AN EMPIRICAL INVESTIGATION OF DEVOPS PIPELINE PROGRAMS USING JENKINSFILES

Nishali D'Mello
Dept. of Computer Science
University of Illinois, Chicago
ndmell2@uic.edu

Manika Maheshwari
Dept. of Computer Science
University of Illinois, Chicago
mmahes3@uic.edu

ABSTRACT

The goal of this project is to empirically investigate a large number of devops pipeline programs and obtain statistical data that describes the content and patterns in devops pipelines. We have searched and obtained Jenkinsfiles and other pipeline artifacts from GitHub. GitHub contains hundreds of pipeline examples and other code artifacts. Our project focuses on the following Research Questions :What are the most and the least frequent operations in pipeline stages? How often timeout periods are used in the pipeline runs and what are the most frequent intervals? What tools are the most and the least frequently used in pipelines? Which agents are most and least frequently used in pipelines?

Categories and Subject Descriptors

D.2.8 [Software Engineering]: Metrics—Product metrics; D.2.8 [Software Engineering]: Metrics—complexity measures, performance measures

General Terms

Mining software repositories, open source, empirical study, largescale software, patterns, pipelines, Jenkinsfiles, artifacts

Keywords

software repository, empirical study

1 INTRODUCTION

The goal of empirical investigation of pipeline programs is to analyze the content and patterns in DevOps Pipelines using statistical methods based on probabilistic reasoning .

GitHub has 24 million users from almost 200 countries and more than 25 million repositories. These repositories include pipelines which are built using Jenkinsfiles.

'Pipeline as code' or defining the deployment pipeline through code rather than configuring a running CI/CD tool, provides tremendous benefits for teams automating infrastructure across their environments. One of the most popular ways to implement a pipeline as code is through Jenkins Pipeline. Jenkins, an open source automation server, is used to automate tasks associated with building, testing, and deploying software.

Jenkins Pipeline is a suite of Jenkins features, installed as plugins, which enable implementation of continuous delivery pipelines, which are the automated processes for getting software from source control through deployment to end users.

Pipeline supports building Continuous Delivery (CDel) pipelines through either a Web UI or a scripted Jenkinsfile. Jenkinsfiles, using a domain specific language based on the Groovy programming language, are persistent files that model delivery pipelines "as code", containing the complete set of encoded steps (steps, nodes, and stages) necessary to define the entire application life-cycle. Pipeline facilitates CI/CD as it's the path to modeling and then automating software delivery; essentially becoming the intersecting point between development and operations. The Pipeline plugin, inspired by the build flow Plugin, also provides other capabilities like the ability to suspend and resume executing jobs, to manage pipeline code with source control, and to share libraries to extend the domain specific language support.

The most common basic pipeline described in a Jenkinsfile consists of three stages: Build, Test, and Deploy. The Build stage is typically where source code is assembled, compiled and packaged. Jenkinsfile is not a replacement for a build tool, but rather orchestrates the build and manages the resulting outputs through scripted steps [4].

This project uses JenkinsFiles of repositories from GitHub to analyze the content and patterns in DevOps Pipelines. To analyze the Pipeline Artifacts, we have focused on research questions and their answers that were procured after analyzing the obtained Jenkinsfiles. To analyze the data collected, we have focused on the following research questions:

1. **What are the most and the least frequent operations in pipeline stages?**

Containing a sequence of one or more stage directives, the stages section is where the bulk of the "work" described by a Pipeline will be located. At a minimum it is recommended that stages contain at least one stage directive for each discrete part of the continuous delivery process, such as Build, Test, and Deploy[1].

We Run a Python Script to Parse through all the Jenkinsfiles. This parser parses through the Jenkinsfiles to get the occurrence of the stages and find out which among them occur the least and most frequently.

2. How often timeout periods are used in the pipeline runs and what are the most frequent intervals?

Timeout is used to set a timeout period for a stage, after which Jenkins should abort the stage[1][2][3].
We use another parser to parse through all the Jenkinsfiles to find out how frequently timeout periods are used. This parser also gives us the most frequent time intervals used in a pipeline.

3. What tools are the most and the least frequently used in pipelines?

A section defining tools to auto-install and put on the PATH.
Pipelines support 3 tools : Maven, Gradle and JDK. We use a parser to parse through all the collected Jenkinsfiles to find the most and least frequently used Tools.

4. Which agents are most and least frequently used in pipelines?

The agent section specifies where the entire Pipeline, or a specific stage, will execute in the Jenkins environment depending on where the agent section is placed. The section must be defined at the top-level inside the pipeline block, but stage-level usage is optional.
Pipelines uses agents like "any" ,none, "node", "label", "docker", etc. We use a Parser written in python to parse through all the collected Jenkinsfiles to find the most and least frequently used Agents.

2 IMPLEMENTATION

In this section we briefly describe how we implemented the infrastructure that we use to obtain answers to research questions.

The infrastructure consists of a Java Code to Download the Repositories from GitHub. And 4 Parsers written in Python to parse through the Jenkinsfiles obtained from these repositories.

2.1 Downloading the Git Repositories

The infrastructure of this project contains a Java Code that downloads Repositories from GitHub which contain a pipeline and thus a Jenkinsfile. We make sure to download only those repositories that have a Jenkinsfile as the entire empirical study of this project is based on analyzing these files as they contain Pipeline artifacts.
One limitation of our implementation that occurs here is the manual segregation of Jekinsfiles from other files in the downloaded repositories. As we were unable to get the Jenkinsfiles from these Repositories programmatically, we copied and pasted them to a separate folder manually.

2.2 Parser to find most and least frequently used operations in pipeline stages.

We Run a Python Script to Parse through all the Jenkinsfiles. This parser parses through the Jenkinsfiles to get the occurrence of the stages and find out which among them occur the least and most frequently.
Following is the working of this parser:

1. Assign the path where all the Jenkins files are stored.
2. Getting the list of files present in a directory.
3. Assign the variables to store the number of Stage Type used which can be:
    a. Build
    b. Test
    c. Deploy
    d. Run
    e. Checkout
    f. Preparation
    g. Artifact
4. Access the files present in the directory one by one in read mode.
5. Access the lines and replace delimiters with space.
6. Break the lines into words.
7. Check if word is equal to Build,
    Build = Build + 1
8. If word is equal to Test, increase the count by 1.
    Test = Test + 1
9. If word is equal to Deploy, increase the count by 1.
    Deploy = Deploy + 1
10. If word is equal to Run, increase the count by 1.
    Run = Run + 1
11. If word is equal to Checkout, increase the count by 1.
    Checkout = Checkout + 1
12. If word is equal to Preparation, increase the count by 1.
    Preparation = Preparation + 1
13. If word is equal to Artifact, increase the count by 1.
    Artifact = Artifact + 1
14. And plot the graph.

2.3 Parser to find how often timeout periods are used in pipeline runs and their intervals

We use another parser to parse through all the Jenkinsfiles to find out how frequently timeout periods are used. This parser also gives us the most frequent time intervals used in a pipeline.
Following is the working of the parser:

1. Assign the path where all the Jenkins files are stored.
2. Getting the list of files present in a directory.
3. Access the files present in the directory one by one in read mode.
4. Access the lines and replace delimiters with space.
5. The timeout can be represented in two forms:
    a. timeout(10)
    b. timeout(time: 1, unit: 'HOURS')
6. Using this functionality, we break the lines into words and store it in the form of list.

7. If that list contains timeout and has length 3, then it is (a) form and print it
8. else if that list contains timeout and has length 5, then it is (b) form and print it

2.4 Parser to find which Pipeline tools are most and least frequently used.

We use a parser to parse through all the collected Jenkinsfiles to find the most and least frequently used Tools.
Following is the working of the Parser:

1. Assign the path where all the Jenkins files are stored.
2. Getting the list of files present in a directory.
3. Assign the variables to store the number of tools used which can be:
    a. Gradle
    b. Maven
    c. JDK.
4. Access the files present in the directory one by one in read mode.
5. Access the lines and replace delimiters with space.
6. Break the lines into words.
7. Check if word is equal to Gradle,
    Gradle = Gradle + 1
8. If word is equal to Maven, increase the count by 1.
    Maven = Maven + 1
9. If word is equal to JDK, increase the count by 1.
    JDK = JDK + 1
10. And plot the graph

2.5 Parser to find which agents are most and least frequently used.

We use a Parser written in python to parse through all the collected Jenkinsfiles to find the most and least frequently used Agents.
Following is the working of the parser:

1. Assign the path where all the Jenkins files are stored.
2. Getting the list of files present in a directory.
3. Assign the variables to store the number of Agents Type used which can be:
    a. any
    b. node
    c. Docker
    d. label
    e. none
    f. windows
4. Access the files present in the directory one by one in read mode.
5. Access the lines and replace delimiters with space.
6. Break the lines into words.
7. Check if word is equal to any,
    any = any + 1
8. If word is equal to node, increase the count by 1.
    node = node + 1
9. If word is equal to Docker, increase the count by 1.
    docker = Docker + 1
10. If word is equal to label, increase the count by 1.
    label = label + 1
11. If word is equal to none, increase the count by 1.
    none = none + 1
12. If word is equal to windows, increase the count by 1.
    windows = windows + 1
13. And plot the graph

3 EMPIRICAL EVIDENCE

In this section, we analyze the results of our implementation These results were obtained as a result of the analysis related to our Research Questions. The Research Questions and their related answers and analysis are as follows:

1. What are the most and the least frequent operations in pipeline stages?
We analyzed 34 Jenkins files and found
Build stage implemented 45 times.
Deploy stage implemented 20 times.
Test stage was implemented 25 times.
Run stage was implemented 18 times.
Checkout stage was implemented 20 times.
Preparation stage was implemented 2 times.
And Artifacts stage was implemented 5 times.

2. How often timeout periods are used in the pipeline runs and what are the most frequent intervals?
We analyzed 34 Jenkinsfiles and found that the most frequent timeout was " timeout(10)" and timeout (Time:1 unit: Days)

3. What tools are the most and the least frequently used in pipelines?
We analyzed 34 Jenkinsfiles and found that most mentioned the Tools as Maven while the rest did not mention any agent.

4. Which agents are most and least frequently used in pipelines?
We analyzed 34 Jenkinsfiles and found the following
Any agent in 11 pipelines.
Node agent in 41 pipelines.
Label agent in no pipelines.
Docker agent in 28 pipelines.
Windows agent in 3 pipelines.

From the above findings we can conclude that even though the build stage is implemented multiple times, deploy stage is employed less times in comparison.
Most of the pipelines implement testing but not all implement testing.
The timeout used by pipelines is very rare and triggers are even rarer in the Jenkins files that we analyzed.
Maven was the tool most used by pipelines and even though JDK and Gradle is supported, it wasn't used in the Jenkinsfiles.
Even though the node agent seems to be used 45 times, it is not the case as Jenkinsfiles also have pipeline features that uses nodes in methods other than Agent.

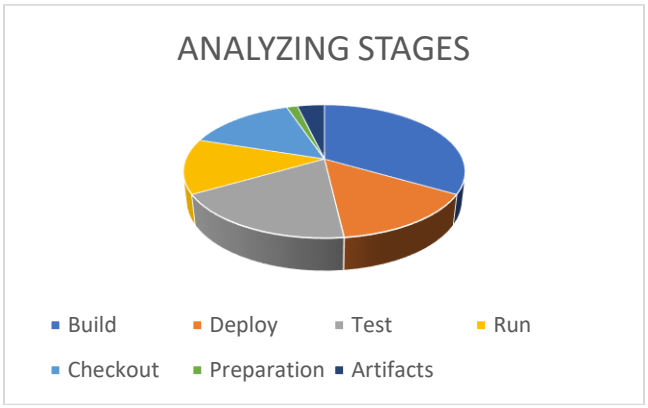

Figure 1: Distribution of Stages

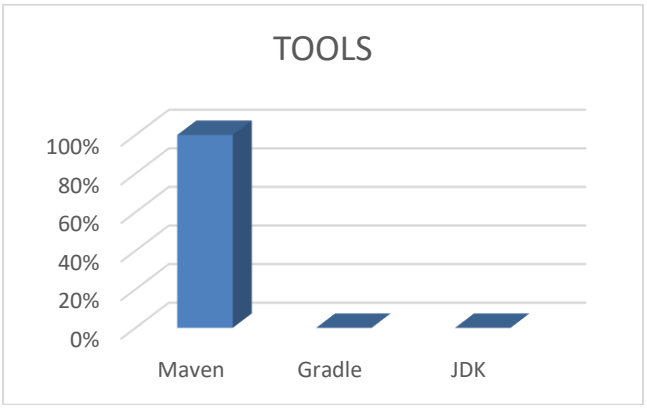

Figure 4: Tools Breakdown

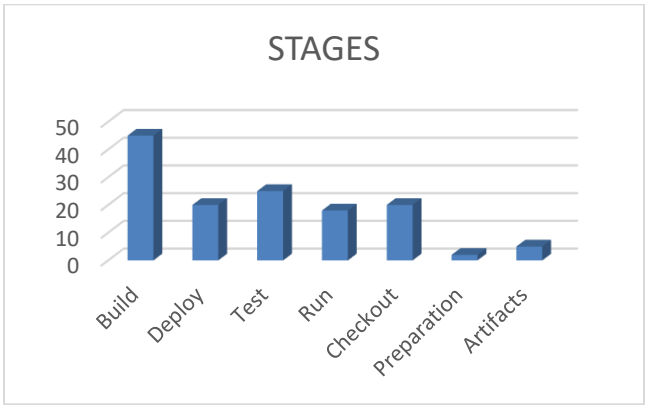

Figure 2: Stages Breakdown

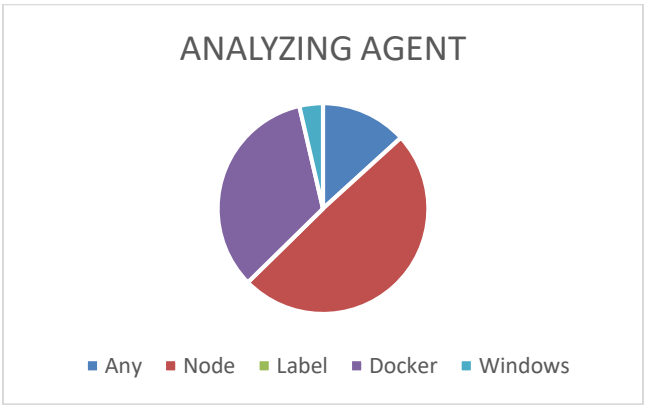

Figure 5: Analyzing Agent

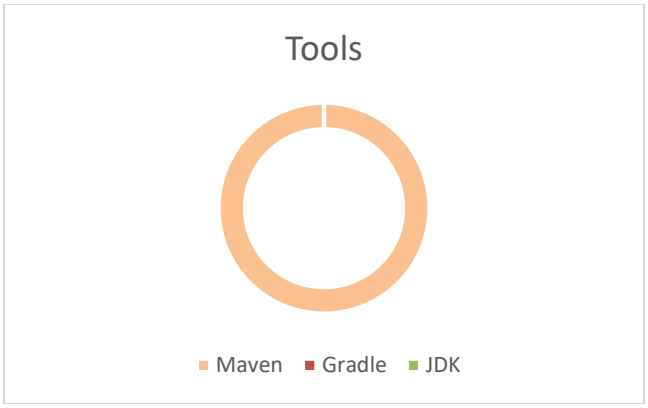

Figure 3: Analysis of Tools

Figure 6: Agent Breakdown

4 RELATED WORK.

Using Jenkins with Kubernetes AWS, Part 2

A Jenkins Pipeline is configured in a text file called Jenkinsfile in Groovy syntax. The Jenkinsfile consists of steps. A "step" is a build step, an instruction for Jenkins to implement. Two kinds of steps are supported: node and stage. A "node" is a top-level step that selects an executor/s on agent/s to run code on. A node is a machine (master or agent) and a label within the node step should match a label on the machine for the node to select the machine. The "node" step creates a workspace, which is a file directory, for a specific Jenkins job for resource intensive processing. The "node" step also schedules the steps defined within it on an executor slot by adding them to the Jenkins build queue. When the executor slot frees up the scheduled steps run from the build queue.

DevOps Pipeline with Docker
Mironov, Oleg (2018)

The result of this project is a fully working pipeline setup that is fully automated and is able to support a fast-paced software development. The pipeline is built against a reference project. Most of the pipeline is configured with a set of different configuration files meaning that from a fresh start it could be brought up with minimal human interaction. It covers all parts of a reference application lifespan from a development environment to a potential production deployment. There is a set of technologies used in the pipeline such as Jenkins, Docker and container orchestration with Rancher.

5 CONCLUSION

We analyzed 34 Repositories and thus 34 Jenkinsfiles with Pipeline Artifacts for our projects. As our code is made for a continuous execution, more repositories and Jenkinsfiles can be analyzed without changing the code for further accommodations.
From the above findings we can conclude that even though the build stage is implemented multiple times, deploy stage is employed less times in comparison.
Most of the pipelines implement testing but not all implement testing.
The timeout used by pipelines is very rare and triggers are even rarer in the Jenkins files that we analyzed.
Maven was the tool most used by pipelines and even though JDK and Gradle is supported, it wasn’t used in the Jenkinsfiles.
Even though the node agent seems to be used 45 times, it is not the case as Jenkinsfiles also have pipeline features that uses nodes in methods other than Agent.

6 REFERENCES

[1] en.wikipedia.org/wiki/Source_control_management
[2] en.wikipedia.org/wiki/Single_Source_of_Truth
[3] en.wikipedia.org/wiki/Domain-specific_language
[4] https://www.coveros.com/jenkins-pipelines-jenkinsfile/
[5] https://jenkins.io/doc/book/pipeline/syntax/