Institute of Architecture of Application Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Master's Thesis

Cloud-native Applications: Authoring and Evaluation of Two Deployment Patterns

Mirna Alaisami

Course of Study:	Computer Science - STE
Examiner:	Prof. Dr. Dr. h. c. Frank Leymann
Supervisor:	Michael Wurster, M.Sc. Matthias Haeussler, NovaTec Mc.
Commenced:	October 4, 2017
Completed:	March 28, 2018

Acknowledgement

First and foremost, I would like to express my sincere gratitude to Professor Frank Leymann for the opportunity to work on this thesis at the Institute of Architecture of Application Systems, University of Stuttgart. He has always motivated and inspired me by his character, teaching style and valuable scientific achievements.

I would also like to thank my master thesis supervisor M.Sc. Michael Wurster at the University of Stuttgart for his continuous support, insightful comments and suggestions, motivation and enthusiasm, and immense knowledge, which helped to achieve and enrich this work.

Besides, I wish to thank my master thesis supervisor Mc. Matthias Haeussler at NovaTec, who has always been there whenever I needed any question or support regarding this thesis. His patient guidance, consistent encouragement, insightful viewpoint and valuable discussions paved the way to fulfil this work.

Last but not least, I would like to express my very profound gratitude to the continuous love, motivation and support of my mother, my brother, my sisters and my husband, and I would like to dedicate this work to my late father, who has always been my constant source of encouragement and inspiration.

Abstract

We live in the era of cloud computing today. Many companies are moving their legacy applications to the cloud or building cloud-native applications from scratch. What makes it more interesting is the fact that 80% of IT budget will be spent on cloud technology by the year 2025 [Soa16]. However, there is no clear single approach on how to migrate or deploy these applications to the cloud, since various platforms provide different options.

The diverse deployment approaches bring along different aspects in terms of both flexibility and responsibility of the deployment. However, one can ask: What are the main differences between these methodologies? What are the pros and cons? Is there a best or more robust methodology? When to use which technique and why? Are there any factors, which may influence your decision? Consequently, this thesis is introduced in order to answer such kind of questions and help developers choose the most suitable approach for their application.

This work uses a systematic literature review in order to investigate the state of the art approaches and technologies used to enable platform-as-a-service deployments. Furthermore, considering the fact that there is still no scientific documentation for the varied approaches as patterns out there, new patterns for the various approaches are authored, described and documented. In addition, comparison matrices, which indicate the pros and cons of the multiple approaches using different application types are constructed, and a sample distributed application is deployed using the introduced patterns, in order to evaluate and validate them. Ultimately, a final discussion of the results is conducted and a decision tree that guides the choice of the appropriate pattern and the suitable platform is built.

Contents

List of Figures

List of Tables

List of Listings

List of Algorithms

1 Introduction

At the heart of cloud computing trend, lie cloud-native applications. They can be defined as applications specifically designed to leverage the advantages of a cloud environment in an efficient manner [FLR+14]. This can be achieved by ensuring the following IDEAL properties: **I**solated state, **D**istribution, **E**lasticity, **A**utomated management, **L**oose coupling [FLR+14]. Thinking about the significant principles of these applications, it can be concluded that: DevOps, continuous delivery, microservices and containers form the four main tenets of cloud-native patterns [BHJ16] [Zim17].

Container technology is an approach, which is used to package full application stacks (code, runtime, system tools, system libraries and settings) as immutable artifact, which can be easily and reliably run in different environments. According to CA Technologies, container technology is set to shape the future of software development standards, making it easier and more efficient for developers to move an application from one system to another aiming at greater interoperability [Pah15] [Sta16]. Thus, containers are a prominent tool to enable DevOps workflows. They act as a common language between development and IT operations and when configured properly; the operations team can utilize containers to assemble environments without compromising security [Wee17]. Consequently, container technology is used as a core technology in most PaaS environments.

In the cloud computing area, different alternative approaches that make use of container technology are mainly used to deploy cloud-native applications. Let us have a look at the cloud-native application example shown in Figure 1.1. In this simple example in which customers can establish some orders to buy products, four components can be seen: a UI; an order service; a catalog service and a customer service, which are interacting through a service registry and each of the services has its own database. Here, developers have different choices in order to deploy this cloud-native application to the cloud. They can rely on uploading the source code of each of the components and the service registry, configure the connection of other components to the service registry and make use of the databases as cloud provided services. The cloud platform will take care of building the containers to run those applications. Developers can even make use of the service registry as a cloud provided service instead of deploying it as a separate component. Alternatively, they can package each of the components, the service registry and the databases inside a container, deploy those containers to the cloud and take care of configuring the connections between them. They can even connect those containers to the databases as external services provided by different cloud providers. If they have some scheduled tasks within those components, which they need to run when an event happens, they can deploy it as a function-as-a-service and benefit from the billing based on consumption model.

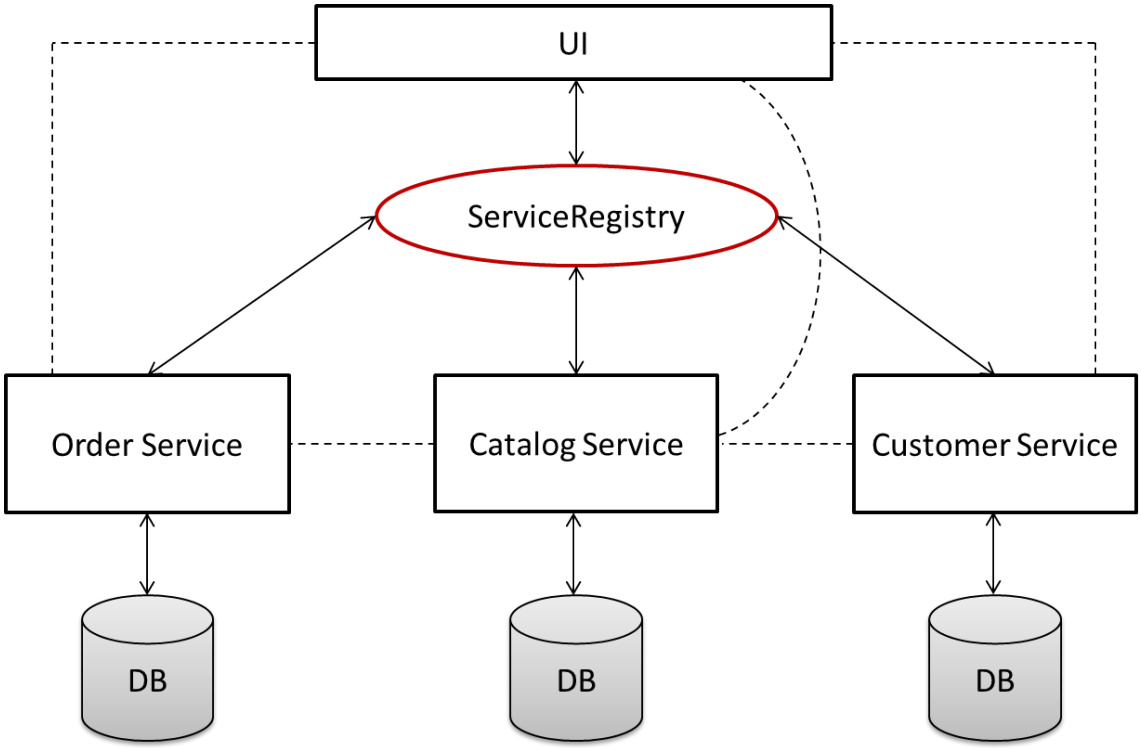


Figure 1.1: Example of a Cloud-native App

Those different deployment choices offer different levels and possibilities for scalability, resiliency, portability, load balancing, logging and monitoring. They even range in their usability, the responsibility they put on the developer and the risks that might occur. They, of course, converge at some points while interfere at others. The world of IT today lacks the capability to take a clear appropriate decision to follow one approach over the other. Companies are somehow confused about the main factors that should affect their choice. This leads to the fact that there are still a lot of open questions related to the multiple solutions in general, regardless which technologies are used to achieve them, which indicates the need for further research and investigation.

1.1 Research Methodology

The research methodology chosen for this thesis is a systematic literature review (SLR). This decision has been made due to the fact that this is a well-defined approach to identify, evaluate, interpret and compare all available studies related to a particular research question, topic area or phenomenon of interest [Kit04] [KC07]. This work seeks to accomplish a deep analysis and evaluation of the platform-as-a-service (PaaS) deployment approaches and technologies. In particular, it aims to have some idea of the extent of studies and activities performed in that field before this thesis was introduced. Also, it pursues to examine the different approaches followed in this field. Besides, it endeavors to address

the state of the art technologies used in the different approaches. Proceeding from the SLR study made in the thesis, this work aims at formulating the different approaches as patterns, which in turn reveals the factors; challenges; pros and cons, which affect evaluating and comparing the different approaches related to PaaS deployment. Moreover, in order to validate and evaluate the authored patterns, multiple matrices, which summarize the pros and cons of each deployment approach considering different application types are designed, and a sample distributed application, which utilizes the resulted patterns, is deployed. Finally, a conclusive discussion of the results along with a decision tree that leads the choice of the convenient pattern and platform are introduced.

1.2 Document Structure

This thesis is structured into five chapters. A short introduction to cloud-native applications deployment approaches and the motivation, the research methodology and the structure of the work are covered in Chapter 1. The systematic literature review that was conducted to explore the current research state is introduced within Chapter 2. The scientific documentation of the deployment approaches as patterns is the content of Chapter 3. The validation and evaluation of the patterns using comparison matrices of the two approaches applied on different application types and the deployment of a sample distributed application are presented in Chapter 4. Ultimately, a summary of this work along with a discussion of the results, the decision tree and the further research opportunities are outlined within Chapter 5.

2 SLR Process

This chapter explains and describes the systematic literature review utilized in this thesis. In this research, the SLR process suggested by Kitchenham is followed [Kit04] [KC07]. Figure 2.1 shows an overview of the used SLR process. It mainly consists of three phases: Planning, Conducting and Reporting. Starting with the planning phase, it fundamentally comprehends the most important pre-review activities. This includes identifying the need for such a systematic literature review, defining the research questions addressed by the systematic review and producing a review protocol that plans for the basic review procedures. Moving to the conducting phase, it contains the major steps, which are essential for executing the SLR according to the defined review protocol. These steps are: performing the search process, filtering of material, extracting data from the collected sources, and

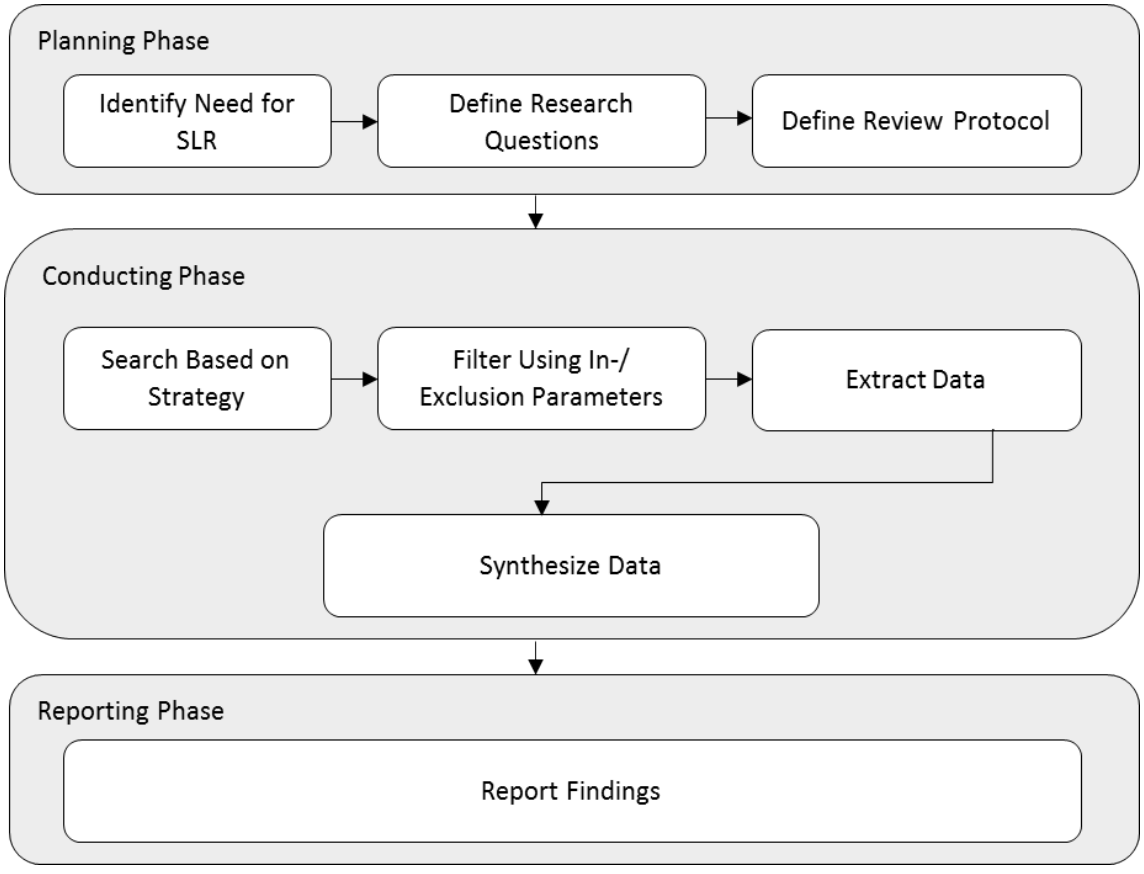


Figure 2.1: SLR Process Overview

synthesizing the data. Concluding by the reporting phase, it gives an account of the results of the systematic review effectively.

2.1 Planning Phase

Firstly, the start is with the planning of the systematic review. This phase begins with addressing the need for the SLR process, passes through formulating the research questions and results in revising the way in which the review will be executed. Within the following sections, the different steps are explained in more detail.

2.1.1 The Need for a Systematic Review

Prior to undertaking the SLR, it is important to clarify the need for it [Kit04] [KC07]. The decision on accomplishing a systematic review arises from the requirement to explore the state of the art approaches and technologies used to enable PaaS deployments in a thorough and unbiased manner. This in turn stands as a substantial prelude to the contribution in formulating those different approaches as patterns at the end of this research.

2.1.2 Define Research Questions

Next, the most important aspect in the planning phase, which is determining the research questions, is reached [CHHK13]. It is a crucial part in studying and analyzing the most raised PaaS concepts and approaches. Within this study, the following research questions are defined:

- RQ1: How much activity in the field of platform-as-a-service research has there been since 2011?
  This question is aimed at giving an overview to what extent existing research has been active in the field of platform-as-a-service deployment before writing this thesis.
- RQ2: Who is active in this research area?
  The motivation behind this question is to provide information about researchers and organizations, who addressed the research topic in this area.
- RQ3: What kind of concepts and approaches are being used for platform-as-a-service deployments?
  This question is formulated to investigate the used concepts and followed approaches to develop and deploy cloud-native applications in PaaS.
- RQ4: What kind of technologies enabling different PaaS approaches are being used?
  The objective of this question is to highlight the most used technologies in platform-as-a-service applications development and deployment.

2.1.3 Define Review Protocol

Another important part in the SLR study, which is necessary to reduce the possibility of researcher bias, is defining the review protocol [Kit04] [KC07]. This pre-defined protocol lies as a foundation for the next actions in the SLR process. On the basis of the specified research questions; the protocol defines various steps. First, it illustrates a search strategy that will be employed to search for primary studies including search terms and resources to be searched. Next, it determines the inclusion/exclusion criteria to select the studies. After that, it clarifies the strategy in which the required information will be obtained from each primary study. Lastly, it explains the approach that will be used for data Synthesis. Within the subsequent sections, an effort will be made to describe in more detail how the different steps of the protocol are designed.

2.1.3.1 Search Strategy

This step is responsible for defining the search process that will be followed in detail. In this context, the databases that will be used for the search procedure are addressed, the search terms are demonstrated, considering their different synonyms and alternative spellings, and finally the sophisticated search phrases, which will be utilized to search for material, are constructed through combining those search terms using boolean operators.

Within this thesis, an automatic search that uses the following data sources suggested by Kitchenham is performed [KC07]:

- IEEE Xplore
- Google Scholar
- Science Direct

Based on the research questions of Section 2.1.2, the following basic search terms are derived: (1) **platform as a service**, (2) **concepts**, (3) **approaches**, (4) **deployment**, (5) **development**, (6) **technologies** and (7) **cloud-native applications**. With respect to other keywords used often in the domain of this study, the following terms are obtained: (8) **container technology**, (9) **function as a service**, (10) **cloud foundry** and (11) **docker**.

In consideration of search terms synonyms, abbreviations, and alternative spellings; different alternative terms are inferred as shown in Table 2.1.

Connecting the previous defined search terms and their alternatives using Boolean operators (ANDs and ORs), results in composing the search phrases that will be finally used for the search procedure. Figures 2.2 to 2.10 illustrate the derived search strings, such that each search string is represented as an And/Or tree.

Term	Synonym/Abbreviation/Alternative Spelling
"platform as a service"	PaaS
concepts	notions, ideas
approaches	ways, methods
“cloud-native applications”	“cloud-native apps”
“container technology”	“container as a service”, "container orchestrator", CaaS
"function as a service"	FaaS
“cloud foundry”	“cloudfoundry”, CF

Table 2.1: Search Terms and their Synonyms, Abbreviations and Alternative Spellings

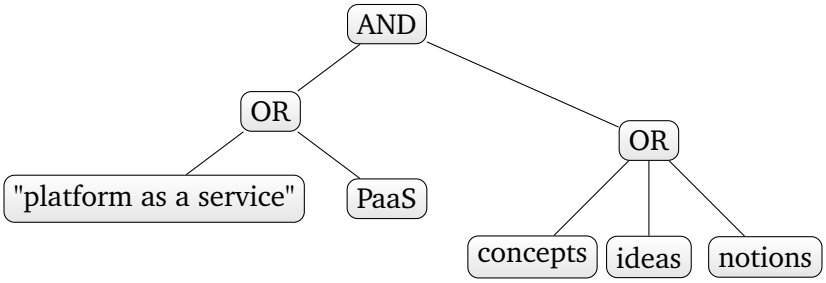


Figure 2.2: Search Phrase 1

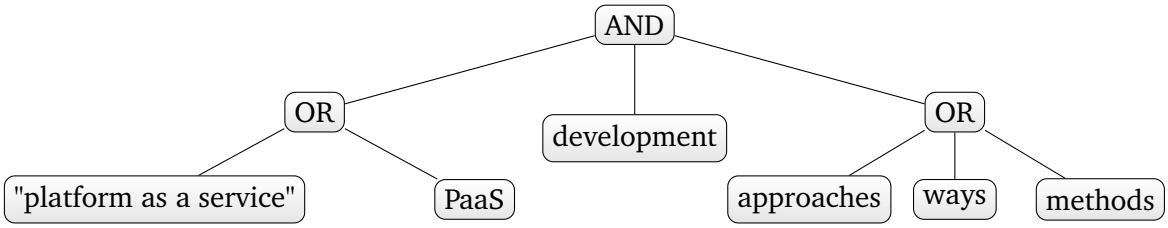


Figure 2.3: Search Phrase 2

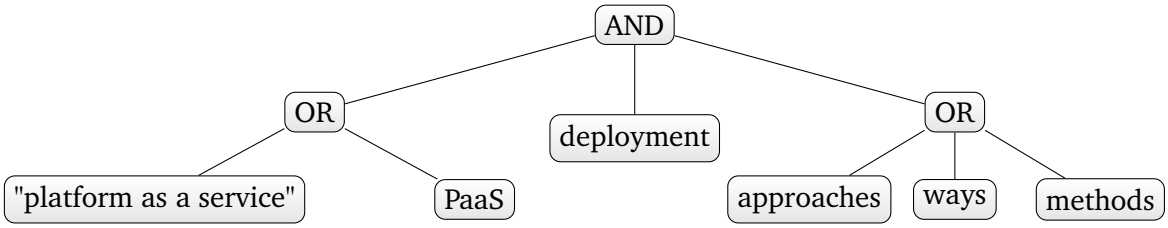


Figure 2.4: Search Phrase 3

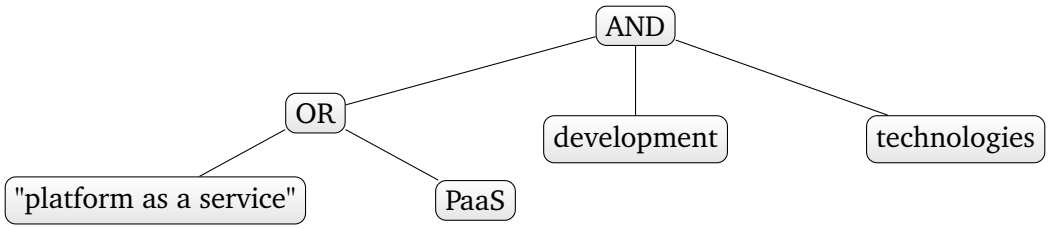


Figure 2.5: Search Phrase 4

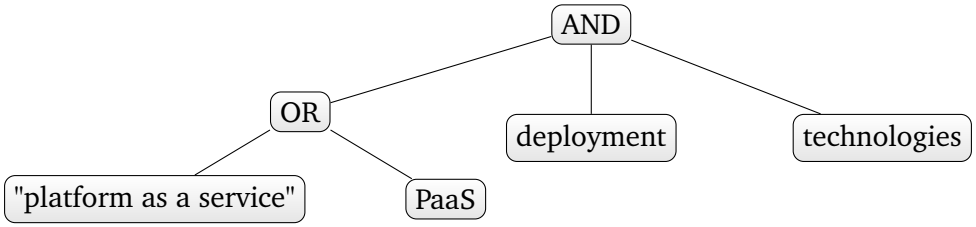


Figure 2.6: Search Phrase 5

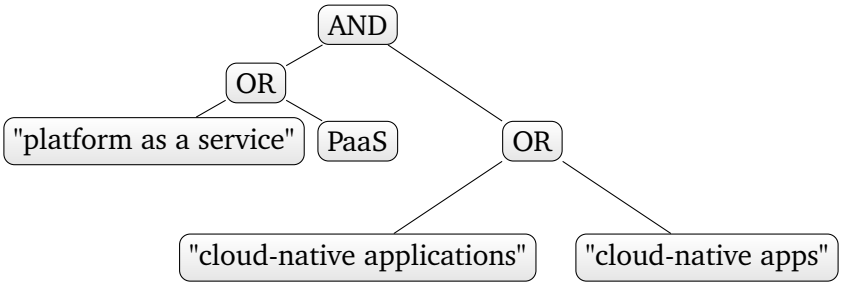


Figure 2.7: Search Phrase 6

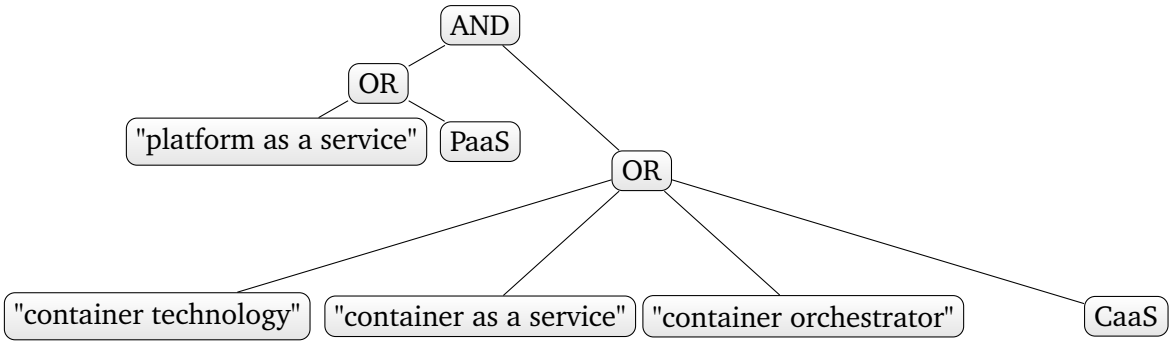


Figure 2.8: Search Phrase 7

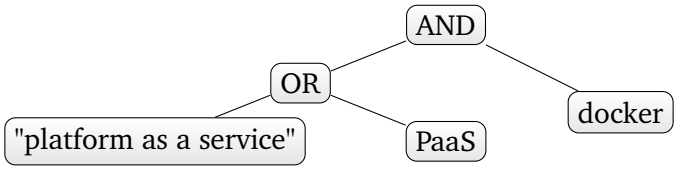


Figure 2.9: Search Phrase 8

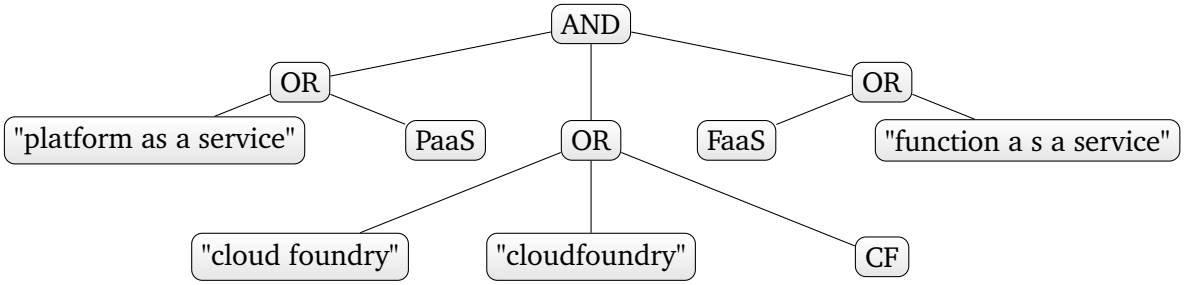


Figure 2.10: Search Phrase 9

2.1.3.2 In- and Exclusion Criteria

In fact, literature searching may lead into a huge number of eligible resources, of which only a small proportion may eventually be included in the review. As a consequence, the review protocol defines a set of predefined characteristics, i.e., a set of inclusion and exclusion criteria, in order to filter results of searching.

A found resource is included in the SLR, if it fulfills all the following inclusion criteria:

- IC1: It concerns PaaS concepts, approaches or technologies
- IC2: It concerns PaaS cloud-native applications development or deployment
- IC3: It is published between 2011 and 2017 (This limitation of time frame is done because the year 2011 coincides with the announcement of "cloud foundry")

A found resource is excluded from the SLR, if it fulfills one of the following exclusion criteria:

- EC1: It is in a language other than English
- EC2: It is in the form of a panel discussion, preface, tutorial, book review, or presented slide, i.e., it has too informal nature
- EC3: Duplicate reports of the same study exist (When several reports of a study exist in different data sources, the most complete version of the study is included in the SLR)

2.1.3.3 Data Extraction

The data from any given source, need to be extracted systematically. Thus, the data extraction step aims at designing data extraction forms, in order to accurately extract relevant data from the selected studies and reduce the opportunity for bias [BKB+07].

Table 2.2 shows the data extraction form that will be used for each study. These forms will be filled with the help of the reference management tool **JabRef** [Jab]. On one hand, data items DE1-DE5 will be used for documentation purposes and answering research questions Q1 and Q2. This includes title, year, authors, organization and country of study. On the other hand, data items DE6 and DE7 help to answer research questions Q3 and Q4. This includes various PaaS concepts, approaches and technologies.

Item Id	Item	Value
DE1	Title	
DE2	Year	
DE3	Author(s)	
DE4	Organization	
DE5	Country of study	
DE6	PaaS concepts & approaches	
DE7	PaaS development & deployment technologies	

Table 2.2: Data Extraction Form

2.1.3.4 Data Synthesis

In this step, the extracted data in the previous step are synthesized, in order to prepare the interpretation of found results. In this sense, all the findings are summarized and tabulated, such that each research question is assessed individually against the findings as follows:

- RQ1: How much activity in the field of platform-as-a-service research has there been since 2011?
  This question will be addressed by simple counts of the number of PaaS related studies per year since 2011. Table 2.3 shows the table form that will summarize data to answer this question.

Year	Num. PaaS Related Studies
2011	
2012	
2013	
2014	
2015	
2016	
2017	

Table 2.3: Data Synthesis Form for RQ1

- RQ2: Who is active in this research area?
  In order to answer this question, it will be investigated whether any specific organization of researches has undertaken a relatively large number of research in PaaS domain since 2011. The table form that will epitomize data to target question is presented in Table 2.4.

Study	Authors	Institution	Country of Institution

Table 2.4: Data Synthesis Form for RQ2

- RQ3: What kind of concepts and approaches are being used for platform-as-a-service deployments?
  In order to target this question, the elicited data are reviewed and the PaaS concepts and approaches resulted from the multiple studies over the years since 2011 are extracted, beside the number of studies in which these concepts appeared. Table 2.5 demonstrates the table form that will be used for collecting data in order to answer this question.

PaaS concept/approach	
Concepts	**Num. Studies**
Approaches	

Table 2.5: Data Synthesis Form for RQ3

- RQ4: What kind of technologies enabling different PaaS approaches are being used?
  To answer this question, an effort will be made to revise the resulted data and extract the PaaS development and deployment technologies elicited from the multiple studies over the years since 2011, beside the number of studies in which these technologies appeared. The table form that sums data to answer this question is shown in Table 2.6.

PaaS technology	Num. Studies

Table 2.6: Data Synthesis Form for RQ4

2.2 Conducting Phase

Secondly, the conducting phase begins, in which the SLR process is executed according to the defined review protocol (see Section 2.1.3). This phase includes searching for sources, filtering them depending on in-/exclusion criteria, extracting data from the collected sources and finally synthesizing the extracted data. The different steps are explained in more detail within the following sections.

2.2.1 Search Based on Strategy

Within this step, the search process is executed. Regarding searching in **IEEE Xplore**, the "command search" and the "metadata only" option are used. This yields **476** studies. For searching in **Science Direct**, the "expert search"; the "computer science" field and the "tak" option (search only title/abstract/keywords), are used. This yields **35** studies. Coming to the search process in **Google Scholar**, the number of results without any filtering is relatively huge (**189461** studies).

2.2.2 Filter Using In-/Exclusion Parameters

Next, the in- and exclusion parameters are applied (see Section 2.1.3.2). After applying them, the number of studies resulted from **IEEE Xplore** is reduced to **17** and the number of studies resulted from **Science Direct** is reduced to **1**. Since the number of studies resulted from **Google Scholar** is relatively huge, the Algorithm 2.1 is followed, which yields **9** sources.

Algorithmus 2.1 Search Algorithm Followed on Google Scholar

```
Filter on (language, year)
if number of sources > 100 then
    Use the "sort by date" option
    if number of sources ≠ 0 then
        Filter based on in- and exclusion parameters
    else
        Use the "sort by relevance" option instead of the previous one and "filetype:pdf"
        if number of sources ≠ 0 then
            Filter based on in- and exclusion parameters
        end if
    end if
else
    Filter based on in- and exclusion parameters
end if
```

Table 2.7 shows references to the resulted studies from the different data sources after the filtering step.

Data Source	Resulted Studies
IEEE Xplore	[APO16], [PHM+16], [Mah11], [CLC11], [Bob11], [GS13], [WASL14], [Coh13], [SSKY16], [GWZW16], [LZL17], [SAS+15], [MPA+16b], [XOR+15], [Pah15], [MPA+16a], [PDKD17]
Science Direct	[RVC+12]
Google Scholar	[Sha16], [BCC+17], [WXL16], [TM16], [SSC17], [MKT17], [Kra17], [Lee17], [GKK17]

Table 2.7: Resulted Studies from the Data Sources

2.2.3 Extract Data

After that, data from each of the 27 resulted sources are extracted by filling out the previously defined data extraction form for each study. The tables that show the extracted data for all studies can be found within the Appendix A.

2.2.4 Synthesize Data

Having extracted all relevant data in the previous step, now the data are synthesized. The previously defined data synthesis forms are filled out, in order to answer the four research questions. The tables that show the synthesized data from all studies can be found within the Appendix B.

2.3 Reporting Phase

Lastly, the reporting phase is reached, in which the results from the data synthesis step are interpreted (see Section 2.2.4) and the answers to the research questions are discussed.

2.3.1 Report and Discuss Findings

The outcome of the SLR process is presented in this section. Within the following, the findings triggered by the four previously defined research questions are outlined (see Section 2.1.2).

- RQ1: How much activity in the field of platform-as-a-service research has there been since 2011?
  The number of studies by year of publication is shown in Figure 2.11. It can be noticed that the number of PaaS publications ranges between one and three within the years from 2011 to 2015. Then, it significantly increases in the years 2016 and 2017. Taking in consideration the bias that occurred because of using the option

that restricts the results to year 2017 in Google Scholar; a significant increase in the number of PaaS publications during the year 2016 can still be noticed. This may be due to the fact that PaaS has gained grand momentum the last two years and investments in PaaS have grown impressively, especially with the emerge of container-as-a-service and containers orchestration [WHJ16] [Kav16] [Hen17].

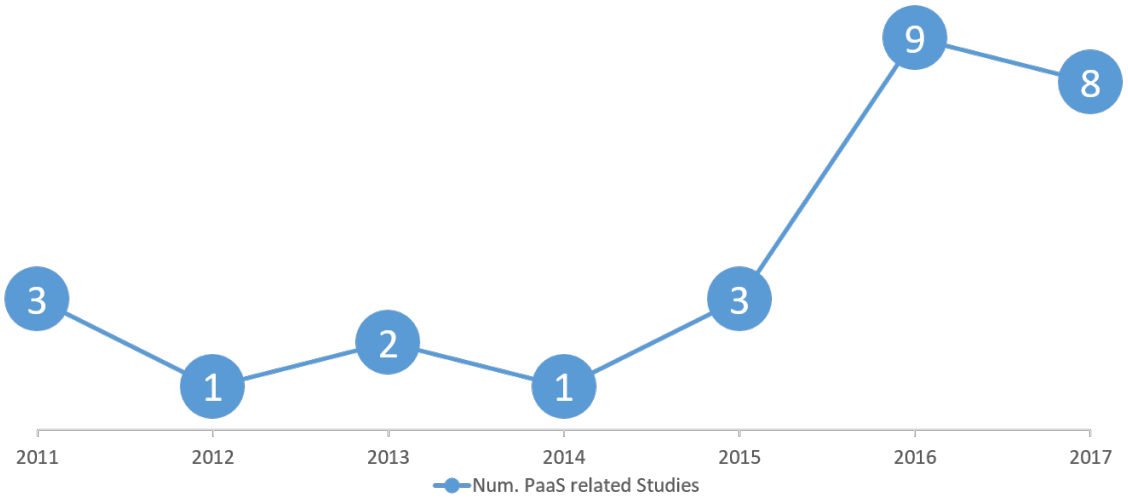


Figure 2.11: Temporal Distribution of the Included Studies

- RQ2: Who is active in this research area?
  The distribution of studies by country is shown in Figure 2.12. Overall, the set of studies is dominated by Asian researchers, who have been involved in twelve of the studies, in particular India, which has been involved in six of the studies, two of them were accomplished at Electronic city (MTech(IIIT), CORI-ISE Dept) and Cisco Systems. Then, the European researchers come in second place with the participation in nine of the studies, specially Germany, which has been involved in three of the studies, two of them were accomplished at Stuttgart University (IAAS). Only four of the studies had North American authors.

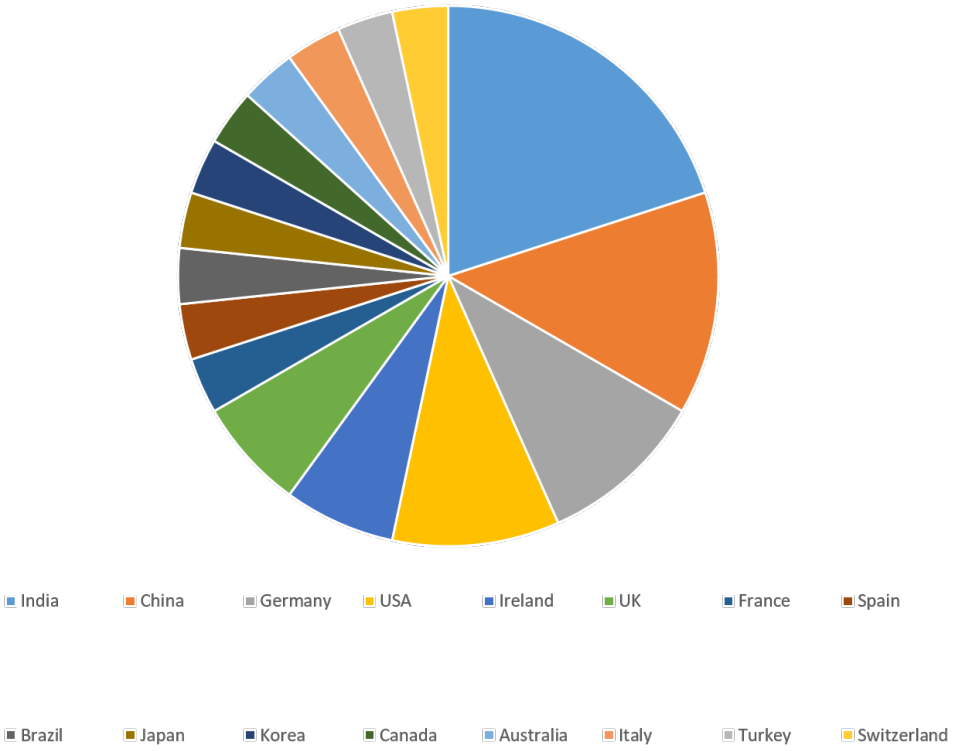


Figure 2.12: Spatial Distribution of the Included Studies

- RQ3: What kind of concepts and approaches are being used for platform-as-a-service deployments?
  The distribution of PaaS concepts by study depending on the number of studies in which they appeared, is shown in Figure 2.13. Based on the analysis of that figure, it can be concluded that the three primary concepts that are related to PaaS are:
  - *Containerization of application in PaaS*
  - *Containers orchestration and cluster management in PaaS*
  - *Application runtime environment and middleware provided by PaaS*

  There are also other important aspects that should be taken into consideration in PaaS such as serverless platforms, security, performance, portability and scalability.

  Considering the main approaches that are followed in deploying applications into a platform-as-a-service; two fundamental approaches were found:
  - **Bring your own code (BYOCD)**, in which developers can simply write and upload their code and let the platform deal with the container handling and any infrastructure below.

- **Bring your own container (BYOCR)**, in which developers have the option to directly deploy applications in the same container format that the platform uses.

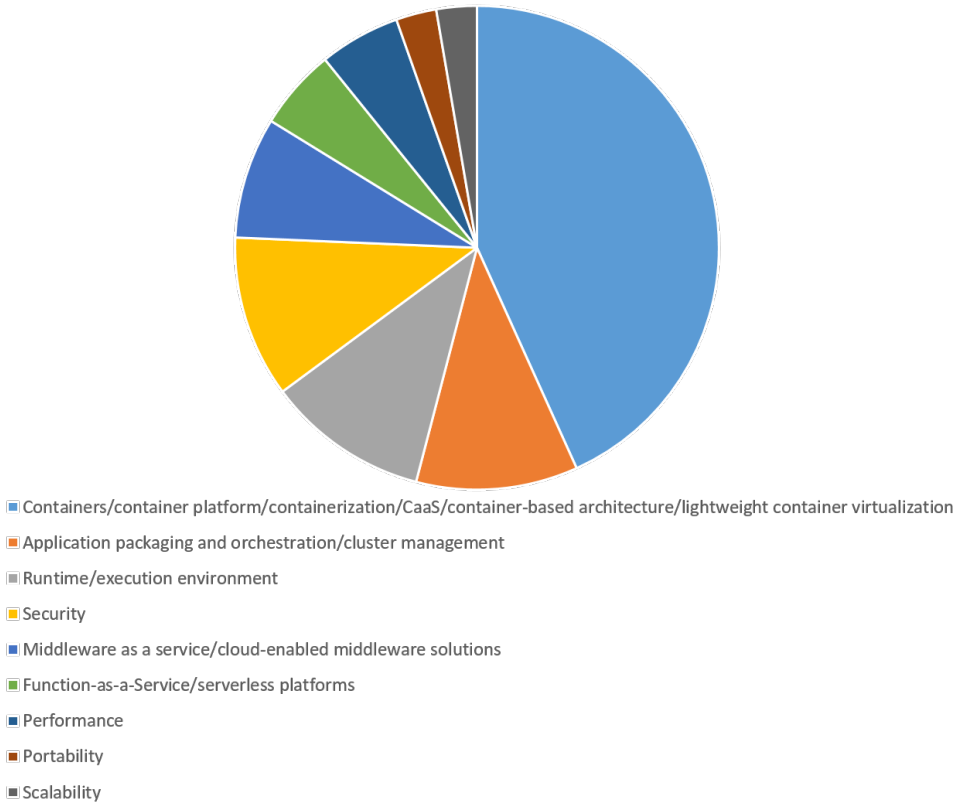


Figure 2.13: Distribution of PaaS Concepts by Study

- RQ4: What kind of technologies enabling different PaaS approaches are being used? The distribution of PaaS technologies by study depending on the number of studies in which they appeared, is shown in Figure 2.14. Analyzing this figure, it can be noticed that the dominant PaaS technologies that are most mentioned in the publications are:

  - *Docker*: The container platform provider that allows applications to run within separate (i.e., logically isolated) containers by leveraging a series of Linux kernel features [LZL17].
  - *Google App Engine*: One of the earliest integrated cloud application tools that serves as a web framework and cloud computing platform for developing and hosting web applications in Google-managed data centers [Coh13] [Goo].
  - *Cloud Foundry*: The industry standard open PaaS that abstracts away infrastructure, so that the developer can focus on application innovation [Pah15] [PDKD17].

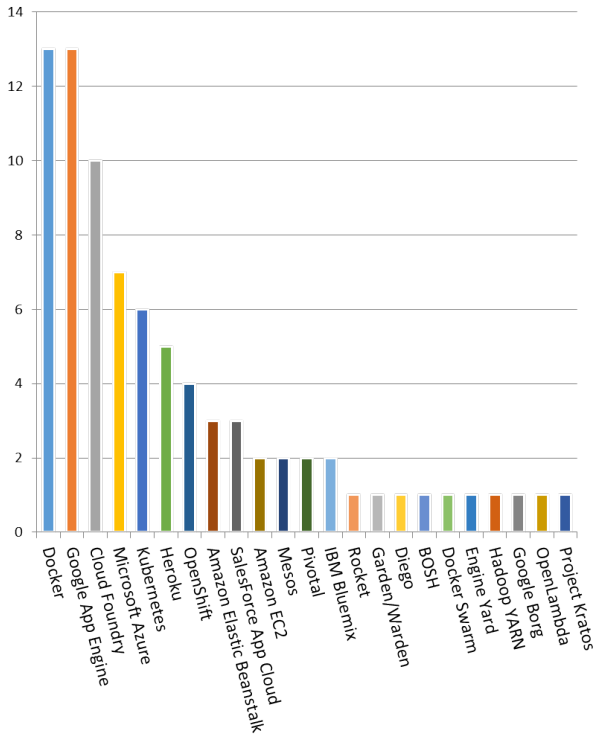


Figure 2.14: Distribution of PaaS Technologies by Study

- *Microsoft Azure*: A cloud computing platform and foundation that is intended for building, sending and managing applications and administrations through Microsoft-managed datacenters [TM16].
- *Kubernetes*: An open-source system that can be configured to allow orchestrating Docker containers. It is based on processes running on Docker hosts that bind hosts into clusters and manage containers [PHM+16].
- *Heroku*: A cloud platform based on a managed container system that lets companies build, deliver, monitor and scale applications bypassing infrastructure headaches [Plab].

Proceeding from the fact that the two inferred approaches followed in PaaS applications deployment, BYOCD and BYOCR, have not been formulated as documented *patterns* before, an effort to investigate and evaluate them will be made, by means of formulating the two deployment approaches as patterns in the next chapter, capitalizing on the found PaaS concepts and technologies.

3 PaaS Deployment Patterns

As it can be seen within the previous chapter, two variants in the space of PaaS application deployment have been detected: BYOCD and BYOCR. Studying the two approaches has taken a lot of concern recently. Based on the achieved SLR process, it could be argued that many attempts have been there to evaluate the two approaches. For example, Beth Cohen, in his article "PaaS: New Opportunities for Cloud Application Development", has focused on giving an overview on some PaaS technologies such as Azure, Cloud Foundry and OpenShift [Coh13]. Also, Claus Pahl, has described briefly some PaaS evolution technologies and their use of Containerization technology in his article "Containerization and the PaaS Cloud" [Pah15]. Moreover, the authors of the article "Towards a Full-Stack DevOps Environment for Cloud-Hosted Applications", talked about different PaaS technologies such as Microsoft Azure, Cloud Foundry, Docker and others [LZL17].

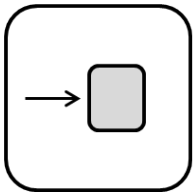
However, the previous attempts have only tried to highlight or compare some specific technologies that belong to the two approaches, but there was no real study that analyzes and evaluates the two deployment approaches in general. This leads to the fact that there are still a lot of open questions related to the two general solutions, regardless of which technologies are used to achieve them.

Relying on the previous discussion and the fact that there is still no scientific documentation for the two approaches as patterns out there, the two PaaS deployment approaches will be described as patterns within the following chapter, documented and evaluated exploiting the different concepts and technologies discovered in Chapter 2.

A pattern can be defined as structured text that follows a well-defined format and captures nuggets of advice on how to deal with recurring problems in a specific domain [Ale78] [Ale80]. Consequently, the two approaches will be analyzed and evaluated through the patterns, highlighting the main differences, pros and cons, and the factors that may influence the developer's choice to prefer one approach over the other. The individual patterns are structured in a way, in which they utilize the pattern format introduced by Alexander [Ale78].

3.1 Bring Your Own Code Pattern (BYOCD)

A platform to which the developer uploads his or her source code and asks to run it on the cloud without caring how. The platform then takes care of everything to deploy and run the code on the cloud.



How can a developer focus on the application business logic and not on the infrastructure, middleware and container implementation, when deploying and operating an application or a piece of code on the cloud?

Context

DevOps is a strong and popular movement in companies nowadays. It mainly refers to the principle of bridging the gap between development and IT operations via establishing cross functional teams, which are responsible for design, development and operating of applications [Htt12].

Looking at DevOps from a development (and not management) perspective, it puts additional responsibilities to operate applications into the same team that developed them. The challenge is that many developers are not familiar or concerned with building containers or tweaking middleware internals in order to run their applications on the cloud. As a result, an approach should be followed in order to remove the undifferentiated heavy lifting from application developers, i.e., the burdensome responsibilities associated with delivering software such as application runtime configuration and containers construction, load balancing and traffic routing, service discovery, databases provisioning and many other tasks [Win17], so that developers can just focus on their application's business code.

Forces

Developers should focus on building their applications well and fast. They should also be able to deploy and run them on the cloud without caring how. This means that the platform should provide the operating system, middleware and language runtime, in such a way as if the developer, who wrote the code himself or herself, has used them to run the application on his or her own laptop.

Besides, the developer should still be able to use necessary backing services (such as datastores, messaging/queueing systems, SMTP services, etc.) for his or her applications in an easy scalable manner that does not affect the application code or requires modifications to it.

Solution

The "application-focused" PaaS standardizes the developer's application stack via *Runtime-Dependencies*. The runtime-dependency is typically a set of rules that wraps whatever is needed to run the uploaded code. It examines the user-provided artifact in order to determine which dependencies to download and how to configure applications and runtime dependencies.

Consequently, when the developer pushes an application source code, the *Platform Controller* takes it and uses the runtime-dependency in order to detect the framework and runtime dependencies and it encapsulates the application along with runtime dependencies into an executable *Container Image*. Then, it runs the containers and manages them using a *Container Orchestrator*. Finally, the required process is started, the application is deployed and a URL is given back to the developer.

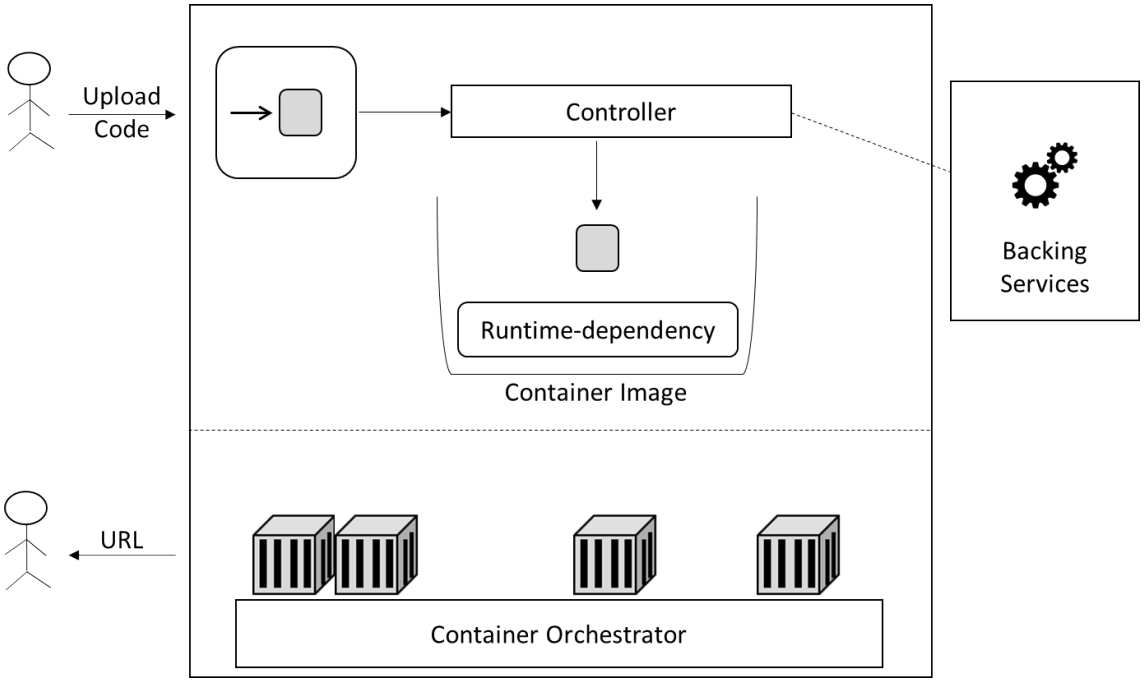
Usually, developers need to make use of additional services in their applications, which are referred to as *backing services* (databases, message queues, email services, etc.). In this solution, applications can easily be bound to a wide set of backing services that are provided and wired completely by the platform, and can be available on-demand. Basically, developers do not have influence on that and have to accept it as it is.

With respect to the uploaded application architecture, it should follow a set of guidelines that make it cloud-native, which facilitates its deployment to cloud platforms. Designing a cloud-native application is nothing more than building your application in a way that takes advantage of the cloud [Hof16]. These cloud-native applications are purposefully designed to be infrastructure unaware, meaning that they are decoupled from the underlying infrastructure and free to move as required [Win16], which meets at the end the purpose of PaaS, to which the applications will be uploaded. In order to optimize building this kind of applications, a collection of design principles that describes the rules and guidelines to be followed, has been developed and referred to as *12-factor app* [Twe]. These rules concentrate mainly on speed, safety, and scale, making no assumptions about the environments, to which they will be deployed. This in turn paves the way for the underlying cloud platforms to deploy cloud-native applications fast and consistently.

Another specialization to this solution is function-as-a-service (FaaS). When talking about uploading or deploying only a piece of code or a function to the cloud, the same fundamental ideas and concepts of the general solution discussed before are basically addressed. However, what makes FaaS special, is its small event-driven functions, which are only run when they are triggered by an event. Here, the developers do not push a whole application to the cloud, they only push small units of works or functions and tell the system how to knit them together.

At the end, whether developers want to deploy a function or a cloud-native application to the cloud, they do not have to be concerned about the underlying infrastructure and container construction. They can build, test, deploy and scale functions or applications easily by uploading their code to the platform that takes care of and facilitates everything else for them.

Sketch



Consequences

Using BYOCD pattern, developers can focus on the business logic of their applications and do not have to care about which container and middleware functions to use. As a result, the operating complexity is decreased and the development productivity is increased, which makes the deployment of functions and applications easy and fast. Moreover, the platform can scale applications (horizontally and vertically) and handle the wiring of backing services automatically. Hence, following this pattern, the platform automatically runs everything, which leads to a higher operational efficiency and less chance of human error.

However, the flexibility of this solution can be low compared to configuring your own development environment. The variety of available runtime-dependencies might be limited. Consequently, the choice to use any language, framework and data store you want, might be restricted. Besides, generally talking, the portability of applications from one platform to another is limited using this pattern. The prerequisite of using certain platform-supported runtime-dependencies, makes it difficult to run the same application on various application-focused platforms without modifying these runtime-dependencies.

Related Patterns

- *Distributed application* [FLR+14]: Cloud-native applications are generally divided into multiple components in order to be distributed among different resources, which significantly simplifies management tasks, such as scaling, failure resiliency, and load-balancing.

This pattern covers different approaches for the decomposition of the application's functionality into multiple independent components that provide a certain function. It also describes how the decomposed application components may be combined to a cloud application.

- *Loose coupling* [FLR+14]: The distributed nature of cloud-native applications enforces the need to avoid dependencies between application components in order to scale components independently and to avoid that failing components impact each other.
- *Stateless component* [FLR+14]: Scaling-out applications should ideally rely on external state information, either passed with every request or retrieved from a storage offering. Consequently, the applications deployed using BYOCD pattern should not keep any internal state as described by the stateless component pattern, in order to simplify scaling them by the platform.
- *Service deployment platform* [Ric]: This pattern describes the use of a deployment platform, which is an automated infrastructure for application deployment that provides a service abstraction, after the developer has architected the system as a set of services that need to be packaged and deployed.
- *Serverless deployment* [Ric]: This pattern describes the use of a deployment infrastructure that hides any concept of servers, physical or virtual hosts, or containers, after the developer has architected the system as a set of services that need to be packaged and deployed. The developer uploads the service's code and the infrastructure takes and runs it.

Known Uses

Many platforms already support the BYOCD pattern. Here, the cases of *Cloud Foundry* and *Heroku* are covered. Starting with Cloud Foundry, it is an open source cloud platform-as-a-service (PaaS), on which developers can build, deploy, run and scale applications [SSC17]. In Cloud Foundry, runtime-dependencies are achieved via *buildpacks*. A buildpack is composed of a set of scripts that provides detection of framework and runtime support, application compilation and application execution. These buildpacks can either be located remotely (e.g., on GitHub) or stay locally on PaaS by means of a process to be packaged and uploaded for offline use [Docc].

The life-cycle of a buildpack contains three stages: detect, compile and release. These life-cycle stages of a buildpack are known as staging in CF parlance [Win17]. Consequently, when a developer deploys a standalone application artifact to Cloud Foundry via *Cloud Controller*, it stages the pushed application and produces a *droplet*, which is an encapsulated version of the application along with all of the required runtime components and dependencies [Win17]. This droplet incorporated with a stack (filesystem) constitutes a *container image*. The resulting container images are run then as isolated containerized processes in Cloud Foundry using the container scheduler and orchestrator *Diego*. This overall process is shown in Figure 3.1.

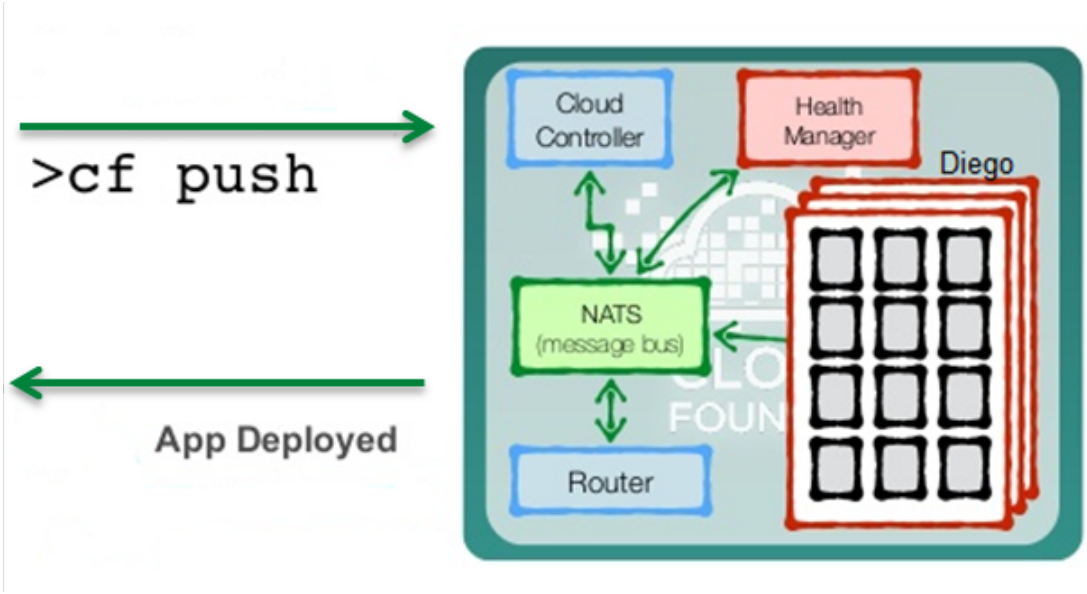


Figure 3.1: Overall Process of Deploying an App to Cloud Foundry [Mar17]

At the end, the buildpack model used in Cloud Foundry leads to a differentiation in roles between the platform operator and the developer [Win17]. The platform operator provides buildpacks, which are then consumed by applications pushed by developers in order to be run and deployed.

Cloud Foundry provides many additional important features such as dynamic routing and load balancing, scaling, service discovery, centralized log aggregation, blue/green deployments and built-in resilience and fault tolerance [Win17].

Moving to Heroku, it is a cloud platform-as-a-service (PaaS) that lets developers build, deliver, monitor and scale applications [Plab]. Heroku is similar in principle to Cloud Foundry. Also here, runtime-dependencies are achieved via *buildpacks*. A buildpack is used as a script to set up the runtime for a language or framework on the platform. These buildpacks can either be defined as custom buildpacks by developers or consumed directly as built-in buildpacks.

A buildpack here typically consists of three core scripts: detect, compile and release. When the developer uploads an application to Heroku using *Git*, it processes the pushed application and produces a *Slug*, which is a pre-packaged copy of the application after it has been processed by the buildpack [KG13]. Using these slugs, Heroku will provision out new *Dynos*. These dynos are isolated smart containers, which provide the environment required in order to run an application [Plab]. The resulting containers are then run, managed and coordinated using the container orchestrator *Dyno Manager*. This overall process is shown in Figure 3.2.

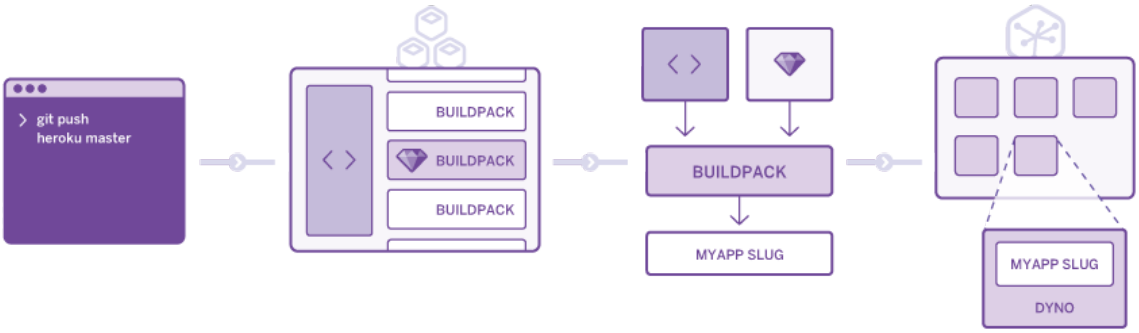


Figure 3.2: Overall Process of Deploying an App to Heroku [Plab]

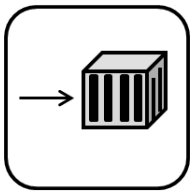
Heroku also enables additional beneficial features such as scaling, code and data rollback, continuous delivery, GitHub integration and App metrics [Plab].

Both platforms, Cloud Foundry and Heroku, provide scaling for applications, either horizontally by adjusting the number of application instances or vertically by adjusting the disk space limit or memory limit for each application instance. They also enable logging, monitoring and built-in resiliency. They automate the recovery of failed applications, components, and processes via self-healing mechanisms.

In conclusion, whether developers use Heroku or Cloud Foundry, they never deal with creation and orchestration of containers, which in turn removes complexity and increases efficiency. This also enables easy and continuous deployment of applications and DevOps-friendly workflows.

3.2 Bring Your Own Container Pattern (BYOCR)

A platform to which the developer uploads his or her application container and asks to run it on the cloud, in a way that allows the developer to have more control over the container construction and application runtime configuration.



How can a developer have more control and focus on creating the application container(s) that can be moved between different environments by himself or herself, and maintain it in the future, instead on focusing only on the application business logic?

Context

Generally, developers tend to focus on controlling what goes on inside the application, while IT operations tend to focus on running, managing, securing and scaling applications. Over decades, the main challenge in application development lifecycle, has been bringing developers and IT operations team closer together. They usually spend hours trying to figure out why an application works in the development environment but not in a production setting [Hig17].

As a result, an approach is needed in order to guarantee the consistent behavior of applications across development and production environments, erasing the tension between development and IT operations team and supporting DevOps.

Forces

The developer should have the freedom to use any programming language to develop the application, on any operating system and infrastructure. At the same time the IT operations team should be able to run, manage and scale applications without disrupting the application code. This leads to the need for a mechanism that allows developers and IT operations to work together in order to build, ship and run their applications anywhere without the need to affect or modify each others' work.

Solution

First of all, let us introduce some important concepts about containerization technology. The basic concept that enabled containerization was *Linux Containers LxC*. LXC is a Linux operating system level virtualization method for running multiple isolated Linux systems on a single host [GJ16]. It provides kernel mechanisms such as namespaces and cgroups in order to isolate processes and control resources on a shared operating system [PL15]. Then, technologies like *Docker*, which is an example of *Application Containers*, appeared in order to ease the leverage of LXC capabilities and provide additional high-level tooling and services.

As a result, application containers essentially emerged in order to create a logical packaging mechanism that abstracts and decouples applications from the surrounding environment, so that they can be moved consistently between different environments. A *Container Image* is a lightweight, stand-alone, executable package of a piece of software that includes everything needed to run it: code, runtime, system tools, system libraries, and settings [Doca]. Based on the image, many containers can begin their lifecycle. Thus, a *Container* is a running instance of a container image [Pah15].

The "container-focused" PaaS, sometimes referred to as CaaS (Container-as-a-service), provides a complete container environment for deploying and managing containers, applications and clusters. It consists mainly of a *Container Image Catalog*, which is a repository for storing container images that can either be hosted by a third-party or as a public/private catalog, a *Container Orchestrator*, which is the core of this solution that differentiates one platform from the other, and a set of developer tools and *APIs*.

The container orchestrator is the substantial component that allows for the management of complex container architectures. It is responsible for tasks such as container deployment, cluster management, scaling, reporting and lifecycle management. Using it, developers can build and maintain containers themselves.

A container orchestrator mainly consists of two types of nodes: *Main Nodes* and Servant Nodes. The main nodes are responsible for managing and orchestrating the servant nodes, by handling API calls and reacting in order to assign workloads to the servant nodes, on which containerized applications will be run.

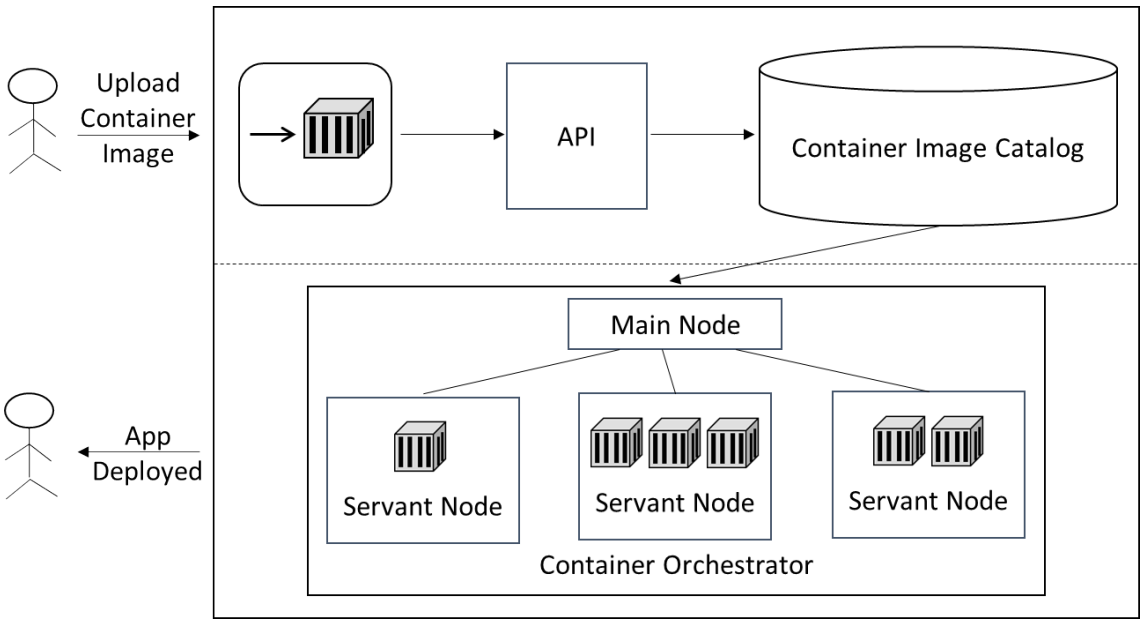
Consequently, the developer encapsulates his or her application code, runtime, system tools, system libraries, and settings within one package, which is called a *Container Image*. Subsequently, he or she pushes the resulting container image to the platform container image catalog using an *API*. Afterwards, the container image is pulled via the main node and tasks are assigned to servant nodes in order to run *Containers*, which are running instances of a container image. Lastly, the required process is started and the application is deployed.

Considering the *backing services* (databases, message queues, email services, etc.) that a developer may use, they in turn can also be deployed as container images to the cloud using this solution and then managed to be connected to the container images belonging to other application parts via configurations provided by the developer.

With respect to the uploaded application architecture, on one hand, it can be cloud-native following a set of guidelines. As mentioned earlier, making use of the design principles and rules described in the *12-factor app* [Twe], creating cloud-native applications can be optimized. This type of applications can be efficiently deployed using BYOCR pattern by wrapping the different services and application parts into container images, deploying them and managing and scheduling them on the platform container orchestrator. On the other hand, nothing prevents deploying monolithic applications using this pattern too. Generally speaking, developers can wrap the application along with all its internal layers, libraries and components into a single container image and deploy it to the cloud.

At the end, this solution provides the needed infrastructure to run application containers with providing more flexibility and control over applications to developers, and that is by letting them build and deploy their own application containers.

Sketch



Consequences

Using BYOCR pattern, portability of container images between computer hosts is achieved, which guarantees that what you run in development is the same as what you run in production environment. This in turn facilitates testing cycles and bug tracking, considering the fact that there is no difference between running an application locally, on a test server, or in production. It also enables DevOps and ensures to deliver software faster by erasing the old "it worked on my machine" challenge between development and IT operations teams [Hig17] [Arm17].

Moreover, this solution provides higher flexibility in terms of configuring your own development environment. Here, developers are not limited by what is supported by the provider. They are free to choose their desired programming languages, frameworks and datastores. Besides, the platform can scale applications (horizontally and vertically) automatically.

However, following this pattern, more responsibility and work remains with the developer, who should take care of creating container images, deploying, managing and maintaining them. The developer is also responsible for the backing services and making sure that the required instances are started with the appropriate scaling settings. This all may lead to more complexity, potential vulnerabilities and time consumption.

Related Patterns

- *Monolithic Architecture* [Ric]: This pattern describes the deployment architecture of an application as a monolithic architecture such as a single Java WAR file or a single directory hierarchy of Rails or NodeJS code. BYOCR pattern can be used to wrap a monolithic application inside a container and deploy it to the cloud.
- *Distributed application* [FLR+14]: Cloud-native applications are generally divided into multiple components in order to be distributed among different resources, which significantly simplifies management tasks, such as scaling, failure resiliency, and load-balancing. This pattern covers different approaches for the decomposition of the application's functionality into multiple independent components that provide a certain function. It also describes how the decomposed application components may be combined to a cloud application. Hence, BYOCR can be used to wrap the different components of cloud-native applications inside different containers and deploy them to the cloud.
- *Loose coupling* [FLR+14]: Considering the fact that BYOCR pattern can be used to deploy cloud-native applications, the distributed nature of these applications enforces the need to avoid dependencies between application components in order to scale components independently and to avoid that failing components impact each other.
- *Service instance per container* [Ric]: This pattern describes the solution of packaging the service as a container image and deploy each service instance as a container to the platform, after the developer has architectured the system as a set of services that need to be packaged and deployed.
- *Stateless component* [FLR+14]: Scaling-out applications should ideally rely on external state information, either passed with every request or retrieved from a storage offering. Consequently, the applications that do not keep internal state as described by the stateless component pattern, can be easily deployed using BYOCR pattern.

Known Uses

Various platforms follow the BYOCR pattern. Since the orchestration platform is the core of this solution, the two cases of *Kubernetes "K8s"* and *Docker Swarm* are covered here. Before diving into the details, it is important to mention the lowest layer software in the orchestration platform that enables all of containerization and orchestration. It is the layer that manages starting and stopping containers, which is referred to as the *Container Runtime* [Liu17]. The most widely known container runtime is *Docker*. Nevertheless, Docker is not the only one, there are others such as rkt, containerd, and lxd.

Starting with Kubernetes, it is an open-source platform designed to automate deploying, scaling, and operating application containers [Kub]. Docker is the most common runtime used in Kubernetes. However, it also supports the usage of other container runtimes such as rkt and containerd, utilizing the Container Runtime Interface (CRI) [Liu17]. The overall architecture of Kubernetes is shown in Figure 3.3. It mainly consists of a *master server* and multiple *nodes* (previously known as minions). The master along with the nodes constitute a *container cluster*. The master is the server that is responsible for the Kubernetes

management tasks [Hig17]. It is composed of three processes: *the API service*, *the replication controller* and *the scheduler*. The nodes are the worker machines and are managed by the master components. Each node runs three services: *the kubelet*, *the kube-proxy* and *the Docker engine*. The master and nodes communicate using Kubernetes APIs. The master decides what runs on all of the cluster's nodes. This can include scheduling workloads such as containerized applications and managing the workloads' lifecycle; scaling; and upgrades [Plaa]. It also manages network and storage resources for those workloads.

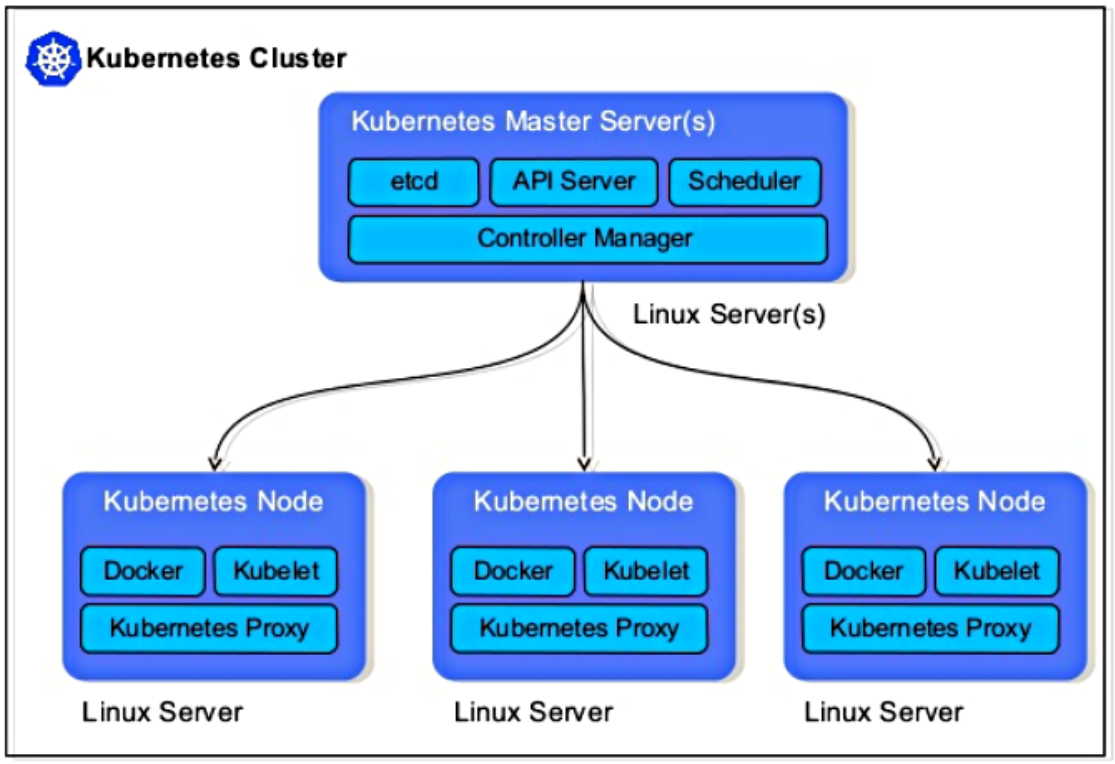


Figure 3.3: Kubernetes Architecture [SCH17]

A very important concept in Kubernetes is the *pod*. The pod represents the smallest unit of deployment, i.e., a single instance of an application in Kubernetes, which might consist of either a single container or a small number of containers that are tightly coupled and that share resources [Kub]. Docker is the most common container runtime used in a Kubernetes Pod, but Pods support other container runtimes as well. Thus, each node may run multiple pods and the master automatically schedules the pods across the nodes.

Kubernetes enables manual as well as automatic horizontal scaling of applications, by changing the number of Pods that encapsulate the application's container(s). It also provides a self-healing mechanism to restart failed containers, replace and reschedule containers when nodes die and kill containers that do not respond to user-defined health check [Kub]. Besides, it automates application configuration in the form of service discovery

and secrets. Moreover, it provides a set of APIs that results in custom deployment workflows such as rolling updates, canary deploys, and blue-green deployments [Hig17].

Moving to Docker Swarm, it is a tool used to cluster, schedule and orchestrate Docker containers. It enables IT administrators and developers to establish and manage a cluster of Docker nodes as a single virtual system [Docb]. Figure 3.4 shows the overall architecture of Docker Swarm. It mainly contains two types of nodes: *managers* and *workers*. Applications here are called *services*. A node is any machine installed and able to run containers (services). Manager nodes dispatch units of work to worker nodes. It is possible to have multiple manager nodes, but they elect one node to be the *leader*, which will perform all the orchestration and cluster management tasks. Worker nodes in turn receive and execute tasks dispatched from manager nodes. By default, manager nodes are also Worker nodes, yet they can be configured to run manager tasks exclusively and not accept any workload, therefore be manager-only nodes [Docb]. In order to maintain the desired state of the cluster, each worker node notifies the state of its tasks to the manager.

Docker Swarm does not enable auto-scaling of services directly. The developer can declare the number of tasks to run for each service and when manually scaling up or down, the Swarm manager automatically adapts by adding or removing tasks [Docb]. However, it provides load balancing capabilities by dividing the workload on the nodes and ensuring that there are sufficient resources for distributed containers. It also supports service discovery by assigning each service a unique DNS name and enabling developers to access services via their names. Moreover, it ensures rolling-back and health checks capabilities.

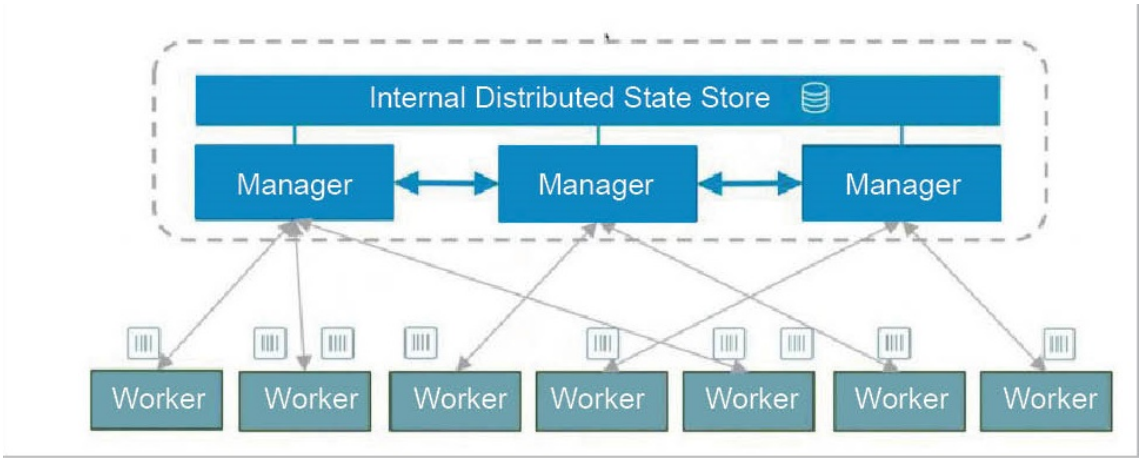


Figure 3.4: Docker Swarm Architecture [Rai16]

4 Evaluation of PaaS Deployment Patterns

In order to validate and evaluate the two introduced patterns in Chapter 3, an effort will be made to come up with different comparison matrices that indicate which pros and cons can be gained by applying the two approaches (BYOCD and BYOCR), using Cloud Foundry and Kubernetes as examples, on different application types. Then, the focus will be on the case of cloud-native applications in order to deploy a Spring CQRS (Command-Query-Responsibility-Segregation) application using the two created patterns and discuss the results. It is important to mention that Cloud Foundry is basically capable of working as a plain container orchestrator, i.e., Docker containers can be pushed directly to CF. However, this is not its initial idea (and probably not its strength). Consequently, within this thesis, CF is only used in its purpose as an application-focused platform.

4.1 Main Application Types

Here, three main application types that may be deployed to the cloud are differentiated: non-cloud-enabled applications, cloud-enabled applications and cloud-native applications.

4.1.1 Non-Cloud-Enabled Application

A non-cloud-enabled application describes a traditional application that was originally developed to run in a traditional data center, in a way that makes it not possible to run this application in a cloud environment. In other words, this application is incapable of running inside a container, because it has some characteristics, which prevent running it in a cloud environment (e.g., having multiple processes application, in which all the processes need to run on one particular host, with a shared file system and a GUI). As a result, a set of steps have to be done in order to make this kind of applications cloud-enabled [ASL13]. As an example, a Windows GUI-based application is a non-cloud-enabled application.

4.1.2 Cloud-Enabled Application

A cloud-enabled application is an application that was not originally developed to run in the cloud, i.e., it was primarily developed to run in a traditional data center. However, it has characteristics that can be changed or customized in order to run it in a cloud environment

[ASL13]. As an example, a legacy Java EE application that is not divided into microservices, but still can be run as a single process application; is a cloud-enabled application.

4.1.3 Cloud-Native Application

A cloud native application is an application that has already started being developed specifically for the Cloud [ASL13]. That means that it has been developed with the cloud principles of multi-tenancy, elastic scaling, statelessness, loose coupling, and easy integration and administration in its design. As discussed before, in order to facilitate building and managing this kind of applications, a number of identified principles and rules have been defined and referred to as *12-factor app* [Twe].

4.2 Matrices of Comparison

Taking into consideration the three mentioned application types, the following matrices that summarize the pros and cons of deploying those applications to both Cloud Foundry and Kubernetes are concluded:

Non-cloud-enabled App	Deployment
CF	• Very unlikely • Requires a specific buildpack to be run • Requires to be run as a single process app
K8s	• Modify to run in a container (migration path) • Requires to be run as a single process app

Tendency

Figure 4.1: Comparison Matrix of Non-cloud-enabled Applications

Cloud-enabled App (runs in a container)	Deployment
CF	• Requires “statelessness” as application behavior • Theoretically stateful applications are still runnable as single instances with binary BPs, however they don’t scale
K8s	• Containers as “easy wrap” black boxes • Capability of using the concept of “StatefulSet”, which is used to manage stateful applications

Figure 4.2: Comparison Matrix of Cloud-enabled Applications

Cloud-native App	Developer Responsibility	Auto-recovery	Manual Scaling
CF	• Push binary • Specify buildpack	• Automatically enabled • Process monitoring • Opt. specify health endpoints	• Supported (Horizontal & Vertical)
K8s	• Build container (Docker file) • Use container registry • Configure deployment (create pods & expose services)	• Automatically enabled • Process health check • Opt. specify liveness & readiness probes	• Supported (Horizontal & Vertical)

Figure 4.3: Comparison Matrix of Cloud-native Applications

Cloud-native App	Auto Scaling	Blue/Green Deployment (zero-downtime)	Load Balancing
CF	• Not a CF Open Source feature • Available through community plugins or in the Pivotal (PCF) implementation	• Manual steps with routes: ○ Push an App “Blue” ○ Update App and Push “Green” ○ Map Original Route to Green ○ Unmap Route to Blue ○ Remove Temporary Route to Green • CLI extensions available (marketplace)	• OOTB for manual & auto-scaling • Default GoRouter
K8s	• OOTB (out-of-the-box) horizontal pod autoscaling • Automatically scales the number of pods based on observed CPU utilization • Creation of a horizontal pod autoscaler using K8s API resource • Vertical autoscaling not implemented	• Out-of-the-box: ○ Deploy & test the new version ○ Switch incoming traffic ○ Shutdown old version if everything is working successfully	• OOTB for manual & auto-scaling • Ingress as the internal load-balancing method • Service of the LoadBalancer type as an alternative

Figure 4.4: Comparison Matrix of Cloud-native Applications - Continued

Cloud-native App	Runtime Support	Risks (potenial human errors)	Backing Services
CF	• Default set of buildpacks: ○ Binary ○ Go ○ Java ○ .NET Core ○ Node.js ○ PHP ○ Python ○ Ruby ○ Staticfile • Marketplace • In update: restage the app to update BPs & make the change effective	Needs to provide: • Location of BP • Application code => Less chance of human error	If available: • Very easy • Direct injection of connection properties If unavailable: • More difficult • User-provided services • Service binding to be done manually
K8s	N/A (BYOCR) • In update: new image	Needs to provide: • Base image • Runtime in image • Application code => Potential vulnerabilities	N/A (no concept of backing services) • Either running in container (manual handling of connection) • Or running external (manual handling of connection)

Figure 4.5: Comparison Matrix of Cloud-native Applications - Continued

4.3 Deploying a Sample CQRS Application Using the Two Patterns

A cloud-native sample application has been used in order to validate the two patterns, via deploying it into *Cloud Foundry* and *Kubernetes*. The CQRS Spring application consists mainly of three microservices, a service registry, and two backing services. The three microservices are: ToDoCommandService, ToDoQueryService, and ToDoUIService. The two backing services include RabbitMQ and MySQL. The interactions between the microservices are managed using a ToDoServiceRegistry. The microservices use the service registry in order to register themselves and look up other services, with which they can communicate directly. The ToDoCommandService uses a RabbitMQ service as a message broker in order to manage the messages in a queue. The ToDoQueryService in turn uses a MySQL service in order to store the ToDoList and update the database whenever a command has been processed. The overall architecture of the cloud-native application is shown in Figure 4.6.

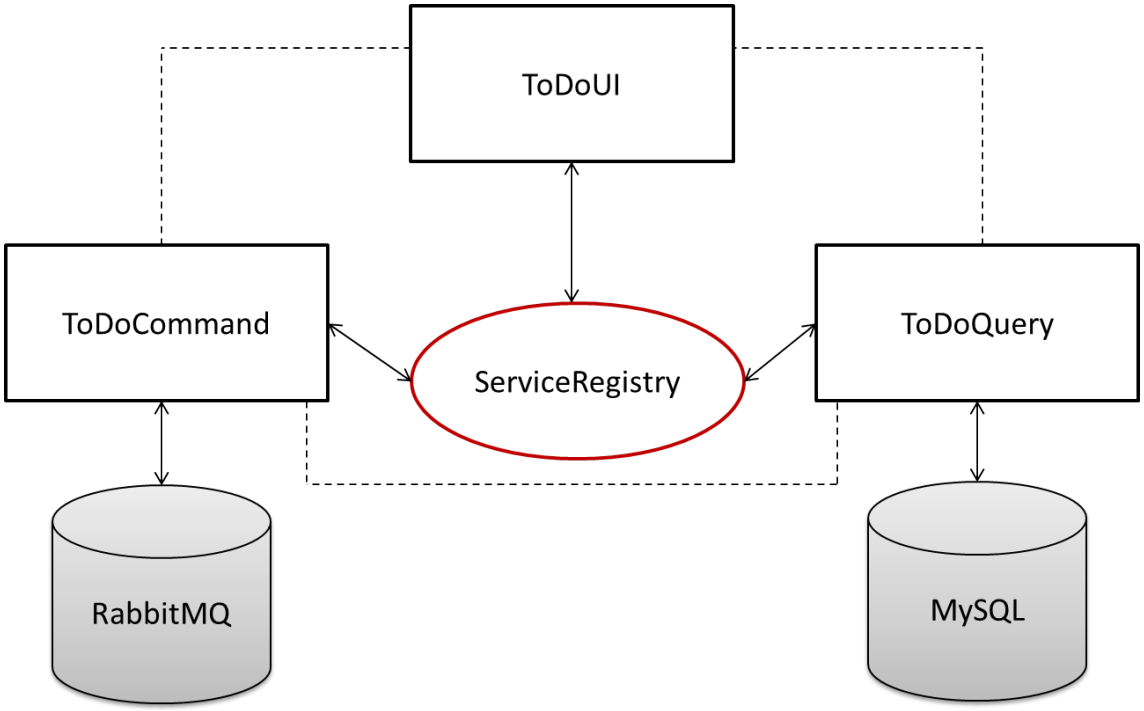


Figure 4.6: Sample Cloud-Native App Architecture

4.3.1 Deploying the Sample App to CF

Throughout this practical example, Pivotal Cloud Foundry is used, as it is the easiest available version. In order to deploy the sample CQRS application to PCF, multiple steps are needed. Within this section, the required steps are outlined in detail. As a tool that facilitates running *PCF* locally, *PCF Dev* is used. As an example of the necessary commands, the commands performed on the ToDoQueryService are described.

The first step to be performed, is to configure the application properties in the application.yml file for each of the microservices and the service registry in a suitable way, in order to manage registering the application into the service registry.

Next, the dependency tree should be framed and the needed components, of each of the microservices as well as the service registry, should be downloaded by executing the following Maven command:

```
mvn clean install
```

The following step that should be done is to login to PCF locally and push each of the microservices and the service registry using the created jar file after performing the previous step. The following cmd commands are used in order to push the ToDoQueryService as an example:

```
cf login -a https://api.local.pcfdev.io –skip-ssl-validation
```

```
cf push ToDoQueryService -m 1G -p target\ToDoQueryService-0.0.1-SNAPSHOT.jar
```

An important step now is to create the backing services, bind them to the related microservices and restage the application in order to make the change effective. As an example, the cmd commands used in order to create the MySQL service and bind it to the ToDoQueryService are shown here:

```
cf marketplace
```

```
cf create-service p-mysql 512mb my_mysql
```

```
cf bind-service ToDoQueryService my_mysql
```

```
cf restage ToDoQueryService
```

It is important to mention that the previous steps of pushing an application to cloud foundry, can also be done by providing a *manifest.yml* configuration file, which defines the deployment options to be used.

Afterwards, the created services and pushed applications can be checked using the following cmd commands:

```
cf services
```

```
cf apps
```

Ultimately, after pushing the applications, creating the services and binding them to the suitable applications, the desired application can be reached for using in the Browser.

The deployed CQRS application into Cloud Foundry has been tested successfully against the following features:

- Recovery by a microservice failure
- Recovery by a backing service failure
- Debugging and Logging
- Horizontal and Vertical manual-scaling

4.3.2 Deploying the Sample App to Kubernetes

Within this section, the various steps, which are needed in order to deploy the sample CQRS application to K8s are described in detail. As a tool that facilitates running *Kubernetes* locally, *Minikube* is used. As an example of the necessary commands, the commands performed on the ToDoQueryService are outlined.

The first important step is to write a *Dockerfile* for each of the microservices and the service registry. These files contain the commands that will be used in order to build *Images*, which in turn will produce *containers* when they are run. Listing 4.1 shows an example of the Dockerfile of the ToDoQueryService:

1. The image inherits from openjdk, the Docker official base image for Java.
2. A new directory is created.
3. The current directory is changed to the created directory to run next commands.
4. The compiled binary (jar file) is copied to the working directory in the container.
5. Finally, the runnable jar file is executed.

```
1  FROM openjdk:8-alpine
2  RUN mkdir -p /opt/ToDoQueryService
3  WORKDIR /opt/ToDoQueryService
4  COPY target/ToDoQueryService-0.0.1-SNAPSHOT.jar /opt/ToDoQueryService
5  CMD ["java", "-Dspring.profiles.active=k8s", "-jar",
       "ToDoQueryService-0.0.1-SNAPSHOT.jar"]
```

Listing 4.1: The Dockerfile of the ToDoQueryService

Another important and critical step now is to create the Service and Deployment *Yaml* configuration files for each of the microservices, the service registry as well as the backing services. These files contain a lot of necessary information for creating containers such as objects metadata, networking info, the used container image, resources requests and the information needed to connect the applications and the services together. Listing 4.2 shows an example of the Service.yml file of the ToDoQueryService.

```
1  apiVersion: v1
2  kind: Service
3  metadata:
4    namespace: todo-cn-app
5    name: todoqueryservice
6    labels:
7      app: todoqueryservice
8  spec:
9    type: NodePort
10   ports:
11   - port: 9082
12     name: todoqueryservice
13     targetPort: 9082
14   selector:
15     app: todoqueryservice
16   type: NodePort
```

Listing 4.2: The Service.yml File of the ToDoQueryService

Next, the application properties in the application.yml file should be configured in a suitable way depending on the deployment.yml files of the microservices, the service registry and the backing services, in order to manage registering the application into the service registry and accessing the backing services.

The following step that should be performed is to frame the dependency tree and download the needed components for each of the microservices as well as the service registry via executing the following Maven command:

```
mvn clean install
```

Then, a Docker image should be built for each microservice using its Dockerfile. The following cmd command is used to build the image of the ToDoQueryService as an example:

```
docker build -t todoqueryservice .
```

After that, the created images are pushed into a Docker registry using the following cmd commands:

```
docker login
```

```
docker push todoqueryservice
```

Within this practical example, the public Docker registry *Docker Hub* is used. However, this is not recommended for productive usage, because the operations team can not have control over this public registry.

Afterwards, the configuration changes are pushed to the K8s cluster and the resources are created using the following cmd command for each of the constructed Yaml files:

```
kubectl apply -f cloud-native-todo-sample\ToDoQueryService\deployment.yml
```

Later on, the created pods and services can be checked using the following cmd commands:

```
kubectl get pods –namespace todo-cn-app
```

```
kubectl get services –namespace todo-cn-app
```

Finally, after creating the pods and exposing the services, the desired application can be reached for using in the Browser.

The deployed CQRS application into Kubernetes has been tested successfully against the following features:

- Recovery by a microservice failure
- Recovery by a backing service failure
- Debugging and Logging
- Horizontal and Vertical manual-scaling
- Horizontal auto-scaling

4.4 Conclusion

Following the comparison matrices and the deployment of the sample CQRS application, the several parts that constitute each of the two patterns (BYOCD and BYOCR) have been validated and evaluated.

As a summary to what has been experienced during building up the comparison matrices and going through the various steps to deploy the sample application; the *runtime-dependencies* comprise the main component of the BYOCD solution, in which the developer uploads the application code and asks the platform to take care of everything else in order to run it. Nevertheless, the *container images* shape the major ingredient of the BYOCR solution, in which the developer uploads the application container and asks the platform to run it in the way he or she wants.

Furthermore, it can be concluded that the responsibility of the developer becomes less and less in the BYOCD pattern, such that he or she can focus on the application business code. This in turn leads to less operating complexity, more usability and less chance

of human error, counter to the BYOCR pattern. However, the flexibility to support any runtime dependency and the portability of container images is much higher with following the BYOCR pattern. Both patterns support scalability, load balancing, auto-recovery and zero-downtime deployments.

5 Discussion and Future Work

This final chapter summarizes the previous research and contributions of this work. Moreover, it discusses the results and concludes a decision tree that guides the choice of the proper pattern and platform. Ultimately, it provides possible further research in the future.

5.1 Summary

Studying and analyzing the various cloud deployment approaches, has gained a noticeable attention by enterprises these days. They are seeking to reach a knowledge about the main differences between the different deployment approaches and their major advantages and disadvantages. Furthermore, they aspire to have some beneficial guidelines that advise them about choosing one approach over the other. However, the world of IT today lacks the existence of a scientific research that studies, analyzes and outlines the multiple deployment approaches along with their different technologies in general solutions, and discusses the considerable differences and factors that may lead the decision to follow one of them over the other.

In order to fill the mentioned gap, this thesis has been introduced. A systematic literature review that investigates the state of the art approaches and technologies used to enable PaaS deployments in a thorough and unbiased manner has been conducted in Chapter 2. Proceeding from the results of the achieved SLR process, a scientific authoring, documentation and description of two new deployment patterns (BYOCD and BYOCR) have been outlined in Chapter 3. With a view to validate and evaluate the described patterns and considering the three different application types that may be deployed to the cloud (non-cloud-enabled, cloud-enabled and cloud-native applications), multiple comparison matrices that consider various aspects when applying the two patterns using CF and K8s as examples, have been concluded in Section 4.2, and the deployment of a sample CQRS application utilizing the two patterns has been accomplished in Section 4.3

5.2 Discussion

On the whole, it can be concluded that non of the approaches is the best or the most robust in general. The two approaches offer different choices with respect to various aspects such as developer responsibility, potential vulnerabilities, runtime support, backing services,

manual and auto-scaling, load balancing, auto-recovery, and zero-downtime deployment. It always depends on the enterprise needs, the application type and the technical and business considerations. However, a useful hint can be suggested in order to help enterprises enclose the factors that may affect their decision to follow the most suitable approach that achieves their application needs perfectly. The working team should always look at the project in hand and the problem they are solving and ask some questions such as:

- What is the type of application that is being dealt with? Is it a non-cloud-enabled, cloud-enabled or cloud-native application?

  When the application is non-cloud-enabled, then some work should be done in order to make it run in a container and the tendency here is toward the BYOCR pattern. Nevertheless, when the application is cloud-enabled, a discussion about its statefulness should be done. If it is stateful, then the orientation is to use BYOCR pattern. If it is stateless or a cloud-native application, then further investigation should be done answering the next questions.

- Is the required time in order to accomplish the deployment short, such that less responsibility and configuration complexity, less chance of human error and more ease of deployment have to be assigned to the developer?

  When the time to deliver is relatively short, then the complexity of the solution and the chance of human error should be low, which heads to the use of the BYOCD pattern. However, further discussion regarding the backing services and runtime support aspects has to be taken into consideration.

- Does the project need runtime dependencies, which are not provided as buildpacks by the platform or in the marketplace? If yes, does the enterprise intend to spend time on building custom buildpacks?

  When the project in hand requires runtime dependencies, which are not provided as buildpacks out there and the enterprise does not intend to spend time on customizing new buildpacks; then the propensity is to follow the BYOCR pattern. Otherwise, further consideration has to be done answering the next questions.

- Are the required backing services provided as cloud services by the platform?

  This question affects mainly the choice of the specific technology or platform to use. Weather a developer tends to use a provided service, link to external services provided by other platforms, provide the service as a user-provided service or package it in a container, depends on the capabilities that each platform offers.

- Is auto-scaling as an OOTB feature a critical aspect for the project?

  This question also does not affect directly the use of one of the patterns over the other. However, it affects the tendency to choose one platform over the other. For example, when the enterprise tends to use an out-of-the-box auto-scaling feature that can be used directly, then an evaluation of the various Cloud Foundry options should be done.

- Does the project need blue/green deployment as an OOTB feature?

  In this question as well, a decision to use one of the patterns over the other can not be properly made. A more convenient decision will be made depending on the capabilities of different platforms. For example, K8s provides more granular configuration options, which can not be done using CF. However, these options have not been covered within this thesis.

At the end, answering some critical questions such as the ones described above and following the resulted comparison matrices presented in Section 4.2, a proper decision can be made to utilize one of the two patterns using the most suitable platform. Figure 5.1 shows a decision tree, which summarizes the questions that should be asked and the paths that should be followed depending on the answers to those questions; in order to take the right decision to follow one of the patterns or evaluate various platforms within those patterns and choose the most suitable one.

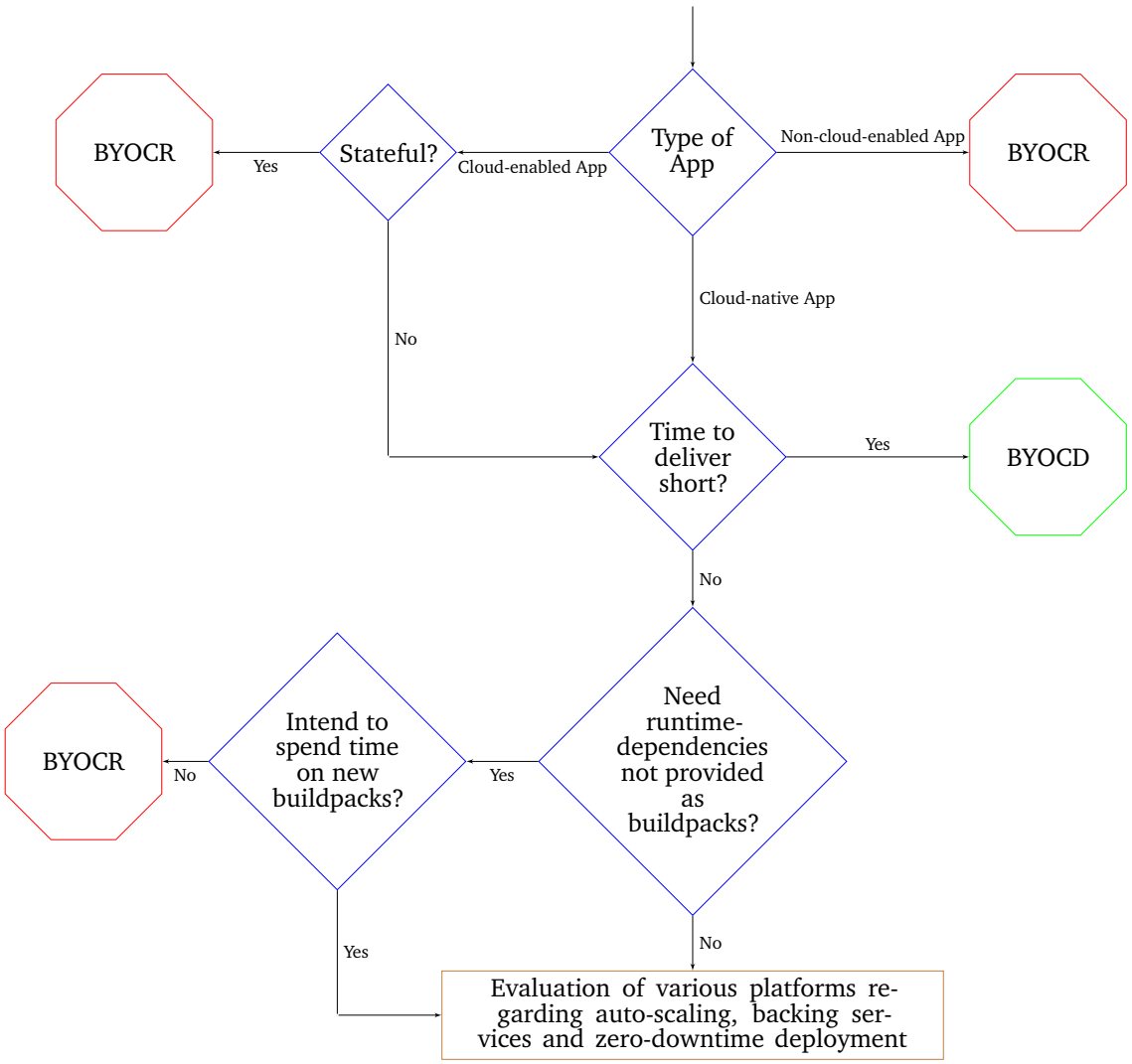


Figure 5.1: BYOCD or BYOCR Decision Tree

5.3 Future Work

Within this thesis, the security concerns have been discarded. This security aspect may constitute by itself a further research sector. Since security issues are some of the biggest concerns in cloud computing, it is important to analyze this challenge in detail within the two patterns and compare the two approaches with respect to that aspect.

Another interesting aspect to address, is the portability of the application from one pattern to the other. Even more motivating is to also study the portability of the application within the same pattern, but between different platforms and technologies. This will raise new questions regarding the steps to be done when converting a containerized application into application code with a suitable buildpack and vise versa. Moreover, it will pave the way to analyze situations in which an application may be moved between different runtime environments and study the challenges resulting from such a decision. Consequently, the portability aspect could be a starting point for further research.

Another research opportunity appears in the context of the effect of the application design patterns on the tendency toward choosing one of the introduced patterns in Chapter 3 over the other. This is a motivating aspect that should be studied in the future.

Furthermore, the serverless concept, its effect on the two introduced approaches and the possibility to support it within them, could be taken as a starting point for further research opportunity.

Ultimately, the research area of PaaS deployment approaches and technologies is quite interesting and the number of yearly publications might still increase over the next years. Therefor, a steady change in the whole culture may occur; bringing along new platforms, strategies and technologies. This in turn may have its effect on the introduced work and should be taken into consideration in the future.

Appendices

A Appendix

Item Id	Item	Value
DE1	Title	Cloud-Based Development Environments: PaaS
DE2	Year	2016
DE3	Author(s)	Mehmet Aydin and Nazim Perdahci and Bahadir Odevci
DE4	organization	Kadir Has University, Mimar Sinan Fine Arts University, Imonacloud.com
DE5	Country of study	Turkey
DE6	PaaS concepts & approaches	application runtime environments, database system and middleware
DE7	PaaS development & deployment technologies	Google App Engine, Microsoft Azure

Table A.1: Filled Data Extraction Form for [APO16]

Item Id	Item	Value
DE1	Title	Cloud Computing: Characteristics and Deployment Approaches
DE2	Year	2011
DE3	Author(s)	Z. Mahmood
DE4	organization	University of Derby (School of Computing & Mathematics)
DE5	Country of study	UK
DE6	PaaS concepts & approaches	runtime environment
DE7	PaaS development & deployment technologies	Google App Engine, Heroku, Engine Yard

Table A.3: Filled Data Extraction Form for [Mah11]

Item Id	Item	Value
DE1	Title	A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters
DE2	Year	2016
DE3	Author(s)	C. Pahl and S. Helmer and L. Miori and J. Sanin and B. Lee
DE4	organization	Free University of Bozen-Bolzano, Athlone Institute of Technology
DE5	Country of study	Italy, Ireland
DE6	PaaS concepts & approaches	Containers, application packaging and orchestration
DE7	PaaS development & deployment technologies	Cloud Foundry, Docker, Rocket, Kubernetes

Table A.2: Filled Data Extraction Form for [PHM+16]

Item Id	Item	Value
DE1	Title	Overview and analysis of cloud computing research and application
DE2	Year	2011
DE3	Author(s)	Y. Chen and X. Li and F. Chen
DE4	organization	Shanghai University (School of Management)
DE5	Country of study	China
DE6	PaaS concepts & approaches	–
DE7	PaaS development & deployment technologies	Saleforce AppExchange, Google App Engine

Table A.4: Filled Data Extraction Form for [CLC11]

Item Id	Item	Value
DE1	Title	Optimal Multitenant Designs for Cloud Apps
DE2	Year	2011
DE3	Author(s)	S. Bobrowski
DE4	organization	salesForce
DE5	Country of study	USA
DE6	PaaS concepts & approaches	container in which you simply deploy your app
DE7	PaaS development & deployment technologies	Heroku

Table A.5: Filled Data Extraction Form for [Bob11]

Item Id	Item	Value
DE1	Title	What are Developers' Preferences on Platform as a Service? An Empirical Investigation
DE2	Year	2013
DE3	Author(s)	A. Giessmann and K. Stanoevska-Slabeva
DE4	Organization	University of St.Gallen, SAP Research Center St. Gallen
DE5	Country of study	Switzerland
DE6	PaaS concepts & approaches	container platform, execution environment
DE7	PaaS development & deployment technologies	Google's App Engine

Table A.6: Filled Data Extraction Form for [GS13]

Item Id	Item	Value
DE1	Title	Characterizing and Evaluating Different Deployment Approaches for Cloud Applications
DE2	Year	2014
DE3	Author(s)	J. Wettinger and V. Andrikopoulos and S. Strauch and F. Leymann
DE4	organization	University of Stuttgart (Institute of Architecture of Application Systems)
DE5	Country of study	Germany
DE6	PaaS concepts & approaches	Cloud-enabled middleware solutions
DE7	PaaS development & deployment technologies	Cloud Foundry, Google App Engine, Amazon Elastic Beanstalk

Table A.7: Filled Data Extraction Form for [WASL14]

Item Id	Item	Value
DE1	Title	PaaS: New Opportunities for Cloud Application Development
DE2	Year	2013
DE3	Author(s)	B. Cohen
DE4	organization	Luth Computer Specialists
DE5	Country of study	USA
DE6	PaaS concepts & approaches	–
DE7	PaaS development & deployment technologies	Cloud Foundry, Azure, OpenShift, Heroku, Google App Engine, SalesForce Cloud

Table A.8: Filled Data Extraction Form for [Coh13]

Item Id	Item	Value
DE1	Title	Overview of PaaS and SaaS and its application in cloud computing
DE2	Year	2016
DE3	Author(s)	A. Singh and S. Sharma and S. R. Kumar and S. A. Yadav
DE4	organization	AMITY UNIVERSITY (Department of Electronics & Communication Engineering, Department of Computer Science & Engineering, Department of Information Technology)
DE5	Country of study	India
DE6	PaaS concepts & approaches	execution environment
DE7	PaaS development & deployment technologies	–

Table A.9: Filled Data Extraction Form for [SSKY16]

Item Id	Item	Value
DE1	Title	Microservices Architecture Based Cloudware Deployment Platform for Service Computing
DE2	Year	2016
DE3	Author(s)	D. Guo and W. Wang and G. Zeng and Z. Wei
DE4	organization	Tongji University (Department of Computer Science and Technology), Tongji Branch National Engineering & Technology Center of High Performance, The Key Laboratory of Embedded System and Service Computing
DE5	Country of study	China
DE6	PaaS concepts & approaches	Containerization
DE7	PaaS development & deployment technologies	Docker, Amazon EC2, Google App Engine

Table A.10: Filled Data Extraction Form for [GWZW16]

Item Id	Item	Value
DE1	Title	Towards a full-stack devops environment (platform-as-a-service) for cloud-hosted applications
DE2	Year	2017
DE3	Author(s)	Z. Li and Y. Zhang and Y. Liu
DE4	organization	Tsinghua University (School of Software)
DE5	Country of study	China
DE6	PaaS concepts & approaches	DevOps environments, Container
DE7	PaaS development & deployment technologies	Google App Engine, Google Borg, Docker, Cloud Foundry, Hadoop YARN, AWS Elastic Beanstalk, Microsoft Azure, Google Kubernetes, Heroku

Table A.11: Filled Data Extraction Form for [LZL17]

Item Id	Item	Value
DE1	Title	Dynamic Tailoring and Cloud-Based Deployment of Containerized Service Middleware
DE2	Year	2015
DE3	Author(s)	S. G. Sáez and V. Andrikopoulos and R. J. Sánchez and F. Leymann and J. Wettinger
DE4	organization	University of Stuttgart (Institute of Architecture of Application Systems)
DE5	Country of study	Germany
DE6	PaaS concepts & approaches	containers, middleware as a service
DE7	PaaS development & deployment technologies	Docker, AWS Beanstalk, Google App Engine, Microsoft Azure

Table A.12: Filled Data Extraction Form for [SAS+15]

Item Id	Item	Value
DE1	Title	Docker container security via heuristics-based multi-lateral security-conceptual and pragmatic study
DE2	Year	2016
DE3	Author(s)	A. R. Manu and J. K. Patel and S. Akhtar and V. K. Agrawal and K. N. B. S. Murthy
DE4	organization	Electronic city (MTech(IIIT), CORI-ISE Dept), Cisco Systems
DE5	Country of study	India
DE6	PaaS concepts & approaches	container as a service, security, lightweight container virtualization
DE7	PaaS development & deployment technologies	Docker, Kubernetes

Table A.13: Filled Data Extraction Form for [MPA+16b]

Item Id	Item	Value
DE1	Title	A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds
DE2	Year	2015
DE3	Author(s)	M. G. Xavier and I. C. D. Oliveira and F. D. Rossi and R. D. D. Passos and K. J. Matteussi and C. A. F. D. Rose
DE4	organization	Pontifical Catholic University of Rio Grande do Sul (PUCRS) (Faculty of Informatics)
DE5	Country of study	Brazil
DE6	PaaS concepts & approaches	containers, container-based architecture, Performance
DE7	PaaS development & deployment technologies	Docker

Table A.14: Filled Data Extraction Form for [XOR+15]

Item Id	Item	Value
DE1	Title	Containerization and the PaaS Cloud
DE2	Year	2015
DE3	Author(s)	C. Pahl
DE4	organization	Irish Centre for Cloud Computing and Commerce
DE5	Country of study	Ireland
DE6	PaaS concepts & approaches	Containers, lightweight virtualization, container-based application deployment, cluster orchestration, Lightweight Virtualization, Application Packaging
DE7	PaaS development & deployment technologies	Docker, Garden/Warden, Diego, Cloud Foundry, OpenShift, Azure, Heroku, Mesos, Kubernetes

Table A.15: Filled Data Extraction Form for [Pah15]

Item Id	Item	Value
DE1	Title	A study, analysis and deep dive on cloud PAAS security in terms of Docker container security
DE2	Year	2016
DE3	Author(s)	A. R. Manu and J. K. Patel and S. Akhtar and V. K. Agrawal and K. N. B. S. Murthy
DE4	organization	Electronic city (MTech(IIIT), CORI-ISE Dept), Cisco Systems
DE5	Country of study	India
DE6	PaaS concepts & approaches	containers, security, Cluster management
DE7	PaaS development & deployment technologies	Docker, Google compute engine, Amazon EC2, Kubernetes, Cloud Foundry, OpenShift

Table A.16: Filled Data Extraction Form for [MPA+16a]

Item Id	Item	Value
DE1	Title	Dynamically Compiled Artifact Sharing for Clouds
DE2	Year	2017
DE3	Author(s)	P. Patros and D. Dilli and K. B. Kent and M. Dawson
DE4	organization	University of New Brunswick, IBM Canada
DE5	Country of study	Canada
DE6	PaaS concepts & approaches	upload and write your code, upload your own OS image, containers, lightweight virtualization
DE7	PaaS development & deployment technologies	Docker, Cloud Foundry, IBM Bluemix, Pivotal

Table A.17: Filled Data Extraction Form for [PDKD17]

Item Id	Item	Value
DE1	Title	Building safe PaaS clouds: A survey on security in multitenant software platforms
DE2	Year	2012
DE3	Author(s)	Luis Rodero-Merino and Luis M. Vaquero and Eddy Caron and Adrian Muresan and Frédéric Desprez
DE4	organization	UMR CNRS, SysFera, Universidad Polite´cnica de Madrid (Facultad de Informa´tica), Hewlett-Packard Labs
DE5	Country of study	France, Spain, UK
DE6	PaaS concepts & approaches	container platform, security
DE7	PaaS development & deployment technologies	Google App Engine

Table A.18: Filled Data Extraction Form for [RVC+12]

Item Id	Item	Value
DE1	Title	A Review of Major Security Issues in Cloud Computing
DE2	Year	2016
DE3	Author(s)	Mitesh Sharma
DE4	organization	JIET College of Engineering (Department of CSE)
DE5	Country of study	India
DE6	PaaS concepts & approaches	security, portability
DE7	PaaS development & deployment technologies	Google App Engine, Windows Azure

Table A.19: Filled Data Extraction Form for [Sha16]

Item Id	Item	Value
DE1	Title	Serverless Computing: Current Trends and Open Problems
DE2	Year	2017
DE3	Author(s)	I. Baldini and P. Castro and K. Chang and P. Cheng and others
DE4	organization	IBM Research, Bentley University
DE5	Country of study	Australia, USA
DE6	PaaS concepts & approaches	Function-as-a-Service, serverless platforms
DE7	PaaS development & deployment technologies	OpenLambda, Project Kratos, Cloud Foundry, Docker

Table A.20: Filled Data Extraction Form for [BCC+17]

Item Id	Item	Value
DE1	Title	Cloud Computing Platform Based on the Docker
DE2	Year	2016
DE3	Author(s)	Z. WANG and L. XUE and Y. LUO
DE4	organization	Anhui Sun Create Electronic Co., Hefei University of Technology (Computer College)
DE5	Country of study	China
DE6	PaaS concepts & approaches	lightweight container engine
DE7	PaaS development & deployment technologies	Docker

Table A.21: Filled Data Extraction Form for [WXL16]

Item Id	Item	Value
DE1	Title	Virtualization in Cloud Computing
DE2	Year	2016
DE3	Author(s)	P. Thakur and M. Mahajan
DE4	organization	CGC College of Engineering Landran (CSE Department)
DE5	Country of study	India
DE6	PaaS concepts & approaches	–
DE7	PaaS development & deployment technologies	Force.com, Google Apps, Microsoft Azure

Table A.22: Filled Data Extraction Form for [TM16]

Item Id	Item	Value
DE1	Title	A Business Driven Scalable Cloud Computing Service Platform (PaaSXpert)
DE2	Year	2017
DE3	Author(s)	A. Sreeramaneni and B. Seo and K. Chan
DE4	organization	Crossent Inc., Seoul National University of Science & Technology
DE5	Country of study	Korea
DE6	PaaS concepts & approaches	performance, scalability, use your own code
DE7	PaaS development & deployment technologies	cloud foundry, BOSH, IBM Bluemix, Pivotal, OpenShift

Table A.23: Filled Data Extraction Form for [SSC17]

Item Id	Item	Value
DE1	Title	FUJITSU Cloud Service K5 PaaS Digitalizes Enterprise Systems
DE2	Year	2017
DE3	Author(s)	O. Matsumoto and K. Kawai and T. Takeda
DE4	organization	Fujitsu
DE5	Country of study	Japan
DE6	PaaS concepts & approaches	Container
DE7	PaaS development & deployment technologies	Cloud Foundry

Table A.24: Filled Data Extraction Form for [MKT17]

Item Id	Item	Value
DE1	Title	Smuggling Multi-Cloud Support into Cloud-native Applications using Elastic Container Platforms
DE2	Year	2017
DE3	Author(s)	Nane Kratzke
DE4	organization	Lübeck University of Applied Sciences
DE5	Country of study	Germany
DE6	PaaS concepts & approaches	Elastic container platforms, Cloud-native applications, container cluster
DE7	PaaS development & deployment technologies	Kubernetes, Docker Swarm, Apache Mesos

Table A.25: Filled Data Extraction Form for [Kra17]

Item Id	Item	Value
DE1	Title	Building Software Defined Systems on HPC and Clouds
DE2	Year	2017
DE3	Author(s)	Hyungro Lee
DE4	organization	Indiana University (School of Informatics and Computing)
DE5	Country of study	USA
DE6	PaaS concepts & approaches	Container Technology, Functions-as-a-Service
DE7	PaaS development & deployment technologies	Docker

Table A.26: Filled Data Extraction Form for [Lee17]

Item Id	Item	Value
DE1	Title	Performance Comparison between Light Weight Virtualization Using Docker and Heavy Weight Virtualization
DE2	Year	2017
DE3	Author(s)	V. Gupta and K. Kaur and S. Kaur
DE4	organization	U-Net Solutions, AD College (Computer Science and Applications)
DE5	Country of study	India
DE6	PaaS concepts & approaches	lightweight container technology
DE7	PaaS development & deployment technologies	Docker

Table A.27: Filled Data Extraction Form for [GKK17]

B Appendix

Year	Num. PaaS Related Studies
2011	3
2012	1
2013	2
2014	1
2015	3
2016	9
2017	8

Table B.1: Filled Data Synthesis Form for RQ1

Table B.2: Filled Data Synthesis Form for RQ2

Study	Authors	Institution	Country of Institution
[APO16]	M. Aydin N. Perdahci B. Odevci	Kadir Has University Mimar Sinan Fine Arts University Imonacloud.com	Turkey
[PHM+16]	C. Pahl S. Helmer L. Miori J. Sanin B. Lee	Free University of Bozen-Bolzano Free University of Bozen-Bolzano Free University of Bozen-Bolzano Free University of Bozen-Bolzano Athlone Institute of Technology	Italy Italy Italy Italy Ireland
[Mah11]	Z. Mahmood	University of Derby (School of Computing & Mathematics)	UK
[CLC11]	Y. Chen X. Li F. Chen	Shanghai University (School of Management)	China
[Bob11]	S. Bobrowski	salesForce	USA
[GS13]	A. Giessmann K. Stanoevska-Slabeva	University of St.Gallen SAP Research Center St. Gallen	Switzerland
[WASL14]	J. Wettinger V. Andrikopoulos S. Strauch F. Leymann	University of Stuttgart (Institute of Architecture of Application Systems)	Germany
[Coh13]	B. Cohen	Luth Computer Specialists	USA

Study	Authors	Institution	Country of Institution
[SSKY16]	A. Singh S. Sharma S. R. Kumar S. A. Yadav	AMITY UNIVERSITY (Department of Electronics & Communication Engineering) AMITY UNIVERSITY (Department of Electronics & Communication Engineering) AMITY UNIVERSITY (Department of Computer Science & Engineering) AMITY UNIVERSITY (Department of Information Technology)	India
[GWZW16]	D. Guo W. Wang G. Zeng Z. Wei	Tongji University (Department of Computer Science and Technology)	China
[LZL17]	Z. Li Y. Zhang Y. Liu	Tsinghua University (School of Software)	China
[SAS+15]	S. G. Sáez V. Andrikopoulos R. J. Sánchez F. Leymann J. Wettinger	University of Stuttgart (Institute of Architecture of Application Systems)	Germany
[MPA+16b]	A. R. Manu J. K. Patel S. Akhtar V. K. Agrawal K. N. B. S. Murthy	CORI-ISE Dept Electronic city Electronic city PESIT VTU Cisco Systems CORI ISE Dept, PESIT, Electronic city PES University	India

Study	Authors	Institution	Country of Institution
[XOR+15]	M. G. Xavier I. Oliveira F. Rossi R. Passos K. Matteussi C. D. Rose	Pontifical Catholic University of Rio Grande do Sul (Faculty of Informatics)	Brazil
[Pah15]	C. Pahl	Irish Centre for Cloud Computing and Commerce	Ireland
[MPA+16a]	A. R. Manu J. K. Patel S. Akhtar V. K. Agrawal K. N. B. S. Murthy	CORI-ISE Dept Electronic city Electronic city PESIT VTU Cisco Systems CORI ISE Dept, PESIT, Electronic city PES University	India
[PDKD17]	P. Patros D. Dilli K. B. Kent M. Dawson	University of New Brunswick University of New Brunswick University of New Brunswick IBM Canada	Canada
[RVC+12]	Luis Rodero-Merino Luis M. Vaquero Eddy Caron Adrian Muresan Frédéric Desprez	UMR CNRS, Universidad Polite´cnica de Madrid Hewlett-Packard Labs UMR CNRS, SysFera UMR CNRS UMR CNRS, SysFera	France, Spain UK France France France
[Sha16]	Mitesh Sharma	JIET College of Engineering (Department of CSE)	India

Study	Authors	Institution	Country of Institution
[BCC+17]	V. Ishakian	Bentley University	USA
	I. Baldini	IBM Research	Australia
	P. Castro	IBM Research	Australia
	K. Chang	IBM Research	Australia
	P. Cheng	IBM Research	Australia
	others	IBM Research	Australia
[WXL16]	Z. WANG	Anhui Sun Create Electronic Co.	China
	L. XUE	Hefei University of Technology (Computer College)	
	Y. LUO	Anhui Sun Create Electronic Co.	
[TM16]	P. Thakur	CGC College of Engineering Landran (CSE Department)	India
	M. Mahajan		
[SSC17]	A. Sreeramaneni	Crossent Inc.	Korea
	B. Seo	Crossent Inc.	
	K. Chan	Seoul National University of Science	
[MKT17]	O. Matsumoto	Fujitsu	Japan
	K. Kawai		
	T. Takeda		
[Kra17]	Nane Kratzke	Lübeck University of Applied Sciences	Germany
[Lee17]	Hyungro Lee	Indiana University (School of Informatics and Computing)	USA
[GKK17]	V. Gupta	U-Net Solutions	India
	K. Kaur	AD College (Computer Science and Applications)	
	S. Kaur	AD College (Computer Science and Applications)	

PaaS concept/approach	
Concepts	**Num. Studies**
containers/container platform/containerization/container as a service/container-based architecture/lightweight container virtualization	16
application packaging and orchestration/cluster management	4
middleware as a service/cloud-enabled middleware solutions	3
Function-as-a-Service/serverless platforms	2
runtime/execution environment	4
cloud-native applications	1
DevOps environments	1
security	4
performance	2
portability	1
scalability	1
Approaches	
upload your own OS-image/container	
upload your own code	

Table B.3: Filled Data Synthesis Form for RQ3

PaaS technology	Num. Studies
Cloud Foundry	10
Docker	13
Rocket	1
Garden/Warden	1
Diego	1
BOSH	1
Kubernetes	6
Docker Swarm	1
Amazon EC2	2
Mesos	2
Pivotal	2
IBM Bluemix	2
Heroku	5
SalesForce App Cloud	3
Engine Yard	1
Hadoop YARN	1
Google Borg	1
OpenShift	4
Amazon Elastic Beanstalk	3
Google App Engine	13
Microsoft Azure	7
OpenLambda	1
Project Kratos	1

Table B.4: Filled Data Synthesis Form for RQ4

Bibliography

[Ale78] C. Alexander. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York, 1978 (cit. on p. 35).

[Ale80] C. Alexander. *The Timeless Way of Building*. Oxford University Press, New York, 1980 (cit. on p. 35).

[APO16] M. Aydin, N. Perdahci, B. Odevci. "Cloud-Based Development Environments." In: *Encyclopedia of Cloud Computing*. Wiley-IEEE Press, 2016, pp. 62–69 (cit. on pp. 30, 65, 82).

[Arm17] J. Armstrong. *Be a Budget Hero with Docker Enterprise Edition*. Aug. 2017. URL: https://blog.docker.com/2017/08/budget-hero-docker-enterprise-edition/ (cit. on p. 44).

[ASL13] V. Andrikopoulos, S. Strauch, F. Leymann. "Decision Support for Application Migration to the Cloud." In: *Proceedings of CLOSER* 13 (2013), pp. 149–155 (cit. on pp. 49, 50).

[BCC+17] I. Baldini, P. Castro, K. Chang, P. Cheng, S. Fink, V. Ishakian, N. Mitchell, V. Muthusamy, R. Rabbah, A. Slominski, et al. "Serverless Computing: Current Trends and Open Problems." In: *arXiv preprint arXiv:1706.03178* (2017) (cit. on pp. 30, 76, 85).

[BHJ16] A. Balalaie, A. Heydarnoori, P. Jamshidi. "Microservices architecture enables DevOps: migration to a cloud-native architecture." In: *IEEE Software* 33.3 (May 2016), pp. 42–52 (cit. on p. 17).

[BKB+07] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, M. Khalil. "Lessons from applying the systematic literature review process within the software engineering domain." In: *Journal of Systems and Software* 80.4 (2007), pp. 571–583 (cit. on p. 26).

[Bob11] S. Bobrowski. "Optimal Multitenant Designs for Cloud Apps." In: *2011 IEEE 4th International Conference on Cloud Computing*. July 2011, pp. 654–659 (cit. on pp. 30, 67, 82).

[CHHK13] J. C. Carver, E. Hassler, E. Hernandes, N. A. Kraft. "Identifying Barriers to the Systematic Literature Review Process." In: *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. Oct. 2013, pp. 203–212 (cit. on p. 22).

[CLC11] Y. Chen, X. Li, F. Chen. "Overview and analysis of cloud computing research and application." In: *2011 International Conference on E-Business and E-Government (ICEE)*. May 2011, pp. 1–4 (cit. on pp. 30, 67, 82).

[Coh13] B. Cohen. “PaaS: New Opportunities for Cloud Application Development.” In: *Computer* 46.9 (Sept. 2013), pp. 97–100 (cit. on pp. 30, 33, 35, 69, 82).

[Doca] Docker. URL: https://www.docker.com (cit. on p. 43).

[Docb] D. Docs. URL: https://docs.docker.com (cit. on p. 47).

[Docc] C. F. Documentation. URL: https://docs.cloudfoundry.org/buildpacks/ (cit. on p. 39).

[FLR+14] C. Fehling, F. Leymann, R. Retter, W. Schupeck, P. Arbitter. *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer Vienna, 2014 (cit. on pp. 17, 38, 39, 45).

[GJ16] M. Greer, K. Jackson. *Practical Cloud Security: A Cross-Industry View*. CRC Press, 2016 (cit. on p. 42).

[GKK17] V. Gupta, K. Kaur, S. Kaur. “PERFORMANCE COMPARISON BETWEEN LIGHT WEIGHT VIRTUALIZATION USING DOCKER AND HEAVY WEIGHT VIRTUALIZATION.” In: (Mar. 2017). 2 Days International Conference on Recent Innovations in Engineering, Science, Humanities and Management, pp. 211–216 (cit. on pp. 30, 79, 85).

[Goo] Google Cloud Engine. URL: https://cloud.google.com/appengine/ (cit. on p. 33).

[GS13] A. Giessmann, K. Stanoevska-Slabeva. “What are Developers’ Preferences on Platform as a Service? An Empirical Investigation.” In: *2013 46th Hawaii International Conference on System Sciences*. Jan. 2013, pp. 1035–1044 (cit. on pp. 30, 68, 82).

[GWZW16] D. Guo, W. Wang, G. Zeng, Z. Wei. “Microservices Architecture Based Cloudware Deployment Platform for Service Computing.” In: *2016 IEEE Symposium on Service-Oriented System Engineering (SOSE)*. Mar. 2016, pp. 358–363 (cit. on pp. 30, 70, 83).

[Hen17] N. Henderson. *Cloud Vendor Revenues Hit $148 Billion in 2016*. Jan. 2017. URL: http://www.channelfutures.com/cloud-services/cloud-vendor-revenues-hit-148-billion-2016 (cit. on p. 31).

[Hig17] K. Hightower. *Kubernetes: Up and Running: Dive Into the Future of Infrastructure*. O’Reilly Media, 2017 (cit. on pp. 42, 44, 46, 47).

[Hof16] K. Hoffman. *Beyond the Twelve-Factor App*. O’Reilly Media, 2016 (cit. on p. 37).

[Htt12] M. Httermann. *DevOps for developers*. Apress, 2012 (cit. on p. 36).

[Jab] JabRef. URL: http://www.jabref.org/ (cit. on p. 26).

[Kav16] M. Kavis. *The PaaS Preview For 2016*. Jan. 2016. URL: https://www.cloudtp.com/doppler/cloud-providers/paas-preview-2016/ (cit. on p. 31).

[KC07] B. Kitchenham, S. Charters. *Guidelines for performing systematic literature reviews in software engineering*. EBSE Technical Report. Department of Computer Science, Keele University and University of Durham, UK, 2007 (cit. on pp. 18, 21–23).

[KG13] C. Kemp, B. Gyger. *Professional Heroku Programming*. John Wiley & Sons Ltd, 2013 (cit. on p. 40).

[Kit04] B. Kitchenham. *Procedures for Performing Systematic Reviews*. Joint Technical Report. Department of Computer Science, Keele University, UK and NICTA, Australia, 2004 (cit. on pp. 18, 21–23).

[Kra17] N. Kratzke. "Smuggling Multi-Cloud Support into Cloud-native Applications using Elastic Container Platforms." In: (2017). 7th Int. Conf. on Cloud Computing and Services Science (CLOSER 2017), At Porto, pp. 29–42 (cit. on pp. 30, 78, 85).

[Kub] Kubernetes. URL: https://kubernetes.io/ (cit. on pp. 45, 46).

[Lee17] H. Lee. "Building Software Defined Systems on HPC and Clouds." In: (Feb. 2017) (cit. on pp. 30, 79, 85).

[Liu17] L. Liu. *Containerd Brings More Container Runtime Options for Kubernetes*. Nov. 2017. URL: http://blog.kubernetes.io/2017/11/containerd-container-runtime-options-kubernetes.html (cit. on p. 45).

[LZL17] Z. Li, Y. Zhang, Y. Liu. "Towards a full-stack devops environment (platform-as-a-service) for cloud-hosted applications." In: *Tsinghua Science and Technology* 22.01 (Feb. 2017), pp. 1–9 (cit. on pp. 30, 33, 35, 71, 83).

[Mah11] Z. Mahmood. "Cloud Computing: Characteristics and Deployment Approaches." In: *2011 IEEE 11th International Conference on Computer and Information Technology*. Aug. 2011, pp. 121–126 (cit. on pp. 30, 66, 82).

[Mar17] W. Mar. 2017. URL: https://www.slideshare.net/Pivotal/pivotal-cloud-foundry-roadshow (cit. on p. 40).

[MKT17] O. Matsumoto, K. Kawai, T. Takeda. "FUJITSU Cloud Service K5 PaaS Digitalizes Enterprise Systems." In: *FUJITSU Sci. Tech. J* 53.1 (2017), pp. 17–24 (cit. on pp. 30, 78, 85).

[MPA+16a] A. R. Manu, J. K. Patel, S. Akhtar, V. K. Agrawal, K. N. B. S. Murthy. "A study, analysis and deep dive on cloud PAAS security in terms of Docker container security." In: *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*. Mar. 2016, pp. 1–13 (cit. on pp. 30, 74, 84).

[MPA+16b] A. R. Manu, J. K. Patel, S. Akhtar, V. K. Agrawal, K. N. B. S. Murthy. "Docker container security via heuristics-based multilateral security-conceptual and pragmatic study." In: *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*. Mar. 2016, pp. 1–14 (cit. on pp. 30, 72, 83).

[Pah15] C. Pahl. "Containerization and the PaaS Cloud." In: *IEEE Cloud Computing* 2.3 (May 2015), pp. 24–31 (cit. on pp. 17, 30, 33, 35, 43, 73, 84).

[PDKD17] P. Patros, D. Dilli, K. B. Kent, M. Dawson. “Dynamically Compiled Artifact Sharing for Clouds.” In: *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. Sept. 2017, pp. 290–300 (cit. on pp. 30, 33, 74, 84).

[PHM+16] C. Pahl, S. Helmer, L. Miori, J. Sanin, B. Lee. “A Container-Based Edge Cloud PaaS Architecture Based on Raspberry Pi Clusters.” In: *2016 IEEE 4th International Conference on Future Internet of Things and Cloud Workshops (FiCloudW)*. Aug. 2016, pp. 117–124 (cit. on pp. 30, 34, 66, 82).

[PL15] C. Pahl, B. Lee. “Containers and Clusters for Edge Cloud Architectures – A Technology Review.” In: *2015 3rd International Conference on Future Internet of Things and Cloud*. Aug. 2015, pp. 379–386 (cit. on p. 42).

[Plaa] G. C. Platform. *Kubernetes Engine*. URL: https://cloud.google.com/kubernetes-engine/docs/concepts/cluster-architecture#master_and_node_interaction (cit. on p. 46).

[Plab] T. H. Platform. URL: https://www.heroku.com/ (cit. on pp. 34, 40, 41).

[Rai16] A. Raina. *Building a multi-host, multi-container orchestration and distributed system using Docker*. Sept. 2016. URL: http://opensourceforu.com/2016/09/building-multi-host-multi-container-orchestration-distributed-system-using-docker/ (cit. on p. 47).

[Ric] C. Richardson. URL: http://microservices.io/patterns/ (cit. on pp. 39, 45).

[RVC+12] L. Rodero-Merino, L. M. Vaquero, E. Caron, A. Muresan, F. Desprez. “Building safe PaaS clouds: A survey on security in multitenant software platforms.” In: *Computers & Security* 31.1 (2012), pp. 96–108 (cit. on pp. 30, 75, 84).

[SAS+15] S. G. Sáez, V. Andrikopoulos, R. J. Sánchez, F. Leymann, J. Wettinger. “Dynamic Tailoring and Cloud-Based Deployment of Containerized Service Middleware.” In: *2015 IEEE 8th International Conference on Cloud Computing*. June 2015, pp. 349–356 (cit. on pp. 30, 71, 83).

[SCH17] C. SCHRODER. *What Makes Up a Kubernetes Cluster?* Apr. 2017. URL: https://www.linux.com/news/learn/chapter/intro-to-kubernetes/2017/4/what-makes-kubernetes-cluster (cit. on p. 46).

[Sha16] M. Sharma. “A Review of Major Security Issues in Cloud Computing.” In: *International Journal For Technological Research In Engineering* (2016), pp. 186–190 (cit. on pp. 30, 75, 84).

[Soa16] J. Soat. *Mark Hurd Predicts the Future of IT: Round 2*. Oct. 2016. URL: https://blogs.oracle.com/mark-hurd-predicts-the-future-of-it:-round-2 (cit. on p. 5).

[SSC17] A. Sreeramaneni, B. Seo, K. Chan. “A Business Driven Scalable Cloud Computing Service Platform (PaaSXpert).” In: *Journal of Korean Institute of Information Technology* 15.1 (2017), pp. 35–44 (cit. on pp. 30, 39, 77, 85).

[SSKY16] A. Singh, S. Sharma, S. R. Kumar, S. A. Yadav. “Overview of PaaS and SaaS and its application in cloud computing.” In: *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH)*. Feb. 2016, pp. 172–176 (cit. on pp. 30, 69, 83).

[Sta16] V. Staff. *Get Ready for a Container-Led Revolution*. July 2016. URL: https://www.vmware.com/ciovantage/article/get-ready-for-a-container-led-revolution (cit. on p. 17).

[TM16] P. Thakur, M. Mahajan. “Virtualization in Cloud Computing.” In: *International Journal of Recent Trends in Engineering & Research (IJRTER)* 02 (2016), pp. 308–315 (cit. on pp. 30, 34, 77, 85).

[Twe] T. Twelve-Factor-App. URL: https://12factor.net/ (cit. on pp. 37, 43, 50).

[WASL14] J. Wettinger, V. Andrikopoulos, S. Strauch, F. Leymann. “Characterizing and Evaluating Different Deployment Approaches for Cloud Applications.” In: *2014 IEEE International Conference on Cloud Engineering*. Mar. 2014, pp. 205–214 (cit. on pp. 30, 68, 82).

[Wee17] D. E. Weeks. *Container Considerations on Your DevOps Journey*. May 2017. URL: https://devops.com/container-considerations-devops-journey (cit. on p. 17).

[WHJ16] A. Williams, S. Hall, J. Jackson. *The Emerging Containers as a Service Marketplace*. July 2016. URL: https://thenewstack.io/emerging-containers-service-marketplace/ (cit. on p. 31).

[Win16] D. Winn. *Cloud Foundry: The Cloud-Native Platform*. O’Reilly Media, 2016 (cit. on p. 37).

[Win17] D. Winn. *Cloud Foundry: The Definitive Guide - Develop, Deploy, and Scale*. O’Reilly Media, 2017 (cit. on pp. 36, 39, 40).

[WXL16] Z. WANG, L. XUE, Y. LUO. “Cloud Computing Platform Based on the Docker.” In: *DEStech Transactions on Computer Science and Engineering* icmsie (2016) (cit. on pp. 30, 76, 85).

[XOR+15] M. G. Xavier, I. C. D. Oliveira, F. D. Rossi, R. D. D. Passos, K. J. Matteussi, C. A. F. D. Rose. “A Performance Isolation Analysis of Disk-Intensive Workloads on Container-Based Clouds.” In: *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. Mar. 2015, pp. 253–260 (cit. on pp. 30, 73, 84).

[Zim17] O. Zimmermann. “Microservices tenets.” In: *Computer Science - Research and Development* 32.3 (July 2017), pp. 301–310 (cit. on p. 17).

All links were last followed on March 27, 2018.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature