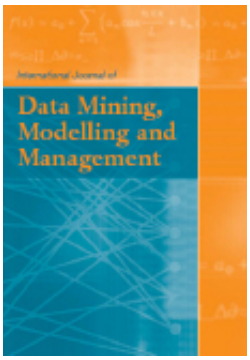


International Journal of Data Mining, Modelling and Management

ISSN online: 1759-1171 - ISSN print: 1759-1163
https://www.inderscience.com/ijdmmm

Capturing uncertainties through log analysis using DevOps

Rajeev Kumar Gupta, Arti Jain, Ruchika Kumar, R.K. Pateriya

DOI: 10.1504/IJDMMM.2023.10055208

Article History:

Received:	12 July 2021
Last revised:	07 December 2021
Accepted:	21 February 2022
Published online:	04 April 2023

Capturing uncertainties through log analysis using DevOps

Rajeev Kumar Gupta*

Pandit Deendayal Energy University,
Gandhinagar, India
Email: rajeevmanit12276@gmail.com
*Corresponding author

Arti Jain

Jaypee Institute of Information Technology,
Noida, India
Email: ajain.jiit@gmail.com

Ruchika Kumar and R.K. Pateriya

Maluna Azad National Institute of Technology,
Bhopal, India
Email: ruchika.kumar9893@gmail.com
Email: pateriyark@gmail.com

Abstract: DevOps is an advancement of agile processes which is mainly used to improve the coordination between development and operation teams. Continuous practices survive within the core of the DevOps which ensures efficient pipelines and high-quality delivery of software. Using such practices in a synchronous, business dynamics compliance and ever-changing needs of clients can meet high performance and reliable final products. This research work is an attempt to propose a simplified solution, guideline and tools support for developing and maintaining quality of continuous practices that are used in the DevOps project. The system automates the correlation among various telemetry data to contribute towards enriching log analysis and reduces manual efforts. The proposed system undergoes in-depth analysis of logs, promotes quality assessments and feedback to developers, which in result, helps in deeper problem diagnosis of the telemetry data. In this work, an empirical study is carried out to gain conceptual clarity on integrated pipeline architecture and to address how automation in continuous monitoring accelerates and extends the feedback loop in the system.

Keywords: agile; DevOps; log analysis; telemetry data; software development life cycle; SDLC.

Reference to this paper should be made as follows: Gupta, R.K., Jain, A., Kumar, R. and Pateriya, R.K. (2023) 'Capturing uncertainties through log analysis using DevOps', *Int. J. Data Mining, Modelling and Management*, Vol. 15, No. 1, pp.53–78.

Biographical notes: Rajeev Kumar Gupta has completed his PhD from the MANIT, Bhopal, India. He is working as a Senior Assistant Professor at the Pandit Deendayal Energy University, Gandhinagar, Gujarat, India. He is a recipient of the Best Young Researcher Award by RSRI in 2019. He has published more than 30 referred articles in various book chapters, conferences and international repute peer-reviewed journals of Elsevier, Springer, IEEE. He has a total of more than ten years of teaching experience. He is a life member of some of the reputed societies – CSI India, IAENG (Hong Kong). He has organised several STTPs/FDPs and have taken several expert lectures at various Institutes throughout India. He has supervised 20 MTech thesis and around 40 BTech projects in various domains. His area of interest includes machine learning, deep learning and cloud computing, software reliability, artificial intelligence, computer vision, data mining and information security.

Arti Jain is working as Assistant Professor (Senior Grade) of Computer Science and Engineering at the Jaypee Institute of Information Technology, Noida, Uttar Pradesh, India. She is having more than 19 years of academic experience. She is member of IEEE, INSTICC, IAENG, SAFAS, IFERP and TERA. She has more than 20 research papers in peer-reviewed international journals, book chapters, and international conferences. She has delivered expert talks in KDPs, FDPs and workshops. She is editorial board member and TPC member of several international journals and conferences. She has supervised MTech thesis and around 100 BTech projects. Currently, she is supervising PhD candidate in the area of social network analysis. Her research interest includes natural language processing, machine learning, data science, deep learning, social media analysis, soft computing, big data and data mining.

Ruchika Kumar is working as a software engineer at the Intel Technology India Pvt. Ltd., Bangalore, India. She has completed her MTech from the Maulana Azad National Institute of Technology Bhopal, India. Her area of interest includes cyber security, software engineering and cloud computing.

R.K. Pateriya is working as a Professor in the CSE Department of Maulana Azad National Institute of Technology Bhopal, India. He completed his PhD from the MANIT, Bhopal. He has published more than 60 referred articles in various book chapters, conferences and international in reputed peer-reviewed journals of Elsevier, Springer, IEEE, etc. He has a total of more than 28 years of teaching experience. He is a life member of some of the reputed societies like CSI India, IEEE, etc. He has organised several STTP/FDP and taken several expert lectures at various institutes. He has supervised eight PhD, 80 MTech thesis and around 120 BTech projects in various domains. His area of interest includes information security and cloud computing.

1 Introduction

DevOps is a set of software engineering practices that brought a revolution of cultural and organisational changes (Katal et al., 2019; Yarlagadda, 2021). In other words, DevOps combines software development (Dev) and information technology operations (Ops) which aims to shorten the systems development life cycle. The DevOps is responsible to provide continuous delivery and that too with higher software quality. It is considered as complementary with agile software development. Also, several of the DevOps aspects have come from the agile methodology. DevOps has broadened the

scope of research as well as the technology market. It has gained popularity in no time because of its objective to utilise knowledge and resources in the best way by eliminating non-value-added processes in the software delivery pipeline and emphasising more on learning at every level.

Developers are expected to be aware of uncertainties in the entire DevOps life cycle. A widely adopted resolution to minimise uncertainties is the 'continuous monitoring'. Rapid feedback on real-time data helps in monitoring such situations. Repetitive measuring and monitoring systems ensure that developers' contributions are oriented towards adding value to the software quality. Moreover, in-depth knowledge of the software delivery pipeline can help to identify and handle the root cause of uncertainties. Continuous monitoring techniques like sensitivity analysis of different parameters of a system, rigorous statistical evaluation and visualisation of results, etc. can help in reducing uncertainties further. Even if uncertainties cannot be handled, qualitative and quantitative analysis predicts the probability of failures.

The aim of the DevOps practices is to mark delivery teams as accountable for the production issues and fixes, whether legacy or novel. In the traditional systems, delivery would only be answerable for the alterations put in by them, within the duration of warranty. In this work, an attempt is made to propose a DevOps-based simplified solution, with suggested guidelines and tools support for developing and maintaining the quality of continuous practices that are used in the telemetry data. The proposed system automates towards enriching the log analysis and reduces manual efforts. It undergoes an in-depth analysis of the logs, promotes the quality assessments, as well as feedbacks to the developers, which in turn helps in the deeper problem diagnosis of the telemetry data.

Stahl et al. (2017) and Rafi et al. (2021) define it as "A superset of continuous practices involving values, principles and procedures" whereas, others stated as "An emerging culture or phenomenon that integrates development, operation and quality" (Kamuto and Langerman, 2017; Pietrantuono et al., 2019). This research project aims to present a simple solution, guideline, and tools for developing and maintaining the quality of continuous practises used in DevOps projects. The system automates the linkage of various telemetry data, which enriches log analysis while reducing manual work.

- Collecting telemetry and metrics data for data analytics and resource management.
- Automation and effective monitoring of the services to map down correlation between telemetry data to analysed data.
- Enriching log analysis context for thorough uncertainty diagnosis by identifying and controlling the sensitive areas.
- Automate the process of continuous monitoring to save time and effort of the users and help them to stay ahead of potential issues with predicted reports and charts.

The paper revolves around build flow of CI/CD pipelines in Jenkins, functional modules, building triggers, log parsing, structuring and cleaning data, and creating visuals for report, and at the end, summarising and analysing the performance.

The paper is structured as follows: Section 2 explains the literature review for evolution of SDLC and emergence of DevOps. Section 3 illustrates the DevOps-based case study of the Intel project. Section 4 details about the execution results. And, finally, Section 6 concludes the paper.

2 Literature review

The literature review is subdivided into two subsections. Subsection 2.1 discusses the evolution of SDLC and Subsection 2.2 discusses the emergence of DevOps.

2.1 Evolution of SDLC

The acronym SDLC stands for 'software development life cycle' it is a procedural approach to convert stakeholders' expectations and requirements into realistic software products. Aligned with companies' strategies and priorities it becomes the home ground for any software production (Frijns et al., 2018; Hemon et al., 2019). Traditional models like waterfall model, interactive model, spiral model, v-model, big bang model, etc. followed a linear progression of requirement gathering, analysing, feasibility study, designing, developing, implementing, verifying, testing, validating, maintaining and ultimately delivering the software.

The waterfall model has been adopted as a base since 1960 for every other model. The factors that contribute to its longevity are included as: simplicity, predictability of deliverables at every stage and segregated roles and responsibilities. It continued to predominate SDLC methodology until the introduction of agile in the early 2000 (Kersten, 2018). Agile is the outcome of incompetence of the conventional methodologies of software development. The basic principle agile is the customer's satisfaction with project's cost control, better team collaboration and minimalistic waste of knowledge and resources (Pingrong et al., 2021).

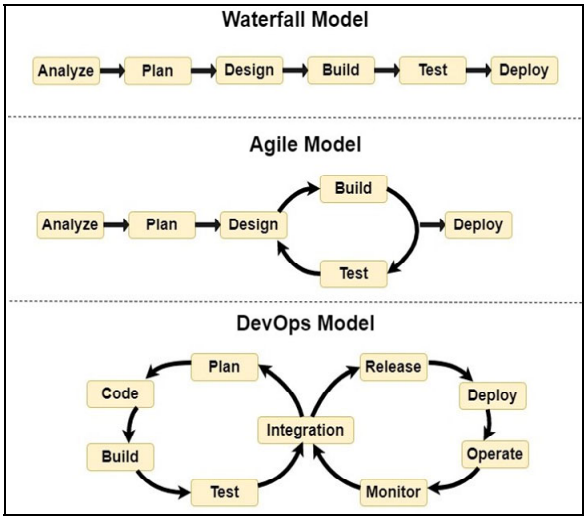
Dörnenburg (2018) has approached by bringing together silo teams and replacing them by a cross functional team based on their specialisations. It would be relatively difficult to be adopted in large enterprises and even worse in case of different vendors and scattered geographical setup locations. With the increasing levels of confusion and frustration, whether to emphasise on timely delivery and on budget delivery or software stability and cost of operations at the sake of expenses and maintainability, awake practitioners are uncomfortable and resistant to adoption of agile.

Instead of just focusing on product, agile needs to be tailored with new techniques and redrawn team boundaries and responsibilities (Gokarna and Singh, 2021). DevOps emerged in the late 2000s which clears the path of production with convenience and reduces life cycle time. This breakthrough in SDLC transforms agile islands and brings a dynamic era of digitalisation and automation (Pietrantuono et al., 2019). DevOps technology framework helps in building effective software delivery pipelines by fusing independent design, development and deployment practices (Koilada, 2019). It is collaboration with new tools and technology has paved new paths of evolution of different software segments. To exemplify, DevOps consists of values, principles, methods, practices and tools. Fedushko et al. (2020) investigate the impact of COVID-19 on network traffic resulting from e-commerce, online education, and other types of activities. This work introduced a site reliability engineering (SRE) approach to assure the reliability and availability of e-portal system. The primary goal of this research is to improve content quality while also identifying anomalous system behaviour and poor infrastructural conditions.

Figure 1 shows how the waterfall model and agile model give birth to DevOps methodology. The main phases of DevOps are – collaboration, plan, build, continuous

integration and continuous development (CI/CD), deployment, operation, continuous monitoring and reporting feedbacks.

Figure 1 Evolution of SDLC methodology (see online version for colours)



The recent Forrester research revealed that till now, about 50% of associations have successfully executed DevOps at an 'escape velocity' (Hemon et al., 2019).

2.2 Emergence of DevOps

DevOps methodology abridges the functional and operational gap among silos teams and handovers a versatile specialist team with amalgamated development and operational responsibilities (Yarlagadda, 2021). According to the reports of 2015, 2016 and 2017 by the Puppet Labs, an important aspect of DevOps is to improve the workflow within an organisation and efficiently share information based on the concept of work being pulled rather than being pushed (Katal et al., 2019).

Although many researchers and academicians have tried to formulate a comprehensive definition for DevOps but have failed to propose one, due to its multifaceted vastness and ambiguity. Some researchers (Stahl et al., 2017; Chen, 2019; Geissdoerfer and Wolisz, 2019; Veres et al., 2019; Rafi et al., 2021) define it as "A superset of continuous practices involving values, principles and procedures" whereas, others stated as "An emerging culture or phenomenon that integrates development, operation and quality" (Kamuto and Langerman, 2017; Pietrantuono et al., 2019). In another work, researchers (Trubiani et al., 2018) have called it a novel trend among practitioners while few focused on its technical stance like automation and toolchain which opened huge entrepreneurial opportunities in the IT market (Koilada, 2019).

Though DevOps became a buzzword, but it can be best explained by CAMS – culture (C), automation (A), measurement (M), sharing (S), term coined by John Willis (Perera et al., 2017; Stahl et al., 2017). Culture, automation, measurement and sharing are the fundamental and mutually reinforced values behind DevOps implementation. Prior culture setup encourages the adoption of agile practices and overcomes conventional

limitations of SDLC approach. "Automation is the key enabler for DevOps adoption" (Perera et al., 2017). Modern DevOps toolset and advanced technology revamped organisational workflow. CAMS recommends transparent, accessible and meaningful measurements of all DevOps constituents. And lastly, sharing of ideas, knowledge, challenges and learnings help in aligning people, practices and technology towards a common goal, i.e., adding value to the IT business.

The in-depth literature analysis of many papers and interviews, it is concluded that few factors which hinder the adoption of DevOps, are risk of disintermediation of roles, lack of education, resistance to change, silo mentality, lack of strategic direction from management (Gokarna and Singh, 2021) and so on.

2.2.1 Continuous practises – 'the heart of DevOps'

DevOps comprises many continuous practices which all together contributes to the agenda – customer satisfaction and better software quality. It includes continuous integration, continuous delivery, continuous deployment, continuous testing and monitoring, and continuous release. Each of them are detailed here.

a *Continuous integration* – Means integrating the developers' work very frequently and iteratively. It frames other continuous practices which in combination eliminates discontinuities between development and operations, preferable to be practiced at large scale.

b *Continuous delivery* – Complies to the actual release of software segments with short release cycles, it brings optimisation of infrastructure management and balances out software release availability and reliability. The potential release candidate undergoes rigorous code analysis, proper documentation, acceptance testing, regulatory compliance assessment, license and requirement verification. It is often used interchangeably with continuous deployment.

c *Continuous deployment* – Refers to the operational placement of the potential release candidate, evaluated in the former stage, in the production environment.

d *Continuous testing and monitoring* – Runs parallely with other continuous practices. Automated testing with regular quality feedback to DevOps and quality assurance teams, evaluates the software candidate's readiness for release. Data collected from the systems in production is passed as inputs for testing and monitoring activities.

e *Continuous release* – Refers to the business practices in order to make the desired software timely and readily available to the stakeholders, i.e., customers and clients.

2.2.2 Automation and software quality

"Quality of the software is the key factor of IT business" (Perera et al., 2017). It ensures business growth with customers' satisfaction as priority. The set of attributes that are mentioned in ISO 9126, an International Standard for Evaluation of Software Quality laid down the six main characteristics of software quality, namely – functionality, reliability, usability, efficiency, maintainability and portability.

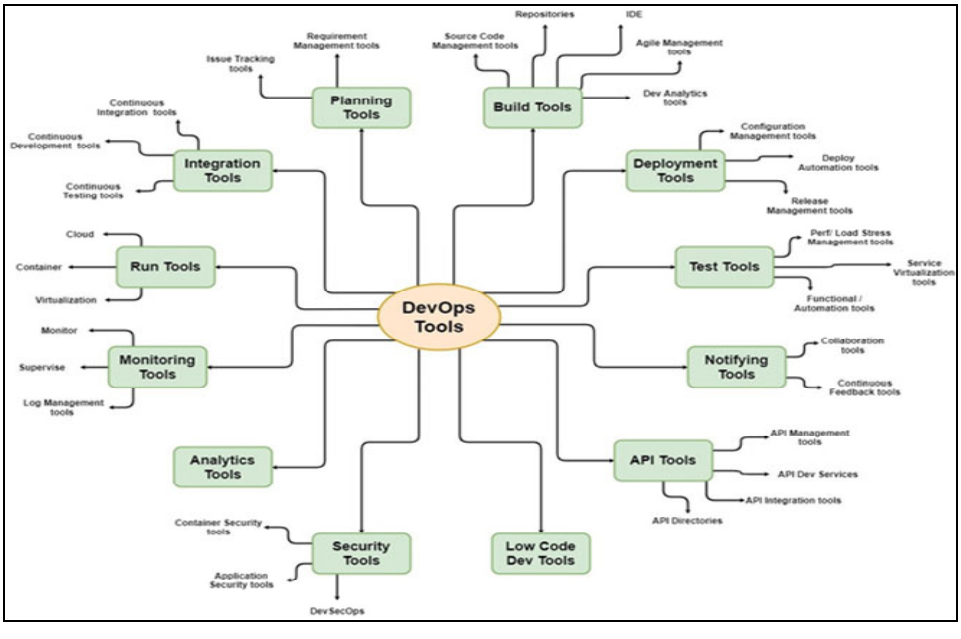
A model is formulated in (1) to represent the relationship between DevOps and quality based on CAMS – culture (C), automation (A), measurement (M), sharing (S), as is given in equation (1) as follows:

$$SQ = 1.409 + 0.176(C) + 0.272(A) + 0.096(M) + 0.172(S) \quad (1)$$

Automation in the business model and high velocity advancement strongly relies on high performing technologies (Alnafessah et al., 2021; Castellanos et al., 2021). With the modernisation of tools used, the release-deployment time gap can be scaled down immensely (Geissdoerfer and Wolisz, 2019). According to an ongoing research by the KBR, the worldwide DevOps market will hit $8.8 billion by the year 2023 (Hemon et al., 2019).

"Quality delivers with short cycle times need a high degree of automation" (Yarlagadda, 2021) and that comes by advancement in tools and technologies.

Figure 2 DevOps tool tree (see online version for colours)



To provide more clarity over the categories and subcategories of varieties of tools that the DevOps ecosystem contains, we have compiled the contributions (Kamuto and Langerman, 2017; Dörnenburg, 2018; Veres et al., 2019; Ganeshan and Vigneshwaran, 2021; Yarlagadda, 2021). The DevOps toolset can be divided into 12 major categories based on the DevOps Lifecycle Mesh (Veres et al., 2019). Figure 2 represents the major categories as – planning tools, build tools, integration tools, deployment tools, run tools, test tools, monitoring tools, notifying tools, analytics tools, application programming interface (API) tools, security tools and low code development tools. Their further sub-categories are as follows:

a *Planning tools* – Includes requirement management tools and issue tracking tools.

b *Build tools* – Includes source control management tools, repositories, integrated development environment (IDE), agile management tools and development analysis tools.

c *Integration tools* – Includes continuous integration and continuous delivery (CI/CD) tools.

d *Deployment tools* – Includes configuration management tools, deploy automation tools and release management tools.

e *Run tools* – Can be cloud-based, containers and virtualisation tools.

f *Test tools* – Includes perf (a performance analysis tool)/load/stress management tools, service virtualisation tools and functional tools.

g *Monitoring tools* – Includes monitoring and supervising tools and log management tools.

h *Notifying tools* – Includes collaboration tools and feedback tools.

i *Analytics tools* – Includes tools for quickly and efficiently data analysis.

j *API tools* – Includes API management tools, API Dev service tools, API integration tools and API.

k *Security tools* – Includes container security tools, application security tools and DevSecOps.

l *Low code development tools* – Includes mobile Dev tools, etc.

3 Case study

This work is carried out at Intel private limited which is well-renowned for manufacturing microprocessors and embedded software worldwide. Our team follows the DevOps methodology to develop and release post-silicon platform specific software to the clients. Every operation committed by the team is strictly carried out in the Agile Scrum model.

3.1 Requirement elicitation and feasibility study

Requirement gathering and its analysis is the crucial phase of the project. Stakeholder's like high level management, developers and the end user of the feedback report and alert system created due to continuous monitoring, their requirements are noted down under categories based on their perspectives. Their expectations give insights into the functional and non-functional requirements of the project.

Further, the feasibility studies of the requirements are outlined in Table 1 that has helped in finalising the tools and technology that are needed and in designing the blueprint of the architecture of the project. List of certain tools with brief description is given in Table 2 which is followed by their detailed description.

1 *Jira* – is a free tool which is used for project management. It is basically used for issue tracking and bug tracking. Features and dashboards provided, helps to follow Agile scrum scheme easily.

2 *Git/GitLab* – is a free, open-source version control tool, widely used for source code management of small as well as large projects. It supports centralised and distributed version systems with the advantages like reliability, scalability and security.

3 *Jenkins* – is an open-source continuous integration (CI) tool written in Java. It accelerates the process of continuous delivery of software segments by integrating them with different scans, test suites and deployment technologies. Easy installation and wide community support make it more preferable by developers for integration of different DevOps stages with the help of various plug-ins.

4 *Skype* – is a simple collaboration tool which provides audio calls, video calls and instant messaging service. This freemium application allows users to communicate on laptops, computers and mobile devices over the internet.

5 *Splunk* – is a data analytics tool used for continuous monitoring. Interactive dashboards and reports with triggered action can be generated by collecting, monitoring, analysing and visualising the system generated real-time telemetry data.

Table 1 Requirement overview

S. no.	*Requirement category*	*Requirements*
1	Business requirements	Save effort and time, lost due to uncertainties at infrastructure and development level
		Ensure smooth functioning among teams – IT, development team, operation team, test and validation team
		Focus on overall productivity and contribution to timely delivery of the error-free final product as committed to end customers
2	Functional requirements	Gain clarity on the blocking issues and figure out where and what caused the uncertainties
		Continuous infrastructure monitoring and uncertainty trend analysis on real-time integration and development
		Early detection and prediction of the issues of potential failure causes of the development segment
		Interactive visual reports and alert mechanism through dashboards with least manual efforts
3	Non-functional requirements	Modularity – to make the modules reusable and inheritable
		Efficiency – to maximise the overall throughput
		Accuracy – to authenticate the data integrity of the reports with the actual output of the jobs
		Automation – to reduce manual interference
		Flexible – easy to use and readable
		Modifiable – to change the logic as and when required
		Platform independent – to avoid storing and processing on any physical server

Table 2 Tools description

S. no.	*Tools used*	*Tool type*
1	Jira	Issue tracking tool
2	Git/GitLab	Source code management tool/version control tool
3	Jenkins	Integration tool
4	Skype	Collaboration tool
5	Splunk	Monitoring tool/analytics tool

3.2 Architectural orientation and technical design

Any DevOps project comprises of five stages majorly:

1 stage 0 – sync

2 stage 1 – build

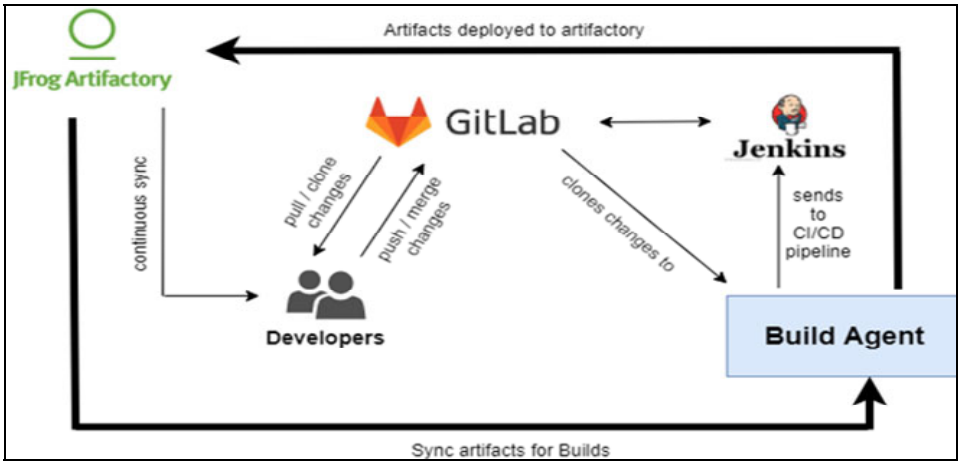
3 stage 2 – integrate

4 stage 3 – validate

5 stage 4 – publish.

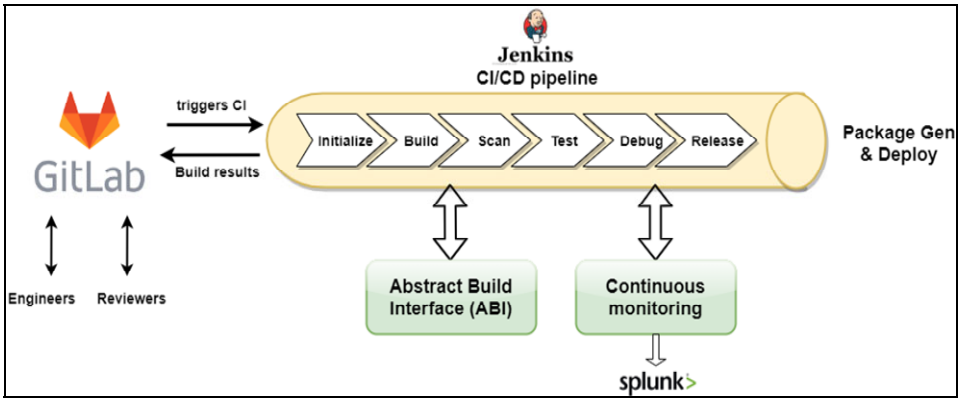
Figure 3 represents the architectural design of the project. Developers push and pull codes from Git, which gets cloned in any build agent. A build agent can be a physical server or container or cloud-based architecture as per the user needs. Whenever any changes happen in Git, the CI/CD pipeline gets triggered in Jenkins. Build agent is in continuous sync to a central Artifactory.

Figure 3 Architectural orientation of the project (see online version for colours)



Integration of these tools (Table 1) with CI/CD pipeline defines the build flow, which is an interim process to enable consolidating source codes, branching and versioning of CI/CD builds as shown in Figure 4.

Figure 4 Build flow of CI/CD pipeline in Jenkins (see online version for colours)



Abstract build interface (ABI) is a DevOps package in Python which provides:

- common functionalities needed during build, signing, security, etc.
- consistent API and shared common libraries
- flexible packages for entire end-to-end CI/CD solutions.

Using ABI, teams have flexibility to choose packages/services as per the requirement without wasting resources. Automatic feature updation and scalability to the business growth are the benefits of ABI.

The generic approach is followed for extraction of relevant data from CI/CD pipeline output, by maintaining a customised agile scrum methodology. When a developer commits code on SCM, i.e., source code management tool GitLab, the commit is listened to by Jenkins, which initiates the build of the components. Every build triggered in Jenkins produces an output console log, where 'what-to-log' and 'where-to-log' is the matter of concern.

'What-to-log' should provide enough information that is needed for uncertainties diagnosis on 'where-to-log', an automated logging practice is adopted. Initially, the console output log is manually tracked down in a physical server in which parsing script is pre-stored. Parsing script is a code written in Python that is responsible for runtime capturing unique build URL, build number, infrastructure details and error messages in every build console output log.

Later, temporary placement of output logs and parsing script are to be node independent and hence ABI shared library is used. The shared library written in Groovy can be easily integrated with the Jenkins file during the build phase in Jenkins. We have systematically analysed logs from historical repositories and extracted the relevant data and derived a certain threshold condition for time stamped data in JavaScript Object Notation (JSON) format collected by Splunk, processed the raw data and filtered out the relevant information by using statistical analysis. Splunk's visualisation support feature has generated interactive reports and feedback with alerts and warning messages. Correlating the analysed data to telemetry data has helped the team members to find out the root cause of failures and also has eased the early detection of several failures.

3.3 Dataset description

Collection of output console logs generated by Jenkins, resulted in a huge dataset with more than 100 fields. The project runs in a dynamic environment which is unpredictable, therefore normalisation of the relation table is avoided initially to prevent unnecessary anomalies and ensure lossless data. Table 3 defines a few most important data fields with schema description and constraints.

Table 3 Dataset schema

Filed name	*Field information*		
	Value data types/constraints	*No. of values; defined values*	*Description*
build_number	number [primary key]	>100 (varies with time range)	Holds unique build number
build_url	string [primary key]	>100 (varies with time range)	Holds unique build URL
host	string	12 (fixed)	Holds the name of main servers
job_duration	number	>100 (varies with time range)	Holds job's time duration
Job_name	string	96 (varies with time range)	Holds job's name
job_result	string	3 (success, failure, aborted)	Holds job's result after Execution
job_started_at	string	>100 (varies with time range)	Holds job's starting time
Stages{}.id	string	>100 (varies with time range)	Holds build's stage unique id
Stages{}.name	string	>100 (varies with time range)	Holds build's stage name
Stages{}.duration	string	>100 (varies with time range)	Holds build's stage time duration
Stages{}.error	string	>100 (varies with time range)	Holds build's stage error message
Stages{}.start_time	string	>100 (varies with time range)	Holds build's stage start time
Stages{}.status	string	4 (fixed); (success, not_built, failure, aborted)	Holds build's stage status after execution
Stages{}.childen{}.name	string	>100 (varies with time range)	Holds build's stage's children name
Stages{}.childen{}.duration	string	>100 (varies with time range)	Holds build's stage's children time duration
Stages{}.childen{}.error	string	>100 (varies with time range)	Holds build's stage's children error message

Table 3 Dataset schema (continued)

Filed name	*Field information*		
	Value data types/constraints	*No. of values; defined values*	*Description*
Stages{}.childen{}.id	string	>100 (varies with time range)	Holds build's stage's children unique ID
Stages{}.childen{}.start_time	string	>100 (varies with time range)	Holds build's stage's children start time
triggered_by	string	>100 (varies with time range); (started by timer, branch indexing, started by user##)	Holds who triggered the build
Queue_id	number	>100 (varies with time range)	Holds unique queue id
Queue_time	number	>100 (varies with time range)	Holds queue time
Metadata~	string	>100 (varies with time range)	Different fields starting with metadata~ holds data details

Figure 5 Data flowchart

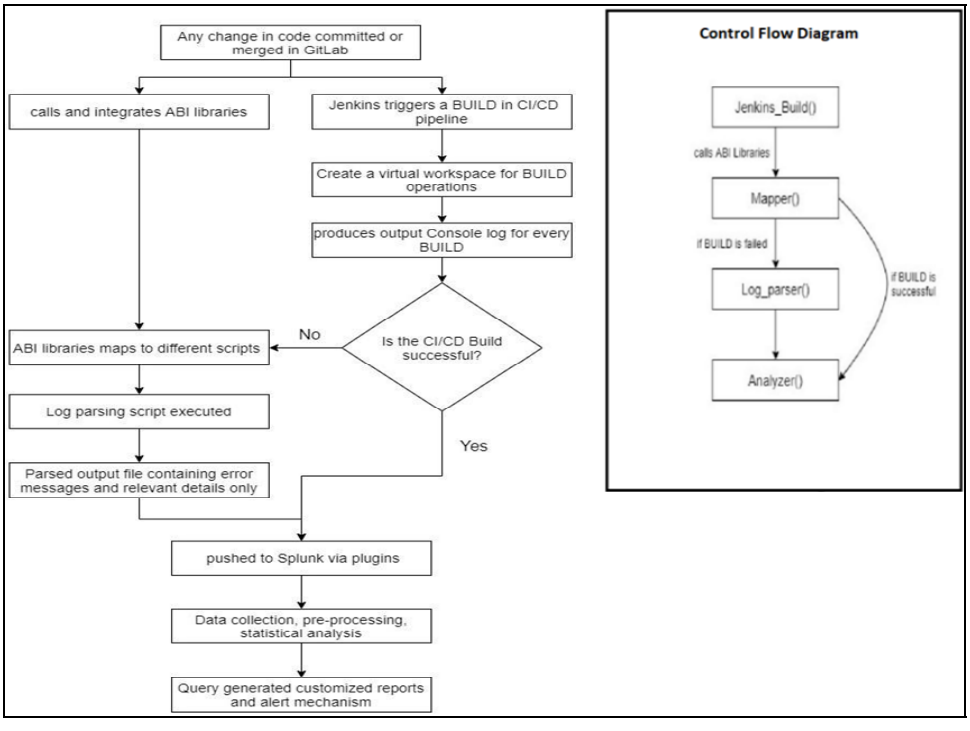


Table 4 Functional modules description

Functional block	*Inputs*	*Functionality*	*Output*
Jenkins_Build()	Any code pushed, committed or merged in GitLab.	Calls ABI Libraries and maps Jenkins to different scripts. Creates virtual workspace for different BUILD operations.	Raw unstructured output console log for every BUILD.
Mapper()	Context containing BUILD configurations and environment setup parameters.	Declares login credentials that are needed for workspace operations.	Raw unstructured output console log for every BUILD.
	Workspace details.	Construct file path for the context downloaded in the workspace. Calls log parsing script to parse output console log. Send parsed output to analyser tool.	Parsed output file containing error message and relevant details of every BUILD.
Log_Parser()	*Version 1:* Raw unstructured output console log for every BUILD.	Fetches error messages from output console log and writes in a file.	Parsed output file containing error message and relevant details of every BUILD.
	Version 2: Raw unstructured output console log for every BUILD.	Fetches error messages from output console log and writes in a file. Creates and identifies error patterns. Assign error pattern to every error message fetched.	Parsed output file containing error message and relevant details of every BUILD.
Analyzer()	*Version 1:* Raw unstructured output console log for every BUILD.	Data collection in SPLUNK database.	Customised interactive reports.
	Parsed output file containing error message and relevant details of every BUILD.	Pre-process unstructured raw data to structured data. Creates and identifies error patterns. Assign error pattern to every error message fetched. Group similar error patterns into categories. Querying database to generate results. Display statistical results and customised reports and enable alert and feedback mechanism.	Alert and feedback mechanism.

Table 4 Functional modules description (continued)

Functional block	*Inputs*	*Functionality*	*Output*
Analyzer()	*Version 2:* Raw unstructured output console log for every BUILD.	Data collection in SPLUNK database.	Customised interactive reports.
	Parsed output file containing error message and relevant details of every BUILD.	Pre-process unstructured raw data to structured data. Querying database to generate results. Display statistical results and customised reports and enable alert and feedback mechanism.	Alert and feedback mechanism.

Figure 6 CI/CD pipeline steps to BUILD triggered in Jenkins

```
Jenkins_Build (Library, Branch, proxy_settings)
Begin
    1. Initialize library = Library;
       // holds ABI Library specified by developer
    2. Create Workspace (Branch, Proxy_settings)
       // BUILD stages need to run in a virtual environment created by parameters
       2.1 Set proxy = proxy_settings;
           // for proxy settings
       2.2 Set Build_branch = Branch;
           // for specifying target branch of project- production/ test/ user defined
    3. Build stages initiated -
       3.1 Initialize Build Environment and configurations
           // holds BUILD parameters- proxy setting, branch, virtual environment
       3.2 Build, Test, Scan
           // predefined BUILD stages ** beyond scope of paper
       3.3 Post condition always (Workspace):
           // changes made only in post BUILD stage running in virtual workspace
           (a) Initialize Context = Build_console_log ;
               // holds output console log generated BUILD stages
           (b) Initialize Build_path;
               // holds BUILD absolute path
           (c) Mapper (Context, Credentials, Build_path, Workspace)
               // function calling
End
```

3.4 Implementation

Whenever a developer pushes or commits any change in code in GitLab, CI/CD pipeline triggers a job called 'build' in Jenkins – Jenkins_Build. It performs two functionalities in parallel – first, it calls and integrates all the required ABI libraries, i.e., Mapper and second, it creates a virtual workspace for different build stages. The ABI library Mapper acts as a mediator between the log parsing script – Log_parser and Jenkins_Build. Build stages running parallely in the workspace, produces an output console log which also contains the overall job result. If the job result turns out to be a failure, Mapper calls Log_parser. This parsing script is responsible for fetching the error messages and the

relevant related details of the build and writing it down in a new file. The complete raw unstructured console output and the parsed file created by Log_parser is pushed to the analysis tool Splunk. The raw data collected, pre-processed and processed to generate result and visuals is taken care by analyser, as represented in the flowchart Figure 5.

Table 4 represents the functional blocks, inputs, functionality and the desired outputs of every block – Jenkins_Build(), Mapper(), Log_Parser() and Analyzer(), respectively.

Figure 7 Mapper() algorithm

```
Mapper (Context, Build_path, Workspace)
Begin
    1. If (Context == Null) or (Build_path== Null) or (Workspace == Null)
       // conditions check on the Input file, absolute BUILD path and
       // virtual workspace details respectively
       Return;
    2. Initialize and declare Login_credentials;
       // holds developer defined Login Id and password for authorization
    3. With (Login_credentials && Workspace)
       3.1 Create Input_File = Context;
           // new file defined holds the BUILD output console log
       3.2 Initialize: Input_File_name = Input_File{name};
           // holds new file's name
       3.3 Initialize:
           Input_File_Path = Workspace + Build_Path + Input_File_name;
           // creating file path by appending the paths-
          // workspace path, absolute Build path and new file path
       3.4 Create Parsed_file = empty;
           // new file created to hold the output data
       3.5 Parsed_file= Log_Parser (Input_File)
           // holds output file returned by the function Log_parser
    4. Initialize: analyzer = true;
       // holds User's choice to send the data for analysis in Analyze_log
    5. If (analyzer)
       5.1 Analyze_Log (Input_File, Parsed_file)    // function calling

End
```

Figure 8 Log_Parser() algorithm

```
Log_Parser (Input_File, Parsed_file)

Begin
    1. Initialize error_pattern[n]; // holds predefined keywords to be searched in the input file
    2. While (Input_file)
       2.1 For every line j in Input_File    // traversing every line in input file
       2.2  {
                For ( 1 to n in error_pattern[] )     // traversing every keyword defined in Error_pattern
                {
                    If (error_pattern(n) found in j)
                       Add j && error_pattern(n) to Parsed_File;
                      // add the line i.e the error message and the keyword matched to the output file
                }
            }
    3. Return Parsed_File;                  // return output file
```

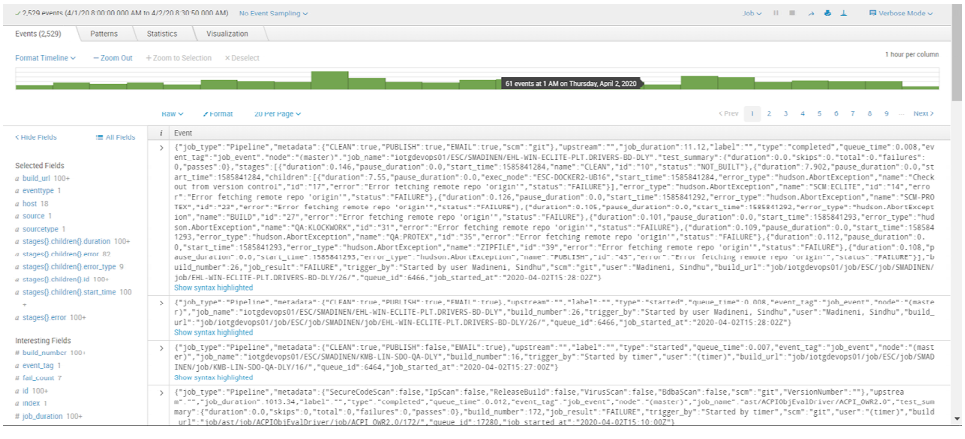
In stated blocks of Table 4 are well explained one-by-one in the algorithms given here. Figure 6 represents CI/CD pipeline steps that are followed at every build that is triggered in the Jenkins, i.e., Jenkins_Build(). Figure 7 represents an algorithm to map Jenkins to other scripts, i.e., using Mapper().

Figure 8 represents algorithm for performing parsing on input data files and storing results in the parsed output file. In version 1, *Log_Parser()* fetches error messages from output console logs generated for every BUILD by Jenkins. The parsing script writes the exact error lines along with the relevant details of BUILD into a new file containing parsed data. And the analytics tool is responsible for creating and identifying error patterns, assigning error patterns to every error message fetched and group similar error patterns into categories.

Figures 9–16 represents Analyze_Log (Input_File, Parsed_File) which illustrates tool-based steps that are followed to yield results from the data contained in the files.

- Figure 9 represents the collection of time stamped raw data from Input_File and Parsed_File.
- Figure 10 processes raw unstructured time stamped data into structured time stamped data.
- Figure 11 stores clean data into the SPLUNK database.
- Figure 12 represents the query generated results.
- Figure 13 creates and identifies error patterns from Parsed_File and assigns error patterns to every error message fetched.
- Figures 14(a) and 14(b) queries the database to generate results and statistical reports.
- Figure 15 displays the results in graphical visuals.

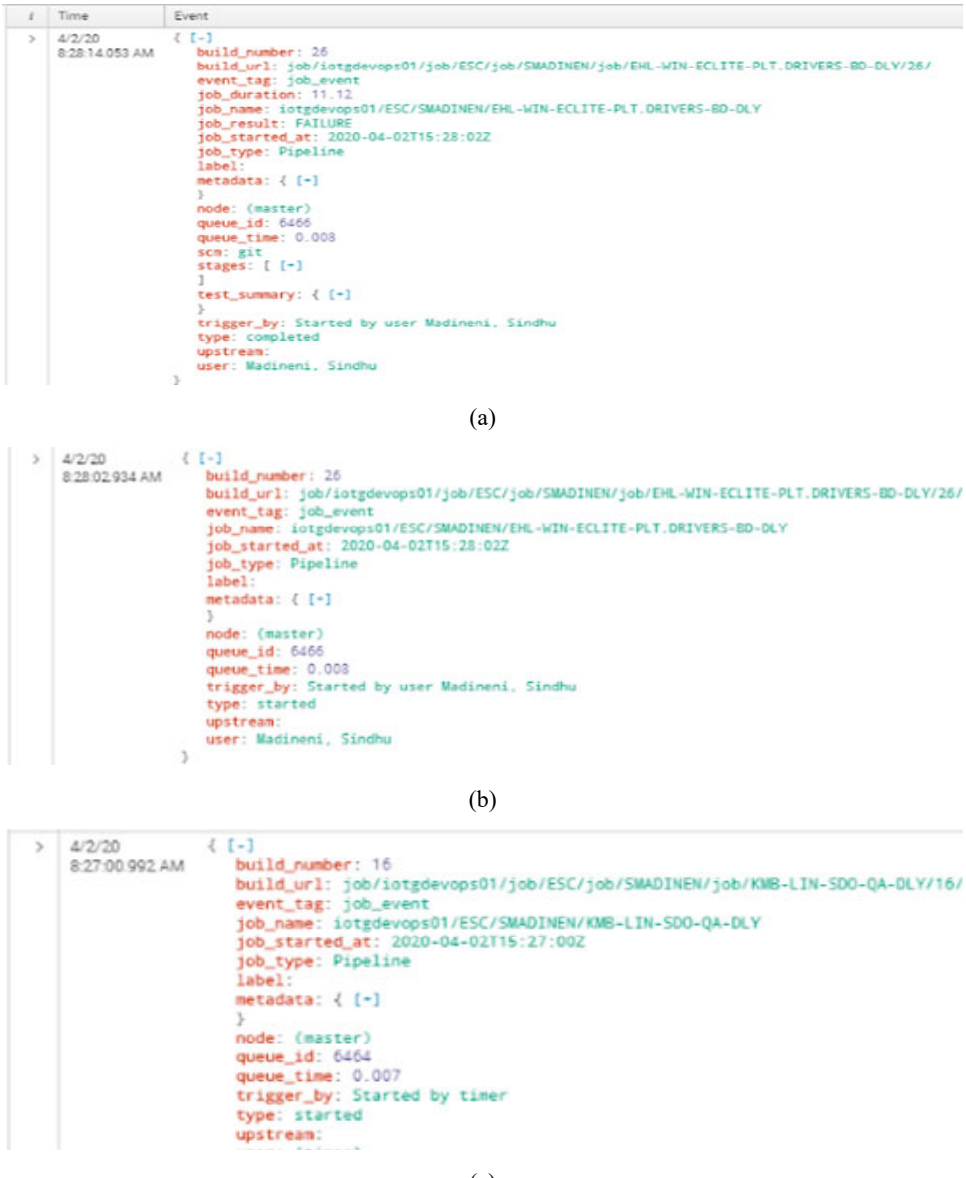
Figure 9 Raw unstructured time stamped data collected (see online version for colours)



In version 1, identifying the error patterns and assigning categories to the raw error message imposed extra overheads on Splunk search engine. In the initial stage when the data is collected via plug-ins is limited and the error patterns are stable, but with time, the

data collected have become large and new error patterns are identified, assigning error categories to raw error messages becomes unmanageable and time taking for the search engine. The report generating time of the analyser tool eventually becomes slow due to the overheads of processing the query and loading results.

Figure 10 Structured raw time stamped data in JSON format (see online version for colours)



(a)

(b)

(c)

Figure 11 Clean data into Splunk database

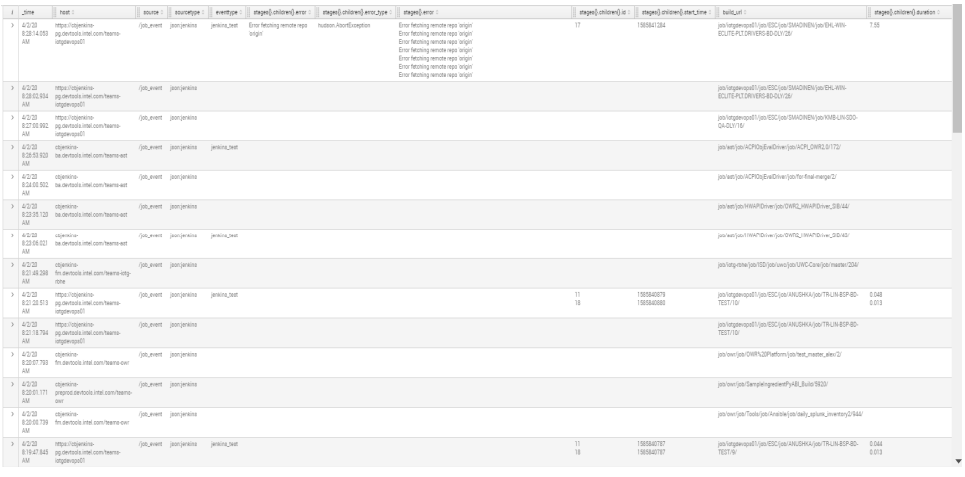


Figure 12 Query generated results (see online version for colours)

errortype	count(build_link)
ABI Exception error	28
Branch failure	2
Exception error	11
Gitlab error	4
Infrastructure error	5
Klockwork error	2
Miscellaneous error	7
Missing property error	11
Unknown ABI error	4
Warning error	23

Figure 13 Processing on query generated results to obtain desired fields (see online version for colours)

Therefore, to overcome the limitations that are faced in version 1, the task of identifying the error patterns and assigning categories to the raw error message is moved in the parsing script in version 2. Logical implementation of the parsing script as in Figure 16 *Log_Parser()* is modified to perform this task and *Analyze_Log()* has handled data collection, pre-processing of unstructured time stamped data and converting to structured

time stamped data, querying the database to generate results and statistical reports and displaying the results in graphical visuals.

Figure 14 (a) Query code to generated reports (b) Query generated reports targeting particular information (see online version for colours)

```
index="jenkins_console" sourcetype="text:jenkins" host="*"
| table _raw,_time,source,host
| rex field=_raw "[ ](?P<errorx>.*)#(?P<error_label>.*)"
| rex field=source "(?P<build_url>.*)console"
| eval host1 = replace(host, "https://", "")
| eval build_link = "https://".host1."/".build_url
| table build_url,host,build_link,errorx,error_label,_time |search errorx=*
| join type=left build_url
    [ search index=jenkins_statistics event_tag="job_event" host="*"
    | dedup host build_url sortby -_time
    | search `utc_to_local_time(job_started_at)`
    | convert timeformat="%Y-%m-%d %H:%M:%S" mktime(job_started_at) as epocTime
    | eval job_duration = if(isnull(job_duration), now() - epocTime, job_duration) ,queue_duration= if(isnull(queue_time), now() - epocTime, queue_time)
    | eval "Duration" = tostring(job_duration,"duration"), "Duration1"=tostring(queue_duration,"duration")
    | eval job_result=if(type="started", "INPROGRESS", job_result)
    | sort -job_started_at
    | search job_name != "TIC Gen*Gen" job_name != "*TIC Gen*"
    | eval host1 = replace(host, "https://", "")
    | eval build_link = "https://".host1."/".build_uri
    | table build_url,host,job_name,job_started_at,Duration,job_result]
| eval errortype = case(like(errorx,"%klockwork%"),"Klockwork error",like(errorx,"%[pipeline]%"),"Miscellaneous error",like(errorx,"%gitlab error%"),"Gitlab error",like(errorx,"%artifact error%"),"Artifact
    error",like(errorx,"%abortexception:%"),"Exception error",like(errorx,"%Keyerror:%"),"Key error",like(errorx,"%namererror:%"),"Name error",like(errorx,"%cache entry deserialization failed%"),"Infrastructure
    error",like(errorx,"%typeerror:%"),"Type error",like(errorx,"%loading libraries failed%"),"load error",like(errorx,"%attributeerror: %"),"Attribute error",like(errorx,"%artifactory%"),"Artifact error",like
    (errorx,"%abi_warning:%"),"Unknown ABI error",like(errorx,"%raise abiexception%"),"ABI Exception error",like(errorx,"%failed in branch %"),"Branch failure",like(errorx,"%abi_exception%"),"ABI Exception
    error",like(errorx,"%could not find%"),"Missing property error",like(errorx,"%: warning %"),"Warning error",like(errorx,"%failed to run ticgen:%"),"TIC GEN Error")|search errorx IN( "*" ) |search job_result
    ="*"
| rename host as "Master"
| dedup build_link errorx
|search errortype IN("*")
| timechart span=5h count by errortype
```

(a)

_time	ABI Exception error	Branch failure	Exception error	Gitlab error	Infrastructure error	Klockwork error	Miscellaneous error	Missing property error	Unknown ABI error	Warning error
2020-04-23 23:00	0	0	0	0	0	0	0	0	0	0
2020-04-24 04:00	0	0	0	0	0	0	0	0	0	0
2020-04-24 09:00	0	0	0	0	0	0	0	0	0	0
2020-04-24 14:00	0	0	0	0	0	0	0	0	0	0
2020-04-24 19:00	0	0	0	0	0	0	0	0	0	0
2020-04-25 00:00	5	1	2	1	0	1	1	0	1	0
2020-04-25 05:00	0	0	0	0	0	0	0	0	0	0
2020-04-25 10:00	0	0	0	0	0	0	0	0	0	0
2020-04-25 15:00	0	0	0	0	0	0	0	0	0	0
2020-04-25 20:00	0	0	0	0	0	0	0	0	0	0
2020-04-26 01:00	0	0	0	0	0	0	0	1	0	4
2020-04-26 06:00	0	0	0	0	0	0	0	0	0	0
2020-04-26 11:00	0	0	0	0	0	0	0	0	0	0
2020-04-26 16:00	0	0	0	0	0	0	0	0	0	0
2020-04-26 21:00	5	1	2	1	1	1	1	2	1	8
2020-04-27 02:00	0	0	0	0	0	0	0	0	0	0
2020-04-27 07:00	2	0	1	0	0	0	1	3	0	3

(b)

The new approach in version 2, resulted in an improved performance of Splunk search engine and quick loading of results. The efficient result generating and displaying time has made it likable and more adaptable in the production environment.

Figure 15 Creating visuals for the reports displaying information

```
index = jenkins_statistics "host"="*" "job_name"="*" build_url=job/tar/job/Owr%2GPlatform/job/Develop/$$$/ |
eval Build_type = if((like{trigger_by,'%Started by time%) OR like(trigger_by,"%Branch indexing%")),
"Production","Testing")| search Build_type = "Production}
| spath pathstages() output()
| files = _read
| eval job_results = if(type = "started", "IN PROGRESS" , job_result)
| search job_results = "FAILURE"
| expand x
| spath input*x
| eval start_time = starttime {start_time, '%Y' =%time  mm : ss : ms}
| eval host1 = replace(host, "https://", "")
| eval BUILD_LINK = "https://",host1."/".build_url
| eval name.status.duration.start_time.children{}.name,children{}.status.children{}.duration.children{}.
exec_node_type.error
| rename children{}.name as "STEPS_NAME", children{}.status as "STEPS_STATUS".children{} , children{}.duration
as "STEPS_DURATION".children{}, server_node as "SLAVE_NODE", name as "STAGE_NAME", duration as "DURATION", start_time as "START_TIME"
| eval child_time = strftime{child_time, '%Y' =%time  mm : ss : ms}
| expand status
| search status!= 'Failed'
| search status!= 'Successful'
| search status!= 'Aborted'
 rename status as 'STATUS'.error_type as "ERROR_TYPE".errpr as "ERROR"
```

Figure 16 Algorithm for parsing on input data file and store result in parsed output file

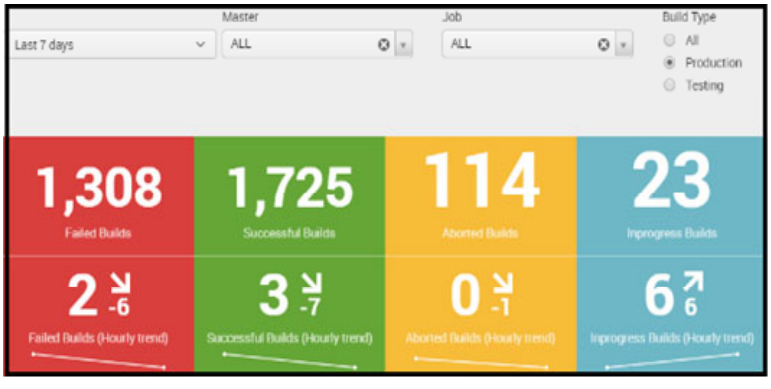
```
Log_Parser (Input_File, Parsed_file)

Begin
1.  Initialize error_pattern[n];      // holds predefined keywords to be searched in the input file
2.  Initialize Miscellaneous[] = Null;
    // holds line of the input file where error message and error pattern does not match error category
3.  Intilialize Flag= false;
4.  Initialize error_category [level_1][level_0];      // pre-defined error categories their error patterns
5.  While (Input_file)
        For every line j in Input_File                 // traversing every line in input file
        {  For (i = 1 to n in error_pattern)           // traversing every keyword defined in Error_pattern[]
           {      If (error_pattern(n) found in j) :
                  {      Flag= true;          // when error pattern is found in line
                         For (p = 1 to level_1 in error_category)
                             // traversing every category defined in Error_category
                         {  For (q= 1 to level_1 in error_category)
                             // traversing every pattern define under each Error_category
                             {  If (error_pattern (i) matches error_category[p][q]):
                                    Add j && error_pattern(n) && error_category[p][q] to Parsed_File
                    // add error message, keyword matched in error_pattern() and error category to output file
                             }
                         }
                  }
                  Else
                         Add j to Miscellaneous[];
           }
        }
6.  return Parsed_File;    //return output file
End
```

4 Execution results

This section summarises the results that are generated on real-time data collected from Jenkins in Splunk database. The result is produced after performing *Analyze_Log* is displayed in graphical visuals to provide more information on builds, platform/ infrastructure and performance trends as shown below:

Figure 17 Summarising build counts (see online version for colours)



4.1 Build analysis

Figure 17 depicts the summary of build counts – 1,308 failed builds, 1,725 successful builds, 114 aborted builds, etc. Figures 18(a)–18(c) depicts the summary of build statistics-based upon: builds per user as in Figure 18(a), top issues as in Figure 18(b), and

builds per branch as in Figure 18(c) respectively. Also, Figure 20 depicts drilldown to specific build details.

Figure 18 (a) Summarising build details per user statistics (b) Summarising build details top issues statistics (c) Summarising build details per branch statistics (see online version for colours)

Builds per user

User	ABORTED	FAILURE	INPROGRESS	SUCCESS
(timer)	28	685	13	1183
anonymous	86	623	10	542

(a)

Top issues

Step Title	Build_count
Check out from version control	350
SSH Steps: sshCommand - Execute command on remote node.	122
Get Artifactory server from Jenkins config	11
Upload artifacts	9
Restore files previously stashed	7
Publish xUnit test result report	5
Find files in the workspace	3
PowerShell Script	3
Download artifacts	2

(b)

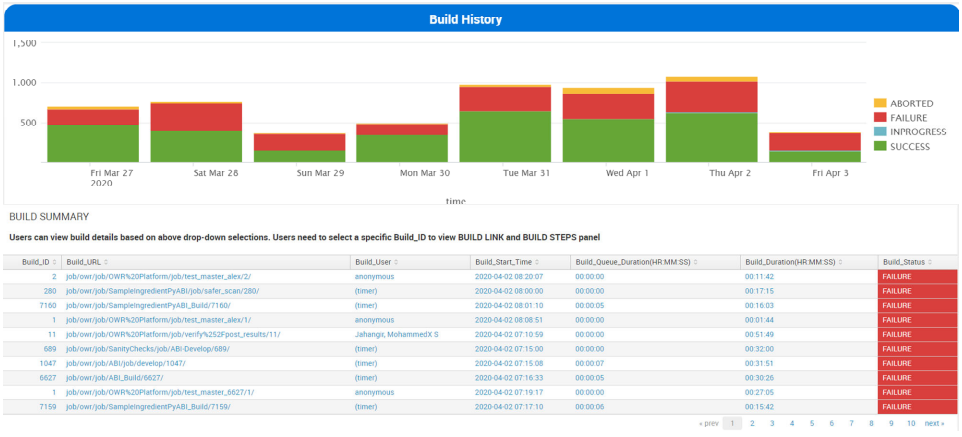
Builds per Branch

Branch_name	Build_count
owr/SampleIngredientPyABI_Build	450
owr/Tools/Ansible/restart_jenkins_service2	323
owr/Tools/Ansible/backup_abrenlldropbox	84
owr/stability_monitoring/stable	80
ast/FFT/FFT_OWR	59
owr/SampleIngredientPyABI/safer_scan	57
owr/OWR Platform/develop	47
owr/Tools/Ansible/daily_splunk_inventory2	45
ast/PowerAndThermal/PECI_Migration	44
owr/SanityChecks/ABI-Develop	42

« prev 1 2 3 4 5 6 7 8 9 10 next »

(c)

Figure 19 Summarising build details (see online version for colours)



4.2 Platform/infrastructure analysis

Figure 20 depicts summarisation infrastructure related details as node/mater correlation. In other words, agent types are distributed for specific masters. Figure 21 depicts the summary of slave server nodes details via agent types' distribution based on standby and overall agent pie chart.

Figure 20 Summarising infrastructure related details as node/master correlation (see online version for colours)

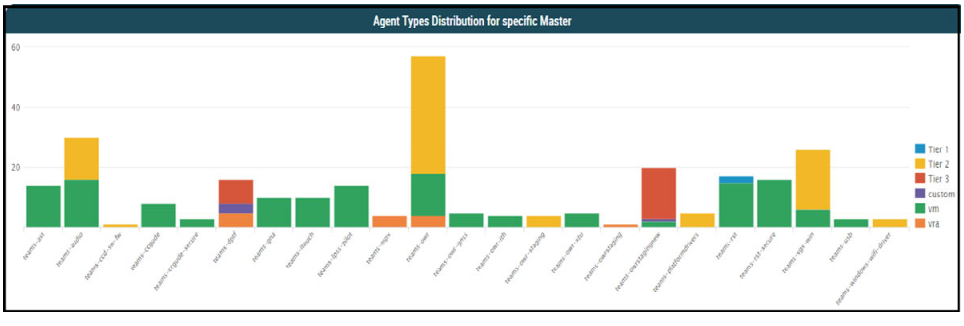


Figure 21 Summarising slave server node details (see online version for colours)

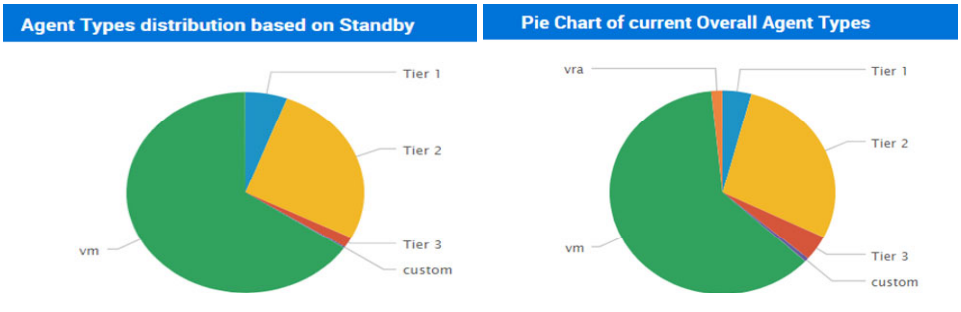


Figure 22 Continuous indicator of the pipeline health based on build's end result

4.3 Performance analysis

Figure 22 depicts a continuous indicator of the pipeline health based on build end result. Figure 23 depicts the summarising infrastructure setup services that are provided by various modules such as Jenkins, Artifactory, Rancher, Klocwork, Teamcity, GitLab, Microservices, Protex, HD-Des, Black Duck Binary Analysis, OneBKC, Splunk, symbols. Figure 24 depicts continuous checks on active servers through statistics, i.e., average build time per team.

Figure 23 Summarising infrastructure setup services provided by the various modules (see online version for colours)

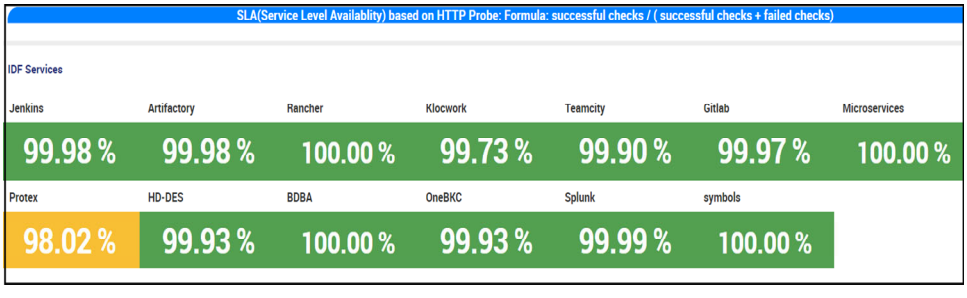
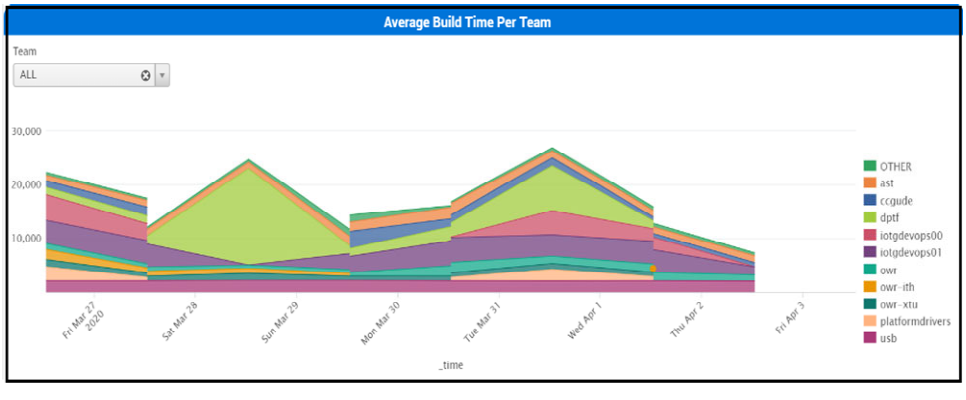


Figure 24 Continuous check on active servers through statistics (see online version for colours)



5 Conclusions and future works

This study demonstrates that certain difficulties must be overcome on the development and operational sides. To begin with, digging up historical repositories is difficult due to the migration to new tools and technology. Creating a benchmark dataset that includes real-time issues concerns is now a time-consuming effort. Secondly, efficient and automated storage of associated source codes and logs in run-time repositories is necessary. Thirdly, collecting and analysing various types of real-timed data must be reliable and always available for infrastructure support. Finally, providing timely feedback, assessment and evaluation reports to team members with specific information is enormously challenging.

This paper proposes a tool for enriching log analysis and reduces manual efforts which automates the correlation among various telemetry data. The proposed solution is quite supportive for developing and maintaining the quality of continuous practices that are used in the DevOps project. This article analyses logs in depth and encourages quality assessments and feedback to developers, which helps to diagnose telemetry data more thoroughly. This research conducts an empirical investigation to establish conceptual clarity about integration pipeline architecture and examine how the automation speeds up and expands the system feedback loop in the continuous monitoring.

All observations of the case study are limited to the organisation's exposure to DevOps methodology. In the future, the amalgamation of machine learning techniques for classification and clustering, can build a more powerful model which will efficiently classify the types of uncertainties and cluster them according to their source and end results. After attaining maturity, the model can predict the potential issues beforehand and forecast them with the evidence produced while analysing the system logs.

References

Alnafessah, A., Gias, A.U., Wang, R., Zhu, L., Casale, G. and Filieri, A. (2021) 'Quality-aware DevOps research: where do we stand?', *IEEE Access*, Vol. 9, No. 9, pp.44476–44489.

Castellanos, C., Varela, C.A. and Correal, D. (2021) 'ACCORDANT: a domain specific-model and DevOps approach for big data analytics architectures', *Journal of Systems and Software*, Vol. 172, No. 4, p.110869.

Chen, B. (2019) 'Improving the software logging practices in DevOps', in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, IEEE, May, pp.194–197.

Dörnenburg, E. (2018) 'The path to DevOps', *IEEE Software*, Vol. 35, No. 5, pp.71–75.

Fedushko, S., Ustyianovych, T., Syerov, Y. and Peracek, T. (2020) 'User-engagement score and SLIs/SLOs/SLAs measurements correlation of e-business projects through big data analysis', *Applied Sciences*, Vol. 24, No. 10, pp.1–16.

Frijns, P., Bierwolf, R. and Zijderhand, T. (2018) 'Reframing security in contemporary software development life cycle', in *2018 IEEE International Conference on Technology Management, Operations and Decisions (ICTMOD)*, IEEE, November, pp.230–236.

Ganeshan, M. and Vigneshwaran, P. (2021) 'A survey on DevOps techniques used in cloud-based IOT mashups', in *ICT Systems and Sustainability*, pp.383–393, Springer, Singapore.

Geissdoerfer, K. and Wolisz, A. (2019) 'Walker: DevOps inspired workflow for experimentation', in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, IEEE, April, pp.277–282.

Gokarna, M. and Singh, R. (2021) 'DevOps: a historical review and future works', in *2021 International Conference on Computing, Communication, and Intelligent Systems (ICCCIS)*, IEEE, February, pp.366–371.

Hemon, A., Fitzgerald, B., Lyonnet, B. and Rowe, F. (2019) 'Innovative practices for knowledge sharing in large-scale DevOps', *IEEE Software*, Vol. 37, No. 3, pp.30–37.

Kamuto, M.B. and Langerman, J.J. (2017) 'Factors inhibiting the adoption of DevOps in large organisations: South African context', in *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, IEEE, May, pp.48–51.

Katal, A., Bajoria, V. and Dahiya, S. (2019) 'DevOps: bridging the gap between development and operations', in *2019 3rd International Conference on Computing Methodologies and Communication (ICCMC)*, IEEE, March, pp.1–7.

Kersten, M. (2018) 'A Cambrian explosion of DevOps tools', *IEEE Computer Architecture Letters*, Vol. 35, No. 2, pp.14–17.

Koilada, D.K. (2019) 'Business model innovation using modern DevOps', in *2019 IEEE Technology & Engineering Management Conference (TEMSCON)*, IEEE, June, pp.1–6.

Perera, P., Silva, R. and Perera, I. (2017) 'Improve software quality through practicing DevOps', in *2017 Seventeenth International Conference on Advances in ICT for Emerging Regions (ICTer)*, IEEE, September, pp.1–6.

Pietrantuono, R., Bertolino, A., De Angelis, G., Miranda, B. and Russo, S. (2019) 'Towards continuous software reliability testing in DevOps', in *2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST)*, IEEE, May, pp.21–27.

Pingrong, L., Xiaoquan, S. and Junqin, Y. (2021) 'Research on the application of DevOps in the smart campus of colleges and universities', in *Journal of Physics: Conference Series*, IOP Publishing, April, Vol. 1883, No. 1, p.12101.

Rafi, S., Yu, W., Akbar, M.A., Mahmood, S., Alsanad, A. and Gumaei, A. (2021) 'Readiness model for DevOps implementation in software organizations', *Journal of Software: Evolution and Process*, Vol. 33, No. 4, p.e2323.

Stahl, D., Martensson, T. and Bosch, J. (2017) 'Continuous practices and DevOps: beyond the buzz, what does it all mean?', in *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, IEEE, August, pp.440–448.

Trubiani, C., Jamshidi, P., Cito, J., Shang, W., Jiang, Z.M. and Borg, M. (2018) 'Performance issues? Hey DevOps, mind the uncertainty', *IEEE Software*, Vol. 36, No. 2, pp.110–117.

Veres, O., Kunanets, N., Pasichnyk, V., Veretennikova, N., Korz, R. and Leheza, A. (2019) 'Development and operations – the modern paradigm of the work of IT project teams', in *2019 IEEE 14th International Conference on Computer Sciences and Information Technologies (CSIT)*, IEEE, September, Vol. 3, pp.103–106.

Yarlagadda, R.T. (2021) 'DevOps and its practices', *International Journal of Creative Research Thoughts*, ISSN: 2320-2882.