PaaS provides development platforms as a service. It makes provision for customizing and deploying applications on the infrastructure [5]. IaaS provides computing infrastructure in the form of virtual machines as a service to customers [7]. Consumers can enjoy flexibility of creating, managing, deploying and customizing their servers to to suit their needs. Cloud computing also provides scalable, reliable and ubiquitous computing representing a paradigm shift in computing that has a potential of transforming IT industry [8]. Cloud computing has gained much popularity both in academic and industrial world; hence prompting large organizations and startups to either move their on-premise applications to the cloud or build a cloud compliant application [9] [10]. However, this migration is potentially complex so cloud computing deployment patterns are proposed to assist these organizations.

Several design patterns are advocated to help make good design decisions, capturing best practices on how system applications should be designed [11]. The benefits of using these patterns include development of reusable and compliant software systems [12]. Interoperability, portability and manageability of software systems are specific benefits of using cloud design patterns [9].

Cloud deployment patterns play a major role in architectural restructuring and migration of on-premise software applications to the cloud [11]. Current cloud computing deployment patterns lack details which make them difficult to use. Furthermore, this lack of detail hampers deployment pattern selection.

We address this issue by comparing previous object oriented software design patterns catalogue [13] with recent cloud computing deployment patterns catalogue [11]. The format, structure, and graphical notations were the main basis of comparison. We used the object oriented design patterns as the baseline for comparison [13]. This is because it has a consistent format of describing patterns and is one of the most highly cited design patterns catalogues.

The contribution of this research is an enhanced cloud deployment pattern description for the shared component pattern. The benefits of the enhanced shared component pattern are: that they are more systematically presented, easier to implement, with more logical headings and detailed pattern description. This will help software developers to make right design decisions as fast as possible [13].

The paper is structured as follows: a description of related work on design patterns is done in section 2. Section 3

Manuscript received April 3, 2015, revised June 20, 2015. This work was supported by the Scottish Funding Council who funded the Scottish cloud horizon project, the Technology Strategy Board and WML UK who provided funds for the conference expenses.

A.A. Adewojo is a researcher with the School of Computing Science and Digital Media, Robert Gordon University, Aberdeen, Scotland. Phone: +441224262575; e-mail: a.a.adewojo@rgu.ac.uk

J.M. Bass is a senior lecturer with the School of Computing Science and Digital Media, Robert Gordon University, Aberdeen, Scotland. e-mail: j.m.bass@rgu.ac.uk

K.Hui is a lecturer with the School of Computing Science and Digital Media, Robert Gordon University, Aberdeen, Scotland. e-mail: k.hui@rgu.ac.uk

I.K. Allison is the Dean, School of Engineering and Computing University of the West Scotland, Paisley, Scotland, email: ian.allison@uws.ac.uk

describes the research method used to implement this research. Section 4 introduces shared component pattern. Section 5 describes how the shared component pattern was improved. Section 6 describes the case study application that was used to validate the pattern. Section 7 discusses how it was implemented and Section 8 summarizes the work done and thoughts on future work. Finally, the appendices contain the enhanced pattern.

II. RELATED WORK

In this section, we discuss existing work on design patterns, pattern languages, cloud computing principles and how these relate to cloud deployment patterns.

Several parts constitute design patterns such as a recurrent problem, its proposed solution, factors that might affect the problem or solution, and rationale behind the solution. However, a recurrent problem in software design is the main motivator of a software design pattern and a formal approach that generalizes the solution to this problem is a key factor to a successful design pattern.

To identify and get acquainted with the standardized format of design patterns, pattern languages, and pattern catalogue, we reviewed the following literatures: Design patterns: elements of reusable object-oriented software by [13]. They defined pattern language and its applicability to object oriented software system. They used a systematic approach to catalogue 23 different object oriented design patterns. These patterns are widely and consistently used in building object oriented system to date. Coad [14] presented seven different patterns for object oriented analysis (OOA) and object oriented design (OOD). Each pattern had an example that illustrates how to use it, also guidelines on how to apply each of them were provided. Martin [15] described software architectures and how design patterns can either positively or negatively affect the software application. They identified rigidity, fragility, immobility and viscosity as major symptoms of rotting design.

Because these patterns are specific to object oriented design, there is a need to review literatures on cloud basic properties, their architectural needs, and design patterns that apply to them. Khorshed et. al [16] describes cloud computing as a system whose data center's resources are delivered as services using virtualization technologies via the internet to provide elastic, on-demand and instant services to its customers. Armbrust et. al [17] identified utility computing, scalability, multi-tenancy, flexibility, manageability, portability as key properties of cloud computing. Based on these principles and their requirements, we reviewed how design patterns in cloud computing are captured to help improve on the identified features of cloud computing. In [11], the basic properties of cloud computing were described and how these properties can be used in our IT infrastructure. Based on these properties they captured best practices on solving recurrent problems in architecting applications for the cloud and migrating existing applications to the cloud. Wilder [18] explored cloud patterns that are useful for architecting cloud-native applications and that can be used to overcome specific challenges that can be encountered in cloud application development. For each of the pattern covered in the book, there is a primer that precedes it. Primer gives a broad understanding of why there is or might be a need for the preceding pattern. This book does not catalogue patterns because the authors argue that each pattern impact multiple architectural concerns, hence it cannot be cleanly catalogued. Having looked at these literatures, object oriented patterns by [13] and cloud computing patterns by [11] inspired us to improve the shared component pattern; a cloud deployment pattern in line with best practices from the software design pattern literature.

III. RESEARCH METHOD

The sequential mixed method approach [19] was used to devise our research process. This will inform the priority of data collection strategy, data analysis, and theoretical perspective of our research. It comprised an inductive development of enhanced cloud deployment pattern and an application case study. The theoretical aspect explores and compared existing design patterns with cloud deployment patterns available from the literature. The format, structure, graphical notations and applicability of these patterns are studied and compared in-line with best practices to develop an enhanced pattern format for the multi-tenancy patterns.

Case study strategy is used to gather and analyze information. The case study approach investigates an observable circumstance in its real-life context [20]. The aim of observing in real-life context is to provide an analysis of the context and processes involved so that the theoretical issues being studied can be well understood [21] and because evidence is gotten from a real-life setting of the case [22].

For our case study, we selected a small business software application. This software application is used to produce business processes. It maximizes a company's efficiency and resources by providing an automated and concise way of defining and communicating what the company does. Its potential users range from company managers to employees who are in the business support unit. This application however has some significant limitations, in its lack of scalability and resource sharing. Hence, we chose this application because the multi-tenancy principle is of utmost importance to it. We applied the newly developed cloud deployment pattern, and tested it with a load tester called LoadUI [23] and by different users. The results establish that the new patterns implementation meets the needs of the application.

IV. SHARED COMPONENT AND CLOUD DEPLOYMENT PATTERNS

The multi-tenancy pattern describes how SaaS application component can be shared between different tenants, the challenges of sharing same instance of software and the solution to these challenges [11] [24]. It is one of the five essential cloud properties [11] [24] that supports resource sharing in cloud deployed SaaS application. Shared component, tenant-isolated component and dedicated component are the three levels of multi-tenancy patterns captured by [11]. In this paper, we focus on the shared component pattern, the first level of the multi-tenancy pattern. It is the basic minimum requirement for resource sharing in a SaaS application. It minimizes resource usage however; this is at the risk of data and processes isolation [11][6].It is used to provide functionality to different tenants without maintaining a notion of tenants itself, hence, tenants can influence each other while this functionality is being accessed [11].

When developing a cloud native application, it is important to consider components of the applications, and cloud infrastructure that can and will be shared with other applications or other cloud instances. To make the software robust and reusable, it is expedient that right design patterns be employed. However, a precise description of this is needed to make an efficient use of it. It is in response to this that we have proposed an enhanced shared component pattern.

V. ENHANCED SHARED COMPONENT PATTERN

Our research observed gaps in the description of cloud deployment pattern by [11]. To address this, we enhanced the shared component pattern to conform to consistent and widely accepted design pattern format. This improvement is based on what we have learnt from revised literatures, our proposed template and best practices in cataloguing a pattern. Table 1 shows the existing cloud deployment pattern headings, the object oriented pattern headings, the gaps we need to fill and the proposed template.

Table I. PATTERN HEADINGS COMPARISON AND TEMPLATE

Cloud Patterns	OOP Patterns	Restructured Pattern
-	Intent	Intent
-	Alias	Motivation
-	Motivation	Applicability
-	Applicability	Structure
-	Structure	Participants
-	Participants	Collaborations
-	Collaborations	Consequences
-	Consequences	Implementation
-	Implementation	Sample Code
-	Sample Code	Known Uses
Known uses	Known uses	
Related patterns	Related patterns (This does not apply to all patterns)	Related patterns
Context		
Solution		
Result		
Variations (This does not apply to all patterns)		
Driving question		

The enhanced shared component pattern now includes intent, motivation, applicability, structure, participants, collaborations, consequences, implementation and sample code. However, some contents of these headings can be found in the original cloud computing patterns definition by [11] but because they are not distinctly spelled out, users can not follow them precisely. The newly enhanced pattern description for Shared Component and its detailed content are elaborately described in appendix 1. The enhanced shared component pattern benefits from a standardized and formal approach of describing patterns; this will then help upcoming developers to efficiently use this pattern and also communicate using well-known names for software interactions.

In the next section, we apply the shared component pattern to a case study application. This is to validate its applicability in cloud deployed applications.

VI. CASE STUDY

The case study application is a business process management (BPM) system that runs on a desktop application intended to be migrated to the cloud. Because it is a desktop application, it cannot be migrated to the cloud as it is, so a web based prototype application of the software was built and deployed on Amazon Web Services (AWS). This effectively turns the application into a SaaS. The goal of this design is to allow the key features of a cloud native application such as maintainability, data accessibility, scalability, multi-tenancy, and interoperability [11].

Appropriate design principles have been used in-line with quality concerns such as performance, modifiability, and usability of the system [25]. Amongst these are loose coupling, modularization, separation of concerns and abstraction [25]. These principles are implemented via these patterns: Model-View-Controller (MVC), Service-Oriented Architecture (SOA), Representational State Transfer (REST), and cloud deployment patterns. MVC was used to decompose this system into three components: data storage, application server, and user interface component. SOA was used to present its data logic and application server component as set of services. REST was used to deliver these services so as to promote interoperability. Shared component pattern and scalability were used to improve resource sharing and user's accessibility to the system. The web application architecture is shown in Fig.1.

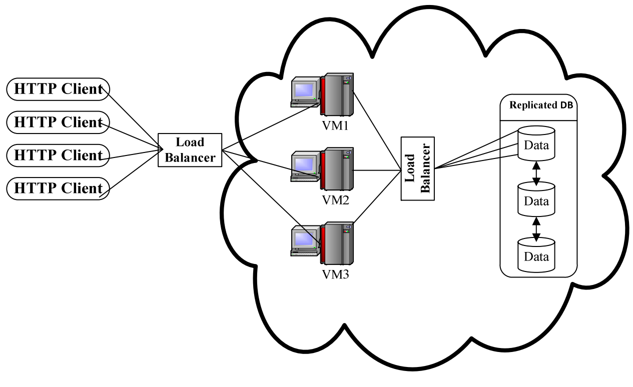

Fig. 1. Web Application Architecture

The web application architecture functions thus: A load balancer intercepts all incoming requests from thin clients and distributes them to an appropriate VM instance. These clients communicate with the web server via HTTP protocol. The VM contains both the application server and the web-server. The web server handles the HTTP protocol, passes request to an appropriate program that is able to handle the request and in response sends a dynamic HTML page for viewing in the web browser. The application server provides access to business logic for use by the client application programs. A load balancer then distributes data that needs to be stored to available data storage. These storage spaces are replicated geographically to avoid problems caused by failure of the storage space.

A no-SQL storage device – SimpleDB by Amazon, which is accessed via a RESTful API [26] is used as the storage space, thus promoting SOA and REST architectures. The different web services runs on a virtual machine (VM) and the VM are easily replicated on demand, thus promoting scalability. Scalability is also improved because the application can run at least in part or parallel and in effect allows resources to be shared efficiently, which is a key feature of SaaS application [24]. This loosely coupled architecture makes the system flexible enough to run different components on different cloud vendors.

VII. DISCUSSION

The shared component pattern was implemented in the data storage component of the case study application. The data storage component was designed to allow multiple customers access a single instance of SimpleDB; however this is at the cost of no isolation of data and processes. It was implemented as follows: A single domain was used to store all companies' data. However each item in a domain is uniquely identified by the company's id. This means different companies share the same domain. This approach offers the highest degree of sharing, but at the risk of data privacy and security. The code snippet below show how this is implemented.

```
    myDomain = "allDomain"; // assigns the
    domain name
    sdb.batchPutAttributes(new
    BatchPutAttributesRequest(myDomain,
    createSampleData()));    //   a   batch
    addition of data to simpleDB is done
    at this stage
     //createSampleData() is a function to
      add all the parts of a process to a
                  linked list
          }
```

Fig. 2.Code snippet to implement Shared Component pattern

The shared component pattern has been validated because we are able to successfully implement it in our case study application.

VIII. CONCLUSIONS AND FUTURE WORK

Choosing the right cloud deployment pattern is a key determinant in building a reusable and compliant SaaS system in software engineering. To aid this process, we presented an enhanced template for describing the shared component pattern, and we also improved the clarity and organization of the shared component pattern in line with design pattern best practice. The shared component pattern has been enhanced to include an elaborate and systematic description of the pattern. This includes Logical headings and detailed expression of each heading.

The Shared Component pattern minimizes resource overheads by making efficient use of critical components. Tenants are each allocated a quota of the shared resource. However, sharing components could compromise privacy, performance and security. This is an area stakeholder and developers need to agree on before commencing to use this pattern.

We argue that our enhanced shared component pattern will help those learning to implement this pattern by reducing the length of time [13] it takes them to understand the implications of using it. It will also provide a common language where they can communicate using well understood names for software interactions.

We also demonstrated the applicability of the shared component pattern as a resource sharing architectural property of a SaaS application in our case study. A successful implementation of it validates this.

The next stage of our study is to apply the proposed template to improve the tenant-isolated and dedicated component pattern catalogue. The Dedicated Component pattern provides exclusive access to application components that provide critical functionality; this is at a higher cost in terms of resource overhead and monetary cost. The Tenant-isolated component pattern represents a compromised implementation between the Shared Component and Dedicated Component approaches. It involves some sharing of resources with intermediate levels of performance, security, privacy and resource overheads.

APPENDIX

A. Intent

Allow multiple tenants to access a component of the application and leverage economies of scale [11]

B. Motivation

To address a large number of customers and in turn leverage economies of scale [11]

C. Applicability

Use shared components when:

Web services are used for authentication and user rights management that are within the scope of one company [11]

D. Structure

The database is created once because tenant shares the database. Tenants also share the table and each row is identified by the tenant and row id because there will be many tables and rows in this database. This does not guarantee security and privacy of data, but it utilizes resources efficiently and reduced cost of database connections. This is shown in figure 3.

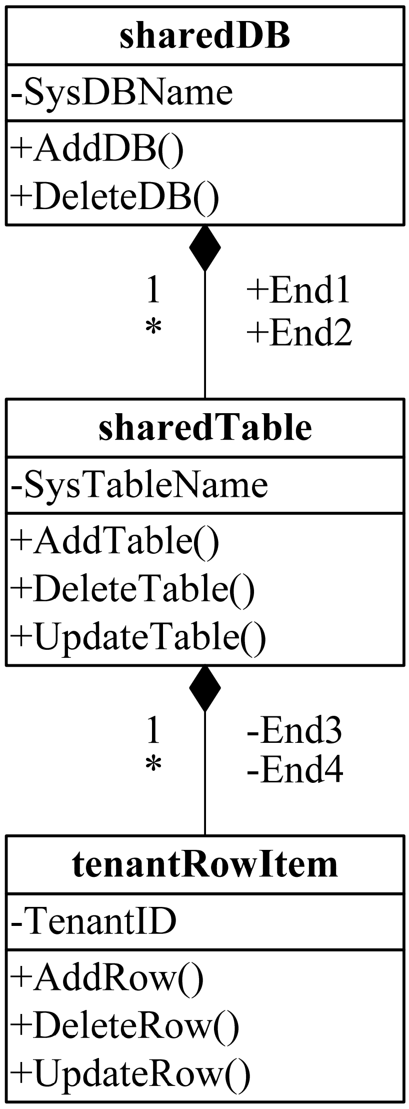

Fig. 3. UML diagram for Shared Component pattern

E. Participants

This defines a database and lets user add tables and rows. Multiple user data can be represented in different row of a table.

F. Collaborations

Key value storage pattern [11]

G. Consequences

The functionality provided by a shared component is unaware of the actual tenant for which request is being executed. Hence, the behavior of one tenant may affect other tenants. Also instances of a shared component can be scaled out depending on the overall workload and the set limit. This in turns reduces the commissioning and decommissioning process [11]

H. Implementation

This allows components to be shared by multiple tenants without much restriction on identifying the tenant. The following issues should be considered when there is a need to use a shared component pattern [7]:

1) Tenant Influence: Influences between tenants that may occur when sharing components have to be avoided.

2) Tenant Requirements: other requirements of tenants that disallows sharing of resources

I. Sample Code

```
CREATE DATABASE [SysDBName] // Create DB
for all tenants
CREATE TABLE [SysDBName].[SysTableName](
TenantID      datatype,   TenantColData1
datatype, TenantColData2  datatype,…) //
Create Table for all tenants
Statement sta = conn.createStatement();
statement.executeUpdate("INSERT     INTO
SysTableName " + "VALUES
(TenantID,'TenantData1', 'TenantData2'),
(TenantID2,'TenantData3', TenantData4'),
(TenantID3,'TenantData5','TenantData6'),
(TenantID4,'TenantData7','TenantData8'))
"  where  TenantID  =   tID);  //Add  Data
based on tenant and row Id
conn.close();//close connection
```

Fig. 4. Sample Code for Shared Component DB Creation

J. Known Uses

National Weather Service provided by the National Oceanic and Atmospheric Administration (NOAA). This provides a web service as a shared component interface that can be integrated in applications [11]

K. Related Patterns

Tenant-isolated and dedicated patterns are related patterns that can be used if sharing of application component is unsuitable for tenants. However if shared component pattern is used, the following patterns may be relevant: management configuration, periodic workload, private clouds and hypervisor [11]

ACKNOWLEDGMENT

We acknowledge the owners of the case study software application. A.A. Adewojo is grateful to S.A. Arekete for the feedbacks while writing this paper.

REFERENCES

[1] C. Ebert, Global Software and IT: A Guide to Distributed Development, Projects, and Outsourcing. Hoboken, N.J.: Wiley-Blackwell, 2011.
[2] M. Y. Vardi, "Globalization and Offshoring of Software Revisited," Commun ACM, vol. 53, pp. 5-5, may, 2010.
[3] J. D. Herbsleb and D. Moitra, "Global software development," Software, IEEE, vol. 18, pp. 16-20, 2001.
[4] R. Moreno-Vozmediano, R. S. Montero and I. M. Llorente, "Key Challenges in Cloud Computing: Enabling the Future Internet of Services," Internet Computing, IEEE, vol. 17, pp. 18-25, 2013.
[5] S. Zhang, H. Yan and X. Chen, "Research on Key Technologies of Cloud Computing," Physics Procedia, vol. 33, pp. 1791-1797, 2012.
[6] P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," NIST Special Publication, vol. 800, pp. 145, 2011.
[7] R. Dargha, "Cloud computing: From hype to reality: Fast tracking cloud adoption." in Icacci, 2012, pp. 440-445.
[8] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica and M. Zaharia, "Above the clouds: A berkeley view of cloud computing," EECS Department, University of California, Berkeley, Feb. 2010.
[9] G. Cretella and B. Di Martino, "An Overview of Approaches for the Migration of Applications to the Cloud," vol. 7, pp. 67-75, 2014.
[10] P. Jamshidi, A. Ahmad and C. Pahl, "Cloud Migration Research: A Systematic Review," Cloud Computing, IEEE Transactions on, vol. 1, pp. 142-157, 2013.

[11] C. Fehling, F. Leymann, R. Retter, W. Schupeck and P. Arbitter, Cloud Computing Patterns
Fundamentals to Deisign, Build, and Manage Cloud Applications. London: Springer, 2014.

[12] F. Palma, H. Farzin, Y. Gueheneuc and N. Moha, "Recommendation system for design patterns in software development: An DPR overview," in Recommendation Systems for Software Engineering (RSSE), 2012 Third International Workshop on, 2012, pp. 1-5.

[13] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns: Element of Reusable Object-Oriented Software. Holland: Addison-Wesley, 1995.

[14] P. Coad, "Object-oriented Patterns," Commun ACM, vol. 35, pp. 152-159, sep, 1992.

[15] R. C. Martin, "Design principles and design patterns," .

[16] M. T. Khorshed, A. B. M. S. Ali and S. A. Wasimi, "Monitoring insiders activities in cloud computing using rule based learning," in Trust, Security and Privacy in Computing and Communications (TrustCom), 2011 IEEE 10th International Conference on, 2011, pp. 757-764.

[17] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin and I. Stoica, "A view of cloud computing," Commun ACM, vol. 53, pp. 50-58, 2010.

[18] B. Wilder, Cloud Architecture Patterns. O'Reilly, 2012.

[19] J. W. Creswell, Research Design: Qualitative, Quantitative, and Mixed Methods Approaches. London: Sage, 2002.

[20] R. K. Yin, Applications of Case Study Research. Thousand Oaks, CA: Sage, 2003.

[21] C. Cassell and G. Symon, Essential Guide to Qualitative Methods in Organizational Research. Sage, 2004.

[22] S. Sarker and A. S. Lee, "Using a positivist case research methodology to test a theory about IT-enabled business process redesign," in Proceedings of the International Conference on Information Systems, Helsinki, Finland, 1998, pp. 237-252.

[23] (12/02/2015). LoadUI - The Home of Load Testing | Open Source Load Testing Tool [Web testing, Server monitoring]. Available: http://www.loadui.org/

[24] C. Bezemer and A. Zaidman, "Challenges of reengineering into multi-tenant SaaS applications," Delft University of Technology, Software Engineering Research Group, 2010.

[25] L. Bass, P. Clements and R. Kazman, Software Architecture in Practice. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc, 2013.

[26] Amazon Web Services, Cloud Computing: Compute, Storage, Database Available: http://aws.amazon.com/