Gradual Deployment in Practice: Experiences from an Industrial Case Study

Eveliina Pakarinen
Cinia, Ltd.
Helsinki, Finland
eveliina.pakarinen@cinia.fi

Tommi Harakkamäki
Cinia, Ltd.
Helsinki, Finland
tommi.harakkamaki@cinia.fi

Tommi Mikkonen
University of Helsinki
Helsinki, Finland
tommi.mikkonen@helsinki.fi

***Abstract*—Short and rapid software development cycles have encouraged deploying software more frequently than ever before. At the same time, much of the features of the software being designed rely on services that cannot be set up for testing configurations, but are constantly live and in production. This has inspired a development approach where new features are gradually deployed, so that they can be tested live, but with limited influence if something goes wrong with the deployment. Putting such an approach to practice requires tools for managing the different software versions as well as their deployment and operations. In this paper, we present a case study on gradual deployment of new versions to a web computing system, executed at a Finnish software company. Furthermore, we also present a proof-of-concept implementation of a tool, which enabled gradual deployment to a cluster-based runtime environment. With this tool it was possible to deploy a new version of a software to a specific group of users or to rollback a deployment automatically in the company cluster environment.**

***Index Terms*—Continuous delivery, gradual deployment, test configurations**

I. Introduction

Continuous integration, continuous delivery, and continuous deployment are examples of activities in a roadmap for continuous software engineering [2]. Especially terms continuous delivery and continuous deployment are used as synonyms in scientific literature even if there can be found differences in their definitions [7]. According to one definition, in continuous deployment software that has passed quality checks is deployed automatically into production with a deployment pipeline while in continuous delivery the deployment is initiated manually [3].

The automation of the deployment of software enables the possibility to deploy an experimental version of the software into production so that feedback can be collected from a subset of the online users [8]. For example while using services such as Facebook, the end users will receive new software several times a day [1]. The data collected from the usage of an experimental version of the software can be used to guide the development activities and this practice can be referenced as continuous experimentation [8].

One of the often mentioned benefits of continuous deployment is that software can be deployed faster and more often to customers than while using traditional software development process [7]. In this scenario it is also possible to deploy bugs faster into production. One goal of continuous experimentation can be to find technical problems in production by utilizing techniques like canary releases [3] and gradual rollouts [3] to do experimentation [8]. These techniques can be used for testing in production to perform multi-phase live testing strategies for which a formal model with a prototype implementation has been proposed [9].

In this paper we define, that in gradual deployment when a new version of a software is deployed into production the new version is first made visible only to a subset of the users of the software. After that the new version of the software is made gradually visible to more and more users. In this paper the implementation technique for this gradual deployment is to use proxies for runtime traffic routing and to use a separate application to control the configurations of the proxies as proposed by [9]. Based on that proposition, a prototype tool was implemented in the case company to be used as part of the deployment pipeline.

In this paper, we present the prototype implementation of the tool for gradual deployment that was developed in the case company. This tool allowed deploying an application gradually in a cluster-based runtime environment and the functionality of the tool was verified by executing multiple test case scenarios of gradual deployment. The study has also provided the empirical results for a Master's Thesis [6], based on which this paper has been written. The work was conducted in a business-to-business company to study strategies for gradual deployment in practice at the implementation level.

II. Case Study

To study how strategies for gradual deployment can be applied in practice in business-to-business context, a case study was conducted. The research question of the case study was *How the gradual deployment of a web application can be implemented in practice at the case company?* During the case study a prototype of a tool for gradual deployment of web applications was developed. With the tool, it was possible to deploy a web application in a cluster-based environment. Multiple test cases were used to validate in practice that the tool satisfied most of the requirements set for it.

A. Case Company

The case company Cinia Ltd.[1] is a Finnish IT service company that offers software consultancy, design and development services for energy sector, healthcare, logistics, security and manufacturing. In its agile development process, the case company utilizes continuous software engineering practices like continuous integration. Many of the customers of the case company have a need to release features frequently and without downtime. There is also a need to make releases gradually for example by introducing a feature first to a limited group of users before releasing it to everyone. Furthermore, there has to be an option to rollback a release if users report problems or issues are detected automatically during or after the release. Based on these needs from the company perspective the goal of the case study was to develop a tool for gradual deployment to be used in the release process.

B. Requirements and Environment

One use case for the tool was to be able to deploy into a cluster-based runtime environment an experimental version of a software from which feedback could be collected. Additionally there was a need to be able to gradually increase the load of the traffic directed to the new software version. Based on the needs of the case company eight requirements for gradual deployment were identified. These requirements were revised in practice with test cases to validate that the tool satisfied the requirements set for it. The requirements, their descriptions and test cases used to validate them are presented in Table I.

To verify the requirements an application under test was needed. The application under test was a simple, stateless back-end web application. For testing purposes four versions of the application under test were developed. The versions are introduced in Table II.

The environment for gradual deployment in this case study was an on-premise Kubernetes[2] cluster. During the case study an Istio[3] service mesh was installed into the cluster to enable runtime traffic routing in the cluster environment.

C. Implementation techniques

To be able to do gradual deployment in the company environment a tool for gradual deployment was implemented. This tool, called AphoDeploy, was used in combination with Istio service mesh to implement automated gradual deployment in the Kubernetes cluster. AphoDeploy is an internal software designed for the specific environment at the case company, so the source code of AphoDeploy is not freely available.

The high-level architecture of the setup in the case study is presented in Fig. 1. The implementation technique for the gradual deployment was to use Istio to do runtime traffic routing in the cluster and to use AphoDeploy to change the configurations of the Istio service mesh. By using this technique it was possible to initiate the configuration changes automatically and precisely at specific points of time.

[1]https://www.cinia.fi/ (Accessed 30.05.2020).
[2]https://kubernetes.io/ (Accessed 11.02.2020).
[3]https://istio.io/ (Accessed 11.02.2020).

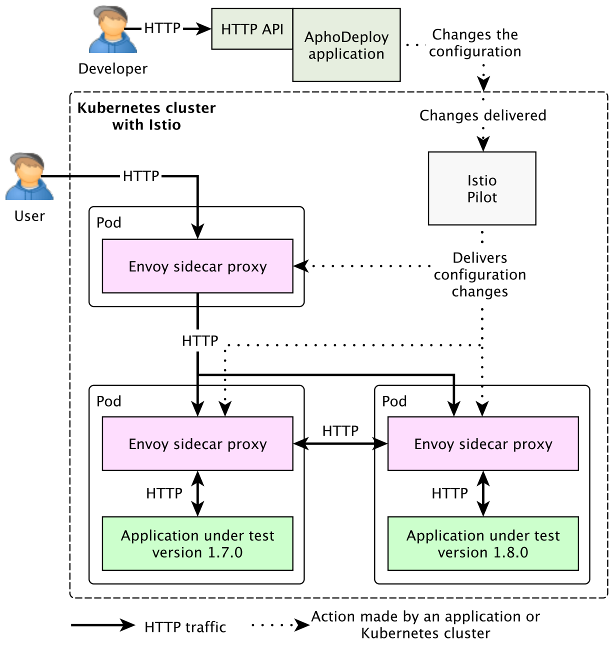
Developer
HTTP
HTTP API
AphoDeploy application
Changes the configuration
Kubernetes cluster with Istio
Changes delivered
User
HTTP
Istio Pilot
Pod
Envoy sidecar proxy
Delivers configuration changes
HTTP
Pod
Envoy sidecar proxy
HTTP
Pod
Envoy sidecar proxy
HTTP
Application under test version 1.7.0
HTTP
Application under test version 1.8.0
HTTP traffic
Action made by an application or Kubernetes cluster

Fig. 1. High-level architecture of the case study setup.

When AphoDeploy changed the configurations of the service mesh Istio Pilot[4] delivered the configuration changes to the Envoy sidecar proxies[5]. The sidecar proxies used this information to do runtime traffic routing without any modifications to the source code of the application under test [4]. In an Istio service mesh the traffic routing decisions can be based on for example the headers of the HTTP requests or the percentage of the traffic load [5]. Both of these techniques were used in this case study.

The strategy to use proxies for runtime traffic routing and to use a separate application to control the configurations of the proxies is proposed by [9]. The combination of AphoDeploy and Istio service mesh is an implementation of this strategy. The best practices for traffic management[6] in an Istio service mesh were also considered in AphoDeploy in the implementation order of the changes to Kubernetes configurations.

The Istio configurations used in this case study were predefined for each of the test cases so that AphoDeploy could directly use the configurations during the execution of the test cases. Each version of the application under test had an own Kubernetes deployment configuration. Additionally for each version there was a deployment configuration with manually injected Envoy sidecar proxy[7]. The architecture of the application under test after the installation and configuration of the Istio service mesh is presented in Fig. 2.

AphoDeploy was designed to be compatible with the com-

[4]https://istio.io/docs/ops/deployment/architecture/ (Accessed 23.02.2020).
[5]https://www.envoyproxy.io/ (Accessed 11.02.2020).
[6]https://istio.io/docs/ops/best-practices/traffic-management/ (Accessed 11.02.2020).
[7]https://istio.io/docs/setup/additional-setup/sidecar-injection/ (Accessed 22.02.2020).

TABLE I
REQUIREMENTS FOR THE GRADUAL DEPLOYMENT

#	Test case	Requirement	Description
1	Manual code analysis	No changes to source code	The selected method for gradual deployment should not affect the source code of the software. There should be no dependency between the method of the gradual deployment and the source code of the software.
2	A	Parallel execution of different software versions	It should be possible to execute multiple versions of the same software simultaneously. The goal is to enable the testing of the new software version while allowing the use of the old software version at the same time.
3	B	The length of downtime	The goal is to have zero downtime while doing gradual deployment.
4	C	Version selection based on user specific information	The goal is to deploy gradually to a specific group of users from whom feedback can be collected. Users are selected by user specific information like the user's organization.
5	D, E	The length of the gradual deployment	The possibility to change the length of the gradual deployment should be enabled. The length should be configurable in the scale from minutes to days.
6	F, G	Version selection based on the amount of traffic	During gradual deployment the version of the software the user is redirected to should be selected based on the amount of traffic. The goal is to enable the feature to gradually increase the traffic load directed to the new software version.
7	B	Cleaning up the old software version	The old software version should not consume the resources of the system after the gradual deployment has been completed.
8	H, I	Rollback during gradual deployment	Gradual deployment should be automatically rolled back if any problems are detected during the deployment. The automatic rollback is used to minimize the problems caused to the users of the software while gradually deploying software.

TABLE II
THE VERSIONS OF THE APPLICATION UNDER TEST

Version	Description
1.7.0	Returns an aphorism successfully.
1.8.0	Returns a different aphorism successfully.
1.9.0	Returns HTTP failure if the header ”Aphorism-Organization” contains characters ”apho-deploy”. Otherwise returns an aphorism successfully.
1.9.1	Returns HTTP failure if the first character of the header ”Aphorism-Organization” equals ”A”. Otherwise returns an aphorism successfully.

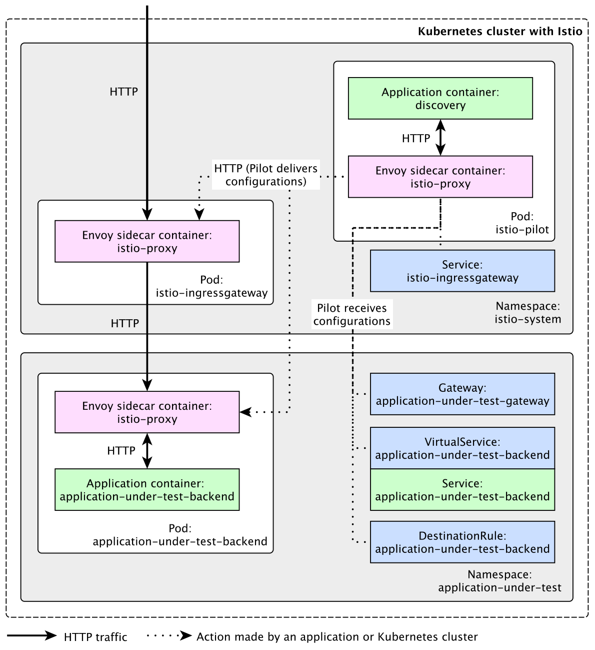
Kubernetes cluster with Istio
HTTP
Application container: discovery
HTTP
HTTP (Pilot delivers configurations)
Envoy sidecar container: istio-proxy
Pod: istio-pilot
Envoy sidecar container: istio-proxy
Pod: istio-ingressgateway
Service: istio-ingressgateway
Pilot receives configurations
Namespace: istio-system
HTTP
Envoy sidecar container: istio-proxy
HTTP
Application container: application-under-test-backend
Pod: application-under-test-backend
Gateway: application-under-test-gateway
VirtualService: application-under-test-backend
Service: application-under-test-backend
DestinationRule: application-under-test-backend
Namespace: application-under-test
HTTP traffic
Action made by an application or Kubernetes cluster

Fig. 2. The Kubernetes architecture of the application under test.

pany cluster environment which is constantly changing and under constant development. The configurations and the implementation of AphoDeploy were adapted to the company cluster environment at a specific time during fall in year 2019. After that the backbone infrastructure and Kubernetes distribution of the on-premise cluster has been changed and exactly the same implementation of AphoDeploy can no longer be used in the new environment.

Consequently, the implementation of AphoDeploy is environment specific, and every time the cluster environment changes the implementation of the gradual deployment tool must be adapted to the new environment. Therefore, AphoDeploy is considered only as a proof-of-concept implementation of a gradual deployment tool in the case company environment.

D. Results

In total, nine test cases were used to test if AphoDeploy met the requirements for gradual deployment. Each test case was executed three times in order to be able to create an average test case run from which the results could be interpreted.

With these test cases it was possible to simulate different scenarios in which AphoDeploy could be used to do gradual deployment. During the execution of the test cases Apache JMeter[8] was used to simulate HTTP traffic load to the application under test. The test case scenarios and the state transitions during the test cases can be seen from the Fig. 3. The figure also shows the results for the execution of the scenarios.

Eight of the nine test cases had a successful result when using AphoDeploy to do gradual deployment for the application under test. The test cases verified that the implementation for changing the runtime traffic routing configurations in

[8]http://jmeter.apache.org/ (Accessed 12.02.2020).

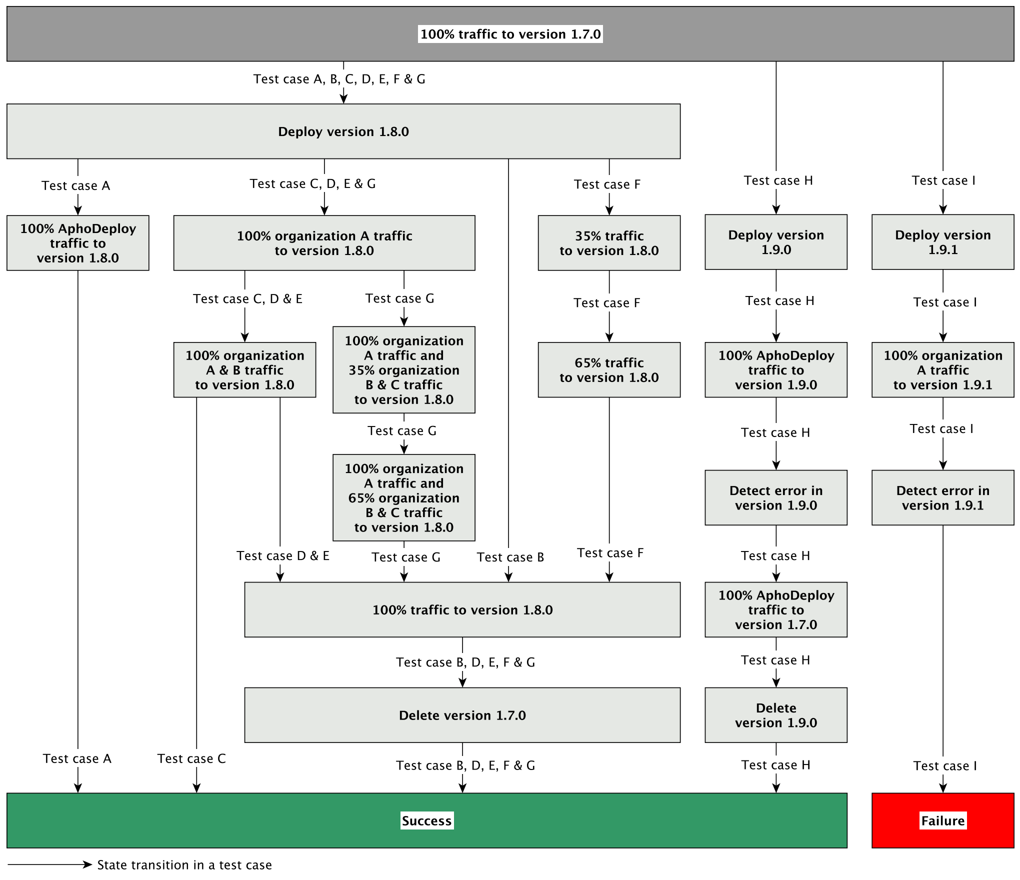
100% traffic to version 1.7.0
Test case A, B, C, D, E, F & G
Deploy version 1.8.0
Test case A
Test case C, D, E & G
Test case F
Test case H
Test case I
100% AphoDeploy traffic to version 1.8.0
100% organization A traffic to version 1.8.0
35% traffic to version 1.8.0
Deploy version 1.9.0
Deploy version 1.9.1
Test case C, D & E
Test case G
100% organization A & B traffic to version 1.8.0
100% organization A traffic and 35% organization B & C traffic to version 1.8.0
65% traffic to version 1.8.0
100% AphoDeploy traffic to version 1.9.0
100% organization A traffic to version 1.9.1
100% organization A traffic and 65% organization B & C traffic to version 1.8.0
Detect error in version 1.9.0
Detect error in version 1.9.1
Test case D & E
Test case B
100% traffic to version 1.8.0
100% AphoDeploy traffic to version 1.7.0
Test case B, D, E, F & G
Delete version 1.7.0
Delete version 1.9.0
Test case C
Success
Failure
State transition in a test case

Fig. 3. Test case scenarios and the results for the test cases.

specific order and after specific time intervals was successful. Furthermore, the test cases verified that the traffic routing could be done using user specific information like the user's organization. User specific information could also be successfully combined with the percentage based runtime traffic routing.

In one scenario AphoDeploy could also perform a rollback when it detected an error during the gradual deployment of the version 1.9.0 of the application under test. In another test case scenario the version 1.9.1 of the application under test returned a failure which AphoDeploy could not detect directly. Because of that AphoDeploy did not do an automatic rollback of this version which led to the failure of the test case.

III. Related Work

In this case study we implemented a proof-of-concept tool, AphoDeploy, which could be used to do gradual deployment in a cluster-based runtime environment. The implementation techniques used in AphoDeploy followed the techniques proposed by [9]. The main technique proposed in the related literature was to use proxies to do runtime traffic routing and a separate application to configure and orchestrate the proxies [9]. Similar technique was also used in this case study.

The use of AphoDeploy as a tool for gradual deployment had also certain limitations which were covered in related literature. One limitation was that the performance impact of using the combination of the Istio service mesh and AphoDeploy in gradual deployment was not considered in this case study. In related literature the performance impact of their prototype implementation was measured to be small [9].

The second limitation was that in this case study AphoDeploy was used to gradually deploy only one application into the cluster-based runtime environment. Related literature covered the possibility to execute over a hundred live testing strategies simultaneously [9]. However, the functionality of AphoDeploy could be enhanced in the future to enable the gradual deployment of multiple applications at the same time.

One of the key aspects of the related literature was using metrics of the environment as a part of the decision making process during the the execution of the live testing strategies [9]. The use of metrics could enable the possibility to detect anomalies during the execution of a live testing strategy and the information which metrics provide could also enable the option to do an automatic rollback if an anomaly is detected [9]. We however did not study these options in this case study.

Additionally, in this case study one failing test case scenario was that AphoDeploy could not identify that the application under test returned a failure for a specific group of users. One possible fix for this failing test case could be to make AphoDeploy aware of the metrics in the cluster environment. By evaluating the metrics while making decisions about rollback or continuation of the gradual deployment the tool could perform automatic rollbacks when needed or decide to continue the deployment. Finally, AphoDeploy was only tested with a stateless web application, which means that AphoDeploy may not be suitable for the gradual deployment of stateful web applications.

IV. Discussion

During the case study we searched an answer to the question how the gradual deployment of a web application can be implemented in practice at the case company. The answer to the research question is that at the case company the implementation must be environment specific and that it can be a combination of existing technologies to create to a custom implementation adapted to the company environment.

We defined eight requirements and based on these requirements implemented a prototype tool, AphoDeploy, which could perform gradual deployment in a cluster-based runtime environment. The functionality of the tool was verified using test cases to execute various scenarios of gradual deployment. The results of the test cases indicated that the tool could perform gradual deployment based on for example user specific information like the user's organization. Although not every part of the gradual deployment was successfully automated by the tool, the concept of the tool was found to be useful. Moreover, a number of tools that were identified proved to be a good starting point for creating a test environment for company cluster environment.

On the downside, the tool was designed to be compatible with the company cluster environment, which is constantly changing and under development. Therefore, when the cluster environment was revisited after the case study, the exact same implementation of the tool could no longer be used in the new environment. By using the requirements and test cases defined during the case study, a new implementation of a gradual deployment tool could be prepared and validated to be used in the revisited environment.

Furthermore, using the implementation details of the prototype tool as guideline it would be possible to adapt the implementation to fit into another environment, and using the test case scenarios it is possible to validate whether the new implementation still fulfills the requirements. The downside of this is that when the cluster would be revisited the next time, chances are that the tool would have to be revisited yet again. This indicates that such tools go hand in hand with the cluster, and making them truly general is not easy.

Threats to Validity. To begin with, the work presents a single case study, so there is a clear threat to validity that the results cannot be generalized. To mitigate this threat, the case study was implemented as a part of company daily operations and not as a research artefact. Furthermore, the fact that the change in the company cluster environment invalidated the prototype implementation it confirms our assumptions on the important role of system specific features.

A further threat to validity is that the first author of this paper has implemented both AphoDeploy and the test cases that validate its functionality. This could mean that the test cases may not be versatile enough and may not contain enough scenarios to find deficiencies in AphoDeploy. This threat has been mitigated by taking in at least one negative test case into the case study.

V. Conclusions

In this paper, we have presented a company case study for a proof-of-concept implementation of a method for gradual deployment in a cluster-based runtime environment. The implementation was designed to be compatible with the company cluster environment in fall 2019 and followed a model defined in [9].

Today, the backbone infrastructure has been revisited, and exactly the same implementation of the tool can no longer be used in the current environment without adapting the implementation to be compatible with the revisited environment. However, the same principles behind the implementation of the tool are also applicable in the new environment. This means that by using these principles a new version of the tool could be implemented in the future to be able to do gradual deployment in the revisited cluster environment.

Acknowledgements

This work has been supported by the Academy of Finland, project 317657. In addition, the authors wish to thank the case company Cinia Ltd. for supporting this work.

References

[1] Dror G Feitelson, Eitan Frachtenberg, and Kent L Beck. Development and deployment at facebook. *IEEE Internet Computing*, 17(4):8–17, 2013.

[2] Brian Fitzgerald and Klaas-Jan Stol. Continuous software engineering: A roadmap and agenda. *Journal of Systems and Software*, 123:176–189, 2017.

[3] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Adobe Reader)*. Pearson Education, 2010.

[4] Istio Authors. Architecture. [Online], Mar 2020. Accessed 15.03.2020. URL: https://istio.io/docs/ops/deployment/architecture.

[5] Istio Authors. Virtual service. [Online], Mar 2020. Accessed 15.03.2020. URL: https://istio.io/docs/reference/config/networking/virtual-service/.

[6] Eveliina Pakarinen. The gradual deployment of a new version of a web application. Master's thesis, University of Helsinki, 2020. In Finnish. Accessed 26.05.2020. URL: http://urn.fi/URN:NBN:fi:hulib-202003191585.

[7] Pilar Rodríguez, Alireza Haghighatkhah, Lucy Ellen Lwakatare, Susanna Teppola, Tanja Suomalainen, Juho Eskeli, Teemu Karvonen, Pasi Kuvaja, June M. Verner, and Markku Oivo. Continuous deployment of software intensive products and services: A systematic mapping study. *Journal of Systems and Software*, 123:263 – 291, 2017. URL: http://www.sciencedirect.com/science/article/pii/S0164121215002812.

[8] Gerald Schermann, Jürgen Cito, Philipp Leitner, Uwe Zdun, and Harald C. Gall. We're doing it live: A multi-method empirical study on continuous experimentation. *Information and Software Technology*, 99:41 – 57, 2018. URL: http://www.sciencedirect.com/science/article/pii/S0950584917302136.

[9] Gerald Schermann, Dominik Schöni, Philipp Leitner, and Harald C Gall. Bifrost: Supporting continuous deployment with automated enactment of multi-phase live testing strategies. In *Proceedings of the 17th International Middleware Conference*, pages 1–14, 2016.