

**CONTINUOUS INTEGRATION AND CONTINUOUS  
DELIVERY PIPELINE AUTOMATION FOR AGILE  
SOFTWARE PROJECT MANAGEMENT**

Indunil Suriya Arachchi

(148204F)

Degree of Master of Science

Department of Computer Science and Engineering

University of Moratuwa  
Sri Lanka

April 2018

**CONTINUOUS INTEGRATION AND CONTINUOUS  
DELIVERY PIPELINE AUTOMATION FOR AGILE  
SOFTWARE PROJECT MANAGEMENT**

Suriya Arachchige Indunil Bandara Suriya Arachchi

(148204F)

Thesis submitted in partial fulfillment of the requirements for the degree Master of  
Science

Department of Computer Science and Engineering

University of Moratuwa  
Sri Lanka

April 2018

## DECLARATION

I declare that this is my own work and this thesis does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to The University of Moratuwa the non-exclusive right to reproduce and distribute my thesis, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books)

Signature:

Date:

The above candidate has carried out research for the Masters thesis under my supervision.

Signature of the Supervisor:

Date:

## ABSTRACT

Adaptation of agile methodologies in software development life cycle has proved an improvement in productivity and quality of systems. In terms of quality, it defines new process and standards requirement where Continuous Integration (CI) principles have filled the gap while improving the quality of system continuously and Continuous Delivery (CD) approach has made faster delivery of software. Continuous Deployment extends the CD features and delivers the software to the production through automation by completing the pipeline. Ultimately, the Continuous Integration Continuous Delivery (CICD) pipeline approach has increased the efficiency and the productivity of agile software projects.

In agile, new features are introduced to system in each sprint delivery, and although it is well developed, the delivery failures are inevitable due to performance issues. By considering delivery timeline, moving for system scaling is common solution in such situations. But, how much system should be scaled? System scale requires current system benchmark status, and expected system status. Benchmarking the production is a critical task, as it may interrupt the live system, which may causes system unstable. New software version should go through a load test, to measure expected system status. The traditional load test methods are unable to identify production performance behavior due to simulated traffic patterns are highly deviated from production.

To overcome those issues, this approach has extended CICD pipeline to having three phase automations process named benchmark, load test and scaling. It minimizes the system interruption by using test bench approach when system benchmarking and it uses the production traffic for load testing which gives more accurate results. Once benchmark and load test phases completed, system scaling can be evaluated. Test bench setup was done on high capacity computer using Ansible automation which provisioned local virtual instances for application servers, Nagios service and load balancing. A simple XML based application which processes cached data by reading files is used to reduce the complexity of test bench approach. Initially, the pipeline was developed using Jenkins CI server, Git repository and Nexus repository with Ansible automation. Then GoReplay is used for traffic duplication from production to test bench environment. Nagios monitoring is used to analyze the system behavior in each phase and the result of test bench has proven that scaling is capable to handle the same load while changing the application software, but it doesn't optimize response time of application at significant level and it helps to reduce the risk of application deployment by integrating this three phase approach as CICD automation extended feature. Thereby the research provides effective way to manage Agile based CICD projects.

Keywords: Continuous Integration, Continuous Delivery, Agile Manifesto, Version Control System, Configuration Management

## **ACKNOWLEDGEMENT**

First I would like to express my earnest gratitude to my supervisor Dr. Indika Perera for the supervision and advice given throughout to make this research a success.

Then, I would like to thank my family members for the support and encouragement that they have given to me.

Finally, I am grateful to my MSc 2014 batch mates and various online community members who supported me during the Research.

S.A.I.B Suriya Arachchi

# TABLE OF CONTENTS

DECLARATION	i
ABSTRACT	ii
ACKNOWLEDGEMENT	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	vii
LIST OF TABLES	ix
LIST OF ABBREVIATIONS	x
CHAPTER 1	1
INTRODUCTION	1
1.1 Background	1
1.2 Research Problem	2
1.3 Research Objectives	3
1.4 Research Overview	3
CHAPTER 2	4
LITERATURE REVIEW	4
2.1 Agile Software Development to CICD	4
2.2 CICD Pipeline	6
2.3 Continuous Integration	8
2.3.1 CI Practices	10
2.3.2 CI benefits	11
2.4 Continuous Delivery (CD)	11
2.4.1 CD benefits	13
2.5 Continuous Deployment	13

2.6 DEVOPS	14
2.7 CICD Tools	16
2.7.1 Repository and Version Controlling	17
2.7.2 Build Tools	19
2.7.3 Automation (Configuration Management)	22
2.7.4 Test Automation	32
2.7.5 Monitoring	33
CHAPTER 3	35
METHODOLOGY	35
3.1 Deployment methods	37
3.2 Benchmark	38
3.2.1 Duplicate Traffic	40
3.3 Load Test	42
3.4 Scale Identification	43
3.5 Provisioning	47
CHAPTER 4	48
DESIGN AND IMPLEMENTATION	48
4.1 Deployment Automation - CICD Pipeline	48
4.2 Benchmark Automation	56
4.3 Load Test Automation	58
4.4 Scaling Automation	60
CHAPTER 5	62
EVALUATION AND RESULTS	62
5.1 Test Bench Setup	62
5.2 Test Bench Performance	65
5.2.1 Initial Bench mark Phase	65

5.2.2 Load Test Phase	69
5.2.3 Scaling Phase	71
CHAPTER 6	74
CONCLUSION & FUTURE WORK	74
6.1 Conclusion	74
6.2 Study Limitations	74
6.2 Future Works	75
REFERENCES	76