DECIDE: An Extended DevOps Framework for Multi-cloud Applications

Juncal Alonso¹, Kyriakos Stefanidis², Leire Orue-Echevarria¹, Lorenzo Blasi³, Michael Walker⁴,
Marisa Escalante¹, María José López¹, Simon Dutkowski²

¹TECNALIA, Bizkaia technology park, Derio, Spain

²Fraunhofer FOKUS, Berlin, Germany

³Hewlett Packard Italiana s.r.l., Cernusco sul Naviglio, Italy

⁴AIMES, Liverpool Innovation Park, UK

Juncal.Alonso@tecnalia.com; kyriakos.stefanidis@fokus.fraunhofer.de; Leire.Orue-Echevarria@tecnalia.com; lorenzo.blasi@hpe.com; Michael.Walker@aimes.net;
Marisa.Escalante@tecnalia.com; MariaJose.Lopez@tecnalia.com;
simon.dutkowski@fokus.fraunhofer.de

ABSTRACT

DevOps represents a model for application development that enables close collaboration between software developers and IT operations with the objective of implementing continuous integration, continuous delivery and continuous development of software applications. This paper proposes an approach for extending the DevOps philosophy with the objective of supporting the development and operation of multi-cloud native applications deployed over heterogeneous cloud resources. The authors present the extended DECIDE DevOps framework and the supporting tool suite developed in the context of the DECIDE H2020 action

CCS Concepts

•Computer systems organization ~ Cloud computing •Software and its engineering ~ Software design engineering

Keywords

Cloud Computing, Multi-cloud, Continuous Design, Continuous pre-deployment, deployment optimization, continuous adaptation, continuous monitoring, DevOps.

1. INTRODUCTION

In the last decade, the software industry has been marked by a growing trend of software enterprises' ability to deploy new features rapidly, in short release cycles. This has been accompanied by a growth in the flexibility of their IT infrastructure to support the digital transformation from product to service economy. Cloud services enable this to some degree, but utilizing them create a dependency to external entities. In a world where new players come, others disappear, and conditions are continuously changing, the enterprises cannot be sure that the architectural decisions that were taken in the past continue to be

valid. The decision on using one or several approaches simultaneously (dedicated internal servers, instances in a shared external server, etc.) is driven by consideration such as profitability, reliability, performance, security, legal or even ecological aspects.

Furthermore, Cloud Service Providers (CSPs) themselves may also fail. Therefore, to ensure the greatest possible measure of protection, an enterprise may choose to adopt a multi-cloud strategy. There are several multi-cloud solutions available for solving specific problems but, to date, little attention has been paid to distributing the cloud risk and the cloud gain. That is, to optimize the potential advantages that each cloud offers in terms of Non-Functional Requirements (NFRs) such as performance, location and cost, and managing multiple clouds from a single technology platform.

The main stakeholders in the development and operation of multi-cloud- applications are the developers and operators. Multi-cloud application developers are relevant to the design and development, while multi-cloud application operators deal with the deployment and provisioning (operating) the services. To support both in the whole process, from design to operation, a DevOps approach is needed. DevOps represents an effort to accomplish the same mutually trusting relationship for Software-as-a-Service as Agile has done for software as a product. DevOps tries to teach operations to move at the same speed and with the same flexibility as development. However, existing DevOps solutions are focused on tailor-made, single-purpose industrial solutions and no generalized, multi-cloud-oriented DevOps framework exists to date.

This problem statement engenders the question; what could be the most appropriate approach to provide a new generation DevOps framework for software developers and operators? In answering that question the necessity for this new DevOps framework to meet several requirements must be taken into consideration. These include:

- Development and operation of NFR aware multi-cloud native applications
- Techniques and mechanisms to design, develop and dynamically (re-)deploy multi-cloud aware applications
- Access provision to reliable, interoperable, monitored and legally secured cloud services

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the Owner/Author.

ICCBDC 2019, August 28-30, 2019, Oxford, United Kingdom

© 2019 Copyright is held by the owner/author(s).

ACM ISBN 978-1-4503-7165-0/19/08

<http://doi.org/10.1145/3358505.3358522>

This paper presents a novel approach for DevOps, targeting multi-cloud native applications, named “Extended DECIDE DevOps framework” and a tool suite which supports it through different modules. The design and implementation of the DECIDE DevOps framework and its tool suite are Key Results (KRs) of the European H2020 project “DECIDE” [9].

DECIDE proposes to extend the ‘traditional’ DevOps approach on both its axes - namely Dev and Ops - to address multi-cloud software needs. In the case of the Dev axis, the authors propose to start with the continuous approach promoted by the DevOps philosophy, right from the architectural design, then to proceed with a continuous deployment simulation followed by the optimization of the application with respect to predefined and prioritized NFRs and then, the definition of a multi-cloud Service Level Agreement (MCSLA) for the application taking into consideration the underlying selected resources. In the Ops axis, DECIDE proposes a continuous monitoring task, targeting both the underlying cloud resources, the application itself, and a self-adaptation or automatic redeployment based on the violation of the thresholds established in the MCSLA for the application or in the Cloud SLAs of the contracted resources.

To sum up, the authors propose to automate the whole process (providing the appropriate mechanisms and tools), and to include new stages both in the Development part and in the Operations part of the traditional DevOps (such as pre-deployment simulation).

2. EXISTING APPROACHES

As stated by RedHat [23] a major outcome of implementing DevOps is a Continuous Integration and Continuous Deployment pipeline (CI/CD). DECIDE goes a step further and extends that pipeline to the previous phase of architectural design, to an intermediate step in between CI and CD, which is pre-deployment and to a later step focused on operations. The CD in DECIDE is also understood in a different way since DECIDE aims is targeted upon multi-cloud applications, which means deploying on multiple clouds. Most CD approaches focus solely upon the deployment on one cloud service provider. The CI and continuous quality is leveraged in DECIDE with the possibility to include open-source tools into the DECIDE DevOps framework.

Xebialabs DevOps platform [24], IBM UrbanCode [13], Microfocus [19], AWS Developers Tools [1] and Microsoft Azure DevOps [2] are similar DevOps solutions in the market. This analysis includes only tool suites that contain the complete CI/CD pipeline. Tools that cover only one of the phases of the DevOps pipeline like the continuous integration (CI) (Jenkins or Travis), dependencies managers (Maven or Gradle), or source code managers (GitHub, GitLab) are not analyzed. The rationale for that is that it was decided to analyze only the tools that can be considered to be the most similar to what the DECIDE DevOps framework aims to offer. Furthermore, the DECIDE DevOps framework is developed such that tools such as Travis or GitHub can be included in the same way as the already included Jenkins and SonarQube. However, when analyzing these tools and comparing them with the Extended DevOps framework that DECIDE aims to offer, covering continuous architecting, pre-deployment, cloud services orchestration and intermediation as well as semi-automatic redeployment it can be seen that not all the phases are fully covered in existing solutions. The integration and provisioning of a complete pipeline, from development to operations, as understood by DECIDE, are only loosely covered in the analyzed tool suites.

For the continuous architectural design phase of the software development lifecycle, the authors of this paper have not found anything similar. Cloud specific patterns are available in literature [10] or have been developed in European Projects [11] but they do not address the peculiarities of multi-cloud aware applications [5]. In the case of the simulation for the deployment, the authors have found a similar tool. The European project MELODIC [18] is aiming to create a deployment simulation tool but the approach varies slightly as they are targeting big data applications and use a model-driven engineering approach. For the configuration, several tools already exist. Among them, Puppet + MCollective [16] and Terraform [22] are the most known ones. MCollective extends Puppet with service orchestration. Terraform describes resources but it relies on the APIs provided by the cloud services and lacks the necessary means to describe relations between software components. For application deployment, Cloudify [4] – TOSCA compliant – proposes a tool suite to orchestrate the application deployment relying on a meta-language and task engine. Activeeon [20] proposes a commercial solution to manage application elasticity by making use of workflow modelling techniques.

3. DECIDE DEVOPS FRAMEWORK

As described above, the DECIDE DevOps Framework is an extension to the traditional CI/CD pipeline. It is realized as an integrated platform of various DECIDE modules and a number of open source tools. The management and orchestration of the DevOps framework modules is done via a unified web based user interface (UI).

Figure 1 depicts the extended CI/CD pipeline as realized in the DECIDE DevOps framework and how the various stages are covered by the different DECIDE modules that will be presented in this paper.

Figure 1. The DECIDE DevOps framework.

The DevOps framework UI allows the developer to access all of the DECIDE modules in an integrated way, transparently unifying their user interfaces and orchestrating their execution when necessary. It also offers editors for the definition of the multi-cloud application properties, the NFRs and the MCSLA. Lastly, it integrates open source tools like Jenkins for continuous integration and SonarQube for code quality.

All of the application properties and all the configuration data that is produced by the various DECIDE modules and the DevOps framework dashboards are stored in a json file that is called the application description. The application description is stored in the application's git repository and serves as the main configuration for all DECIDE modules.

While the detailed functionality of each DECIDE module is presented individually in the following sections, we will start with an overview of the whole application lifecycle within the DevOps framework. For the remainder of the paper, we will use the term "developer" as a unified term for all the roles (designer, developer, tester, operations engineer, etc.) that take part in the application lifecycle.

In the first stage of the lifecycle the developer defines the basic application properties such as the location of the Git repository, the number and basic characteristics of the application's microservices as well as the application's NFRs. This is done via the provided dashboard that is part of the DevOps framework's UI.

Moving on to the design stage, the ARCHITECT module suggests a set of architectural patterns that fit to the defined NFRs and application properties. The developer selects the ones that intends to adopt, and the selection is stored in the application description.

For the actual development, integration and testing of the application DECIDE does not dictate the use of a specific technology or development environment. The DevOps framework offers a selection of open source tools (such as code quality tools) that help this process.

When the application is ready for deployment, the lifecycle passes through four different stages before the actual deployment takes place. First the OPTIMUS module runs a number of simulations that determine the best deployment scenarios based on the application characteristics so far. Then the ACSmI module handles the discovery of the CSP services that are needed for the deployment scenario. The developer defines the MCSLA for the application using the MCSLA editor module and, lastly, ACSmI handles the contracting of the desired CSP services that are available and can fulfill the MCSLA.

The next stages are deployment and configuration management. Both are handled automatically by the ADAPT module. These stages include the provisioning of the needed CSP resources as dictated by the previous stages and the configuration of the provisioned CSP infrastructure.

Moving on to the operations stages, ACSmI continuously monitors the provided CSP resources while ADAPT checks if there are any violations of the MCSLA. In case of a violation, ADAPT handles the redeployment of the application component in alternative CSP resources that can fulfil the same MCSLA.

In the following sections we present the detailed functionality of each of the main DECIDE modules. The collection of those modules is called the DECIDE tool suite.

4. DECIDE TOOL SUITE

4.1 Architect

ARCHITECT is the first module in the application lifecycle and supports the developer in the preparation of the application for a multi-cloud deployment scenario by providing a set of (multi-)cloud architectural patterns, which can be applied to the application. It also recommends the most appropriate ones based on the defined application NFRs. The pattern recommendations

can also be used within the rest of the DECIDE modules to evaluate the most optimal deployment scenarios.

An architectural pattern provides a general, reusable solution to a commonly occurring problem in the development or deployment of software components. DECIDE distinguishes between two types of patterns. Those that provide a description or template for solving a particular problem and are independent of the implementation details (Cloud Provider Independent Patterns (CPIP)) and those that target specific implementation techniques or rely on the use of specific software components (Cloud Provider Specific Patterns (CPSP)). With the former (CPIP) the formulation of high-level solutions that cover a broad problem space is possible, whereas with the latter (CPSP) tailored solutions are provided, for instance to optimize an application in a very specific context.

DECIDE defines four groups of architectural patterns: 1) Fundamental Patterns, the standard recommended patterns for all cloud applications that are in line with the DECIDE DevOps framework. They describe cloud service models and cloud deployment types analogous to the NIST cloud definition [17] and cover the conditions under which a certain service model and deployment type should be used for a cloud application [10]. 2) Development Patterns, which provide the developer with best practices for building multi-cloud applications. The "Service State Management" pattern is one example. 3) Deployment Patterns that address the deployment schemes and strategies for multi-cloud applications. Examples include "Blue/Green deployments" and "Single Node Multi-Containers". 4) Optimization Patterns that aid the developer in improving the application NFRs by taking adequate measures in optimizing the application code to reflect these requirements. Optimization patterns can be multi-cloud in nature, but their design could also optimize the use of cloud resources, such as elasticity. An example is using the cloud persistence layer instead of implementing it as part of the application.

The core of the ARCHITECT module consists of a library of cloud and multi-cloud architectural patterns and a recommendation mechanism.

This library contains the detailed description of the pattern, any models or figures that complement the description and links to the source for each pattern. The various sources for the CPIP and CPSP patterns that are included in the ARCHITECT library as well as the pattern descriptions can be found in [5]. ARCHITECT also provides a user interface for accessing the pattern descriptions.

On the other hand, the recommendation mechanism analyses the application's NFRs and suggests the most appropriate development, deployment and optimization patterns. To facilitate this type of recommendation, each pattern in the ARCHITECT library has been analyzed in terms of its qualitative and, in regard of CPSP patterns, quantitative impact to the various NFRs including availability, scalability, etc. The results of this analysis are stored as metadata to each pattern and are used as input to the recommendation mechanism.

The developer can select which of the recommended patterns is willing to adopt. This selection, along with the whole set of recommended patterns, is stored in the application description and can also be used by the rest of the DECIDE modules.

The implementation of ARCHITECT consists of the pattern compendium, a standalone service which contains the library and

the recommendation mechanism and two user interfaces. One is part of the DevOps framework web interface and the other is a plugin for the Eclipse IDE with tight integration with OPTIMUS.

4.2 Optimus

The objective of OPTIMUS is to provide the five best possible application deployment schemas, based on the NFRs and the properties of the multi-cloud application. OPTIMUS starts when the developer has implemented and tested the application as shown in Figure 1. OPTIMUS provides the following functionalities:

Application Classification: This functionality involves associating the components (microservices) that form the multi-cloud application into a group of cloud services where they can be deployed through a classification type. For this purpose, the profiling of the microservices of the multi-cloud application is a way to match the characteristics of those microservices with those of the group of cloud services features where they will be deployed. This classification is based on the information provided by the developer and the information that the system holds and manages regarding available cloud services.

Theoretical deployment generation: When the classification is made, and the NFRs defined by the developer, OPTIMUS invokes ACSmI and obtains the cloud services that fulfil the deployment requirements of the microservices. This invocation request is composed of generic characteristics of the cloud services and the list of resources that the microservices need. This functionality requires interacting with the ACSmI API to obtain the list of cloud services that meet the requested criteria. The request to ACSmI is built and performed based on the classification of the microservices, the characteristics associated to it and the application NFRs.

Deployment Simulation: The combination of the different optimized possibilities of deployment, considering the theoretical and individual deployment possibilities for each microservice and the list of cloud services (from ACSmI) that suit them, is ranked to select the five best deployment schemas and presented to the developer for selection. This process is implemented using a genetic algorithm for combine and optimize problems. The algorithm that implements this functionality is the NGSA-II. This algorithm is “one of the most popular multi objective optimization algorithms with three special characteristics, fast non-dominated sorting approach, fast crowded distance estimation procedure and simple crowded comparison operator” [15]. This process is also followed when a violation is detected by ACSmI or ADAPT. In this case, ADAPT will launch a new OPTIMUS simulation. This new simulation takes into account the previous deployment schema history and also discards those cloud services that have had SLA violations in a specific timeframe.

4.3 ACSmI

The Advanced Cloud Service (meta-) Intermediator (ACSmI) aims to provide the means for the discovery, contracting, managing and monitoring of different CSP offerings. ACSmI provides the means to continuously assess the fulfilment of SLA from cloud service offerings, including legislation compliance. ACSmI comprises four main components: Discovery, Contracting, Billing and Monitoring

ACSmI interacts in several steps in the lifecycle illustrated in Figure 1:

- ACSmI Discovery is used by OPTIMUS to get the list of cloud services that fulfil the requirements.
- Once the deployment schema is selected and the MCSLA is agreed, the process of contracting the services starts. ACSmI Contracting provides automatic cloud services contracting.
- When the contract starts, the developer can use the ACSmI Billing component to know about the use and the cost of the contracted cloud services
- Finally, once the application is deployed, ACSmI Monitoring starts to monitor whether the SLAs of the contracted services are fulfilled.

ACSmI implements the following functionalities:

Discovery of services stored in the service registry: ACSmI Discovery implements the discovery of the services based on NFRs such as availability, performance, location, legal level and cost. This functionality is used by:

- OPTIMUS: ACSmI provides OPTIMUS with the services that are in the registry that fulfil the requirements. Not only the services that fulfil completely all requirements are selected, but rather ACSmI presents a short list with the information of the requirements fulfilled and the coverage degree.
- MCSLA: ACSmI provides MCSLA with the information regarding the SLOs of the service that has been selected.

Adding services in the ACSmI service registry: The fields to be completed are adapted depending on the type of service to be endorsed. ACSmI implements the endorsement process collecting the information required to complete the SLOs for NFRs like availability and performance. It also provides information (attaching contracts and other information) that enables the legal experts to answer a set of questions that allow the assignment of a legal level based on the analysis of this information provided in the questionnaires, both by the CSPs and by the legal expert. The registry also records the SLA violations (if any) of each service, as this information is provided by ACSmI Monitoring

Enabling the contracting of the selected cloud services: Contracting may be accomplished either through the ACSmI Contracting or by the developer directly with the CSP. Currently, ACSmI Contracting supports automatic contracting for the following CSPs: Amazon, Cloud Sigma and Arslys.

To enable the “automatic” execution of contracts: ACSmI contracting includes and manages the connectors needed to facilitate the start-up of the contracted services.

Monitoring of the CSPs: ACSmI Monitoring implements the detection of any violation according to the SLOs and the related alerting mechanism. In order to detect the violations, this component meters the behavior of the services, records these metrics in a time series database and assesses the fulfilment against the SLA.

ACSmI Billing component is able to monitor the use of the cloud services and provide the cost in a defined billing period. This cost is provided at two levels:

- At cloud service level: this enables the monitoring of the cost of each contracted cloud service
- At application level: this provides the information of the cost for all the cloud services involved in an application.

4.4 MCSLA editor

The accumulation of a number of SLAs from different CSPs is defined in DECIDE as a multi-cloud native application composite SLA (MCSLA). When a multi-cloud native application is ready for deployment, an MCSLA has to be defined. The purpose of the MCSLA is twofold; it acts as the contract between the end-users and the developer of the application and it is used for monitoring purposes by ADAPT which assesses it at runtime to ensure that it is satisfied. A cloud SLA is typically composed of a number of Service Qualitative Objectives (SQO) and Service Level Objectives (SLO) as defined in ISO/IEC 19086-1. The SLOs and SQOs represent, amongst other factors, the non-functional requirements of an application and its underlying infrastructure.

The MCSLA Editor serves as the user interface through which the developer will specify the multi-cloud SLAs agreed with the client. The MCSLA editor provides the developer all of the possible SLOs and SQOs, which may partly incorporate default values, aggregated values or overwritten values depending on the values resulting from the contracted CSPs.

Once a developer sets an MCSLA and communicates to the end-users, certain terms should not be changed. For this reason, two layers are defined: an external one, which has to be respected and cannot be changed when a re-deployment takes place, and an internal one that collects the SLAs from the various providers where the application has been or will be deployed. The external SLA is a composition of the internal SLAs plus the application SLOs. In case of a candidate redeployment involving different services or CSPs, the internal SLAs change accordingly, but their composition should still satisfy the external SLA for the candidate to be acceptable. If no such candidate exists, the adaptation (i.e. re-deployment) fails.

4.5 ADAPT

ADAPT implements the deployment, monitoring and adaptation parts of the DevOps lifecycle. It is a tool that allows automatic deployment and adaptation of a multi-cloud application and re-deployment in another multi-cloud configuration when certain conditions are violated.

The targets of ADAPT are cloud native applications developed according to the principles of micro-services architecture and to be deployed on multiple clouds using container technology. The information needed by ADAPT to perform its functionalities is provided by the application description.

The functionalities of ADAPT can be summarized as follows:

- Provisioning of the needed cloud resources from different providers by invoking the unified API offered by ACSmI
- Configuration of the provisioned infrastructure
- Automatic deployment of container-based applications based on the configuration indicated in the Application Description
- Collection of monitoring data from both the application and the cloud providers hosting it (in collaboration with ACSmI)
- Continuous analysis of the collected metrics to identify possible violations of the MCSLA
- Triggering the adaptation and performing it through re-deployment in order to restore the correct working conditions in case the SLA has been violated

The ADAPT implementation leverages on proven open source technologies and tools, such as Terraform [22], Influx DB [14], Grafana [12] and Telegraf [21].

ADAPT starts by reading the application description, from which it generates the needed Terraform configuration files describing the expected status of the multi-cloud infrastructure. Executing those configuration files results in the provisioning and configuration of the cloud resources from different providers indicated in the deployment schema. When the infrastructure is ready, ADAPT uses the same mechanism, generating infrastructural code from the application description and executing it, to deploy the containers running the application's microservices, each one in their respective provider.

After the deployment, ADAPT extracts the SLA from the application description, dynamically creates the configuration needed to collect the required metrics and starts analyzing the monitored values to identify possible violations. Metrics from the cloud providers are collected by ACSmI and analyzed by ADAPT along with those from the application.

In case of a violation ADAPT intervenes and decides how the adaptation should proceed: if the application's technology risk has been classified as low, the adaptation is automatic, otherwise the operator is alerted, and the re-adaptation should be manually confirmed. In the automatic flow ADAPT triggers OPTIMUS to create a new deployment configuration and the workflow then transitions back to ADAPT to obtain a redeployment.

Redeploying an application involves stopping it, un-deploying the current configuration and deploying the new one. Application adaptation using redeployment for each SLA violation may impact business continuity. Techniques to avoid such an impact have been studied and have been reported in [6]. Some of the studied techniques apply to the application and can be described as development patterns (e.g. a microservices architecture, stateless containers, replication), whereas other techniques involve the re-deployment procedure itself (e.g. Blue-Green deployment).

5. VALIDATION AND RESULTS

The main objective of the evaluation process was to assess if the key results of the project accomplished the success criteria defined in the Description of Action of DECIDE project.

The validation strategy adopted makes use of an iterative process, undertaking scheduled evaluations, with feedback to developers, of the initial, interim, and final releases of DECIDE. In order to ascertain the development progress and final efficacy of DECIDE, a user centric approach was adopted and three widely differing use cases, with different requirements, were developed. These were a clinical data entry tool (StreamLine), a change-tracking center (CTC), and a block-chain based energy-trading platform.

Each of the use cases utilizes different elements of the DECIDE tool suite and, combined, they enable a robust evaluation of the quantitative and qualitative impacts which DECIDE will have on the design, development, deployment and monitoring of multi cloud applications.

A combined set of prioritized use case requirements was established and mapped against the DECIDE project KRs in [7], with success criteria identified and was used to inform the designs of the tool design and the evaluation/validation processes set out in the document reporting the Use Case Validation [8].

The evaluation dimensions are defined as: Availability, Efficiency, Usability [3], Flexibility, Interoperability, Reusability, Reliability, Testability and each of the KRs is assigned appropriate dimensions with individual evaluation strategies.

The use case developers utilized their applications to evaluate them against these criteria at two release points in the development, initial [8] and interim, with final to be completed when released. These interim results were collated, moderated and fed back to the developers of the DECIDE modules.

The results of the evaluation of the initial release modules were limited at the point of reporting, principally because many of the use case requirements had not been planned for implementation in the initial release. However, evaluating the initial modules indicated clearly that, even at that point, a significant number of the mutually agreed (by Use Case owners and module developers) requirements had been fulfilled, with the remainder to be addressed in iterative releases until finally completed in the final release.

Evaluation of the interim release suffered from some of the same issues that beset the initial release in that whilst the modules were substantially more developed, not all of the functionality was present (nor scheduled to be). In addition, the interim release was the last release of the pre-integration version of the tools so the workflow issues, which were evident in the initial release of the tools, persist. This has meant that results, thus far, are limited to qualified, predominantly positive, comments, particularly around usability and function. However, the progress in the development of features meeting the requirements from the use cases' requisites [7] is discernable. One positive quantitative result is a reported 50% efficiency improvement when using deploying a multi cloud application using the ADAPT module. This provides confidence that post integration evaluation will demonstrate the designed for functionality and usability benefits first indicated in the initial.

6. CONCLUSIONS

This paper presents a tool-based approach for the adaptation of the traditional DevOps philosophy to the specific case and needs of applications distributed over different cloud resources (multi-cloud applications).

The solution described is intended to solve some of the challenges of multi-cloud applications design, development, deployment and adaptation, providing a novel DevOps approach and creating new tools to support it, addressing the identified challenges: The novel concept and tools created for the implementation of the extended "DevOps" approach for multi-cloud native applications presented in this paper have been validated in three use cases (clinical data entry tool, a change-tracking center, and a blockchain based energy-trading platform). Initial evaluation reported significant efficiency improvements when using DECIDE framework for developing a multi cloud application.

Both the DECIDE Framework and the described tools are still in active development by the DECIDE project's partners. The software has been put in open source, so other developers can also contribute to it. The current version is still a prototype but future evolutions are planned to improve both quality and functionalities. For example, some new features currently being analyzed include

the support for stateful containers, deployment on Kubernetes, new monitoring metrics and the integration of new cloud providers. In addition, whole system stress tests will be undertaken, continuously running multiple applications for a long period of time, to verify the system's stability and improve where possible.

7. ACKNOWLEDGMENTS

The project leading to this paper has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 731533.

8. REFERENCES

- [1] AWS Developer Tools: <https://aws.amazon.com/products/developer-tools/>.
- [2] Azure DevOps: <https://azure.microsoft.com/en-us/solutions/devops/>.
- [3] Brooke, J. 1996. SUS-A quick and dirty usability scale. *Usability evaluation in industry*. 189, 194 (1996), 4-7.
- [4] Cloud & NFV Orchestration based on TOSCA: <https://cloudify.co/>.
- [5] DECIDE Consortium 2019. *D3.3 Final architectural patterns for implementation, deployment and optimization*.
- [6] DECIDE Consortium 2019. *D4.3 Final DECIDE ADAPT Architecture*.
- [7] DECIDE Consortium 2018. *D6.2 Final Use Case Requirements Capture*.
- [8] DECIDE Consortium 2018. *D6.5 Use Cases Evaluation*.
- [9] DECIDE H2020: 2016. <https://www.decide-h2020.eu/>.
- [10] Fehling, C. et al. 2014. *Cloud Computing Patterns Fundamentals to Design, Build, and Manage Cloud Applications*.
- [11] Fleck, M. et al. 2014. Towards pattern-based optimization of cloud applications. *CEUR Workshop Proceedings* (2014), 16-25.
- [12] Grafana monitoring platform: <https://grafana.com/>.
- [13] IBM UrbanCode: <https://developer.ibm.com/urancode/>.
- [14] Influx DB database: <https://www.influxdata.com/>.
- [15] Lwakatare, L.E. 2017. *DevOps Adoption and Implementation in Software Development Practice*. University of Oulu.
- [16] Marionette Collective: <https://puppet.com/docs/mcollective/current/index.html>.
- [17] Mell, P. and Grance, T. 2011. The NIST Definition of Cloud Computing. Recommendations of the National Institute of Standards and Technology. *NIST Special Publication*. (2011).
- [18] MELODIC H2020: 2017. <http://www.melodic.cloud/>.
- [19] Microfocus Enterprise DevOps: <https://www.microfocus.com/en-us/trend/enterprise-devops>.
- [20] ProActive Workflows & Scheduling - Job Scheduler: <https://www.activeeon.com/products/workflows-scheduling/>.
- [21] Telegraf agent: <https://www.influxdata.com/time-series-platform/telegraf/>.
- [22] Terraform framework: <https://www.terraform.io/>.
- [23] Understanding DevOps: <https://www.redhat.com/en/topics/devops>.
- [24] Xebialabs DevOps platform: <https://xebialabs.com/products/>.