Cloud Migration Patterns: A Multi-Cloud Service Architecture Perspective

Pooyan Jamshidi[1], Claus Pahl[1], Samuel Chinenyeze[2], Xiaodong Liu[2]

[1] IC4 – the Irish Centre for Cloud Computing and Commerce, Dublin City University, Ireland
{pooyan.jamshidi,claus.pahl}@computing.dcu.ie
[2] Centre for Information & Software Systems, School of Computing, Edinburgh Napier University, UK
{S.Chinenyeze,X.Liu}@napier.ac.uk

Abstract. Many organizations migrate their on-premise software systems to the cloud. However, current coarse-grained cloud migration solutions have made a transparent migration of on-premise applications to the cloud a difficult, sometimes trial-and-error based endeavor. This paper suggests a catalogue of fine-grained service-based cloud architecture migration patterns that target multi-cloud settings and are specified with architectural notations. The proposed migration patterns are based on empirical evidence from a number of migration projects, best practices for cloud architectures and a systematic literature review of existing research. The pattern catalogue allows an organization to (1) select appropriate architecture migration patterns based on their objectives, (2) compose them to define a migration plan, and (3) extend them based on the identification of new patterns in new contexts.

Keywords: Cloud Architecture, Cloud Migration, Migration Pattern, Multi-Cloud.

1 Introduction

Cloud migration [1] benefits from the cloud promise of converting capital expenditure to operational cost [2]. Mixing cloud architecture with private data centers adds operational efficiency for workload bursts while legacy systems [3] on-premise still support core business services. Instead of re-architecting applications, they can be re-hosted from on-premise to possibly multiple cloud architectures, either private or public ones. We are concerned with the migration of legacy *on-premise* software to *multi-cloud* architectures. Multi-cloud deployment [4] is particularly effective in dealing with the following challenges:

- Users are widely distributed where they are located around multiple data centers.
- Country regulations limit options for storing data in specific data centers, e.g., EU.
- Circumstances where public clouds are used jointly with on-premises resources.
- Cloud-based application must be resilient to the loss of a single data center.

Current migration solutions are coarse-grained, making detailed planning difficult. For these cloud migration processes [1], a migration plan as a verifiable artefact is not considered. The plan is prepared at either a very broad strategic level with no technical value or very thorough and technical not suitable for non-technical stakeholders. Thus, the repeatabil-

ity of migration processes decreases. Architecture migration patterns can make this repeatable and transparent.

We address (i) how to reorganize multi-tier applications into disjoint groups of service components, such that (ii) each such group can be deployed separately in different platforms (i.e., cloud platforms, on-premise platform) while preserving and in most cases enhancing the desired properties of the application. We report on 9 fined-grained core and 6 variant cloud-specific architecture migration patterns, extracted based on empirical evidence from a number of migration projects [5], best practice for cloud architectures [4], [6] and a systematic literature review [1]. Our main contribution is a set of fine-grained service-oriented migration fragments that allows application developers and architects to plan the migration and communicate the plan and the decision with non-technical stakeholders.

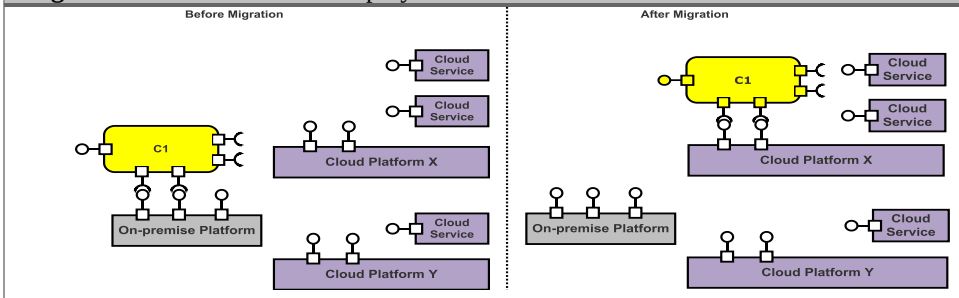
The patterns define architectural change in the application re-engineering and deployment setting, through which an application is gradually modernized and deployed in a multi-cloud. A migration plan is defined as a composition of selected patterns for specific situations.

Cloud migration methods define activities to plan, execute and evaluate migration [7]. To account for the situational context of applications, e.g., security, performance, availability needs, existing approaches [1] suggest a trade-off between flexibility and ease of migration using a fixed set of migration strategies. We propose an assembly-based approach based on our experience in situational method engineering [8] where a method is constructed from reusable method *fragments* and *chunks* [9]. This allows creating a migration plan from scratch by combining existing migration building blocks in the form of migration patterns. The usability of the approach is evaluated through a cloud migration case study at the end.

2 Background

We first introduce architecture migration patterns and the multi-cloud deployment setting.

Migration Pattern MP1: Re-deployment



Definition: An application (component) is re-deployed (moved, re-hosted) as-is on cloud platform(s)
Problem: Resource constraints limit scalability, Need to improve performance, Single point of failure, Reduce cost, Modernization
Solution: Re-deploy on cloud environments, make use of elastic resources, multiple cloud deployment for failover and scalability.
Benefits: Improved Backup and Failover, Coarse-grained scalability at application level, Simple coarse-grained re-deployment.
Challenges: Existing architecture constrains portability, deployment time/cost, scalability, integration may introduce complexity.

Migration Patterns. For each *migration pattern*, an architectural migration schema has to be defined. A migration pattern is represented by an architecture diagram of the service architecture deployment before and after migration, i.e. a migration pattern is a *transformation*

triple consisting of source and target architecture together with the applied pattern as the transformation specification. Each architecture is represented by well-defined architectural elements including services and connectors, deployment platforms (on-premise and cloud-based) and cloud services. The notation here is loosely aligned with UML component diagrams, with specific component types color-coded. A service component can either be atomic or contain internal components allowing for hierarchical decomposition. For example, the migration pattern MP1 consists of a coarse-grained component that consumes services of an on-premise deployment platform. These can be coordination services that orchestrate different components in larger compartments or simply configurable IaaS resources providing required operating system or storage features. After migration, this component, instead of using on-premise platforms, uses public cloud platform services offered. Thus, the application component is re-deployed as-is on a cloud platform. The current architecture is mirrored in the cloud, but can take advantage of virtualization to not only reduce operational expenditure, but also to create multiple instances of the application to improve scalability and failover without increasing capital expenditure. The key risk is that underlying architecture issues are not addressed. A monolithic legacy application in the cloud is still monolithic with limitations such as lack of scalability. Scalability is coarse-grained and cannot easily be achieved if, e.g., the architecture does not allow the database to be updated by multiple instances.

Multi-Cloud. In order to build highly scalable and reliable applications, a *multi-cloud deployment* is appropriate. Our objective is to provide architectural guidance for migrating cloud-based systems that run on multiple independent clouds. Multi-cloud denotes the usage of multiple, independent clouds by a client or a service. A multi-cloud environment is capable of distributing work to resources deployed across multiple clouds [10]. A multi-cloud is different from *federation* where, a set of cloud providers voluntarily interconnect their infrastructures to allow sharing of resources [10]. *Hybrid deployment* can be considered as a special case of multi-cloud where an application is deployed in both on-premise as well as cloud platforms. This deployment model is essential in cases where critical data needs to be kept in house in corporate data centers. We reviewed different application types and their requirements that necessitate multi-cloud deployment – see the supplementary materials here [11].

Note that we primarily target Platform-as-a-Service (PaaS) clouds that provide middleware services to host and manage application services. PaaS clouds like Microsoft Azure or Cloud Foundry generally provide mechanisms to support the re-architecting activities here.

3 Research Methodology

The first step to identify migration patterns was to identify the concerns of organizations moving on-premise applications to the cloud. We have identified four categories based on feedback from industry partners in our IC4 research centre [5]:

- **Availability**. Cloud environments typically guarantee a minimum availability.
- **Management**. Use runtime information to monitor and support on-the-fly changes.
- **Scalability**. Scale out to meet bursts in demand and scale in when demand decreases.
- **Resiliency**. Provide ability for systems to gracefully handle and recover from failure.

Focus Groups / Expert Interviews. We used focus groups to identify migration process concerns. The organizations involved were consultants for SME migration and larger multinationals – technology providers and systems integrators [5]. Through migration expert in-

terviews, we looked at common processes for migration towards cloud as a framework for more fine-grained patterns. These covered IaaS, PaaS and SaaS migration projects.

Systematic Literature Review (SLR). We recorded existing cloud design and architecture patterns [4][6]. A major role in this process played a SLR on cloud migration [1]. We detected shortcomings associated with these design patterns when we applied them in migration planning. The patterns were either limited to specific platforms [4] or fine-grained at a very technical level [6]. To redesign an on-premise application with these patterns, it requires deep knowledge of vendor-specific services as well as fair understanding of detailed design documents. Thus, a migration plan based on these patters cannot be communicated with non-technical stakeholders. Thus, we generalize the architectural elements of these cloud architectures with general concepts of software architecture, i.e., components, connector, on-premise/cloud platform, cloud service, cloud broker.

Empirical Analysis and Pattern Synthesis. We analyzed migration projects for a range of CRM and retail systems as well as PaaS platform services. We generalized emerging patterns, considering patterns retrieved from the SLR based on different architecture scenarios that satisfy the migration concerns. Coarse-grained on-premise applications are not agile enough to respond to variations in workload. In the cloud, the deployment of high-usage components can be optimized independently of low-usage ones. Re-architecting into independent components reduces dependencies and enables optimization for scalability and performance. However, challenges remain: (1) on-premise application modernized in isolation, not part of a consistent architecture, (2) modernization performed primarily for technical reasons resulting in sub-optimal response to business change, (3) architectures determined bottom-up from existing APIs and transactions may need re-evaluation for multi-clouds.

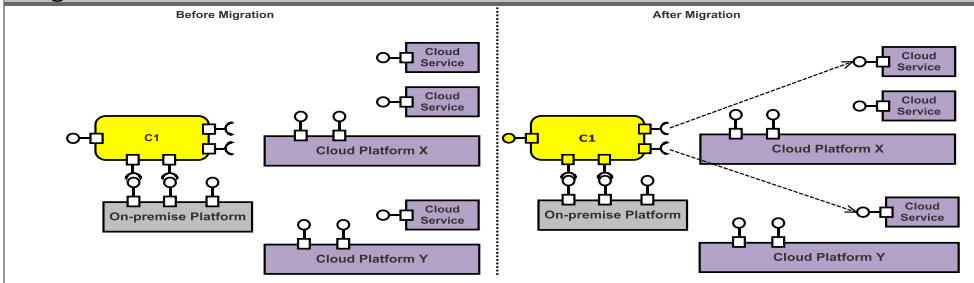
4 Cloud Architecture Migration Patterns

Some applications are integrated and support core business processes and services, but many of them support utility needs, are certainly non-core applications and are independent. The latter category may be obvious candidates for direct re-deployment. For the former integrated core ones, refactoring (re-architecting or redesigning) is more appropriate. Our migration patterns are sequences of architectural changes in the application deployment setting, through which the current application is gradually modernized.

To obtain unambiguous pattern descriptions and to ground pattern-based migration planning, we provide a template-based definition of migration patterns. This definition is based on the semantics of architectural schemas before and after migration. In some migration patterns, it may only be possible to deploy application components in a public cloud. However, for those patterns that consider re-architecting, the application can be deployed in hybrid public/private platforms. Due to space limitations, we do not describe all patterns fully, for more details refer to [11]. We use a template-based description of patterns. The usability of the patterns in migration planning will be shown through a method engineering process in Section 5 and through a case study in Section 6.

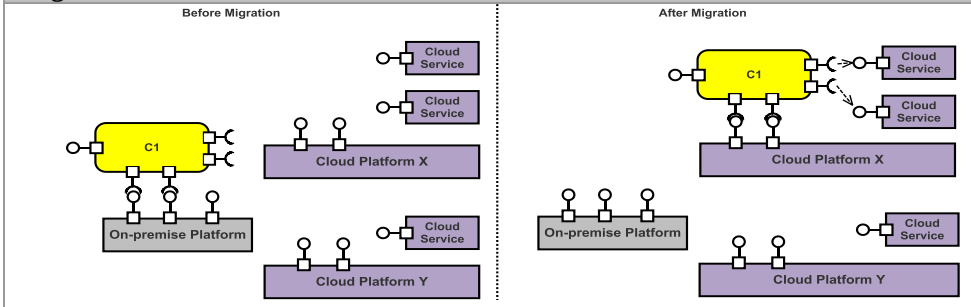
For space reasons, only the core patterns are presented. The patterns missing from this list are variants of some core patterns (which will be summarized afterwards). The core patterns highlight the different construction principles for the cloud architecture: re-deployment, cloudification, relocation, refactoring, rebinding, replacement and modernization.

Migration Pattern MP2: Cloudification



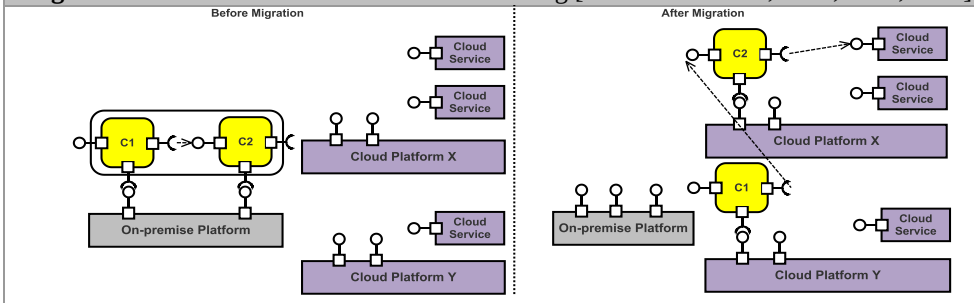
Definition: Application hosted on-premise as-is but use public cloud services for added capabilities instead of on-premise ones.
Problem: Need to improve reusability, extensibility, Avoid redundancy by consuming existing publicly accessible cloud services
Solution: Extend the on-premise application by integrating with existing public cloud services.
Benefits: Improved time to market.
Challenges: Integration may introduce greater complexity.

Migration Pattern MP3: Relocation [see variant MP4]



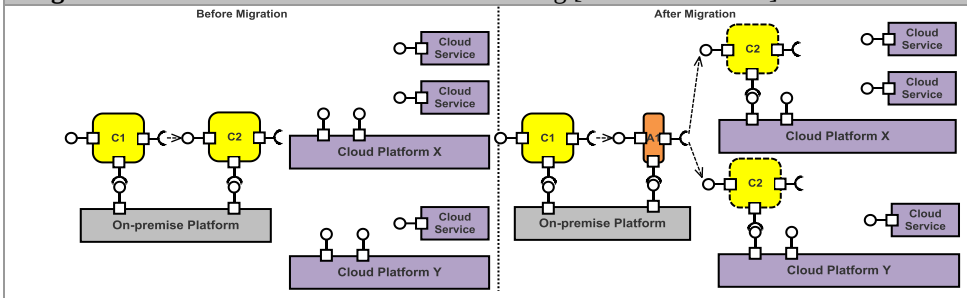
Definition: Component re-deployed (relocated) on cloud platform is cloudified but without evolution in the application architecture.
Problem: Enhance performance without significant architecture change, without capital expenditure for on-premise hardware.
Solution: Use cloud services to improve throughput by leveraging Queues, Database partitioning/sharding, NoSQL, Cache
Benefits: As component re-hosting in cloud and optimized performance.
Challenges: The type of application requests changes over time for example proportion of read only calls reduces, Cloud provider does not provide the necessary services to wrap the optimizations around the application without re-architecting.

Migration Pattern MP5: Multi-Cloud Refactoring [see variants MP6, MP7, MP8, MP9]



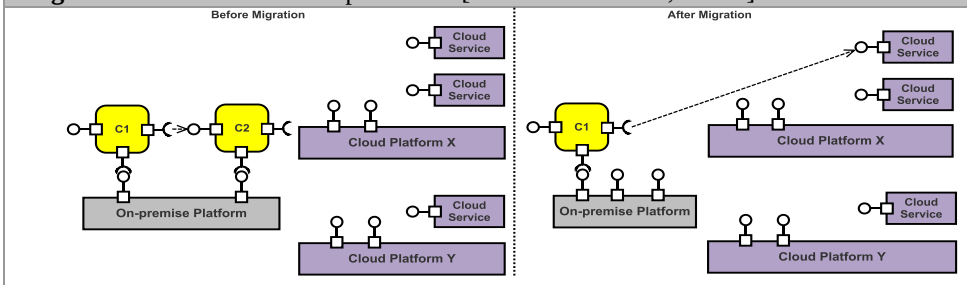
Definition: An on-premise application is re-architected for deployment on cloud platform to provide better QoS.
Problem: Coarse-grained applications are not agile enough to respond to requirement changes or variations in workload, and cannot take full advantage of the performance improvements that can be offered by cloud platforms.
Solution: Application re-architected into fine-grained components; deployment of high-usage comp. optimized independently of low-usage ones; parallel design for better throughput to multi-cloud platforms; components as independent integrity units.
Benefits: Optimal scalability/performance, range of multi-cloud deployment options, agility to respond to business/IT change.
Challenges: On-premise application is modernized in isolation; Modernization is performed primarily for technical reasons, Component architecture is only determined bottom-up may need to be re-evaluated because of multi-cloud environment.

Migration Pattern MP10: Multi-Cloud Rebinding [see variant MP11]



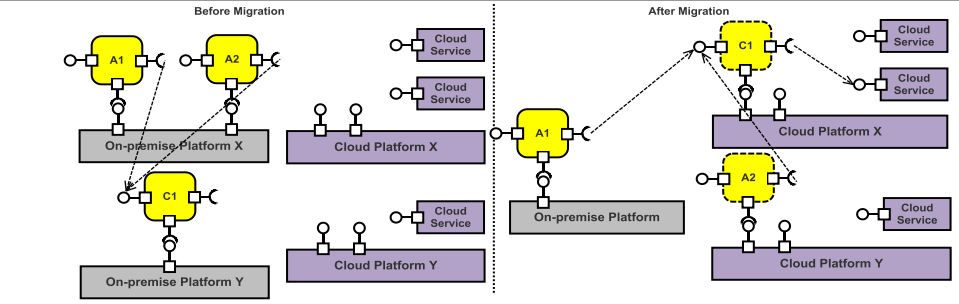
Definition: A re-architected application is deployed partially on multiple cloud environments and enables the application to continue to function using secondary deployment when there is a failure with the primary platform.
Problem: Failure such as a bug or configuration error that impacts cloud services may cause a failure to a cloud platform.
Solution: Architecture for resilient systems (routes users to closest data center) used for failover: monitor services, if unavailable, traffic is routed to healthy instances. On-premise adapter (bus or load balancer) provides integration of components
Benefits: As unhealthy services become healthy again, traffic can be delivered, returning system responsiveness to maximum.

Migration Pattern MP12: Replacement [see variants MP13, MP14]



Definition: Individual capabilities in a re-architected solution are re-provisioned rather than re-engineered.
Problem: Some existing components provided by current application are not the best alternative to meet business requirements.
Solution: Analyze and identify capabilities to be replaced by cloud services (capabilities that can be supported by re-architected system), identify alternative cloud services with benefit over re-engineering of current capability to replace components
Benefits: The solution is improved though best-in-class cloud services, Re-engineering costs and effort are saved.
Challenges: Cloud services presume specific communication protocol that make the replacement a challenging tasks.

Migration Pattern MP15: Multi-Application Modernization



Definition: Different on-premise applications A1/A2, C1 are re-architected as a portfolio and deployed on cloud environment.
Problem: The re-architecting of on-premise applications in isolation does not remove inconsistencies in data or duplicated functionalities, nor reduce the cost of their combined operation or maintenance.
Solution: Current applications are analyzed jointly to identify opportunities for consolidation/sharing. Separation of service and solution architecture enables the identification of components (capabilities) that are shared by more than one solution.
Benefits: Consistent information / rules in shared components, Reduced operation / maintenance costs for shared components,
Challenges: Lack of business commitment to shared capabilities.

Variants for the following core patterns can be identified [11]:

- MP3 **Relocation**: MP4 (relocation for multi-clouds)
- MP5 **Multi-Cloud Refactoring**: MP6 (hybrid refactoring), MP7 (hybrid refactoring with on-premise adaptation), MP8 (hybrid refactoring with cloud adaptation), MP9 (hybrid refactoring with hybrid adaptation)
- MP10 **Multi-Cloud Rebinding**: MP11 (rebinding with cloud brokerage)
- MP12 **Replacement**: MP13 (replacement with on-premise adaptation), MP14 (replacement with cloud adaptation)

Further variants can be added, but we will show the sufficient completeness of the given set to model common PaaS migration scenarios in the use case evaluation.

5 Assembly-based Situational Architecture Migration

To enable migration planning as a tractable process, appropriate building blocks have to be selected and combined. Migration patterns embed desirable principles for the target architectural deployment. Migration patterns represent fine-grained migration activities to be combined into a migration plan, ensuring that combined patterns do not violate pattern properties. For example, a pattern for the replacement of an on-premise component can be combined with a pattern for refactoring. This ensures that an architecture migration plan can be created incrementally. Figure 1 shows this pattern composition process. The patterns form a sequence of activities by which an application is gradually migrated and refined.

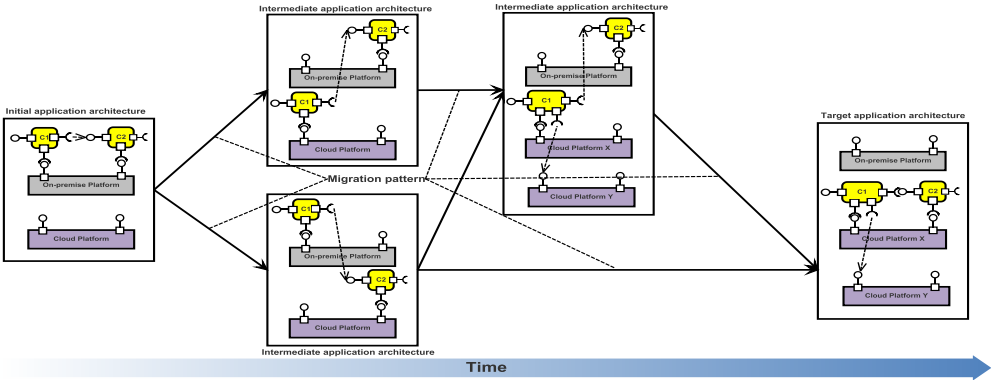


Figure 1. Migration Transition Graph.

A migration transition graph provides a generic migration plan based on situations and possible migration patterns. The graph nodes are current architectural configurations and edges are migration patterns. The directed nature of the graph shows sequencing of patterns. Since multiple edges can enter a node, the model is able to represent many candidate plans. There are initial and target architectures, but also intermediate application architectures. Migration plans are triples <*source config*, *pattern*, *target config*> that correspond to a migration step to achieve the target configuration from a specific configuration following a particular pattern. Note that one path from the source configuration (current on-premise application architecture) to the target (multi-cloud application architecture) will be chosen.

Table 1 shows the patterns base as a mapping of migration patterns and concerns for which they are suitable. These patterns can be used to form a plan (see Figure 1). This mapping is used to narrow down the related patterns and we can select the final pattern by comparing the situation through the "benefit" part in the pattern template. The selected patterns can be integrated based on the presence/absence of overlaps between patterns. The flexibility of this approach is restricted only by the set of available migration patterns. The patterns can be extended over time, e.g., by integrating a new solution to new problems. For a more detailed description of the assembly-based approach, see the supplementary material [11].

Table 1. Cloud Migration Pattern Selection.

Objective	MP1	MP2	MP3	MP4	MP5	MP6	MP7	MP8	MP9	MP10	MP11	MP12	MP13	MP14	MP15
Time to market	☺	--	✖	✖	✖	--	--	--	--	✖	✖	☺	☺	☺	☺
New capabilities	✖	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	--	--	--	--
Reduce operational cost	☺	☺	--	--	✖	--	--	--	--	✖	✖	☺	☺	☺	☺
Leverage investments	☺	☺	--	--	--	☺	☺	☺	☺	☺	☺	✖	✖	✖	☺
Free up on premise resources	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺	☺
Scalability	✖	--	--	--	☺	☺	☺	☺	☺	☺	☺	--	--	--	--
Operational efficiency	☺	--	--	--	☺	--	--	--	--	☺	☺	☺	☺	☺	--

6 Case Study and Validation

The *usability* of the migration patterns shall be evaluated through a case study. We use a sample migration project based on our work with Microsoft Azure as a PaaS cloud for illustration and validation. This project acts as a representative for a range of migrations we examined (and for the latter two categories also implemented). These include several CRM systems (e.g., larger configurations based on commercial products), online retail solutions and services utilizing cloud storage solutions. Usability refers to the suitability of the pattern set to provide options and facilitate staged migration plans. Thus, we need to demonstrate the utility of all patterns, but also that the set is sufficiently complete to model a range of cases.

Context. A financial services company decides to migrate in-house applications to the cloud. It uses Microsoft technologies, but it also has legacy systems deployed on UNIX. Some applications have external ports, while others are exclusively for internal use. The importance of the applications ranges from marginal to critical. A significant portion of the IT budget is spent on maintaining applications with marginal importance.

Challenges. New applications take long for deployment, causing problems with adapting to changes. For any application, requirements must be analyzed, procurement processes must be initiated and networks must be configured. The infrastructure is used inefficiently. The majority of servers are underutilized. It is difficult to deploy new applications with the required SLA to the existing hardware. Applications in a public cloud platform can take advantage of economies of scale and have automated processes for managing.

Concerns. An objective is to improve the user experience. Some applications vary in usage (e.g., used once every two weeks, like salary-wages, but rarely at other times). They would benefit from the cloud-based increased responsiveness during peak times. A second objective is to expand ways to access applications. Applications located in the public cloud are available over the Internet, but authentication concerns exist. A third goal is portability,

i.e., it can be moved between a cloud and a private data center without modification to application code or operations. Furthermore, a tractable migration plan is essential.

Application. The migration starts with the *Expense* application. This allows employees to submit and process expenses and request reimbursements. Employees can tolerate occasional hours of downtime, but prolonged unavailability is not acceptable. Most employees submit expenses within the last days before the end of each month, causing high demand peaks. The infrastructure for the application is scaled for average use only. The application is deployed on-premise. It requires high volume storage because most stored receipts are scanned.

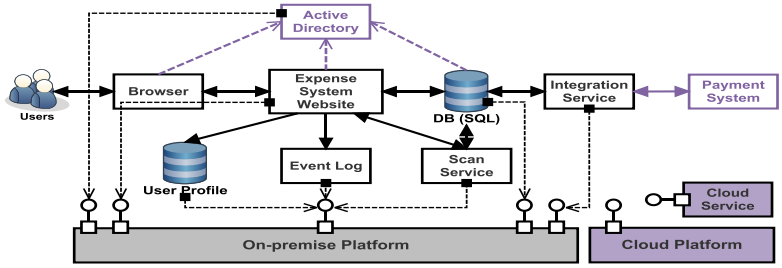


Figure 2. Application Architecture before Migration to the Cloud.

Expense is an ASP.NET application. It uses Windows authentication for security. To store user preferences, it relies on ASP.NET profile providers. Exceptions and logs are implemented with Enterprise Library's Exception Handling Application Block and Logging Application Block. It uses Directory Service APIs to query data. It stores information on SQL Server. Receipts are stored in a file system. The architecture is illustrated in Figure 2.

The migration plan. The existing servers, networks, and associated systems such as power supply and cooling are managed by the company. We present a set of migration steps and decisions made to reach a tractable migration plan by adopting the presented patterns.

Step1. Move the application to a cloud platform unchanged providing infrastructure reliability and availability. Management costs for running the hosted operating system and OS licenses must be considered, but development costs can be reduced as applications do not need to be refactored. Migration patterns **MP1, MP3, MP4** suit, of which **MP1** was selected, because only copy-as-is to the cloud without need for environmental services required.

Step2. An alternative is to adapt Expense to run as hosted on a platform by an external partner. This would avoid costs of porting the application to a different system and reduces management cost. There is work involved in refactoring the application to run in cloud-hosted roles. **MP5-MP11** can be selected. Since the user profiles were to be kept on-premise. Pattern **MP6** was selected because there was no need for any interface adaptation (as in MP7-MP9) or multi-cloud deployment (as in MP10 and MP11).

Step3. Abandon the own payment application and rent a typically more generic cloud service, which needs to be evaluated regarding security, performance, and usability. **MP12, MP13, MP14** suit, but a need to integrate Expense with a Payment service, favors **MP13**.

Step4. For an external hosting decision, data storage facilities offered by cloud platforms are required. Expense requires a relational database system and NoSQL storage to store receipt images. **MP12** was selected as Azure SQL and Storage offerings meet requirements.

Step5. Remote applications need to be integrated with other cloud services and on-premise for data access and monitoring. A systems operation or authentication tool could be used for monitoring, requiring remote services to be integrated. **MP7, MP8, MP9, MP12, MP13, MP14** can be selected. Due to a need for some adaptations, **MP14** was selected.

Step6. Although only employees use Expense, the payment sub-system also used by other applications must always be available. **MP10, MP11** can be selected, but if the development of failover rebinding is to be avoided, a broker as in **MP11** is utilized (e.g., to deploy the payment system on Amazon and keep a mirror on Azure to route requests in case of failure).

Step7. Value-added services from the cloud such as caching can maximize performance when retrieving data or can cache output, session state and profile information**MP3** was selected to accommodate these environmental services of the cloud provider.

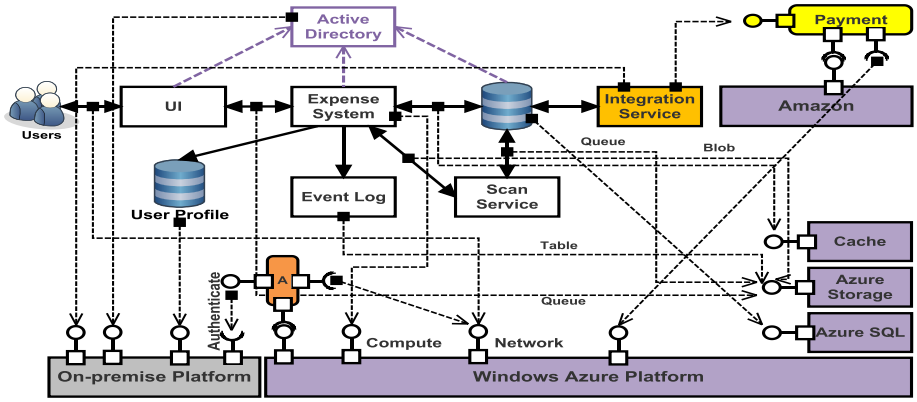


Figure 3. Application Architecture after Migration to the Cloud.

Migration path. A possible migration path is presented below. The result is the architecture in Figure 3. The migration steps are illustratively represented in [11]. Depending on the concerns of an organization, different combinations of hosting, data store and cloud services are possible. For example, MP1 step 1 follows a gradual migration by adopting the hosting approach, but uses SQL Server hosted in a VM before moving to an Azure SQL Database. Using MP3 instead would take advantage of storage capabilities (table/blob storage) and caching instead of relational databases to improve performance early rather than late.

Migration Step	Requirement	Chosen Patterns
1	Minimal code changes to application and familiarity with platform	**MP1**
2	Granular control of resource usage and opportunity for auto-scaling	**MP6**
3	Lower cost although some limitations on feature availability	**MP13**
4	Replacing on-premise storage with cloud offerings	**MP12**
5	Integration with cloud utility services	**MP14**
6	Highly available service replacement	**MP11**
7	Better user experience, improved efficiency, and load leveling	**MP3**

Discussion. For the migration plan we had different requirements, but were able to find a satisfactory patterns solution. Thus, the requirement satisfaction in this case is achieved and

met by the proposed patterns [8]. Technically, we can only conclude that the migration patterns are complete and useful for all situations arising from the use case. However, we have analyzed and considered other migration, e.g., different IaaS/PaaS/SaaS migration processes [5]. The storage refactoring options relating to relational, table and blob storage, particularly addressed by patterns MP1 and MP3, are specifically addressed in [12]. This paper highlights the re-architecting options that advanced PaaS clouds offer, but also shows that while in this paper quality concerns such as scalability or availability are covered, their quantification and a trade-off analysis with cost aspects is not covered. Often, which specific paths are chosen is driven by more in-depth quality concerns. Our solution focuses on functional architecture aspects and only includes quality and cost concerns qualitatively.

7 Related Work

We conducted a review [1] aiming to identify, taxonomically classify, and systematically compare the existing research focused on planning, executing, and validating migration of legacy systems towards cloud-based software based on earlier architecture evolution work [13]. We found a lack of repeatable and verifiable practices as one of the key reasons that cloud migration is not a fully mature domain. In the context of the Cloud-RMM migration framework [1], our work here can be categorized as a contribution to migration planning.

Cloud migration approaches range from *decision making* to *enabling legacy software migration* with approaches reporting *best practice*, *experience and lessons learned* in between. Decision making for cloud adoption (e.g., [14]) is inherently complex and influenced by multiple factors such as cost and benefits through migration [15]. In contrast, some approaches enable the actual migration of legacy software in terms of procedures and model transformation (e.g., [16]). Some other work reports on lessons learned and best practices [17] – providing empirical evidence for migration research.

A number of migration strategies and best practices have been suggested in terms of patterns in [18][19][20]. These are rather informal and do not consider a multi-cloud setting. The objective there was mainly classification of existing best practice into migration strategies. The key advantage and novelty of our work, more than a set of patterns, is the notion of assembly-based situational migration at the architecture level, specifically towards pattern-based migration planning for multi-cloud deployment. It enhances the state-of-the-art by a tractable planning approach based on composable patterns.

8 Conclusion and Outlook

We identified cloud migration patterns, which in combination allow planning the migration of applications for multiple cloud platform deployment. The introduction of migration patterns complements existing migration practices and allows for an engineering approach towards constructing and evaluating the migration plan. The migration patterns are reusable and composable architectural change patterns that we see as building blocks of an overall migration process, reflected through a migration plan as a sequence of pattern applications.

Future work will include the development of a migration pattern repository as a tool that facilitates migration planning as well as application of the patterns to new domains and migration cases. To demonstrate the usability and completeness of the patterns beyond busi-

ness-oriented SaaS and standard PaaS-level services such as storage, currently we are in the process of evaluating others for migration planning in three cases with our industry partners. We also plan to formally represent the relations between migration patterns in order to form a pattern map and work toward a pattern language for migration practices.

Acknowledgments. The research work described in this paper was supported by the Irish Centre for Cloud Computing and Commerce (an Irish national Technology Centre funded by Enterprise Ireland and the Irish Industrial Development Authority) and the Royal Irish Academy/Royal Society International Cost Share Grant IE131105.

9 References

[1] P. Jamshidi, A. Ahmad, and C. Pahl, "Cloud Migration Research: A Systematic Review," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 142–157, 2013.
[2] M. Armbrust, "Above the clouds: A Berkeley view of cloud computing," 2009.
[3] R. Khadka, A. Saeidi, and A. Idu, "Legacy to SOA Evolution: A Systematic Literature Review," in *Migrating Legacy Applications*, 2012.
[4] B. Wilder, *Cloud Architecture Patterns*. Oreilly, 2012.
[5] C. Pahl, H. Xiong, and R. Walshe, "A Comparison of On-premise to Cloud Migration Approaches," *European Conference on Service and Cloud Computing ESOCC'13*, 2013.
[6] C. Fehling, et al., *Cloud Computing Patterns*. Vienna: Springer Vienna, 2014.
[7] V. Tran, J. Keung, A. Liu, and A. Fekete, "Application migration to cloud," *SECLOUD '11*.
[8] M. F. Gholami, M. Sharifi, and P. Jamshidi, "Enhancing the OPEN Process Framework with service-oriented method fragments," *Softw. Syst. Model.*, 2011.
[9] I. Mirbel and J. Ralyté, "Situational method engineering: combining assembly-based and roadmap-driven approaches," *Requir. Eng.*, 2006.
[10] N. Grozev and R. Buyya, "Inter-Cloud architectures and application brokering: taxonomy and survey," *Softw. Pract. Exp.*, vol. 44, no. 3, pp. 369–390, Mar. 2014.
[11] P. Jamshidi and C. Pahl, "Cloud Migration Patterns - Supplementary Materials," 2014. [Online]. Available: http://www.computing.dcu.ie/~pjamshidi/Materials/CMP.html.
[12] E. Gamma, et al., *Design patterns: elements of reusable object-oriented software*. 1994.
[13] P. Jamshidi, M. Ghafari, A. Ahmad, and C. Pahl, "A framework for classifying and comparing architecture-centric software evolution research," *17th European Conference on Software Maintenance and Reengineering CSMR'2013*. IEEE. pp. 305-314. 2013.
[14] S. Frey, W. Hasselbring, and B. Schnoor, "Automatic conformance checking for migrating software systems to cloud infrastructures and platforms," *J. Softw. Evol. Process*, 2013.
[15] S. C. Misra, "Identification of a company's suitability for the adoption of cloud computing and modelling its corresponding return on investment," *Math. Comput. Model.*, 2011.
[16] S. Frey and W. Hasselbring, "The cloudmig approach: Model-based migration of software systems to cloud-optimized applications," *Int. J. Adv. Softw.*, 2011.
[17] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch, "How to adapt applications for the Cloud environment," *Computing*, vol. 95, no. 6, pp. 493–535, Dec. 2012.
[18] L. Wilkes, "Application Migration Patterns for the Service Oriented Cloud," 2011. [Online]. Available: http://everware-cbdi.com/ampsoc.
[19] N. C. Mendonca, "Architectural Options for Cloud Migration," *Computer*, 2014.
[20] C. Fehling, et al., "Service Migration Patterns -- Decision Support and Best Practices for the Migration of Existing Service-Based Applications to Cloud Environments," ICSOC, 2013.