DOSSIER-Cloud

DEVOPS-BASED SOFTWARE ENGINEERING FOR THE CLOUD

http://www.dossier-cloud.eu

Deliverable D4.5

Survey paper / Technical report on DevOps Automation and Software Service Composition

Document details:

Editor :	Lambros Odysseos
Contributors :	Andreas Andreou, Andreas Christoforou, Luciano Baresi, Mike Papazoglou, Damian Tamburri
Date:	August 2018
Version:	7.0

Document history:

Version	Date	Contributor	Comments
1.0	21/06/18	Lambros Odysseos	Initial document, structure and content
2.0	05/07/18	Lambros Odysseos	First draft
3.0	10/07/18	Andreas Chistoforou	Corrections, second draft
4.0	16/07/18	Luciano Baresi, Damian Tamburri, Andreas Andreou	Corrections, third draft
5.0	23/07/18	Andreas Chistoforou	Corrections, fourth draft
6.0	28/07/18	Andreas Andreou	Corrections. Final review for approval
7.0	03/08/18	Luciano Baresi, Mike Papazoglou	Approved final version

Contents

1. Introduction 4

2. Survey Methodology 5

3. Literature Review 7

3.1. Which and how automated procedures support DevOps practices? 8

3.2. Which automation tools are being used to support DevOps teams? 9

3.3. Which are the most important services characteristics that affect automatic services synthesis? 14

3.4. What is the level of services synthesis automation? 17

4. Conclusions 18

5. References 19

1. Introduction

Targeting at faster application delivery and born from agile methodologies, the adoption of the DevOps approach has recently started to grow in software development and operation processes. Among various process tasks like testing, deployment, reconfiguration, etc., the DevOps approach favors a new culture between development and operations teams aiming to achieve a closer collaboration and communication without silos and barriers between these teams, and to emphasize automation and sharing.

DevOps describes the main stakeholders of producing and supporting a distributed software service or application that possess a mixture of code skills and system operation skills [1]. Authors in [2] state that DevOps consists of four dimensions: Culture, automation, measurement and sharing. Culture mainly refers to organizational structure and the corresponding new processes to support collaboration; it also takes into consideration the efficient management of human resources in activities such as team staffing. Automation of practices is the goal behind the support of new adopted tools and processes. Measurement in DevOps is defined as "monitoring high-level business metrics such as revenue or end-to-end transactions per unit time". Metrics at a lower level are important for measuring the delivery process, as well as the way people work. Sharing is an essential component towards knowledge and control spreading across teams.

Many organizations migrated from the agile software development approach to the DevOps approach. It is proved that DevOps is much more effective and productive than other development methodologies, although there are many variations as its application to different organizations varies. As DevOps is widely spreading [3], there is a strong need for automating certain processes or tasks.

By automating DevOps activities, we ensure the continuous delivery of new pieces of software in the application and bug fixing [4]. The most vital key point for achieving DevOps automation is simply human communication and collaboration. To this end, certain tools are used to ensure best practices, including those used to aid processes primarily for Building, Testing, Monitoring, Collaboration, Source Control Management, and Database Management. Considering human communication as one of the fundamental the building blocks of DevOps automation, a series of tools are available and can be used to ensure better productivity outcomes in Software Engineering.

DevOps is usually tightly connected to a new Software Engineering trend, software service composition. Service composition is derived from Service-Oriented Architecture (SOA) [5] and Microservices Architecture (MSA), [6], which both rely on services as the main component but vary in terms of service characteristics. In this survey we mainly deal with services that consist of smaller functional software components that are available and exposed over the cloud and support specific business operations. From this point forward and for the remainder of the survey, reference to services will also imply microservices as nowadays the two terms are indistinguishable.

Service composition relies on locating and combining small functional pieces called services and reinforces the automatic software development under the DevOps methodology. In order to achieve that a series of tasks should be followed in order to locate the available services, define the candidate (suitable) ones and finally select those that, when combined together, lead to the closest matching of the functional and non-functional requirements. The final step of this process is to integrate the matching services by combining them together through a commonly decided API and communication gateway.

Although this combination may seem irrelevant, it actually supports the DevOps automation process and affects Software Engineering in a positive manner. To be more specific, it affects Software Engineering simply because the development approach is different. As regards how the DevOps process is automated, the latter relies primarily on the fact that service composition comprises a number of critical tasks than may be automated apart from software building, like communication, coordination, monitoring, problem solving, deployment etc.

As DevOps is not a de-facto approach, a series of research questions were set from the beginning of this study, which may clarify many things regarding what DevOps actually is, how it can be automated and, finally, how it can be combined with one of the latest software development approaches, that is, software service composition. This deliverable presents an empirical study that outlines the significant concepts of the relevant scientific knowledge through four research questions. In order to provide answers to these research questions, a series of articles from various data sources were gathered, examined and analyzed. More information about how this study was conducted is provided in Section 2 which describes the survey methodology. Section 3 follows with the findings from the relevant literature review, while Section 4 concludes this survey.

2. Survey Methodology

The four research questions that motivated this study are the following:

RQ1: Which and how automated procedures support DevOps practices?

RQ2: Which automation tools are being used to support DevOps teams?

RQ3: Which are the most important service characteristics that affect automatic service synthesis?

RQ4: What is the level of services synthesis automation?

We may differentiate between two categories in the questions above; the first category is related to the automation procedures and tools that support DevOps (RQ1, RQ2), while the second focuses on a specific way of developing software services and systems under DevOps (RQ3, RQ4).

In order to give answers to the aforementioned research questions and complete this literature review, we collected and studied various publications and articles from different data sources and digital libraries like IEEE Xplore, ACM Digital Library, Google Scholar etc. Also, some other data sources were accessed like websites, previous survey papers etc. The way this investigation was approached was by searching within the selected sources for articles relevant to specific keywords like "DevOps", "DevOps automation", "Microservices", "Service Synthesis", "Service Ontology", "Microservice properties", "Microservice attributes" etc. Through a qualitative analysis and assessment of the papers that emerged from this research, we have managed to collect 48 papers in total. Some papers were completely discarded from our research pool because they were out of scope. A few papers were partly considered as they were relevant only in some of their sections. For example, some papers were dealing mostly with service decomposition but also mentioned vital service properties. The decomposition part was skipped due to the fact that it is out of scope and we mainly focused on the service attributes part which is helpful for the automation of services synthesis. We also discarded papers that were referred by other papers already included in the pool. The final number of papers collected in our investigation list after narrowing down our research to fit the purpose of this study was fourteen (14). To provide a clear picture about the conducted research, a chart that shows the years each paper was published is shown in *Figure 1*. It is obvious and important to state that this research was conducted by relatively recent data sources and publications.

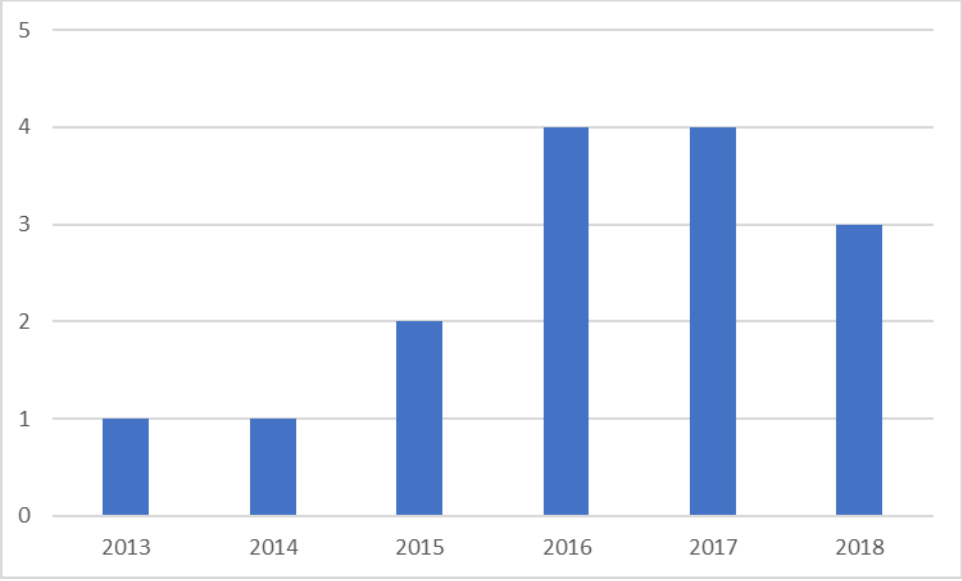

Figure 1: Number of papers examined by year

3. Literature Review

In this section, a literature review is provided according to the data sources and publications related to the subject of this brief survey. The answers to the research questions posed above are provided and discussed after describing the main findings of the conducted study. The review is divided into four sections, answering one-by-one each of the research questions.

Before moving to the survey findings, let us briefly define DevOps and DevOps automation. In order to automate processes regarding the application of the DevOps methodology in an organization, we start by acknowledging the fact that the two teams responsible for developing and operating the software must be inseparable. Their productivity rates and gaps must be detected early enough so that corrective measures can be taken. But the question is how can we detect these difficulties? The answer is to use some metrics and measurements that monitor the whole process. These metrics are able to determine whether a weakness becomes a reality. But which measurements have to be applied? There is no silver bullet on the usage of metrics regarding DevOps because DevOps itself is something that varies in the first place due to its nature of being the methodology with the highest level of agility.

Metrics and measurement activities on a DevOps environment can be divided into two categories. The first category includes DevOps-focused metrics which are dedicated to assess, (a) how well developers and operators are doing in terms of cost, time, resource utilization, (b) levels of quality of service, and (c) customer response and satisfaction. Some of these metrics are delivery cycle time, mean time to detect problems and weaknesses, mean time to repair problems, quality at the source code, etc. The second category includes cloud infrastructure metrics which are dedicated to sense the distributed environment in terms of services provided such as availability, security, reliability etc. Therefore, these two groups of metrics are strongly coupled aiming to sense the distributed environment and to collect meaningful data in each phase and activity so as to trigger the automation tasks and procedures.

Build, deployment and operation activities focus on automated procedures for adjusting and reconfiguring the DevOps environment. As mentioned before, these procedures are driven by quality metrics and follow specific workflows and rules. Automation mostly involves approaches, algorithms and tools for (a) monitoring the delivery of services to clients and offering decision support or decision making for reconfiguration that ensures compliance with service level agreements, (b) performing resource management and leveling towards green Cloud, and (c) offering synthesis or composition of services to build larger functional entities or applications.

In a more advanced and formal level, DevOps automations can be aided through Decision Support Systems (DSS) or Decision-Making Systems (DMS), which are specific systems that may automatically monitor SLAs and the reconfiguration of the software systems, or, in general, of the whole environment. The abovementioned systems essentially implement algorithms and

methods including Artificial or Computational Intelligence, like Artificial Neural Networks, Evolutionary Algorithms, Optimization Models, Fuzzy Logic etc. to measure the level of service delivery (e.g. time and performance) and adjust the availability of resources to improve it. One important goal within this approach is that it provides the advantage of reinforcing communication between the developers and the operators whilst increasing their productivity rates at the same time.

3.1. Which and how automated procedures support DevOps practices?

As previously mentioned, it is obvious that some procedures that may be automated, support various DevOps practices. But which of those automated procedures achieve this support, to what extent and how? As DevOps varies in application from one case or organization to another, some procedures may enhance productivity whilst some others may not. This mostly depends on the DevOps team and each individual organization can choose a combination of procedures to be applied in order to achieve DevOps automation in its own procedures or tasks.

In order to provide an answer to the question of this section, we should first define what DevOps aims at. The concept of DevOps is understood differently by individuals. The most common concepts or notions behind DevOps are Communication and Collaboration, Continuous Delivery, Automated Pipeline, Continuous Feedback and Continuous Deployment.

Regarding the second part of the research question, which aims to identify how automated tasks supports DevOps practices, we must first note that like with all new processes, the application of DevOps provides certain benefits but also introduces some challenges. According to the literature examined, the number of challenges in comparison to the benefits is minimal. This means that DevOps provides much more benefits than challenges or blind spots. These benefits include code version control and parallel deployment, scheduled deployment and testing, quality assurance and continuous integration within the project, real-time automated monitoring, cloud and database management, innovation in the development process, rapid delivery etc.

The work in [7] describes a systematic mapping study performed to explore DevOps. The main findings of this study were that DevOps is supported by culture of collaboration, automation, measurement, information sharing and Web service usage. More specifically, automation is supported by various design patterns such as, use cloud storage for storing big files, process asynchronous jobs using queues, use of a Real-time user monitoring tool etc.

Aiming to provide clarity and understanding of DevOps, the research work in [8] applied a systematic literature review, in which catalogues of DevOps concepts, practices, tools, benefits and challenges were developed. The analysis of the results showed that despite the fact that DevOps may be considered as an automation centric approach, at the same time the key concepts are human communication and collaboration, continuous delivery and automated pipeline. The main outcome of the analysis addressed in this study was that the key theme is

around “automation”, while a large number of tools were identified that are responsible to support the automation of the DevOps pipeline.

Authors in [9] identified the significance of the automation processes and the crucial role they have in applying DevOps practices. A systematic classification of DevOps artifacts was presented and it was also shown how different kinds of artifacts can be transformed into TOSCA specifications.

The study in [3] describes an empirical research which aims to highlight the factors influencing DevOps implementation. One of the main findings of this research is the significance of automation pipeline implementation and its critical role towards delivering DevOps benefits. Based on this research, the basic software development activities, like planning, development, testing and deployment, constitute the DevOps capability enablers. These activities, however, require the support of technical practices, that is, the technological enablers. Technological enablers support DevOps capabilities by automating tasks such as *Build*, *Test*, *Deployment*, *Monitoring*, and *Recovery*.

Finally, the authors in [10], investigate the elements that characterize the DevOps phenomenon using a literature survey and interviews with practitioners actively involved in the DevOps movement. Four main dimensions of DevOps were identified: Collaboration, automation, measurement and monitoring. According to the literature review and the practitioners which were interviewed by the authors, automation in DevOps is required in operations processes and increased test automation is necessary in the software development process. Particular emphasis is placed on the need for operations processes to be flexible, repeatable and fast by eliminating manual processes.

3.2. Which automation tools are being used to support DevOps teams?

As has been highlighted in this survey, in order to benefit from the outcomes of applying DevOps practices, various automated procedures have to be applied. But what comes first is communication between people. DevOps cannot tackle problems and will not ensure all the positive outcomes if communication is non-existent. The basic building block of DevOps is the harmonious communication and cooperation between the Development team and the Operation team. This is the basic building block of DevOps and from there we can enhance DevOps by the so-called automation tools.

This section aims to answer the second research question which seeks for possible tools that can be used to automate DevOps procedures. To make the connection with the previous section, DevOps practices are supported by a series of automated procedures that are mostly provided by DevOps automation tools, like the following in descending order of frequency: *Iaas/PaaS*, *Continuous Integration*, *Continuous Deployment*, *Configuration and Provisioning*, *Containerization*, *Monitoring*, *Build*, *Collaboration*, *Source Control Management*, *Testing* and,

lastly, Logging/Security. These are not specific tools but rather tool categories. Depending on the field that the organization that runs on DevOps aims to amplify, it may choose the best tools in its own case that lie within the specified category. For example, if a DevOps team is aiming at enhancing its productivity in the Continuous Deployment process, there are some tools that will aid it to achieve this. This research question basically provides information supporting that there are a series of tools which can be used in each case and that these tools those are divided in categories. Therefore, depending on the organizational needs, the available tools can be adapted to fulfill the targets of the specified category (e.g. productivity).

A literature review performed by the authors in [8] identified twelve categories of DevOps tools: *Source Control Management*, *Continuous Integration*, *Continuous Deployment*, *IaaS/PaaS*, *Monitoring*, *Database Management*, *Containerization*, *Configuration* and *Provisioning*, *Logging/Security*, *Build*, *Testing* and *Collaboration*. According to the frequency of their occurrence in the total number of reports, *IaaS/PaaS, Continuous Integration and Continuous Deployment* appear to be the most important categories. The DevOps tools that are reported by this work are listed in *Table 1*.

Table 1:DevOps Tools List (extracted as reported in [8])

Category	**Tools**	**Features**
Source Control Management	Github	• Github is a web-based Git (private and public accounts) repository designed for version control and source code management. • Github provides team collaboration • Provide logs containing (commit history, tracking labels, pull requests, code review comments, email notifications, task lists, readme code information file)
	Bitbucket	• Similar features to Github • Offers both free public and private commercial accounts
Continuous Integration	Codeship	• Use Docker abilities to automate development and deployment • Enable developers to create their own test units - Provide team notifications with code changes and test results • Deploy and run code in parallel simultaneously with tests • Integrate many programming languages (Java, Ruby, Python, PHP, GO) • Integrate many platforms (Heroku, AWS) • Integrate various databases (MySQL, MongoDB)
	Travis CI	• Used to build, test, deploy code hosted on Github • Enables automated continuous integration with Github

		• Notify team with test results through email, postings or any IRC channel • Support various programming languages (Java, C, C++, C#, Perl, Python, Ruby, Node.js) • Provide its own command-line UI • Enable parallel deployment and testing
Continuous Deployment	Codeship	• Enable multiple deployment sequential or parallel • Enable developers to run deployments commands on an authenticated remote server using SSH. This feature allows developers to trigger deployment/update on external systems for stakeholders.
	Travis CI	• Enable developers to setup continuous deployment schedule. • Enable developments to automate deployment schedule. • Integrated deployment with Github.
IaaS/PaaS	Heroku	• Heroku is a PaaS that support (Ruby, Java, Node.js, Python, PHP) • Heroku Git server handle application pushes with repository • Heroku integrates with Github, Bitbucket • Enables automated continuous deployment • Provide logs and maintain version control of code • Heroku Logplex collects all application reporting
Monitoring	Nagios	• Nagios is an open source application that monitors systems • Nagios also provides remote monitoring through its Remote Plugin Executor which supports SSH and SSL encrypted tunnels. • Nagios enable developers to build reporting units using programming languages (Shell Scripts, C++, Perl, Ruby, Python, C#, etc.) • Nagios also provide a powerful tool for DevOps driven SD applications that consist automated log file rotation and creating in a parallel enabled service distribution.
	New Relic	• New Relic provides insight into an SD application at runtime • New Relic delivers unique monitoring log metrics of cloud application development and it deployment from UI to backend • New Relic provides continuous automated reporting on health, status, runtime, build, deployment and performance or a cloud application.
Database Management	MongoDB	• MongoDB is a free open source, cross-platform, document-oriented database program.

		• Classified as NoSQL database application, MongoDB avoid tradition table-based relational database in favor of JSON-like documents with dynamic schema. • MongoDB provides developers with Ad hoc queries, Aggregation using MapReduce and Server-side JS.
Logging/Security	Loggly	• Loggly is a cloud-based log management and analytical service. • Loggly summarizes automatically a software application log and provides real-time analysis for software processes. • Loggly increases delivery speed and provide guided-data log to DevOps team based on application troubleshooting results. • Loggly manages logs from any source or application test units coded in any language (Java, PHP, Node.js, Python, .NET, JS, Docker, Linux, Windows, Apache)
	Papertrail	• Cloud based log monitoring system - Integrates with Heroku metrics logs • Integrates with HipChat collaborative tool
Build	Codeship	• Codeship provides build capability for DevOps team form end-to-end in the development pipeline.
	Travis CI	• Travis CI provide a powerful build environment that can be setup in travis.yml
Testing	Cucumber	• Runs automated acceptance tests written in a behavior-driven development style. • Cucumber merges SD specifications and test documentation into one cohesive log. • Cucumber uses Gherkin, a language that defines Cucumber test cases which is designed to be human readable non-technical.
	Junit	• Junit builds test functions from normal functions by providing @Test annotation to the method header. • Automated Test units are composed of collection of annotated Java methods that handle particular exceptions or provide runtime report about a component or process behavior.
Collaboration	Slack	• Slack is a cloud-based collaboration tool it improves DevOps team communication by offering an IRC-like features which can handle files exchange from integrated could such as Trello, Google Drive, DropBox, Heroku, Github, etc.

	HipChat	• HipChat is a web-based service for internal private chat and messaging • HipChat supports group and one-on-one chats, it also support video calling (group and pair) between team members • HipChat relays messages through SMS services as well and allows a user a 5GB storage capability • HipChat integrates the team progress from different repositories such as Bitbucket, Github, etc.

The authors in [11] present a brief overview of the most recent DevOps technologies, such as delivery tools. Through this overview, some tools appear to be mandatory in automating DevOps and particularly important is the selection of the right tool for an environment or project. The tools being discussed in this work are categorized, according to DevOps phases, into three categories and, based on the tool type, into five categories. The categories based on DevOps phase are *Build*, *Deployment* and *Operations*. The categories based on tool type are *Build*, *Continuous Integration*, *Configuration Management*, *Logging* and *Monitoring.* The full list of the automation tools presented in this work is provided in *Figure 2*, along with various other features.

Automation tools for DevOps.

Tool	DevOps phase	Tool type	Configuration format	Language	License
Ant	Build	Build	XML	Java	Apache
Maven	Build	Build	XML	Java	Apache
Rake	Build	Build	Ruby	Ruby	MIT
Gradle	Build	Build	Based on Groovy	Java and a Groovy-based domain-specific language (DSL)	Apache
Jenkins	Build	Continuous integration	UI	Java	MIT
TeamCity	Build	Continuous integration	UI	Java	Commercial
Bamboo	Build	Continuous integration	UI	Java	Commercial
Puppet	Deployment	Configuration management	DSL similar to JSON (JavaScript Object Notation)	Ruby	Apache
Chef	Deployment	Configuration management	Ruby-based DSL	Ruby	Apache
Ansible	Deployment	Configuration management	YAML (YAML Ain't Markup Language)	Python	GPL (GNU General Public License)
Loggly	Operations	Logging	—	Cloud based	Commercial
Graylog	Operations	Logging	—	Java	Open source
Nagios	Operations	Monitoring	—	C	Open source and GPL
New Relic	Operations	Monitoring	—	—	Commercial
Cacti	Operations	Monitoring	—	PHP	GPL

Figure 2: List of DevOps Tools (Source: [11])

3.3. Which are the most important services characteristics that affect automatic services synthesis?

It is self-evident that services have a series of vital properties that define them. But in the specified trend of services-based software development there is no concrete evidence as to which attributes should describe a service and which of them are vital for promoting their integration or reusability. This is a gap in service-oriented software engineering which makes this the research area a greenfield. During this section we demonstrate how we managed to locate and define the most important attributes that every service must bear to promote reusability and consequently facilitate their automatic synthesis.

One of the most vital characteristics of a microservice is the programming language in which it was implemented, although it is trivial to the end-user or programmer. The calculation complexity and performance must be defined, and security mechanisms and auditability must be provided. It is very important to describe the domain of application and/or the service itself. Furthermore, in order to build reliable services, load balancing schemes have to be provided along with monitoring mechanisms. Depending on the application, services may provide data storage information by means of explaining to the programmer the way data is stored as services use their own data storage [12].

As it has already been mentioned, endpoints are critical so that a clear picture is formed about how data is exchanged throughout the process. If message passing and communication protocols are provided the developer knows better how to implement the requirements. Lastly, a service may have methods by means of executing a series of small tasks instead of a single one [13].

A more complicated but interesting study suggests some principles and patterns when developing services, the organization of which must be ensured around business capability (domain). Infrastructure automation can be used by providing some sort of intelligence within a service, especially at the endpoints. Data control has to be decentralized and services have to be designed for failure as failures occur often. These principles may be, and it is suggested to be, reinforced with some kind of patterns like “Aggregator Microservice Design Pattern”, which is a development approach that causes services to invoke others for data retrieval or processing. Another pattern comes in contrast to the basic idea of the service architecture. It is well known that services mostly use REST APIs to exchange communication messages. A new pattern known as “Asynchronous Messaging Microservice Design Pattern” suggests the use of message queues. In this context, it is clear that services may change forms through principles and design patterns [14].

Garriga [15] proposed a taxonomy for the properties and features of services, which is depicted in Figure 3. This taxonomy is quite useful as it can form the starting point for selecting services to integrate, provided that the required functional and non-functional characteristics of the final system which will be developed through this integration are recorded.

As previously mentioned, the programming language is important. This is stated as “the right tool for the right job” [15]. Also, a service may have some obligations or constraints. Services may have their interaction models define if they are synchronous or asynchronous. Based on this, the data exchange protocol (REST, RPC, message queues) and the data storage protocol (SQL, NoSQL, graph, document) are derived. Finally, services may have some security or management constraints, like whether a microservice provides public or private access at the specified context, or how it is supposed to react to failures. There is no such thing as *standard development principles* in services; however, optimal combinations of technologies can be used to aid the service functionality. Of course, this does not mean that the specified parameters and information accompanying a service must not be provided as a documentation to the developer.

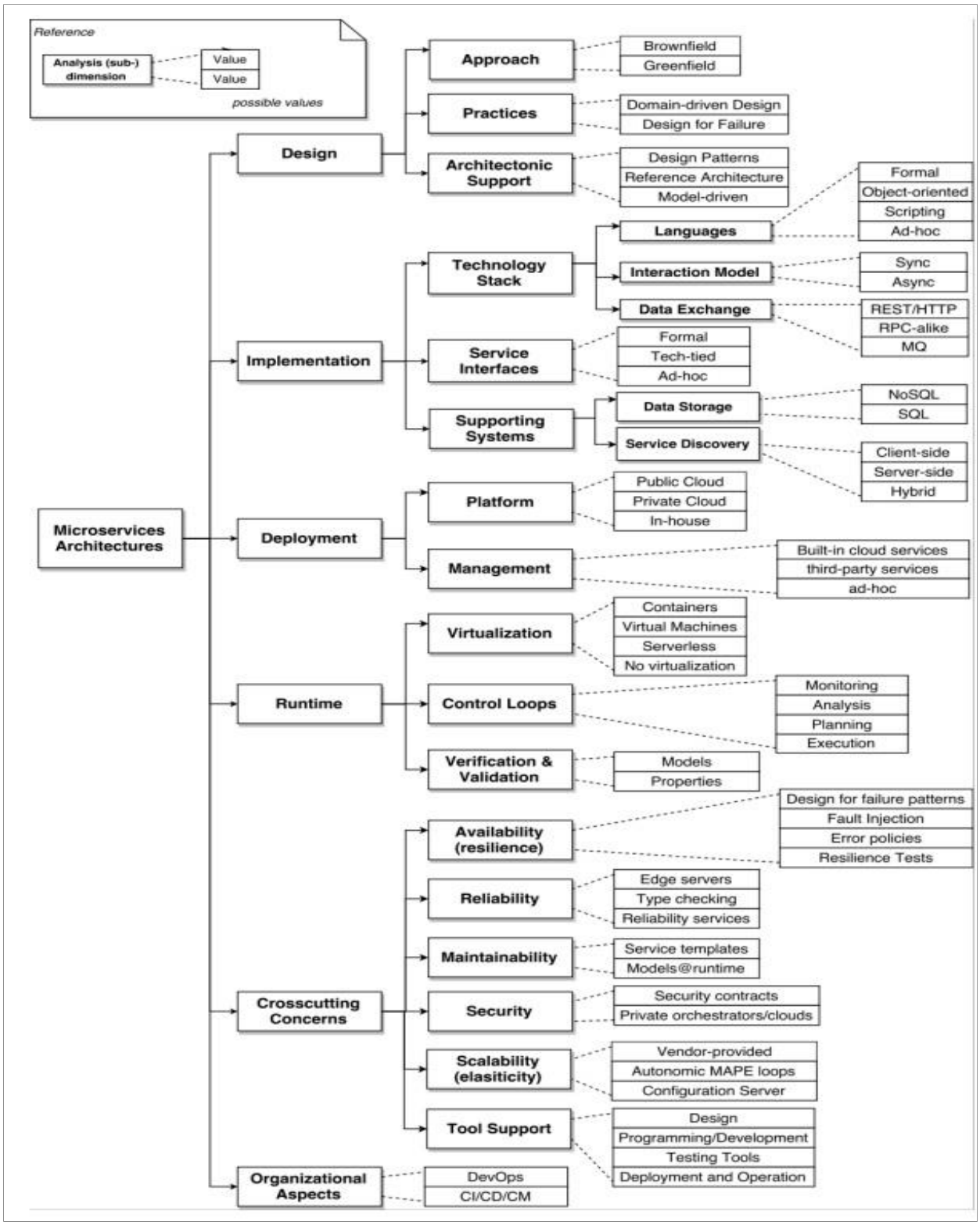

Figure 3: Microservice attributes depending on the field of application (Source: [15])

3.4. What is the level of services synthesis automation?

Quite a few of the leading vendors offer a variety of services providing the agility of developing multi-domain applications. Some of them refer to services also as software components.

This section aims at answering the fourth research question which is to identify the level of service synthesis automation.

Despite our efforts to identify any related papers which propose or deal with service synthesis towards an integrated software solution or development paradigm this was not feasible. This reveals that the corresponding research area is at its infancy and there is ample room for innovative ideas. Probably the most enlightening paper that is closest to solving this problem of service synthesis and automation is [18]. All the proposed layers in this paper can be used as described with possible enhancement with a few more features to customize them so that they fit the new purpose. Therefore, [18] is a good starting point for attempting to provide automation in software synthesis.

According to [18], a key-concept must be introduced before the utilization of the five layers described, and this is ontology. Through a dedicated ontology schema, the matching between the required properties of a group of services and the features offered by services in the market will be made feasible. Therefore, we must firstly form a pool of components, which are essentially the services that are made available by vendors and provide a standardized form of description for their properties; following that, we need also to describe the desired functional features of the system under development, i.e. the software requirements, which will dictate what the final software product is supposed do. Then, both descriptions (of available services and of sought ones) are inserted into a component profiling process which uses an ontology-based language like EBNF as in the case of [18] or other means (e.g. TOSCA) to perform a standardized profiling of the functional and non-functional service descriptions. This is the first layer of the framework in [18], which is the Description Layer. From then, we are able to move to the next step which is pushing the profiled parts into the Location Layer. This layer aims at locating services that match the software requirements. Then, the third layer follows, that is, the Analysis Layer, which simply analyzes the results from the candidate (located) services and yields a list of possible candidates that match the requirements sought. Then, the Recommendation Layer follows which suggests to the end-user which services may be finally used in order to cover all the possible requirements and ensure product dependability. The final layer is the Build Layer which attempts to bind all the recommended components from the previous step to deliver the final software product. In a nutshell, by following the layered structure proposed in [18] the automation of services synthesis may become a reality in the near future.

4. Conclusions

This deliverable attempted to provide answers to the four research questions that were set from the beginning: The first research question was “Which and how automated procedures support DevOps practices?” The second one, “Which automation tools are being used to support DevOps teams?” The third, “Which are the most important services characteristics that affect automatic service synthesis?” And the fourth, “What is the level of services synthesis automation?” The findings of each research question were presented and analyzed up to a degree depending on the number and quality of the papers located to support the answers. The identification of these papers was based on the standard process utilized when conducting surveys, with the sources being primarily top multidisciplinary journals of IEEE, ACM, Elsevier etc., or top notch conferences in the broader area of services.

For the first research question, we have stated that organizations can use a combination of procedures to support DevOps which varies in application depending on the company’s environment. Some procedures that support DevOps practices are Communication and Collaboration, Continuous Delivery, Automated Pipeline, Continuous Feedback and Continuous Deployment. Now, regarding how these procedures may support DevOps practices, they can by providing Code Version Control, Parallel Deployment, Enable Scheduled Deployment, Enable Scheduled Testing, Provide Quality Assurance, Continuous Integration within the project, Real-Time Automated Monitoring, Cloud and Database Management, Innovation and new Ideas of Development, Rapid Delivery and a series of much more.

For the second research question, we have examined a pool of resources to identify tools that are capable of being used to support DevOps teams. These tools are not given as entities but however as categories or families of tools that aim a purpose. These families of tools include Iaas/PaaS, Continuous Integration, Continuous Deployment, Configuration and Provisioning, Containerization, Monitoring, Build, Collaboration, Source Control Management, Testing and lastly Logging/Security. Depending on the organization’s needs, a selection of tools can be combined in order to enhance DevOps practices.

For the third research question, we must say that services have a lot of attributes that can be classified into vital and trivial. This is meant to the end-user/developer. There are some attributes that are trivial and do not mean anything to the developer like the programming language used to develop the specified component at the first place. However, as vital attributes we can consider application domain, functionality, security and auditing mechanisms, message exchange protocols, data storage, complexity, performance and any possible principles and patterns that were used during the component’s developing process.

For the fourth and last research question of this deliverable, we can conclude the findings within a few steps. Firstly, there was minimal literature on this subject due to that it is a latest trend in software engineering. A promising technique, which may be adopted and adapted, targets

automatic component reusability. Along the same lines, the available services and software requirements have to be transformed into a standardized description form like EBNF or TOSCA. Then this description is used as input in a five layer framework as proposed by (Andreou & Papatheocharous, 2016), to define which services match the product requirements

Concluding, although in some of the research questions the available literature was poor, we managed to collect adequate information and present our findings. We hope that these findings will aid and enforce the work and outcome of other researchers.

5. References

[1] P. Duvall, “Breaking down barriers and reducing cycle times with devops and continuous delivery” 2012.

[2] J. Humble and J. Molesky, “Devops: A software revolution in the making,” *Cut. IT J.*, vol. 24, no. 8, pp. 6–12, 2011.

[3] M. Senapathi, J. Buchan, and H. Osman, “DevOps Capabilities, Practices, and Challenges,” in *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018 - EASE’18*, 2018, pp. 57–67.

[4] N. FORSGREN, “DevOps Delivers.,” *Commun. ACM*, 2018.

[5] B. Lublinsky, M. Rosen, M. J. Balcer, and K. T. Smith, *Applied soa : service-oriented architecture and design strategies*. Wiley, 2013.

[6] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, “Microservices: Yesterday, Today, and Tomorrow,” in *Present and Ulterior Software Engineering*, Cham: Springer International Publishing, 2017, pp. 195–216.

[7] F. Erich, C. Amrit, and M. Daneva, “Report: DevOps Literature Review Architectually Significant Requirements View project Engineering the Architecturally Significant Functional Requirements in Global Outsourcing Projects View project Report: DevOps Literature Review,” 2014.

[8] G. Bou Ghantous, A. Gill, and G. Bou, “Association for Information Systems AIS Electronic Library (AISeL) DevOps: Concepts, Practices, Tools, Benefits and Challenges Recommended Citation,” 2017.

[9] J. Wettinger, U. Breitenbucher, and F. Leymann, “Standards-Based DevOps Automation and Integration Using TOSCA,” in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, 2014, pp. 59–68.

[10] M. Soni, “End to End Automation on Cloud with Build Pipeline: The Case for DevOps in Insurance Industry, Continuous Integration, Continuous Testing, and Continuous Delivery,” in *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, 2015, pp. 85–89.

[11] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “DevOps,” *IEEE Softw.*, vol. 33, no. 3, pp. 94–100, May 2016.

[12] P. Di Francesco, I. Malavolta, and P. Lago, “Research on Architecting Microservices: Trends, Focus, and Potential for Industrial Adoption,” in *2017 IEEE International Conference on Software Architecture (ICSA)*, 2017, pp. 21–30.

[13] G. Granchelli, M. Cardarelli, P. Di Francesco, I. Malavolta, L. Iovino, and A. Di Salle, “Towards Recovering the Software Architecture of Microservice-Based Systems,” in *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*, 2017, pp. 46–53.

[14] C. Pahl and P. Jamshidi, “Microservices: A Systematic Mapping Study Centre for Next Generation Localisation View project SOA and Cloud Security and Assurance View project Microservices: A Systematic Mapping Study,” 2016.

[15] M. Garriga, “Towards a Taxonomy of Microservices Architectures,” Springer, Cham, 2018, pp. 203–218.

[16] A. Balalaie, A. Heydarnoori, and P. Jamshidi, “Microservices Architecture Enables DevOps: Migration to a Cloud-Native Architecture,” *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May 2016.

[17] G. Kecskemeti, A. C. Marosi, and A. Kertesz, “The ENTICE approach to decompose monolithic services into microservices,” in *2016 International Conference on High Performance Computing & Simulation (HPCS)*, 2016, pp. 591–596.

[18] A. S. Andreou and E. Papatheocharous, “Towards a CBSE Framework for Enhancing Software Reuse: Matching Component Properties Using Semi-formal Specifications and Ontologies,” Springer, Cham, 2016, pp. 98–121.