Butler

Prototype project report

Schweitzer Engineering Laboratories

\

Horizon group

Brendan Crebs

Introduction

I. Introduction

In software engineering the practice of continuous integration(CI) has become an increasingly important part of the software development process. Continuous integration is the idea that code from multiple contributors should be frequently built and tested as it is being pushed to a main branch. A tool that automates this process solves a number of common development problems which if solved, will lead to significantly greater efficiency in the long run. The aim of this project is to develop a generalized CI tool called BuTLer(build, test, lint) which we seek to integrate into a number of repositories at SEL.

One of the primary motivations of this tool is to have it generalized so that it will work in a number of drastically different code bases. At SEL, there are already a number of hardcoded CI tools that some teams have developed for themselves. Our team seeks to provide a standardized CI tool that can parse through any repository and construct a series of build, test, lint and publish tasks if the tool is being run on a publish branch. For SEL this could greatly increase the productivity of a number of teams who do not use CI or use inferior methods.

II. Background and Related Work

More broadly the software domain of this project would be automation, more specifically the domain would be *"continuous integration"* which was defined in the introduction. One of the biggest problems a CI tool like Butler seeks to solve is "integration hell" where it becomes incredibly difficult to integrate new code due to integration errors which arise when a child branch greatly deviates from the master branch. Having constant feedback on how your code may break things upon integration prevents a multitude of issues from arising and saves the time and resources that would be needed to fix them. In the domain of continuous integration one of the most well known and state of the art CI tools would be Jenkins. Jenkins is a highly flexible tool that primarily provides a syntax for a "pipeline" feature which can be setup to run a number of CI steps on a dedicated build server[1]. The flexibility of Jenkins allows for more in depth and specific CI tools like Butler to be easily integrated as a step in the pipeline. Jenkins itself is not sufficient for the task that Butler tries to solve. Without Butler we would have to define an enormous number of hardcoded tasks for Jenkins to perform. Instead, Butler will automatically determine project packages, construct build, test and lint tasks for each package and execute said tasks. The automated nature that Butler will have of parsing through the entire repository looking for application packages is a functionality that other CI tools do not supply. To accomplish this without leaving Butler error prone we will supply a syntax for special Butler configuration files that can be thought of as similar to makefiles where a team could specify Butler's behavior in their repository.

III. Project Overview

Butler will be implemented as a CLI tool written in Golang. Butler could be run locally, but most importantly it would be run on a build server as a step in a Jenkins pipeline. Each time code would be pushed from a local branch to the remote origin Butler would be run. When it is run it will construct a series of tasks that can be separated into build, test, lint, and in certain situations publish tasks.

Butler will start by reading configurations from a Butler config file that will be defined by the team. Since Butler could be operating in a variety of different repositories that use different languages, it is important that the users supply some information such as where to find dependencies for a language and what the relevant test and lint commands would be for that language. A .butlerignore file will also be supported which will allow a user to input paths for Butler to not check. After this, it will determine the git branch and the git diff of the current branch compared to the main branch. Checking the git diff will allow us to dynamically determine what application packages to create tasks for. We can then avoid always making tasks for all packages in the repo. This will greatly improve performance by excluding irrelevant packages. Determining the git branch is also very important. If a branch is master or a publish branch, then we will indeed create tasks for every application package along with the creation of special publish tasks. More on that later.

After the setup for Butler has been completed it will go on to satisfy the first objective which is linting. Linting is checking the code for stylistic issues, redundancies, or code that could potentially break. Butler will walk through the repo it is running in and execute a language specific linting command for each package. Anything flagged by the linter will cause the Butler to fail. It is our intention to make it so it is required that the most recent run of Butler on a non-master branch is successful before that branch can be merged with the master branch. This is so we can avoid merging incorrect code.

The next objective we want Butler to achieve would be test tasks. In a similar fashion to linting Butler will execute a language specific test task for every relevant package. The test command will execute unit tests and also gather coverage. Failure of any test or failure to achieve 100% test coverage will result in Butler failing the build. In addition to running unit tests, we also want Butler to run special fuzz tests. These fuzz tests will generate millions of random inputs for each package in an attempt to break them or find vulnerabilities. Discovery that an input breaks a package will result in Butler failing.

The next objective would be build tasks. Build tasks will be generated based on docker files that accompany an application package. In the various repositories at SEL there exist many applications within individual repos which Butler will have to parse through looking for relevant dockerfiles. To perform build tasks Butler will simply execute the instructions contained within the dockerfile adjacent to an application package.

On child branches from the main branch Butler will finish with build tasks and print a large collection of data collected from the build to a “butler results” json file. The user will also be told whether the build succeeded or not in the terminal. If it fails, we want the user to know exactly why it failed by printing the failing package and the specific task that failed such as a particular unit test.

As mentioned earlier we want functionality for a “publish task” which bundles up application packages into their distributable form then pushes them to antifactory. These tasks will only occur on a publish branch or a merge into the master branch.

It should be noted that due to the enormous number of expected tasks, Butler will take a long time to run. So, performance considerations are of high importance here. My planned solution is to make Butler multithreaded. Butler will determine how many available threads there are and will then spin off a child process on each thread. These child processes will be to execute the tasks that we constructed.

It is our intention for Butler to be used in conjunction with Jenkins pipeline. We plan to instruct users to set up a pipeline and a build server that will be connected to their Bitbucket repo. This build server will run on a virtual machine however, the instance of Butler will run on a container on that virtual machine. The reason for separating them is because we also plan to distribute a specific container image for Butler to be run on. This image will be set up so that Butler will have access to any tools it needs on other team’s build servers.

IV. Client and Stakeholder Identification and Preferences

My client is Schweitzer Engineering Laboratories where my boss Matt Halladay will serve as my mentor and contact liaison. Stakeholders in this project will first be myself and my boss Matt. Other stakeholders will include my team who will be my first client when it is finished. Other future stakeholders would include any team that decided to adopt Butler as a CI tool in their development process. Other stakeholders that I will not be directly interacting with would be companies that supply dependencies such as Jenkins.

The clients for this project would need clean, working code that is easy to use. This will primarily be accomplished by providing extensive and easy to read documentation. This documentation would also include guides for usage such as editing the Butler config and setting up a build server. The needs of my boss and team who are expecting this application will be prioritized above all others.

V. Glossary

Dockerfile: a script used by docker to build container images.

Fuzzing: the testing practice of feeding code enormous amounts of random inputs in order to find bugs or vulnerabilities.

Golang: A widely used object oriented programing language.

Jenkins: A well known open source CI tool.

VI. References

[1]“Jenkins user handbook, Pipeline” Jenkins, 2023. [Online]. Available: https://www.jenkins.io/doc/book/pipeline/. [Accessed: 20-Sep-2023].