Spring 3-16-2024

Defining DevOps Projects Workspaces

Jordan Shropshire

Prathyusha Rayapati

Jeffery Holifield

DEFINING DEVOPS PROJECT WORKSPACES

Jordan Shropshire
University of South Alabama
Mobile, AL
jshropshire@southalabama.edu

Prathyusha Rayapati
University of South Alabama
Mobile, AL
pr2121@jagmail.southalabama.edu

Jeffrey Holifield
University of South Alabama
Mobile, AL
jholifield@southalabama.edu

ABSTRACT

Software development organizations are increasingly reliant on DevOps teams to develop, test, and deploy software for internal and external customers. These teams of software and systems engineers are responsible for delivering microservices on cloud infrastructure. They use a wide variety of externally-hosted DevOps tools and services to accomplish their tasks. For a given project a DevOps team will rely on a unique combination of tools such as code repositories, continuous integration agents, test suites, vulnerability scanners, and cloud infrastructure. It can be difficult to define, observe, and manage this collective workspace. However, managers wishing to stay abreast of team performance and project status should possess an accurate depiction of its workflows. Thus, this research proposes a novel method for accurately defining DevOps project workspaces. The method involves parsing activity logs and fusing data points to render integrative views of project landscapes. The method's evaluation and research implications are provided.

Keywords

DevOps, cloud, internal development platform, workflow, management

INTRODUCTION

Microservices are the building blocks of Software-as-a-Service (SaaS) companies. They represent both an architectural and organizational approach to software development. They are characterized as small, independent services that communicate over well-defined interfaces. Microservice architectures are designed for cloud architecture. They are easier to scale, faster to develop, and allow for more flexibility (De Lauretis, 2019). Further, they reduce time-to-market and enable innovation. Each component in a microservice architecture can be developed, modified, and redeployed independently of other services. This independence increases the resilience of a microservice architecture. One component can fail independently of the others. These services are developed and maintained by small teams called DevOps teams.

A DevOps team consists of software developers and system administrators. These individuals work collaboratively throughout a microservice's lifecycle – from its inception through its development, deployment, and to its retirement. This means they are not only responsible for planning and coding the microservice, but they also have to deploy it. When the microservice fails, they are responsible for resolving faults and bringing it back into production.

It has been stated that responsibility has shifted left in terms of DevOps. Teams are responsible for more than just software development and deployment (Bogner et al., 2019). They must also consider testing, security, storage and other factors. Thus, they now must also master a number of tools and platforms in order to render microservices. This includes tools for tasks such as collaboration, codebase management, testing, deployment, execution, and monitoring. The combination of tools uniquely implemented for a project is considered the project workspace. DevOps teams have broad responsibility and wide latitude for managing project workspaces.

Owing to variations in teams and projects, it can be difficult for third parties to accurately define project workspaces (Wang et al., 2021). By definition, microservices are deployed using different combinations of tools and resources. Even if two microservice projects use the same toolchains, there will be separate implementations for each project (Bogner et al., 2019).

For instance, each project will have its own code repository, different continuous integration scripting, and a unique combination of cloud-based resources. Further, tool implementations may not adhere to naming conventions or resource tagging. This makes it difficult to non-team members to know details such which specific code repository to associate with a given project (Wang et al., 2020). Furthermore, if there has been turnover within a project team then new members will be of little help in tracking down project details. This points to a need for a method for workspace mapping which relies on independent sources of data.

Thus, the purpose of this research is to introduce a novel method for developing integrative models of DevOps project workspaces. The proposed new method uses a combination of a manual and automated techniques to define a project workspace. The proposed method can be summarized in terms of 4 activities: log collection, data extraction, activity mapping, and workspace definition. Log collection consists of collecting recent raw log files from the pool of organizationally-adopted DevOps tools and services. For data extraction, log files are parsed and specific data elements are recovered. Activity mapping involves sorting data elements extracted from the various log files into groups and then creating sequences of activity. For workspace definition, the DevOps tools and services adopted for the project are identified via the activity sequence. The four activities can be performed manually or automatically using processing scripts.

The present study is one of the first to recognize the complexity of DevOps Workflows and propose a method for systematically defining project workspaces. Although project contributors may be aware of the development and deployment landscape, this information is not readily-apparent to those who are not immediately involved. However, once-removed parties have need for this information. For instance, managers need this information in order to track progress and worker performance, security teams need implementation particulars in order to assure the integrity of the deployment pipeline, and financial analysts need to link project expenses such as pay-as-you-go resource usage with specific microservices.

The remainder of the research introduces the proposed method in more detail and describes its implementation. The next section contains the literature review. It is composed of two subsections: one on microservices and another on DevOps. The next section describes the theoretical development by summarizing the proposed method and describing each of the four main components in separate subsections. Next, an evaluation of the proposed method is described. The outcome of the evaluation is described in the results section. Finally, concluding comments are provided.

In order to appreciate the proposed new method some contextual information is necessary. Hence, this section describes the concept of microservices then it discusses DevOps. It focuses on tools which support DevOps workflows.

LITERATURE REVIEW

In order to appreciate the proposed new method some contextual information is necessary. Hence, this section describes the concept of microservices then it discusses DevOps. It focuses on tools which support DevOps workflows.

Microservices

Contemporary software-as-a-service offerings are comprised of suites of microservices. Each microservice runs as its own process and communicates with a lightweight API (Liu et al., 2020). Microservices are built around business capabilities and are independently deployable via established DevOps pipelines. Microservices are minimally integrated. A group of microservices may be written in different languages and use different storage schema even though they all contribute toward a single product vertical (Doerrfeld, 2023). A major benefit of microservice architectures is that they are horizontally-scalable. It is possible to add or remove microservice instances according to real time demand (Wang et al., 2021).

DevOps

The DevOps lifecycle is a series of iterative, automated deployment processes or workflows. Sometimes, the path between processes is known as a pipeline (Toh et al., 2019). Within the organization, DevOps requires communication, collaboration, and shared responsibility among stakeholders. Hence, DevOps teams are commonly comprised of software engineers and IT operations specialists (Shropshire et al., 2017). At the technical level, DevOps requires a commitment to automation that keeps projects moving within workflows. This allows for feedback and measurement that enable improvement in software quality and performance.

The demands of DevOps place a premium on tooling that supports automation, scalability, and integration. There are numerous types of DevOps tools (see Figure 1). Many of these tools fall into one of the following 5 categories: source code repositories, CI/CD agents, test automation frameworks, configuration management tools, and infrastructure tools.

• Source code repositories: provide version-controlled coding environments which allow multiple developers to contribute to the same codebase. Code repositories are usually integrated with CI agents so that when code is committed to the repository a number of downstream tasks are automatically queued.

• CI (Continuous Integration) Agents: perform automated code checkout, building, testing, and deployment. They start tests, security checks, and perform other tasks prior to moving a containerized software to its final storage location.

• Test automation tools: include libraries, tools, and code to automate unit, integration, and functional tests, performance analysis, penetration testing, and usability testing.

• Configuration management tools: also called deployment or orchestration tools. They configure and provision fully versioned infrastructure according to script.

• Infrastructure tools: resources that may include public or private cloud infrastructure. In some cases, the infrastructure tools are considered to be separate but related to DevOps.

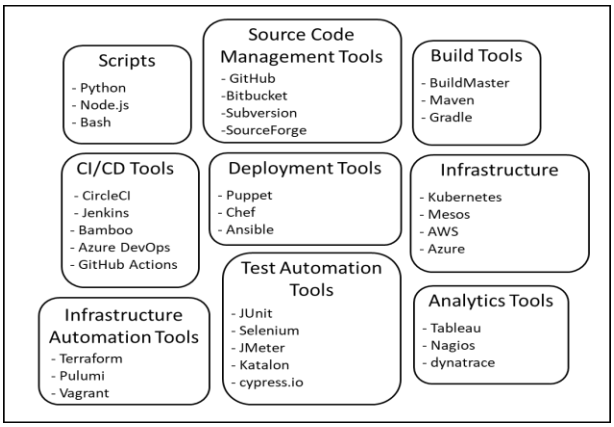

Figure 1: Examples of DevOps tools

Having provided background information on microservices and DevOps, the next section will introduce a new method for determining where and how specific tools are employed in conjunction with various projects.

Related Work

Although this is one of the first studies oriented towards manually mapping DevOps pipelines, several related have been conducted in this general area. One study (Zampetti et al., 2021) combined a qualitative and quantitative analysis of the evolution of CI/CD pipelines. This study developed 16 metrics to measure pipeline drift over the lifetime of a microservice. Another study (Zheng et al., 2022) developed a solution called BuildSonic to monitor CI pipeline configurations and detect sub-optimal configurations. Additionally, Bello *et al.* (2022) proposed a new framework for integrating continuous delivery services into software-defined systems. The proposed framework provided a means of documenting the entirety of the pipeline in order to ensure replication and rebuilding if necessary.

THEORETICAL DEVELOPMENT

It has been suggested that the typical commercial Software-as-a-Service product consists of a median of 53 microservices (Li et al., 2021). Given that senior engineers and product manager may have multiple teams and dozens microservices to shepherd, it is not possible for them to actively participate in the development of every project. However, they must monitor, manage, and control the progress of each microservice while it is in development and remain abreast of its functionality after it enters production (Cinque, 2022). This requires visibility into each project's workspace. A well-defined project workspace will contain details such as: which code repositories are associated with a specific microservice, which continuous integration agents are being used, where images are stored, and how code is deployed (De Lauretis, 2019). This information will allow third parties to make informed decisions regarding resource allocation, create cost projections, anticipate support requirements, and monitor service quality ((Wang et al., 2021)).

Hence, this section introduces a novel method for defining DevOps project workspaces. It consists of four activities: log collection, data extraction, activity mapping, and workspace definition (See Figure 2). The first activity includes collecting recent raw log files from the pool of organizationally-adopted DevOps tools and services. For the second activity, log files are parsed and specific data elements are recovered. The third activity involves sorting data elements extracted from the various log files into groups and then creating sequences of activity. For the fourth activity, the DevOps tools and services adopted for the project are identified via the activity sequence.

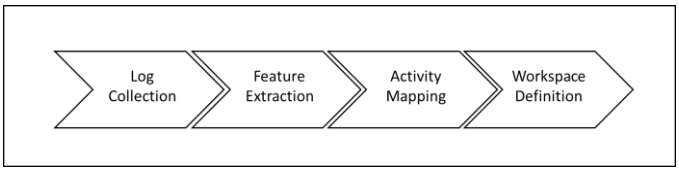

Figure 2: Overview of Proposed Workflow

Log Collection

This activity begins with identification of all the organizationally-adopted DevOps tools. Organize all the tools into their respective categories within the taxonomy of DevOps tools. Separately collect the logs for each tool for each DevOps contributor. It will be necessary to separately analyze each DevOps contributor's activities in order to sensibly group their activities into workflows. The extant of log length is dependent on project velocity. If a team is actively committing code to a microservice will be ample data within a short timeline. However, if a project is relatively stable then it may be necessary to retrieve logs which go back further.

Data Extraction

Once the logs for each tool have been collected, the next step is data parsing. The goal is extract data which defines individual instances of using DevOps tools. These are herein called activity instances. In latter sections these individual activity instances will be combined into groups which describe activity patterns.

Review the log for each tool within the code repository category. Within each log, look for activity instances. In this context an activity instance could be a git push, pull, commit, or something else. For each activity instance, extract the following data: username, time, repository code path, etc. Save these instances.

Inspect the log for each tool within the CI agent category. Search for activity instances. For this category, an activity instance could be containerizing code, initializing security checks, and/or code testing. For each instance of continuous integration activity, extract the following data: username, time, accessed code, script, etc. Retain these instances.

Check the logs for each tool in the test automation category. Within each log, look for instances of unit tests, code evaluations, and performance analyses. These are test automation activity instances. For each instance of test automation activity, extract username, date, time, test name, tool name, script name, and software/code name.

Examine the logs for each tool in the configuration management category. Identify activity instances. These will involve image creation and instance deployment. For each instance of configuration management activity, extraction username, date, time, project name, project ID, image ID, and image repository path.

Finally, check the logs for each tool in the cloud infrastructure management category. Identify activity instances. These will involve resource orchestration, network configuration, and instance management. Extract cloud resource ID, project ID, image ID, and instance ID as well as username, date, time. Save each activity instance.

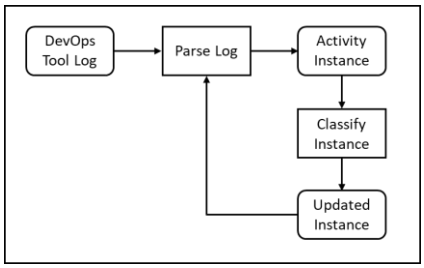

Figure 3: Log collection and data extraction cycle

Activity Mapping

After all the individual activities have been extracted the next step is to amalgamate them into groups which describe activity patterns within DevOps projects. These groups consist of starting activities, middle activities, and ending activities. Collectively, they trace the route that code takes as it leaves the developer's computer, passes through DevOps tools, and heads to towards the location where it will run as a microservice. In general, the process for creating an activity group is as follows:

First, the ending activity is selected. Based on the ending activity instance, valid candidates for middle activity instances are identified. Based on the middle activity instances, valid candidates for starting activity instances are last selected.

Prior to constructing activity groups, the activity instances within each DevOps tool category are time-sorted. From the organization's list of adopted DevOps tools, the first, middle, and last categories of tools are identified. The first category of DevOps tool is a code repository. None of the other steps can be taken if code is not available. Activity instances from this category are called start activities. The last category of DevOps tool is always cloud infrastructure, as this is when the code is executing as a microservice. The activity instances from this category are called ending activities. Activity instances from all other DevOps tool categories are considered middle activities. They occur sometime after the code is pushed to a repository but before it is running on the cloud.

To construct an activity group, the earliest instance of an ending activity is first identified. Next, all the middle activity instances which concluded prior to the start of the ending activity are selected. Finally, all the potential starting events which concluded prior to the start of the earliest middle activity are added.

At this stage each activity group contains a finite endpoint as well as a number of possible starting and middle activity instances. It is now necessary to further filter the activity group to determine which specific starting activity and which middle activity instances should be associated with the ending activity. For instance, which code push, software test, and container creation event should be associated with the specific container startup event. Hence, all the middle activities are analyzed according to tool category. Within each tool category, activity instances which are identical except for time and event identification number are clustered. One representative activity instance is retained from each cluster while the other instances are removed. Further, all activity instances with a status of fail are removed. The same tasks are performed for the starting activity so that a small group of possible start and middle activities are included in each activity group.

Workspace Definition

Next, all the activity groups are sorted according to the infrastructure project ID associated with the ending activity instance. These are activity groups which target the same cloud resource group and hence the same microservice. All the activity groups associated with the same microservice are sorted again according to the repository code path. The activity groups with the most prevalent repository are kept. Now the subset contains activity sequences which begin with the repository same code path and end with a software instance running on the same cloud deployment space. The left-hand sort continues until the majority set of middle activity instances is defined. The collective sequence of activities is the candidate path that the code is presumed to take as it leaves the developer's computer, updates the repository, triggers other DevOps tools, and finally enters production as a containerized script running on cloud infrastructure.

The route from IDE to microservice instance includes all the DevOps tools and resources which are needed to deploy the service. The route not only identifies the tools, but the implementation specifics. Within this information it is possible to define the project workspace. Information such as repository address/ID, CI agent configuration, test scripts, security routines, image repositories, cloud resource orchestration scripts, and instance ID are available. The information can be organized into a flow chart which captures the activity sequence. Alternatively, it could be captured within a JSON file.

EVALUATION

A small-scale evaluated was performed in order to empirically evaluate the proposed method of workspace definition. The research question is: Does use of the proposed method affect workspace mapping accuracy? The test hypothesis is that there are significant different in performance between groups who use the proposed method and groups who do not.

The test was conducted in partnership with a public sector organization which provides 3 microservices for its internal use. The organization agreed to allow the researchers to acquire activity logs from each the various DevOps tools that it uses. For each tool, at least 48 hours' worth of activity logs were acquired. The logs were given to 34 undergraduate students who are family with DevOps. The treatment group consisted of 17 individuals were acquainted with the proposed method for defining DevOps project workspaces. They were told to apply the developed method to the log files and define project workspaces. The other 17 were not trained to use the proposed method and were instructed to use their own judgment. These were the control group. Neither group was told how many microservices existed or given any other information about the organization. They were expected to submit workspace flow charts for each identified microservice. The individuals were given the logs and expected to complete their tasks within lab session which lasted approximately two hours.

The 3 project workspaces were collectively defined in terms of 47 points. Points were assigned to definitional attributes such as specific repository IDs, specific test scripts, and specific infrastructure IDs. For grading, each subject's set of workspace definitions were compared against the 47 points and a match percentage was rendered. 100% would indicate a complete match

while a 0 would indicate no matches. The goal is to determine how closely each individual's workspace definition matched the actual project workspace.

RESULTS

It was determined that a two-sample t-test is an appropriate method to evaluate the difference in performance between individuals trained to use the proposed new method and individuals who are equally familiar with the problem space but not trained to use the method (Hair et al., 1998). The data is normally distributed, the measurements are continuous, the variances between the groups are equal. For each group, the average match score, standard deviation, and sample size were determined (see Table 1). Next, the pooled standard deviation was calculated and the t-statistic was derived. The t statistic was compared against the t value. The difference is significant at the 0.05 level. This provided sufficient evidence to reject a null hypothesis that the difference between the two groups would be the same. It can be inferred that match percentage differences between groups who use the proposed method and those who do not. The results of the evaluation broadly support the efficacy of the new method. The outcomes are summarized in Table 2, below.

Group	Control(Default Method)	Test (Proposed Method)
Mean	0.7400	0.8400
Standard Deviation	0.1100	0.1300
Standard error of Mean	0.0267	0.0315
N	17	17

Table 1. Accuracy at Defining DevOps Workspaces

T statistic	2.4212
Degrees of Freedom	32
Standard Error of Difference	0.041
Two-tailed P value	0.0213

Table 2. Two sample test results

CONCLUSION

Software-as-a-Service (SaaS) organizations depend on DevOps teams to design, deliver, and support microservice applications which are resilient, effective, and scalable. In order to deliver reliable microservices these teams use a number of tools and resources. Although an organization may formally adopt a finite set of DevOps tools, the implementation of toolsets varies among projects. Managers wishing to observe, monitor, and safeguard their microservices should possess accurate definitions of each microservice's workspace.

Hence, this research proposed a new method for mapping microservice workspaces. In a small-scale evaluation is was determined that the use of the method effects accuracy at workspace mapping. The average score of the group using the proposed method was significantly higher than the control group. It can therefore be inferred that the structured approach to defining workspaces is a valuable contribution to the field of DevOps.

Beyond DevOps, this research has implications for the field of platform engineering. Platform engineers combine disparate DevOps tools into homogenous development platforms. Internal development platforms alleviate the cognitive load associated with using a variety of tools. If properly designed they allow developers to focus on designing microservices which meet customer needs. In order to achieve their aims, platform engineers require accurate depictions of existing project workspaces. This research provides an important method for mapping workspaces.

Future research should focus on automating the proposed method so that it easily scales for large SaaS organizations. These studies should focus on automating the process of documenting the pipeline configurations and rendering their architecture using a markup language such as YAML or JSON. This would free developers from having to manually track their implementation and allow for deterministic reconstruction. This could be implemented as a service which uses a configuration agent such as Travis. If successful, it may not be necessary for humans to perform the data extraction, grouping, and filtering tasks which are part of the method. Further, machine learning algorithms could be applied to observe patterns and render even

more accurate mapping than humans. With these advances, it would be possible to stay abreast of workspace changes in near real time. This would allow managers to make better decisions regarding microservice management.

REFERENCES

1. Bello, Y., Figetakis, E., Refaey, A., Spachos, P. (2022) "Continuous Integration and Continuous Delivery Framework for SDS," 2022 IEEE Canadian Conference on Electrical and Computer Engineering, pp. 406-410.
2. Bogner, J., Fritzsch, J., Wagner, S., Zimmerman, A. (2019) "Microservices in Industry: Insights into Technologies, Characteristics, and Software Quality," 2019 IEEE International Conference on Software Architecture, pp. 187-195.
3. Cinque, M., Della Corte, R., Antonio, P. (2022) "Microservices Monitoring with Event Logs
4. and Black Box Execution Tracing," IEEE Transactions on Services Computing, 15(1) pp. 294-307.
5. De Lauretis, L. (2019) " From Monolithic Architecture to Microservices Architecture," IEEE International Conference on Software Reliability Engineering Workshops, pp. 94-96.
6. Doerrfeld, B. (2023) "Strategies To Consider When Adopting Platform Engineering," DevOps.com, retrieved from: https://devops.com/strategies-to-consider-when-adopting-platform-engineering/
7. Hair, J., Anderson, R., Tatham, R., Black, W. (1998) Multivariate data analysis (5th ed.). Upper Saddle River, NJ: Prentice Hall.
8. Li, S., Jia, Z., Zhong, C., Zhang, C., Shan, Z., Shen, J., Babar, M. (2021) "Understanding and addressing quality attributes of microservices architecture: A Systematic literature review," Information and Software Technology, 130(2) pp.1-23.
9. Liu, G., Liang, Z., Qin, M., Zhou, H., Li, Z. (2020) "Microservices: architecture, container, and challenges," IEEE International Conference on Software Quality, Reliability and Security Companion, pp. 629-635.
10. Shropshire, J., Menard, P., Sweeney, B. (2017) "Uncertainty, personality, and attitudes toward DevOps," Twenty-third Americas Conference on Information Systems, pp.1-10.
11. Toh, M., Sahibuddin, S., Mahrin, M. (2019) "Adoption Issues in DevOps from the Perspective of Continuous Delivery Pipeline," ICSCA '19: Proceedings of the 2019 8th International Conference on Software and Computer Applications, pp 173-177.
12. Wang, Y., Kadiyala, H., Rubin, J. (2021) "Promises and challenges of microservices: an exploratory study," Empirical Software Engineering, 26(1), pp. 1-44.
13. Waseem, M., Liang, P., Shahin, M. (2020) "A Systematic Mapping Study on Microservices Architecture in DevOps," Journal of Systems and Software, 170(2), pp. 1-30.
14. Waseem, M., Liang, P., Shahin, M., Di Salle, A., Marquez, G. (2021) "Design, monitoring, and testing of microservices systems: The practitioners' perspective," Journal of Systems and Software, 182(4), pp. 4-48.
15. Zampetti, F., Geremia, S., Bavota, G., Di Penta, M. (2021) "CI/CD Pipelines Evolution and Restructuring: A Qualitative and Quantitative Study," 2021 IEEE International Conference on Software Maintenance and Evolution.
16. Zhang, C., Chen, B., Hu, J., Peng, Z., Zhao, W. (2022) "BuildSonic: Detecting and Repairing Performance-Related Configuration Smells for Continuous Integration Builds," Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering, pp. 1-13.