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

A.A. Adewojo, J.M. Bass and K.Hui
School of Computing Science and Digital Media
Robert Gordon University
Aberdeen, Scotland
a.a.adewojo@rgu.ac.uk

Abstract—Cloud computing provides scalable and reliable computing. Software organizations can benefit from this by moving their existing applications to the cloud. However, this migration is potentially complex. This is because migration of existing systems to the cloud is not a straightforward process and there is not enough skill and expertise in managing this. We therefore argue that cloud computing deployment patterns can assist these organizations to migrate their applications to the cloud. This research compares the format, structure and notations of previous object oriented design patterns with recent cloud computing design patterns. Firstly, the gaps in cloud computing design patterns are identified. Secondly, the multi-tenancy pattern, a cloud computing design pattern, was re-structured to conform with consistent and widely accepted design pattern format. Finally, the three variants of multi-tenancy pattern were implemented to illustrate their contrasting strengths and weaknesses by applying it to a case study application, which involved the migration of a database driven desktop application to the cloud. The result shows that: (i) there is an improvement in the structure and clarity of the cloud computing patterns; (ii) deployment patterns can positively improve the quality of a cloud deployed application; (iii) Information conveyed to software developers is enhanced.

Keywords—SaaS; Cloud Computing; Deployment Patterns; Cloud Migration

I. INTRODUCTION

Cloud computing, a paradigm shift in computing that has a potential of transforming the IT industry [1] as it provides scalable and ubiquitous computing services. It has gained much popularity in the computing world with many organizations moving their on-premise applications and IT infrastructure to the cloud [2]. However, this migration is potentially complex so cloud computing deployment patterns are proposed to assist these organizations.

We argue here that the development of software services for worldwide deployment presents a significant challenge for software engineers. However, cloud deployment patterns can significantly alleviate these challenges. It plays a major role in architectural restructuring and migration of on-premise software applications to the cloud [3]. Current cloud computing deployment patterns lack details which make them difficult to use. Furthermore, this lack of detail hampers deployment pattern selection.

II. RESEARCH METHOD

We have addressed this issue by comparing previous object oriented software design patterns [5] with recent cloud computing deployment patterns [3]. The format, structure, and graphical notations were the main basis of comparison. We used the object oriented design patterns as the baseline for comparison [5]. 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 catalogue. The benefits of the enhanced cloud deployment patterns are: that they are more systematically presented, easier to implement, with more logical headings and detailed pattern descriptions. The enhanced pattern will help software developers to make right design decisions as fast as possible [5]. This will also enable them to understand the design tradeoff between different approaches of applying the multi-tenancy pattern.

We have used a sequential mixed method approach [9] 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 patterns and an application case study. The theoretical aspect explored and compared existing design patterns with cloud deployment patterns available from the literature. The format, structure, graphical notations and applicability of these patterns were studied and compared in-line with best practices to develop an improved pattern format for the multi-tenancy patterns.

The case study strategy was used to gather and analyze information. This approach investigates an observable circumstance in its real-life context [6]. 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 [15] also we need evidences of a real-life setting of the case study [8].

In line with this we selected a small business software application. A business process management (BPM) system that runs on a desktop application intended to be migrated to the cloud. As a desktop application, it cannot be migrated to the cloud as it is, so a web based prototype system was built. 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.

We applied the newly developed cloud deployment pattern and tested it with a load tester called LoadUI and by different users. The results established that the new patterns implementation met the needs of the application depending on the user’s priority that is cost minimization, data privacy, and security.

III. MULTI-TENANCY AND CLOUD DEPLOYMENT PATTERNS

One of the five essential cloud properties is resource sharing [3]. The multi-tenancy architecture is one of the architectural patterns for SaaS that supports this property [10]. 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.

A well-developed cloud native application requires a combination of different design patterns to function properly. Software developers who use these patterns for software development requires a detailed and clear catalogue of applicable patterns. [3] presents a design pattern catalogue for cloud applications; one of the catalogued pattern captures three types of multi-tenancy patterns namely: shared component, tenant-isolated component, and dedicated component. These patterns applies to different components and application needs. The degree of isolation between tenants is the key determining factor for choosing the correct pattern.

However, we have observed gaps in the description of these patterns. In response to this we restructured the multi-tenancy pattern to conform with consistent and widely accepted design pattern format. These restructuring is based on what we have learnt from revised literatures and best practices in cataloguing a pattern. The restuctured 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 [3] but because they are not distinctly spelled out, users cannot follow them precisely.

IV. CASE STUDY AND OUTCOME

The multi-tenancy pattern was implemented in the data storage component of the case study application. To implement Shared component and Tenant-Isolated component, the data storage component was designed to allow multiple customers access a single instance of SimpleDB with either row or domain id as the customer’s identity. In the case of dedicated component, each customer had access to only their own SimpleDB instance.

The whole application was deployed on Amazon Web Services (AWS). AWS elastic beanstalk was used for the deployment. Sticky sessions were used to remember user sessions and a cloud watch was setup to monitor the health and utilization of the application. The application was thoroughly tested, to ascertain the usefulness of the variants of multi-tenancy patterns. The result shows the strength and weakness of these patterns and also that these patterns can be successfully implemented.

V. 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 have improved the clarity and organization of three multi-tenancy cloud deployment patterns in line with design pattern best practice. The pattern improvements are as follows: the structure has been elaborated and systematically articulated, a detailed description of how each pattern can be utilized has been provided, and underlying theories of the pattern properties are explained.

The three enhanced patterns are Shared Component, Tenant Isolated and Dedicated Component. 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 and security. The Dedicated Component pattern provides exclusive access to components that provide critical functionality. This exclusivity maximizes privacy and security, but at a higher cost in terms of resource overheads and amount of money paid per database connection. Also, the addition of new tenants is expensive in terms of performance. The Tenant-Isolated component pattern represents a compromised implementation between the Shared Component and Dedicated Component approaches. The tenant-isolated pattern involves some sharing of resources with intermediate levels of performance, security, privacy and resource overheads.

REFERENCES


