Automation of Regression test in Microservice Architecture

Mohammad Javad Kargar
Department of Computer Engineering
University of science and Culture
Tehran, Iran
kargar@usc.ac.ir

Alireza Hanifizade
Department of Computer Engineering
University of science and Culture
Tehran, Iran
alireza.hanifi@gmail.com

Abstract—**Nowadays Microservice, as one of the most important architectural approaches towards Cloud Computing, has caught the attention of many developers. To give a simple definition of microservice, it could be said: Each microservice is completely independent, and implements a part of the business, which is composed of several microservices. Each microservice could be deployed, updated, and scaled, without any impact on other microservices, and it is all automated. Among Agile methods, Continuous Delivery has played an important role in development process of Microservice-based systems. Continuous Delivery provides faster delivery of changes and faster getting customer's feedbacks. Continuous Delivery consists of many sections, one of which is Software Test. One of the challenges of automated deploying of new releases to production environment, is software reliability, that in case it is breached, the beneficiaries would suffer considerable losses. Regression test is one of the tests used for ensuring reliability, which compares two system versions based on various metrics. This paper proposes an automated method in running this test, which places Regression test in Continuous Delivery steps, which tests operability of the last developed version as a black box, and prevents writing test unit. Finally, microservice developers could ensure the operability of the developed microservice dependencies through investigating obtained comparisons.**

Keywords—microservice; regression test; continuous delivery; DevOps;

I. INTRODUCTION

Microservice architecture has many applications in distributed systems, specially in Cloud Computing. One of the principles used in this architecture is decomposing the system to smaller components. Each component of system, which is called a microservice, would consume resources independently. Microservice has all the conditions of a regular system and all of the system layers. They are connected to each other using lightweight mechanisms such as http. Role of each microservice is to create a business value in system which could be implemented by independent teams and different technologies[1].

To deploy software system in production environment without service loss, Canary Release[1] and/ or Blue Green Deployment[2] methods are used. These methods that are addressed in Zero Down Time, are types of parallel updating and could be used by Continuous Delivery[2]. Software changes are deployed along the pipeline in short iterations, using Continuous Delivery. Different operations is done in the pipeline, such as Compile, Build, Test, Deploy and Delivery[3].

One of the issues that arises during releasing a new version, is operational failure. For instance, one of the system features which was changed during the latest version, fails and does not respond appropriately. This would affect the user and lowers his reliance to the system. To overcome this issue, the software must be tested using different tests before being published to the production environment. In the past, running these tests by software test experts would result an increase in time and financial costs. Nowadays with the development of automated test methods and DevOps, these test are no longer runned by human resources and this important role has been assigned to the system. Thus, the speed of deploying new releases to production environment could be increased significantly[4].

One of the important software tests that is used during version publishing, is Regression test. In the Regression test, system could be compared to the previous version, based on measurable metrics such as resource usage rate, system failure rate, system performance and etc, and prevent releasing of the latest version, if needed[5].

Automation is performing a process without any human assistance. To automate Regression test, first the significant qualitative factors of the system must be determined and then the system must be tested as a black box automatically, and the results must be compared to the previous version. Finally, the system could suggest a release to the developer, and he/she could make a decision based on the provided results. One of the most important objectives of Agile methods, is increasing the speed of deploying new releases to production. On the one

[1] https://martinfowler.com/bliki/CanaryRelease.html
[2] https://martinfowler.com/bliki/BlueGreenDeployment.html

978-1-5386-5364-7/18/$31.00 ©2018 IEEE

hand, this would increase update hazards and risks which in order to overcome them, software qualification is required that is time consuming. Thus, it reduces the speed of releasing new version. To prevent time loss, modern methods could be used, so that with higher speed, reliability is maintained as well. For this purpose, [6] proposes a model for comparing different companies. In this model, the software versions are checked based on two factors, reliability and publishing speed, and are divided to four categories, cautious, balanced, problematic, madness. Fig 1, demonstrates this comparison. This paper presents an automated method, which aims to move from madness category towards balanced, and reduce updating hazards.

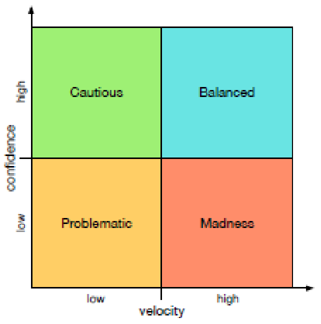

Fig 1. comparison of software releases based on reliability and publishing speed

Releasing a new version in microservices is known as one of the important challanges, since each microservice is dependent upon one or more microservices, which their operability could be affected by the published version. That is why the microservices must have backward compatibility, so that each version of microservices must also support the previous version inputs. Regression test could be used to investigate that. This paper, proposes an automated method that reduces production costs, which reduces the costs of running this test considerably, with placing the Regression test in Continuous Delivery stages, and also using input traffics.

II. Related Works

Possibility of automating of non-functional system tests in continuous integration process is studied with cooperation of Ericsson experts. In their proposed method, they have designed a system as a Verdict Mechinary, which grants the permission for deploying new release to the production environment. Using this system, the test case system could be evaluated based on different metrics such as RAM usage, CPU usage and the number of observed errors in logs. Finally, according to the system information, these information are studied at various checkpoints along time, and if the differences are more than what system expects, it would prevent releasing the new version to production environment[7].

Software performance, as one of the most significant evaluation components, could affect the software users. In this regard, software beneficiaries aim to produce a system with competitive functionalities. Generally publishing several versions in short time intervals, would affect the system performance. Thus, developers aim to not affect system performance during the development process. Regression test of performance, is considered as one of the main tests for this purpose, which must be done after each change.

One of the challenges of this test, is its performing cost, which some methods are proposed to overcome it. To shorten the feedback time, it would be better if after each commit, the test is ran automatically, and the results are presented to the developers. However this method is impractical, since it increases costs of performing the test significantly. Thus, in [8] a method is proposed which allows the developer to choose Regression test. It checks the system source code and identifies the modified codes, then required tests for each commit are determined, so there would be no need for the parts of the code which have not been modified, to be tested, and in doing so it would save resources. Therefore, each developer could view the relevant results for his/her modifications after each commit, so transferring issues to production environment would be reduced. This method would save 83% of performing Regression test costs, which is significant.

In order to save resources, it would be better if tests are prioritized, so that each one is run when it is needed. Also, prioritizing these tests often requires expert resources. However, since this would increase the costs, [9] proposes an automated method to prioritize Resgression tests. This method, analyzes available data based on multiple factors, such as recent error, modified source codes, and past tests runtime, and selects the required tests.

One of the advantages of Cloud Computing is cost reduction of infrastructure. Thus, it is important in cloud native softwares that the available workload would be computed. One of the benefits of this, is estimation of future workload. Tankovic et al. analytic method study distribution of input traffic to microservices, and identify parts of the system which have heavy workload during different time intervals[10].

III. Research Method

Regression test must be performed based on the prioritized metrics that are appropriate for the system. To identify them, first available system features and architecture must be studied, and after that the desired metric could be determined. This paper proposes an automated method, which studies functionality of Regression test system to be done with higher speed and less cost. In order to use this method, several steps must be taken, which first are outlined, and then each step will be described in details.

This paper aims to automate Regression test, using Continuous Delivery in microservice architecture. Continuous Delivery steps are designed according to system features and architecture. Thus, at first we must study and analyze a system based on microservice architecture in general. Then, architecture of test case system would be presented, and its significant features are identified. After that, Continuous Delivery steps are implemented using one of the available tools. At the end of these steps, the automated Regression test is placed. Finally, performance of the proposed method is investigated, with creating a test and running it.

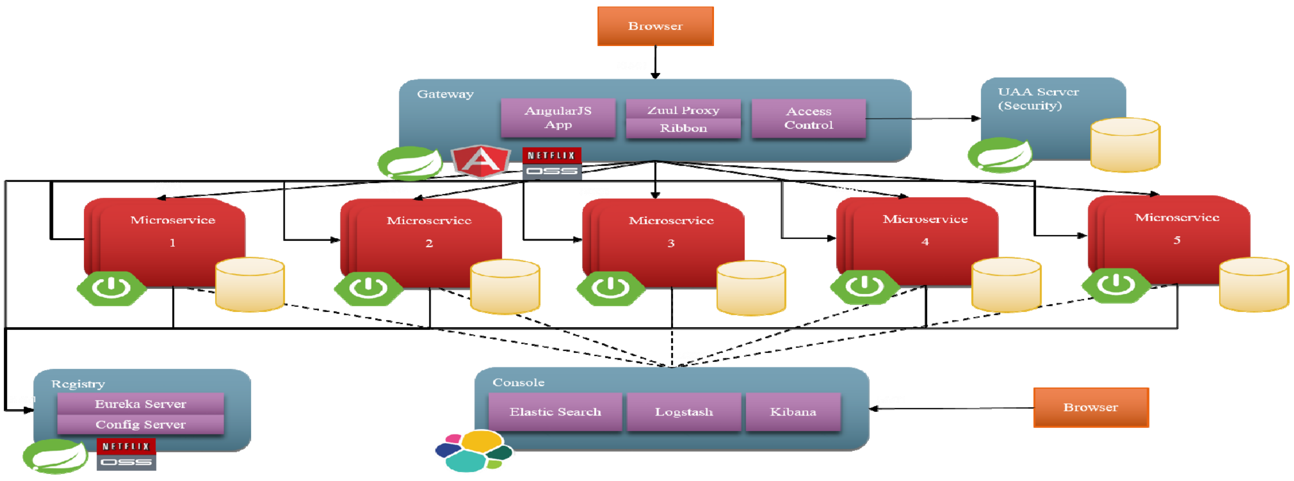

Fig 2. Overall design of National Iranian Oil Products Distribution system architecture

IV. The Proposed Method

The test system is National Iranian Oil Products Distribution Company sales system, which is being rewritten using microservice architecture. Each time a new release is published, its functionality is evaluated using Regression test. Fig 2, shows a general view of this architecture.

The main business of this system, is selling oil products to fuel transfer stations (gas stations), (powerhouses and manufactures) and other oil consumers. This system, interacts with several systems, such as "financial", "storage", "price determination", "transport fee payment" and etc, and is known as the primary system of the organization.

A. *Experimental system's architecture*

In rewriting oil products distribution company sales system, some of the systems which are being used independently and are distributed across the country, must be implemented as a centralized, integrated one. For this purpose, parts of this system which have common processes businesswise and are interconnected, are implemented in a separate microservice. One of the main reasons behind using microservice architecture to implement this system is isolating the development team, updating each microservice completely independent of others, and also making it scalable.

Implementing software systems based on microservice architecture, often arises several challenges, which relevant design patterns are used to overcome each one. The design patterns used in this system are listed below:

- Access Token Pattern[3]
- Service Instance per container[4]
- Service discovery pattern[5]

[3] http://microservices.io/patterns/security/access-token.html
[4]http://microservices.io/patterns/deployment/service-per-container.html
[5] http://microservices.io/patterns/client-side-discovery.html

B. *Definition of Continuous Delivery steps*

Jenkins tool which is one of the most popular and tools in this field, is used to implement this step. Jenkins is plugin based tool which is implemented using Java.

There are many ways in Jenkins to create the steps. In this paper pipeline is used. In this method, a script of groovy language could be used to define Continuous Delivery steps one by one. First, the required arrangements are prepared for starting the operation, by creating a user account and defining a token in each project (microservice) to connect git to Jenkins, and then, for each microservice, an item is created in Jenkins.

Jenkins is informed that a newer version of system is ready to be built, when master branch codes are sent to git server, using webhook. This is when continuous deployment steps begins. These steps are defined by groovy script.

1) *Checkout*

In the first step, the latest version of code is pulled from master branch of git server, using the user account created for Jenkins in Gitlab.

2) *Check Java*

In test case system implementation, repository design pattern is used. Thus, all of the Continuous Delivery process is done in a container docker[7]. Therefore, first a Java image is considered as a base, so the building steps are done in it. Then, Java related environmental variables are set. At the end of this step, Java version is checked by running java -version command, so that Java deployment would be ensured.

3) *Clean*

Since these steps are done periodically, each time the last compiled codes in Jenkins workspace are available, thus the previous compiled codes must be removed so that

[6] http://microservices.io/patterns/apigateway.html
[7] https://www.docker.com/

no conflicts occur. To do this, clean phase of maven tool is used, which is a tool for dependency management in Java.

4) Package and Deploy Artifact

In this step, deploy phase is executed using maven features. Three phases of compile, package and test are deploy prerequisites, which are executed sequentially. In compile phase, source code is compiled by available JRE in docker, and then the developed codes are checked using test units which are defined in the code. After that, a war output is built. Next, in deploy phase, war file would be deployed along with the version in a maven product repository. So after each push, a version along with date and hour is created in maven repository. If we would need to go back to a previous version, we could use the versions saved in this repository. Artifactory[8], an open source tool, is used to create maven repository in this project. In Artifactory, private repositories could be created which grant access to created packages by authentication.

5) Build Docker

In the 5th step, in order to publish an executable output as a container, first a docker image must be built. For this purpose, a dockerfile is developed in microservice codes, which all the steps of executing the output is developed within it. Docker tool could run war output, with command execution and steps provided in this file.

6) Deploy Docker

In this step, the desired image must be deployed in a repository, so that it could be used in the next steps. Docker hub, is a free docker repository, in which the created images could be published publicly or privately. However, since Iran is facing sanctions, this service is not available. Therefore, a private docker registery has been set up, so that image files could be deployed in it.

7) Deploy to Development

In this step, an executable image of microservice is available, which could be run in a production environment. However, in order to prevent probable issues, this image must be run in a test environment, be checked and evaluated. To execute microservices in production environment, kubernetes[9] tool has been used, which Google has been using it for deployment automation and scalability in production environment for years, and in 2014 it became open sourced. Using this tool, a cloud native, which is composed of different clusters, could be created, and microservices system containers could be run there. In kubernetes, different spaces could be created between the clusters virtually, using namespace concept. For instance, in this project three namespaces have been created which are: production environment, secondary environment, and development environment.

The last stable version of microservices are available in production environment, and users requests are executed in this environment. The stable version is often available in secondary environment as well, qualitative tests are performed in secondary environment before being deployed in production environment. Development environment contains the latest developed version of microservices as well.

In this step, the latest version available of the microservice which was deployed in docker registry, must be executed in kubernetes. However, before publishing it in production environment, first it must be deployed in development environment so that Regression test could be run.

Kubectl could be used to interact with Kubernetes. Required preparations are made in Jenkins server, so that deployment of the new version of microservice is done in development environment, using kubectl.

8) Regression Test

In order to ensure the deployment of the developed version in production environment, it is necessary that the new version is checked by Regression test, before execution. In this step, Regression test is defined in Continuous Delivery. For this purpose, a tool named Diffy[10] is used, which was developed by Twitter in the late 2016. This tool allows you to use input traffic as input for testing the developed microservice, which could result in considerable cost reduction of testing. This tool runs Regression test by comparing microservices response outputs of the running version in production environment to the new developed version which is accessible in development environment. To run this test user input is needed. User input requests to the production emvironment could be used. However, often services do not respond deterministically. Thus, only the system parts which do respond deterministically must be tested. In this regard, three production environments are required to run this test. The latest stable system version is accessible in both production and secondary environment, and development environment contains the latest developed version. System could determine deterministic and non-deterministic responses by sending input requests to both production and secondary environment. For instance, in a service output sending request time feature is responded in 0.01 second. This part must be overlooked in comparison in Regression, since this feature would definitely differ in production and development environment. Fig 3, shows the Regression test process.

To run the test, system input traffic must be sent to Diffy tool. After this step execution, the differences between stable and developed versions would be apparent.

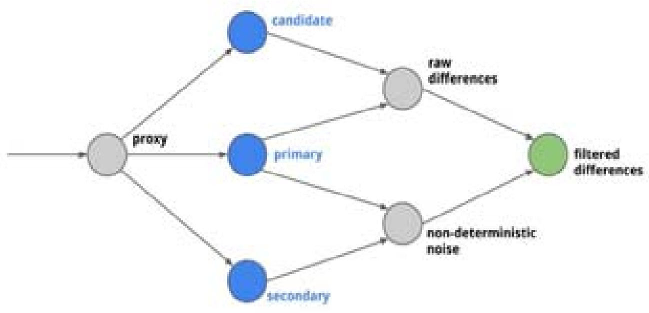

Fig 3. how Regression test works

[8] https://jfrog.com/artifactory

[9] https://kubernetes.io/

[10] https://github.com/twitter/diffy

After the user reviewed the differences, the final decision for deploying the system in production environment could be made. In this step, given the user's final approval, the new microservice will be deployed to production environment, using rolling deployment technique. Using this technique, considering Zero Down Time, microservice would be updated without any service loss. In kubernetes the process is done gradually, so that if any issues arise in the new release, the least possible damage is done. Updating is done in a way that the number of stable versions are reduced gradually, and the number of new versions increase, until all microservice instances are updated. In case of any issue, the user could replace the new version with the latest stable version.

V. Conclusion

National Iranian Oil Products Distribution Company sales system, has many users across country, thus its availability is very important. This system is implemented based on microservice architecture, and consists of several microservices. Some of these microservices need to exchange data for some operations. Therefore updating a microservice must not affect other microservices functionality.

Oil orders issued by this system, are calculated by price determination microservice. Thus, this service is highly sensitive. In this paper, an automated method is proposed for taking Regression test, which could be used to reduce testing costs.

Price determination service, which is a deterministic one, could calculate order price based on different input parameters. Therefore, testing this service manually would be very costly. Automation of this test and using input traffic could prevent writing several test units. Fig 4, demonstrates the proposed method steps. One of the downsides of this method, is limited responding of deterministic services.

References

[1] F. Martin and J. Lewis, "Microservices," 2014. [Online]. Available: http://martinfowler.com/articles/microservices.html.

[2] S. Danilo, "ParallelChange," 2014. [Online]. Available: http://martinfowler.com/bliki/ParallelChange.html.

[3] L. Chen, "Continuous Delivery: Huge Benefits, but Challenges Too," *IEEE Softw.*, vol. 32, no. 2, pp. 50–54, Mar. 2015.

[4] M. Callanan and A. Spillane, "DevOps: Making It Easy To Do The Right Thing," *IEEE Softw.*, pp. 1–1, 2016.

[5] J. Waller, N. C. Ehmke, and W. Hasselbring, "Including Performance Benchmarks into Continuous Integration to Enable DevOps," *ACM SIGSOFT Softw. Eng. Notes*, vol. 40, no. 2, pp. 1–4, 2015.

[6] G. Schermann, J. Cito, P. Leitner, and H. C. Gall, "Towards quality gates in continuous delivery and deployment," *IEEE Int. Conf. Progr. Compr.*, vol. 2016–July, pp. 1–4, 2016.

[7] M. Fagerström, E. E. Ismail, G. Liebel, R. Guliani, F. Larsson, K. Nordling, E. Knauss, and P. Pelliccione, "Verdict machinery: on the need to automatically make sense of test results," *Proc. 25th Int. Symp. Softw. Test. Anal. - ISSTA 2016*, pp. 225–234, 2016.

[8] A. B. De Oliveira, S. Fischmeister, A. Diwan, M. Hauswirth, and P. F. Sweeney, "Perphecy: Performance Regression Test Selection Made Simple but Effective," *Proc. - 10th IEEE Int. Conf. Softw. Testing, Verif. Validation, ICST 2017*, pp. 103–113, 2017.

[9] P. E. Strandberg, D. Sundmark, W. Afzal, T. J. Ostrand, and E. J. Weyuker, "Experience Report: Automated System Level Regression Test Prioritization Using Multiple Factors," *Proc. - Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 12–23, 2016.

[10] N. Tankovic, N. Bogunovic, T. G. Grbac, and M. Zagar, "Analyzing incoming workload in Cloud business services," *2015 23rd Int. Conf. Software, Telecommun. Comput. Networks, SoftCOM 2015*, pp. 300–304, 2015.

[11]

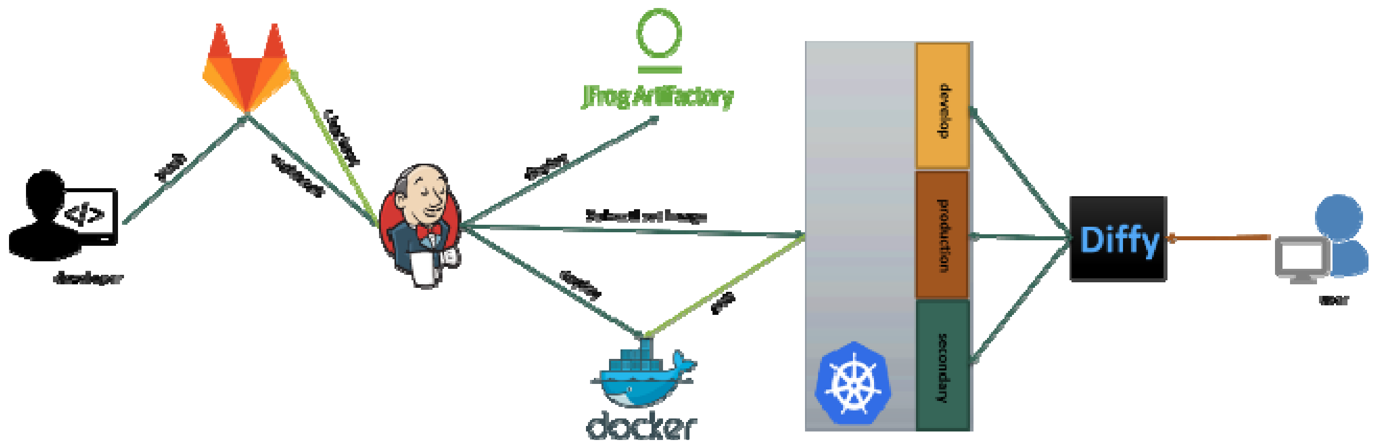

Fig 4. The proposed method