A design study on the migration of an on-premise software application to the cloud

Bachelor of Science Thesis in Software Engineering and Management

NIMA FARAHBAKHSH-FARD
JOHAN RINGSTRÖM
PIERRE LEVEAU

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

A design study on the migration of an on-premise software application to the cloud

NIMA FARAHBAKHSH-FARD
JOHAN RINGSTRÖM
PIERRE LEVEAU

Supervisor: ROGARDT HELDAL

Examiner: JAN SCHRÖDER

University of Gothenburg

Chalmers University of Technology
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
UNIVERSITY OF GOTHENBURG
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2018

Abstract—**This research paper aims to highlight two things; the difference in development time when migrating an on-premise application to the cloud with two different migration strategies and the performance of the application after migration with the two strategies. This design study features an experience report where the two migration strategies are evaluated in the aspect of development time. The experience report serves to answer research question 1; "What are the differences between the rehosting (lift-and-shift) strategy and the refactoring (making it cloud native) strategy in the aspect of development time?". Research question 2; "How will the performance of the application differ after being migrated to the cloud using the rehosting (lift-and-shift) strategy and the refactoring (to cloud native) strategy?" was answered by measuring execution time for the mobile applications methods. The study shows that although the development time for refactoring was longer than for rehosting, it did not differ as much as expected for inexperienced cloud platform users who are migrating a small 3-tier application. The refactored application performed better than the rehosted application in the method execution time tests.**

FF. INTRODUCTION

In recent years the migration of on-premises application to the cloud has become an important topic for researchers and industry [1]. There are several advantages suggested as to why to deploy your software on to the cloud, where cost savings is the most stressed reason. This comes from the fact that the cloud architecture provides the applications the ability to scale when needed and that the software companies can buy as much storages resources and computational power as they need when needed [2]. This has highlighted the importance of migrating on-premise software applications to the cloud to get the benefits of cloud computing.

Recent systematic literature reviews have shown that there has been a growth of maturity of the cloud migration research field, but that there is a need for more results of cloud migration evaluation [3]. When studying literature on cloud migration it is very hard to find studies that deal with the migration of smaller on -premises application, which is what we are going to focus on in this study.

There are several ways to migrate your application to the cloud. David S. Linthicum suggest seven paths to go when considering migrating an application to the cloud. The seven R's of migration paths are: replace, reuse, refactor, replatform, rehost, retain and retire [4]. Which path to take when migrating an application to the cloud is one of the hardest parts of application migration and although a best practice is starting to emerge, the industry is still struggling to choose the right path [5].

This study will be a part of this cloud migration evaluation domain and its purpose is to investigate two different strategies of migrating a small legacy 3-tier mobile application to the cloud and compare advantages and disadvantages of the two strategies within the specific scenario. The study is motivated by the need to understand what advantages and disadvantages there are, with different paths when migrating an application to the cloud.

This study focuses on the rehosting and the refactoring strategy to migration. Rehosting, or the "Lift and shift" strategy, describes a scenario where an application is migrated to the cloud with as little code change as possible [12], whereas the so called refactoring migration strategy describes a scenario where you redesign the application to comply with the cloud platform infrastructure to use the cloud in a more efficient way [12], that is to say, to make it cloud native. In our case we will refactor a small legacy 3-tier mobile application.

This approach of comparing the rehosting and refactoring strategies will add knowledge to the cloud migration research domain that helps practitioners to decide what migration strategies to take when migrating an application to the cloud. The reason for evaluating these two approaches is that they are the most commonly used approaches when migrating to the cloud. We believe that the standardized nature of these approaches makes them the most interesting to conduct research about.

Development time will be compared of the two migration strategies and performance testing will be conducted on the mobile application after migrating with the two different strategies. This study does not aim to be generalizable but strives to complement previous research with an investigation of a migration endeavor within specific circumstances.

II. Research Questions

RQ1: What are the differences between the rehosting (lift-and-shift) strategy and the refactoring (to cloud native) strategy in the aspect of development time?

RQ2: How will the performance of the application differ after being migrated to the cloud using the rehosting (lift-and-shift) strategy and the refactoring (to cloud native) strategy?

Based on previous research, our assumptions for these research questions are that while a rehosting ("lift and shift") strategy will save you development time, it will not provide us with the performance benefits that refactoring to a cloud native architecture will. As mentioned previously, regarding the lack of study on this field, these assumptions may not be accurate when migrating smaller applications.

III. Literature Review

Definitions and background

We have in this paper already used the term cloud several times as if it has a well-defined meaning but this is not the case [14]. The birth of cloud computing could be argued to be set to 2006 when Amazon Web services launched the first general purpose public cloud service (Simple Storage Service, S3) and from this date many cloud providers have emerged [14]. AWS, Google cloud and Azure are the biggest providers today. All these cloud providers are defining what the cloud really is [14].

National Institute of Standards and Technology (NIST) defines the cloud as "...a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction. This cloud model is composed of five essential characteristics, three service models, and four deployment models.". [15] With the scope of this study this definition holds fairly well. One obvious flaw is that it only includes three services, Software as a service(SaaS), Platform as a service(PaaS) and Infrastructure as a service (IaaS). Now you can easily add several more to that list, Function as service(FaaS), Backend as a Service(BaaS) being two examples [24].

The cloud provider we will migrate our on-premise application to is the Google Cloud (GC), with its cloud management interface, Google Cloud Platform (GCP), which gives the developers remote access to its resources [16]. GCP is one of the most important and fastest growing cloud provider today and has a range of big companies, such as HTC, Coca-Cola and Spotify [16]. Services they provide are numerous and divided into different areas such as, Compute Engine, Databases and storage, Cloud AI, IoT, Big Data and many more. In our case, when migrating the on-premise application to the cloud we have used:

Rehosting:

For this approach we have used google cloud platform as a Infrastructure as a service [24] (IaaS) and the cloud SQL as storage.

- Compute Engine provides virtual machines which run on Google's data centers and fiber network and it supports scaling from one instance to global cloud computing [17].
- Cloud SQL is a database service with which you can fully manage a relational MySQL database on the cloud [18].

Refactoring:

For this approach we used the Firebase platform, which is integrated with the Google Cloud Platform (GCP) but more directed at mobile and web applications, to create a cloud native backend (Back end as a Service, BaaS [24]) for our mobile application. Firebase libraries were used in the android code to create connections to the backend, making it possible to connect the android application directly to the database without using any servers.

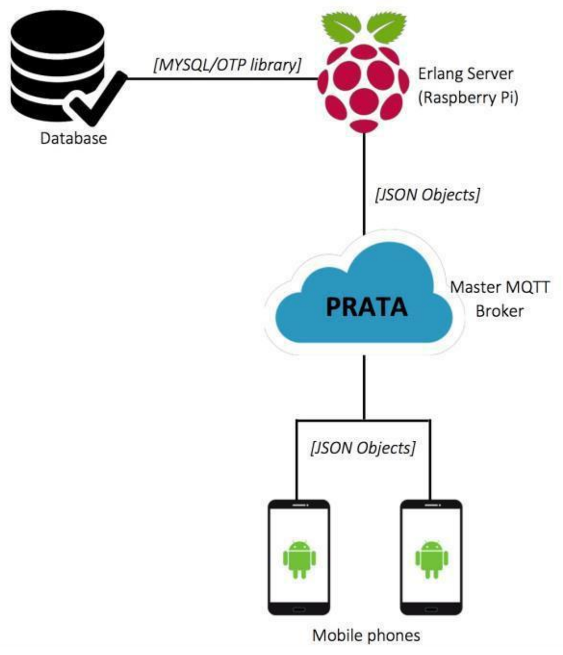
The following Firebase SDK's were used

- Firebase Realtime Database SDK, the data is stored as JSON and synchronized in realtime to all connected clients. [20]
- Firebase Authentication SDK, adds a unique user id to each user and their data can only be accessed by that user id. The users can however share their data with other users if they want to. [21]

This BaaS solution is one of several ways to go when refactoring a legacy application to be cloud native. This decision was made because the Realtime Database synchronizes with all clients connected and the fact that it was JSON based. These features are very compatible with the legacy application and was able to fulfill all the requirements of the legacy application. Another way of refactoring an application to become cloud native, is to split the legacy monolithic application into small, so called microservices [8]. This approach is generally more suited with bigger applications [8] where it makes more sense to divide the functionality into smaller loosely coupled services. In our case refactoring to microservices would have become a more time consuming and complex endeavor and could also have influenced the performance in a negative way, because it is commonly observed that microservices can introduce additional lag. [23]

The on-premise application (*Figure 1.*), that has been migrated to the cloud, consists of 4 components; an Android mobile application (1859 lines of code), an Mqtt broker, an Erlang server (573 lines of code) and a MySQL database. The premise of the application is to create grocery lists and share them with friends or family. The Mqtt broker provides a publish and subscribe functionality to the application which allows users to send and receive grocery lists between each other. (The lines of code in the Grocode application only pertains to the java classes of the application and excludes xml files, gradle builds, resources, etc.) [19].

Figure 1: On-premise application.



Relevant literature

In the previously mentioned systematic literature review by Jamshidi he proposed a framework for characterizing cloud migration studies. He identifies four major themes:

- Maturity level, which is concerned with the methodology.
- Migration characterization, which is concerned with the intention of the migration, the migration task, migration type and the means of migrations.
- Migration support, which is concerned with tool support, automation and support.
- Constraints, which is concerned with architectural style, target platform and cloud stack layer.

All these characterization parameters can describe different variations and approaches when studying cloud migration. Our study, with its motivation to contribute to the knowledge body of how to choose the right migration strategy is most closely related to studies that focus on migration types (migration strategy) and/or means of migration and especially experience and lessons learned studies.

There is plenty of existing literature on the problem of knowing how to migrate a legacy application to a cloud platform [5][6][7] and plenty of articles with regards to how and if the architecture should be updated [8][9]. These books and articles also address the issue of migration to the cloud. Most of them look at either the practical way of doing this or how to adjust the architecture when migrating. We are doing much of the same things but also implementing aspects such as looking at the performance of the application after migrating using our two different approaches.

Solution approaches

As stated earlier the problem of migrating to the cloud has been a "hot topic" in recent years. There are several blog posts and articles [4] online where people propose the solution of using a cloud native structure, for the software application. The general idea is that a cloud native structure is an efficient way of using the cloud for each business' individual agenda and that it provides a big boost in scalability. To achieve a cloud native architecture, we must refactor our application.

Besides refactoring, the other strategy that we are evaluating is rehosting (lift-and- shift), where a legacy application is moved to the cloud without redesigning the application. This does not however guarantee that the application will be able to take full advantage of the cloud.

We also discussed, but later discarded other options when migrating to the cloud such as replatforming. With replatform migrations, the core architecture of the application is kept but some parts, such as the database for example, can be moved to the cloud to achieve desired benefits without spending the resources that refactoring requires [10].

IV. Research Methodology

Our study is a design research paper where we have migrated a small legacy 3-tier application to the cloud using the two migration strategies; rehosting and refactoring. The application used is an android mobile application called "GroCode", developed by seven students from the Software Engineering and Management program in Gothenburg.

Motivation for our chosen research methodology

In the article "Design science in information system research" by Hevner et al. [11] he discusses what constitutes design science research. They identify seven guidelines for a design science approach. The first guideline is Design as an Artifact, the second is Problem Relevance and the fourth is Research Contributions. These guidelines can be said to answer the question when it is appropriate to do a design science study.

- Guideline 1: Our research will produce two different cloud migration methods as artifacts.
- Guideline 2: Our artefacts are technology-based and, as described in the introduction, our research on cloud migration of a small legacy three tier application is important and relevant for practitioners in the industry.
- Guideline 4: This study's design artefacts aims to create two methods that will contribute to the cloud

migration, in the sense that it will give a scientific evaluation of the two methods and help practitioners to make well educated choices when migrating a small mobile application to the cloud.

Hevner et al. also stresses the importance of the novelty of the artefact designed. [11] As already discussed these methods cannot be said to be new, they are well described practices when migrating legacy applications to the cloud. The novelty of our artefacts lies in the specific circumstances of how they are developed. That is, as previously stated, that there is little research done on the area of cloud migration methods on smaller applications.

Another method to answer the research questions could have been to conduct a case study, that would have focused on the migration process in a real-world context. Due to a limited time frame this was not possible and would also have been harder to exactly estimate the time spend on the migration process, due to a less controlled environment and it would have been hard to find a company that migrated the exact same application to the cloud using two different migration methods. The fact that the context is to some extent controlled, is also a drawback for this design sciences study. Instead of a real-world context, the migration process has been conducted by the researchers themselves which can be said to be a validity threat as discussed in the validity threat section and can also to some extent be of less relevance to the industry.

Another limitation with design science is that it only tries to establish how well an artifact works not try to theorize about why it works [11]

The fifth guideline, Research rigor, for conducting a design science, addresses how the research is conducted [11]. In this study, this puts the focus on how the artefacts. the migration methods, was done and later how the evaluation of them was conducted.

Data collection

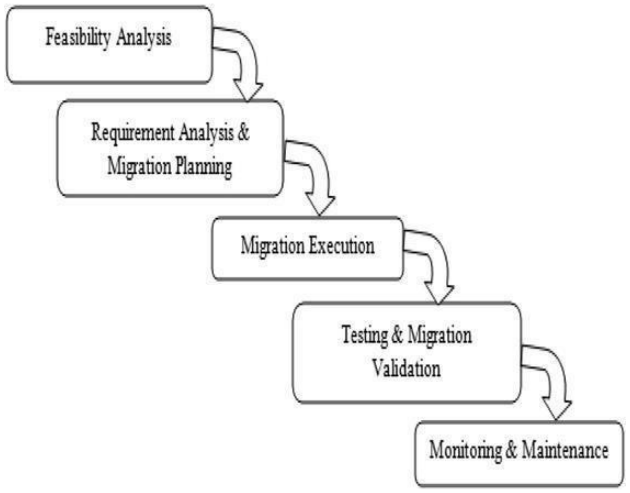
To answer the research questions the researchers, three third year students from the Software Engineering and Management program in Gothenburg with little or no experience of cloud migration, have conducted two migrations, following two different strategies, of the GroCode on-premise application, to the cloud. The reason for these two migrations were to be able to evaluate the actual migration strategies in regard to time (RQ1) and the deployed applications in regard to performance (RQ2).

The data has been collected in two phases; first, an experience report [13] of the migration processes, where the time spent on the migrations is logged. Second, performance testing in the form of execution time of methods in the two applications when deployed on the cloud.

The experience report consists of a log where the researcher documented the process of the two migrations. In the log the researchers have documented the challenges and unexpected obstacles the two strategies introduced during the migration processes and these are presented in an upcoming section of this paper. The main reason for this experience report was to, in a structured way, log the time we spent on the migration process, and in the end to be able to elicit and compare the effort made when migrating the application to the cloud.

There are many different tools you can use to track time. The most of them are developed to assist people that work and charge their clients by the hour. We looked at different alternatives to track the time we spent on the migration processes. In the end we ended up tracking time by logging at what time of the day we started working and at what time we stopped working. The important thing in this case was to know what to and not to track and be very mindful not to log time spent on tasks that are irrelevant for the migration and not to miss logging time spent on relevant activities. We had to define what development time, from RQ1, really is.

Relevant activities, for the migration process, are tightly connected to the different phases of a software lifecycle. In the article “Exploring the factors influencing the cloud computing adoption: a systematic study on cloud migration” Rashmi Rai et al. introduce a five-phase migration model inspired of the software lifecycle models [22]


Feasibility Analysis
Requirement Analysis & Migration Planning
Migration Execution
Testing & Migration Validation
Monitoring & Maintenance

The relevant activities that we have tracked are:

- Feasibility analysis: Knowledge gathering about cloud migration in general.
- Migration planning: Planning for how to migrate the application. Decision making of which cloud provider to use, which parts to migrate and which services to use.
- Migration Execution: Migrating the application and data using the platform, services and strategies that was planned for in the previous phase. Refactoring and change of code.
- Testing and migration validation: Testing the migrated application to validate the migrated system

and debugging and fixing problems that was encountered.

The last phase, Monitoring and maintenance is not within the scope of this study. The process is iterative in the sense that it was possible to go back to a previous phase if so needed.

We started the migration by gathering general knowledge about cloud migration. The time we spent on this phase was considered as development time spent for both the rehosting and refactoring strategies since what we learned during this phase helped carry out both migration processes.

The next step was to start planning for migrating with the rehosting strategy and later to execute the migration and testing and debugging the deployed application. The rehosting process was followed by more knowledge gathering and planning, where all researchers looked at different ways of migrating our application to the cloud and making it cloud native and when we felt confident enough, we started with the execution phase and the refactoring process. All the steps explained were thoroughly documented in the aspect of time and various challenges that we faced. This can be read about in detail in the appended experience report document referenced earlier in this section.

To answer RQ2 we have collected quantitative data by conducting execution time tests (in milliseconds) on key mobile application methods. The reasoning behind testing the performance inside the application is because we want to know how fast the application executes methods (where the method send/fetch data from database) since the mobile user interface (UI) and application as a whole stayedthe same on both approaches. To measure execution time for the lift and shift approach we used the built in android studio logger TimingLogger. And traces were used from the Firebase Performance API for the cloud native approach. While Firebase offered this feature for both approaches with their monitoring tool we learned that it updated very slowly (every 12 hours) and therefore choose to go with TimingLogger for the other approach. These two are essentially the same since they both measure the time (milliseconds) from start() to stop() in application code. All the methods were tested five times each and an average or median will be measured based on the distribution.

Validity Threats

To look for validity threats and try to minimize the impact of these threats we used the four categories; Construct validity, internal validity, external validity and reliability [25].

- Construct validity pertains to what extent the researchers is measuring the right things in regard to the research question [25]. In our case a threat was that we would measure other activities that we had not defined as development time. To mitigate this risk, we focused on having a very structured working process where we worked together in no longer then two-hour time slots. This to be able to stay focused and observant so that no one drifted away on activities that was not relevant to the migration process.

  Initially we realized the validity threat of us being in a changed state after we finish our first migration since we have learned new things and gained experience when working with the cloud platform. While working with the second migration strategy though, we quickly realized that the experience we had gained in the first migration process was of very limited advantage to us since the refactoring strategy and making the application completely cloud native requires learning to use new platforms, such as firebase, and hardly any of the knowledge gathered in the first migration process was of any use.

- Internal Validity is concerned with the risk of drawing wrong causal relationships [25]. This study's research questions are not of the kind that are seeking a cause and effect type of answer. This is in line with the design science methodology that are more focused on establishing how efficient an artefact is not why this is the case. But implicitly our questions relate the results of the migration strategies and because of the fact that we, the researchers did the migration ourselves the processes were totally transparent to us, which mitigates the risk of drawing the wrong casual relationships in regard to development time and RQ1.

  In regard to performance and RQ2 it is harder to mitigate the risk of drawing wrong casualties. Actually, it is not possible, within the scope of this study, to try to determine exactly what is the cause of the performances of the migrated application, other than to relate it to the specific migration strategy used.

- External Validity puts the focus on whether a study is generalizable or not [25]. As stated before this study does not strive to be generalizable but aims to complement previous research with an investigation of a migration endeavor within specific circumstances such as the small size of the application that is migrated and the inexperience of the ones completing the migration.

- Reliability: if a study is reliable it can be replicated by other researchers, i.e. the study should not be biased or subjective and the researcher's choices, research design, methods and data should be clearly described and made available [25]. In this study we can see a potential risk of confirmation bias where we try to confirm our assumptions. This risk was mitigated by checking ourselves to make sure that we do not have any preference in regard to the outcome. The whole research process is made transparent, from how we gathered data, wrote an experience report, and conducted our performance tests.

Evaluation

The third of Hevner et al. guidelines, Design evaluation, stresses the importance of well executed evaluation methods [11]. To evaluate our results, we will take what Hevner et al. describe as an analytical approach [11]. In the case of RQ1 we use a static analytical approach, where the time consumed when migrating the application, with two different strategies, is compared. In the case of RQ2 we use a dynamic analytical analysis, where we take the data collected from performance tests of the deployed applications and compare the results. To conclude the advantages and disadvantages of using the two migration strategies we cross-referenced the performance of the application, after using the migration strategies, and the time consumed outcome.

In the six guidelines, Design as a search process, Hevner et al. describes design science as an iterative process, as a search for an effective solution to a problem [11]. Alternative designs are created and later evaluated, in this case two different solutions are created to a problem and later evaluated with each other.

V. RESULT

Results for research question 1

The results of our migration strategies will here be displayed by presenting how many hours in total (all developer's hours added together) was spent on knowledge gathering and executing the migrations every week of the processes. This section is a shorter summary of the experience report referenced earlier.

week 1:
- A total of 60.5 hours was spent on gathering knowledge about the google cloud platform. (counts for both rehosting and refactoring)

week 2 (rehosting starts):
- The rehosting process was started by migrating our Erlang server and our MySQL database to the cloud.
- The server migration started on Monday and was finished on Wednesday.
- The database was migrated on Thursday and Friday.
- A total of 46 hours was spent on migrating the Erlang server to the cloud.
- A total of 42.5 hours was spent on migrating the database to the cloud. Debugging was still required.

week 3:
- A total of 40 hours was spent on debugging the migrated application back-end on the cloud

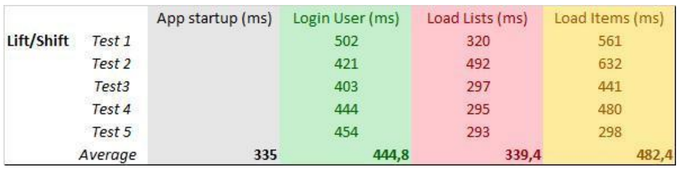
week 4 (refactoring starts):
- A total of 17 hours was spent on optimizing the rehosted application.
- A total of 50.5 hours was spent on reading about various approaches to the rehosting process.

week 5:
- A total of 99.5 hours was spent on getting familiar with the firebase platform and migrating our MySQL database to it.

week 6:
- A total of 119 hours was spent on customizing our android application to work with the firebase database and optimizing the new application.
- Total time spent on the rehosting process: 206 hours.
- Total time spent on the refactoring process: 269 hours.

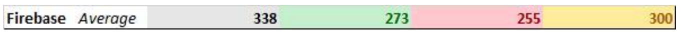
Results for research question 2

Performance testing



		App startup (ms)	Login User (ms)	Load Lists (ms)	Load Items (ms)
Lift/Shift	Test 1		502	320	561
	Test 2		421	492	632
	Test3		403	297	441
	Test 4		444	295	480
	Test 5		454	293	298
	Average	335	444,8	339,4	482,4

Figure 2. Lift/Shift tests



Firebase	Average	338	273	255	300

Figure 3. Firebase test.

In these tests we focused on the execution time (in milliseconds) for the key methods in the mobile application. These methods are as follows:

- Login: Time for a user to authenticate with the application.
- Fetch the users lists.
- Fetch the items in a list.

App startup was measured just to guarantee that there wasn't any big performance issue with the specific application before testing the methods. For the lift and shift approach we did 5 tests on each method and calculated an average value because of the normal distribution (Figure 2) and for the cloud native approach we collected data from 5 tests for a period of 12 hours and calculated the average of that (Figure 3).

VI. DISCUSSION

Our assumptions were that migration with the rehosting strategy would be significantly easier and less time consuming to carry out. We realized quite early on in the first migration process that the lift-and-shift method would not be a matter of simply moving our software components to the cloud platform and getting it to work with minor configurations. As is shown in the results section, the development time spent on the rehosting process is not far from that spent on the refactoring process. We realize that one major factor for this outcome is the fact that we, the developers, did not have any experience of working with cloud platforms before this study. If we had worked with the cloud platform before, some of the obstacles that we faced could probably have been avoided or dealt with more efficiently. This further emphasizes that the aim of study is to complement previous research with a scenario with specific parameters, as mentioned in the introduction. Therefore, the parameters for this study should be taken into consideration when viewing the outcome of the study which should act as complementary information rather than a best practice. For example, one of the major issues that we had with the rehosting process was that the migrated database did not automatically generate some values as it did when it was used on the previous, non-cloud based, host. This issue caused the SQL queries in the server to return errors. Getting familiar with the cloud platform was also something that required more time and effort than we initially expected.

The refactoring process proved to be more time consuming than the rehosting process as we had expected. What required most time was figuring out how to migrate our application in a way that makes it cloud native. We decided to stay on the Google Cloud platform and use the Firebase where the android application communicated directly with the Firebase Realtime database. Our assumptions for the development time of the two migration strategies proved to be accurate even for our small application, although the differences were smaller than expected. The motivation of our assumptions and the reasons for how the results turned out are the same, refactoring generally takes more time than rehosting because architectural changes are bigger, thus forcing changes in the different components of the application to make them compatible with each other. Rehosting requires no changes in the architecture and less code changes than refactoring.

For the performance testing we assumed that a refactored approach would have an edge in execution time based on the added cloud native functionality, which the results also shows. Having an average execution time 38% faster for the login method, 25% faster for loading user lists and 38% faster for loading items lists than the rehosting application. If you then take the development time into account and compare these two metrics one could argue that refactoring your application to the cloud is more beneficial. Another advantage with the refactoring strategy is that a cloud native application more easily can make use of all the services and tools that the cloud provides.

VII. Conclusion

This study has provided us with valuable insight to the world of cloud computing. We set out to find answers to the frequently asked question of what method to use when migrating an application to the cloud. We found that with our small 3-tier application the difference in development time between the rehosting approach and the refactoring was not significant enough to go with a rehosting migration method. If you then take into account that the refactoring approach gave us better performance and options to incorporate different cloud services one could argue that this approach would be more beneficial. In the case of an organization wanting to conduct a migration with similar parameters, this study helps prove that it could be worth it to consider migrating in a way that makes the application cloud native.

VIII. References

[1] Jamshidi, P., Ahmad, A., & Pahl, C. (2013). Cloud migration research: a systematic review. IEEE Transactions on Cloud Computing, 1(2), 142-157.

[2] Hajjat, M., Sun, X., Sung, Y. W. E., Maltz, D., Rao, S., Sripanidkulchai, K., & Tawarmalani, M. (2010, August). Cloudward bound: planning for beneficial migration of enterprise applications to the cloud. In ACM SIGCOMM Computer Communication Review (Vol. 40, No. 4, pp. 243-254). ACM.

[3] Rai, R., Sahoo, G., & Mehfuz, S. (2015). Exploring the factors influencing the cloud computing adoption: a systematic study on cloud migration. SpringerPlus, 4(1), 197.

[4] Linthicum, D. S. (2017). Cloud-Native Applications and Cloud Migration: The Good, the Bad, and the Points Between. IEEE Cloud Computing, 4(5), 12-14.

[5] Bond, J. (2015). The enterprise cloud: Best practices for transforming legacy IT. " O'Reilly Media, Inc.".

[6] Daconta, M. C. (2013). The Great Cloud Migration: Your Roadmap to Cloud Computing, Big Data and Linked Data. Outskirts Press.

[7] Passmore, E. (2016). Migrating Large-Scale Services to the Cloud. Apress.

[8] Balalaie, A., Heydarnoori, A., & Jamshidi, P. (2015, September). Migrating to cloud-native architectures using microservices: an experience report. In European Conference on Service-Oriented and Cloud Computing (pp. 201-215). Springer, Cham.

[9] Zhang, W., Berre, A. J., Roman, D., & Huru, H. A. (2009, October). Migrating legacy applications to the service Cloud. In 14th Conference companion on Object Oriented Programming Systems Languages and Applications (OOPSLA 2009) (pp. 59-68).

[10] Suleman, A. (2018). The Best Cloud Migration Path: Lift And Shift, Replatform Or Refactor?. Retrieved from https://www.forbes.com/sites/forbestechcouncil/2018/03/23/the-best-cloud-migration-path-lift-and-shift-replatform-or-refactor/#631ae0fd4f51

[11] Von Alan, R. H., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. MIS quarterly, 28(1), 75-105.

[12] JWoods, J. (2011). Five Options for Migrating Applications to the, Cloud: Rehost, Refactor, Revise, Rebuild or Replace. In Gartner The Future of IT Conference.

[13] Ringström, J., Farahbakhsh-Fard, N. and Leveau, P. (n.d.). Experience Report. [online] Available at: https://docs.google.com/document/d/1X64bGsbJhd5elfIDsgMUjrkU7IiePXB1sH4i5qt3pGU/edit.

[14] Nane Kratzke, Peter-Christian Quint, Understanding cloud-native applications after 10 years of cloud computing - A systematic mapping study, Journal of Systems and Software, Volume 126, 2017.

[15] Peter Mell, Timothy Grance, The NIST Definition of Cloud Computing, Computer Security Division Information Technology Laboratory National Institute of Standards and Technology, Special Publication 800-145, 2011.

[16] S. Challita, F. Zalila, C. Gourdin and P. Merle, "A Precise Model for Google Cloud Platform," 2018 IEEE International Conference on Cloud Engineering (IC2E), Orlando, FL, 2018, pp. 177-183.

[17] Compute engine, [Online] Available: https://cloud.google.com/compute/ [Accessed: 11-April-2018].

[18] Cloud SQL, [Online], Available: https://cloud.google.com/sql/ [Accessed: 12-April-2018].

[19] The-Embedded-8,[Online]Available: https://github.com/The-Embedded-8, [Accessed: 10-May-2018].

[20] Firebase Realtime Database, [Online] Available: https://firebase.google.com/docs/database/, [Accessed: 25-April-2018].

[21] Firebase Authentication, [Online] Available: https://firebase.google.com/docs/auth/, [Accessed: 25-April-2018].

[22] Rashmi RaiEmail, Gadadhar Sahoo, Shabana, Mehfuz, Exploring the factors influencing the cloud computing adoption: a systematic study on cloud migration, SpringerPlus, 2015.

[23] N. Alshuqayran, N. Ali and R. Evans, "A Systematic Mapping Study in Microservice Architecture," 2016 IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA), Macau, 2016, pp. 44-51.

[24] Lucas F. Albuquerque,, Felipe Silva Ferraz, Rodrigo F. A. P. Oliveira, Sergio M. L. Galdino, Function-as-a-Service X Platform-as-a-Service: Towards a Comparative Study on FaaS and PaaS, The Twelfth International Conference on Software Engineering Advances, 2017.

[25] Per Runeson, Martin Höst, Guidelines for conducting and reporting case study, Empir Software Eng, 2009