DEPLOYMENT PIPELINE DEVELOPMENT AT SCALE: AUTOMATING SOFTWARE AS A SERVICE

A Thesis presented to

the Faculty of the Graduate School

at the University of Missouri-Columbia

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

---

by

BROCK D. WEEKLEY

Dr. Dong Xu, Thesis Supervisor

MAY 2022

The undersigned, appointed by the dean of the Graduate School, have examined the thesis entitled:

DEPLOYMENT PIPELINE DEVELOPMENT AT SCALE:

AUTOMATING SOFTWARE AS A SERVICE

presented by Brock Weekley,

a candidate for the Degree of Master of Science and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Dong Xu

Dr. Prasad Calyam

Dr. Yaw Adu-Gyamfi

ACKNOWLEDGMENTS

I would like to extend my gratitude to my advisor, Dr. Dong Xu, for providing guidance, critiques, and kind words at every part of the process as I pursued my research. I would not have been able to accomplish this without him. I would also like to thank my committee members, Dr. Prasad Calyam and Dr. Yaw Adu-Gyamfi, not only for finding time to collaborate and serve as my committee members, but also for playing integral parts in my growth as a student and Software Developer. To all of my committee members, your support, knowledge, and drive are not lost on me, thank you for striving to be the best at what you do.

To my family, who provided constant support, and to my parents who listened, advised, and loved me every step of the way, thank you from the bottom of my heart. To my friends, thank you for forcing me to have fun in between all of the hard work. You all reveled in my success, consoled me in my failures, and taught me to be the person I am today through this journey.

Brock Weekley

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

LIST OF FIGURES v

ABSTRACT vii

CHAPTERS

1. INTRODUCTION 1

1.1 Background 2

1.1.1 Cloud Computing 2

1.1.2 Software as a Service 3

1.1.3 Deployment Pipelines 5

1.2 Motivation 6

1.3 Problem Statement 8

1.4 Literature Review 9

1.4.1 Existing CI/CD Pipelines 10

1.4.2 Market Needs 15

1.5 Thesis Organization 21

2. APPLICATION DEVELOPMENT 21

2.1 Interface & Commands 23

2.2 Package Management 30

2.3 GitHub 33

2.4 Containerization 35

2.5 Golang 38

2.6 Code & Style Injection 40

2.7 Mobile Integration 42

2.8 Test Driven Development 43

2.9 Open Source License 45

3. USE CASES 46

3.1 Individual Developer 47

3.2 Startup Company 48

3.3 Enterprise Company 50

3.4 Integration With Existing Clouds 51

3.5 Test Results and Comparisons 52

4. FUTURE WORK 62

BIBLIOGRAPHY 65

LIST OF FIGURES

Figure Page

1 An Example Deployment Pipeline Managed with Jenkins ………………… 6

2 Top 10 CI Systems Used With GitHub.com ……………………………..….. 7

3 Number of Projects Using CI Over Time ……………………………………. 11

4 Reasons Developers Gave For Not Using CI ………………………………… 12

5 A Typical Blue-Green Deployment Diagram ………………………………… 17

6 The Banquet Logo ……………………………………………………………. 23

7 Banquet Initialization Through CLI ………………………………………….. 24

8 A Diagram of Banquet Command-Line Flow ……………………………….. 27

9 Server Initialization and Endpoint Options …………………………………… 29

10 An Example of Packages Downloaded From NPM …………………………... 32

11 Banquet Provided Access to Private Repositories on GitHub ………………... 34

12 Banquet's Go Mod File and Dependencies ………………………………….... 39

13 Example 1 of Style Injection ………………………………………………….. 41

14 Example 2 of Style Injection and Code Injection …………………………….. 42

15 Some Banquet Unit Tests …………………………………………………….…. 44

16 Deployment Time for Jenkins, Shared Runner, and Specific Runner ……...… 53

17 Deployment Time for Jenkins, Shared Runner, Specific Runner, and Banquet 54

18 Test Results for Deployment 1 ………………………………………………. 54

19 Test Results for Deployment 2 ………………………………………………. 54

20 Output of Banquet Init Command …………………………………………… 56

21 Prompt Answers to Banquet Dish Add Command ………………………….. 57

22 Output of Banquet Dish Add Command ……………………………………. 58

23 Deployed Application Created by Deployment 1 …………………………… 59

24 Results of Banquet Unit Tests ……………………………………………….. 60

ABSTRACT

In recent years, adaptability has become the most important aspect of software. As technology companies grow, sustainable scalability and increased security become the main goals for developers, while the company's goals become increased scalability and decreased development time. In this overlap of goals, the delivery model of Software as a Service has gained particular notoriety. This model has inspired the development of new types of tools, such as Content Management Systems, Continuous Integration Continuous Deployment Pipelines, and Progressive Delivery Tools. These tools, while beneficial, offer persistent disadvantages to the development of Software as a Service applications, such as high costs or high initial development time. Many companies, in all size classifications, have decided to build multiple applications for multiple clients, despite a large aggregate of code and design shared between apps. Within this market, there is a need for tools that allow developers to automate the process of creating SaaS apps.

In this thesis, a new tool has been developed that allows for the automatic style change and packaging of any web application for use by many companies at once. The tool, titled Banquet, is a command-line tool and API which will allow programmers to create CICD pipelines in a fraction of the time of normal development. This enables a low-cost, low development time solution for building Software as a Service that is abstract and multifaceted to fulfill as many use cases as possible. The tool utilizes some of the most prominent technologies of pipeline development and is open for customization to allow more technologies to be added. This makes Banquet essential for serving elaborate apps for many companies.

Chapter 1

INTRODUCTION

In 2022, the size of the public cloud services market is expected to grow 47.2% from 2020, a size of $270.033 billion to $397.496 billion over two years. The largest segment of this market, Software as a Service (SaaS) products, is expected to reach $145.377 billion. In this market and the world economy that is shifting due to factors including the global pandemic, cloud services continue to grow, providing adaptable and innovative solutions at every level of the tech industry [1].

The SaaS model eliminates the need for companies to create, manage, or even install software. Generally, a customer of a SaaS product subscribes to utilize the software at their company, allowing instant access and scalability to its employees. This reduces hardware, development, and infrastructure costs, making it an attractive option for many big businesses and industries [2].

This market growth and the efficiency of the SaaS model attract hundreds of thousands of new developers to the Cloud Computing industry each year. Despite this mass entry into the market, existing cloud tools are complicated, costly, and require extensive training [3]. As the cloud market shows no signs of stagnation, and the SaaS model has proven to be increasingly valuable, it will also become increasingly important for software developers to have access to efficient, cost-effective, and intuitive means of creating SaaS applications at scale.

1.1 Background

1.1.1 Cloud Computing

Colloquially, the “cloud” has become slang for any operation that a user does not see, often visualized as a physical entity of the company or brand where data goes. Data storage, authentication, or other server-side operations are often consolidated into this one idea for simplicity. As a newer concept in the computing industry, communicating trust in this model to end-users has shown to be complicated, creating challenges in streamlining development. Designing less complicated, lower barrier to entry tools for Cloud Computing could be essential in establishing a better public definition for the model and building increased trust.

Cloud Computing is academically defined as a “model for enabling convenient, on-demand network access to a shared pool of configurable computing resources” [4]. It is a revolutionary paradigm that allows for the acquisition of pay per use and seemingly infinite on-demand resources. With agile methodologies and DevOps culture, the cloud can help organizations innovate and quickly respond to market demand to provide continuous delivery of services [5]. This definition is beneficial for prospective developers in understanding the process of building a cloud for production. Public and private clouds alike typically share key similarities that create this definition of a “cloud”. The popular uses for the cloud model require convenience, elasticity, and adaptability. In order to achieve these characteristics, the development of cloud services can, at the least, take many years. Many of the most notable clouds are never complete, but continue to change and grow. Signficant development time leads to a growing industry of Cloud Computing developers as labor needs are fulfilled.

These aspects make Cloud Computing a powerful paradigm for managing resources and providing services to end-users. This model can be further specified into three distinct categories: Software as a Service, Platform as a Service, and Infrastructure as a Service [4]. Defining clouds in this way seems to be more efficient in communicating the specific use and specialization of a product. As such, the Cloud Computing tool developed for this thesis, Banquet, was designed to lower the barrier to entry for Software as a Service applications specifically, allowing more developers to create better SaaS products.

1.1.2 Software as a Service

Software as a Service stands out as an industry-defining category of cloud computing. It is a vital model for creating multi-tenant applications, meaning that an application can be developed one time and then tooled for multiple clients. The main characteristics of a SaaS application result in an increased need for customization, requirements, and integration. SaaS applications must be highly customizable to be used by many clients effectively. This flexibility can be found in design, including colors, images, text, and layout, as well as in the subjects of security, connection types, and more. As many clients use a SaaS application at one time, more frequent updates are typically needed to accommodate more requirements from these multiple clients. SaaS applications must also have the ability to integrate with many different protocols, such as JSON, HTTP, REST, and SOAP [6].

Recently, more models have emerged that incorporate SaaS. Service Oriented Architecture (SOA) and Microservice Architecture are two examples. SOA is “an enterprise-wide approach to software development of application components that takes advantage of reusable software components, or services” [7]. SOA is typically composed of many different SaaS applications across an entire company or service. Microservice Architecture is an approach that involves many loosely coupled components that work independently of each other to form an application [7]. Microservice Architecture is often used to create SaaS applications. These two models are great examples of how the uses of SaaS continue to grow and how a tool such as Banquet can make developing bleeding-edge Cloud software more intuitive. Banquet can easily manage the deployment of multiple SaaS applications in communication with each other, allowing for a Service Oriented Architecture [8].

End-users in this model can range from enterprise companies to single developers, depending on the type of software. This range makes multi-tenancy and its corresponding models a core problem in Cloud Computing. Effective multi-tenancy can improve the viability, efficiency, and stability of products. In contrast, poor development of multi-tenant software will result in wasted resources and inefficiency [9]. Currently, the most popular way to develop and deploy multi-tenant software is with deployment pipelines. Without a proper pipeline, there is excessive manual labor required to create a new app instance in the SaaS model, and developing these proper pipelines can take many years. The SaaS model would benefit significantly from increased velocity in the development of deployment pipelines.

1.1.3 Deployment Pipelines

Continuous Integration Continuous Deployment, or CICD, is a popular workflow for application development. The process of CICD is as follows: developers begin by planning a design or implementation, then create that implementation through code, followed by building, testing, and releasing. It aims to automate the build process, with goals of early bug discovery, quality assessment, short release cycles, and increased productivity [10]. It is also likely the most beneficial concept to understand in this thesis, as Banquet excels at building CICD pipelines.

A pipeline is a term for any group of software that facilitates the design principles of CICD. In enterprise software specifically, it can often be difficult to maintain code quality and development speed simultaneously. In maintaining much-needed development speed, technical debt often builds quickly. CICD pipelines let developers meet both needs contemporaneously [10]. Tools such as Jenkins Ansible and GitHub Actions are standard pipeline services used for deploying applications. These tools are well documented and suitable for building basic pipelines, but face several disadvantages when building pipelines for SaaS applications.

Pipeline development faces many challenges, including bad practices resulting in restructuring or inefficiencies, difficulty in maintaining and documenting, unnecessary additions, performance bottlenecks, and the inability to adapt to application changes [10]. Developing proper pipelines can take many years at minimum. As time moves forward, developers leave, projects change, and technology evolves, which will make most pipelines obsolete. This degradation is unsustainable due to the high development time and cost. Thus, a crucial challenge to software development involves determining if there is a better way to build pipelines.

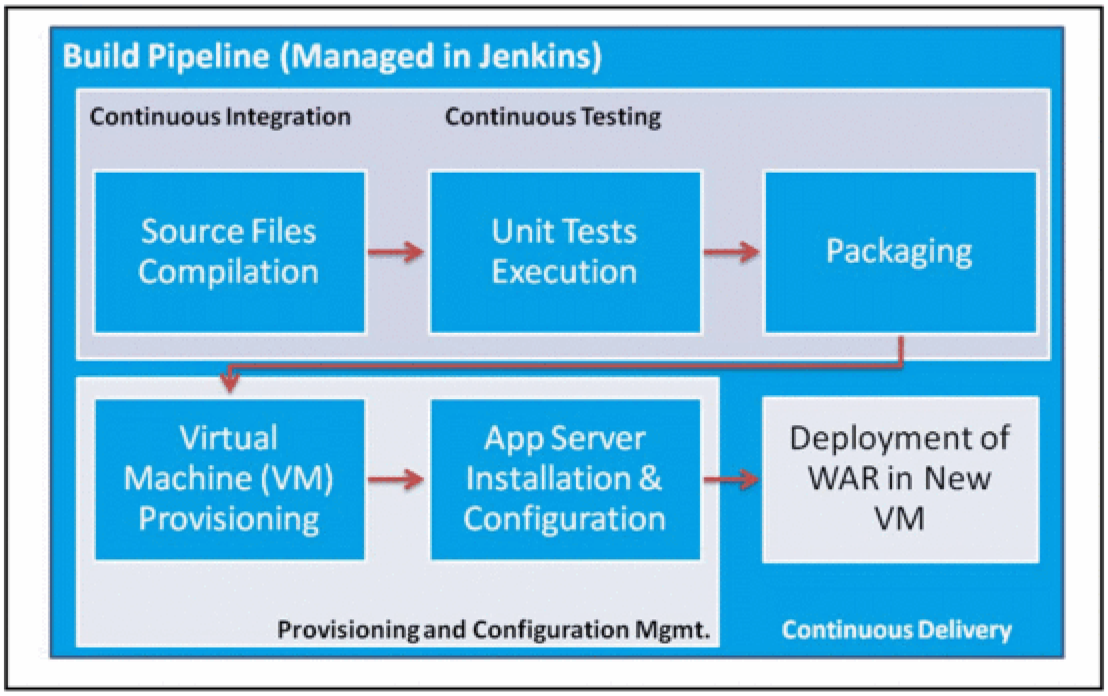

Build Pipeline (Managed in Jenkins)
Continuous Integration
Continuous Testing
Source Files Compilation
Unit Tests Execution
Packaging
Virtual Machine (VM) Provisioning
App Server Installation & Configuration
Deployment of WAR in New VM
Provisioning and Configuration Mgmt.
Continuous Delivery

Fig 1. An Example Deployment Pipeline Managed with Jenkins [5]

1.2 Motivation

Cloud Computing as an industry is growing by billions of dollars every year, including the category of Software as a Service. At every size category of tech development, there is an increasing need for specialized tools to aid in the development of deployment pipelines. Due to this surge in need, more developers are entering the Cloud Computing industry every year, seeking to build new models and efficiencies surrounding SaaS. As new developers enter the industry, it becomes painfully clear that the current process for creating deployment pipelines is outdated in a world that requires increasing flexibility and fast development. The solution to this is to automate the way developers do CICD.

Currently, automation for CICD does exist. According to GitHub, in 2017, the top 5 most used CI systems in order were: Travis CI, Circle CI, Jenkins, AppVeyor, and CodeShip. GitHub also stated that many projects utilized more than one of these automation tools in their project. However, academic surveys of users of these services have found that most are dissatisfied with the current tools [11].

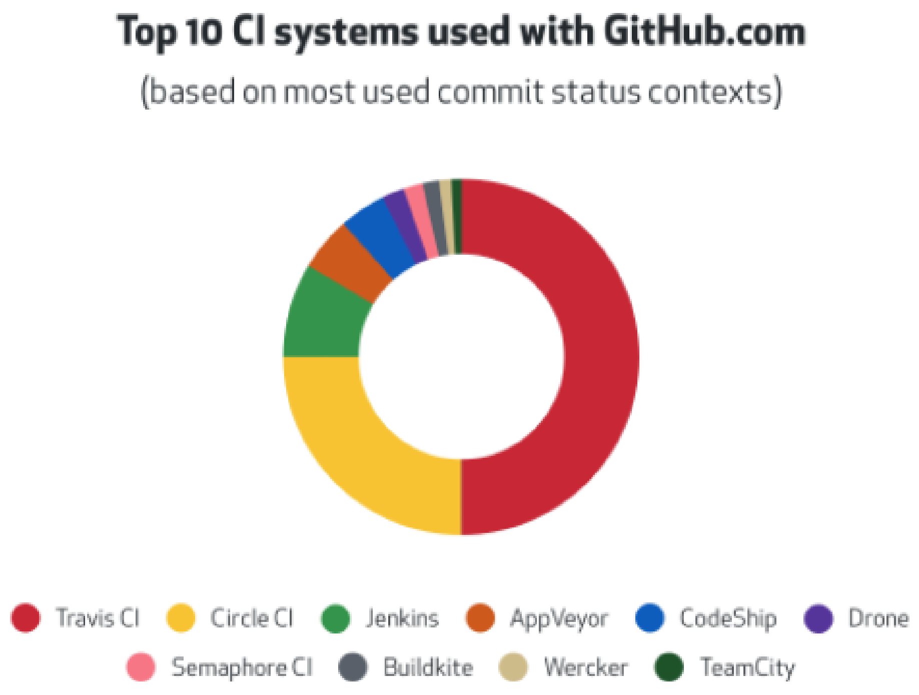


Fig 2. Top 10 CI Systems Used With GitHub.com [12]

The issue with current implementations is summarized in three categories: flexibility, complexity, and cost. CICD pipelines are typically built with a YAML file, which specifies a build process for a specific application. YAML, or “YAML Ain’t

Markup Language", is a Unicode-based data serialization language that is often used in configuration files [13]. YAML is the most popular form of configuration file for pipeline services, but there are other options, such as the Jenkins software's solution, "Jenkinsfile" [14]. YAML is a fantastic language for building a single build process, but it is not dynamic. One build process cannot build many different types of apps. In terms of complexity, building pipelines with these tools is not easy. "It can be a repetitive, time-consuming, manual, and complex process. Additionally, enterprise applications are so diverse and composed of so many technologies, open-source tools, and commercial tools, that it is becoming difficult to manage application delivery lifecycle for organizations while dealing with these complexities" [5]. Finally, the current services that exist for pipeline automation are expensive and not one size fits all solutions as "finding a cloud service provider (CSP) has become an intricate judgment. Nowadays it's not a question of which option we should work with, but rather, how to achieve the right performance and dispense risk across multiple vendors - while optimizing cost." [2].

1.3 Problem Statement

Banquet as a tool is designed to drastically reduce the development time of deployment pipelines and increase the scalability and customization of SaaS products. As an open-source project, it is open to tooling for specific needs by a wide range of developers, including mobile developers. In the current market, there are many services that facilitate the creation of a deployment pipeline, but few, if any, that easily allow Software as a Service customization or ample flexibility in pipeline use cases. Banquet focuses on multi-tenancy and content management that has the potential to change the way developers do SaaS.

Banquet achieves a better solution to the three core problems of existing pipeline development services. In flexibility, Banquet's CLI or REST API allow for automation and dynamic configuration that a singly configured YAML process does not offer. As a CLI, the initial complexity of utilizing Banquet is low, but the tool remains powerful in its ability to automate and integrate with other tools. As an open-source tool, hardware is the only limiting cost as Banquet is free to use. This tool seeks to lower the barrier to entry that developers entering the Cloud Computing and SaaS market face, decreasing development costs and increasing usability.

1.4 Literature Review

The cloud market is developing a new need for an elegant, low-cost solution to automating Software as a Service development. Most of the modern Internet uses cloud computing services in search of elastic and cost-effective needs [15]. The cloud market and software development, in general, are developing new requirements to meet these needs. Changes are coming at a rapid pace to deliver new services and products because the application delivery lifecycle is pertinent to the success of a business [5]. A huge gap exists between the requirements of software and the application delivery provided by the traditional approach to development, resulting in business loss [5]. In this literature review, papers from the most recent ten years studying the current needs of the cloud market, existing CICD tools, and the growing need for innovation in this area are summarized.

1.4.1 Existing CI/CD Pipelines

Due to a broad range of benefits, CICD is becoming a widely adopted practice in both industry and open-source. In a qualitative study of 8,000 GitHub pipeline projects, the findings show two main reasons that pipelines are changed. The first reason is to add functionality that did not previously exist within the pipeline. The second is to modify the pipeline to cope with a system or technology's change. Out of a studied 615 commits, nearly half of these changes had the purpose of increasing flexibility within a pipeline. The projects utilize the most popular existing pipeline services including: AppVeyor, Bamboo, Circle-CI, GitLab-CI, Jenkins, Semaphore, Travis-CI, CloudBees, and Wrecker [10].

In a similar study of 34,544 open-source projects on GitHub, the findings show that, out of projects that utilized some form of pipeline technology, 90.1% used Travis-CI for their pipeline services. The next most popular in order are Circle-CI, AppVeyor, and CloudBees. Of the surveyed projects, only 40.27% were utilizing any form of a pipeline, which shows that there are still many projects that perform deployments manually [16]. From developers surveyed from these projects, the findings show that 94% of developers were 'Definitely' or 'Most Likely' going to use a pipeline in their next project. This tentative interest means that it is safe to predict that pipeline adoption rates will continue to increase [16]. However, the limitations of existing tools and technologies are inhibitors to the goals of continuous practices [3].

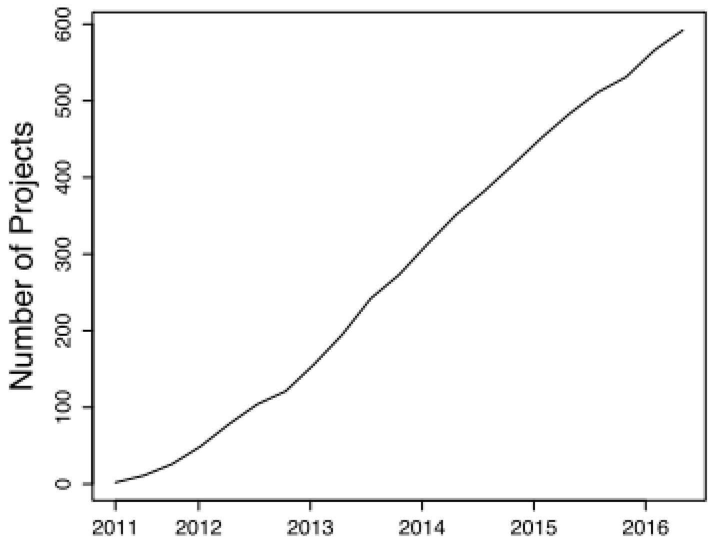
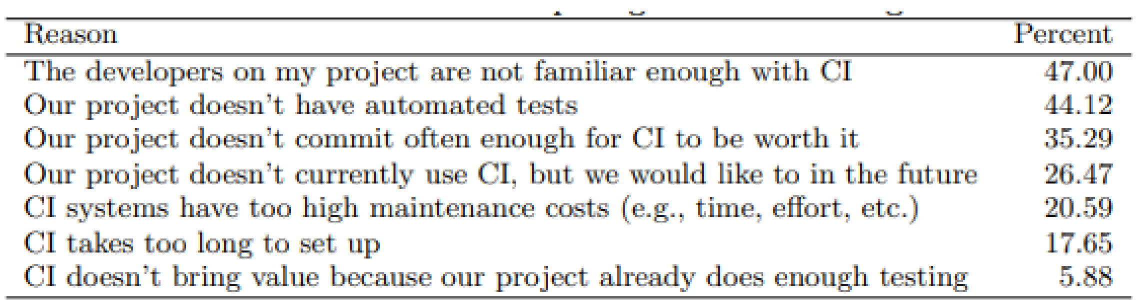


Fig 3. Number of Projects Using CI Over Time [16]

Of the 59.73% of surveyed projects that were found not to be utilizing a pipeline for automated deployments, the developers of these projects were asked why they were not utilizing some form of Continuous Integration. With 47% of respondents, the number one reason was that the other developers on the project were not familiar enough with Continous Integration. This response showcases that the barrier to entry is incredibly high for new developers looking to create some form of a pipeline. With 20.59% of respondents, the fifth-highest reason cited high maintenance costs (e.g., time, effort, etc.) [16]. These responses are in line with the motivations for developing Banquet, which seeks to reduce entry barriers and maintenance costs.



Reason	Percent
The developers on my project are not familiar enough with CI	47.00
Our project doesn't have automated tests	44.12
Our project doesn't commit often enough for CI to be worth it	35.29
Our project doesn't currently use CI, but we would like to in the future	26.47
CI systems have too high maintenance costs (e.g., time, effort, etc.)	20.59
CI takes too long to set up	17.65
CI doesn't bring value because our project already does enough testing	5.88

Fig 4. Reasons Developers Gave For Not Using CI [16]

The most studied pipeline tools in industry studies are GitHub, Subversion, and Jenkins. In a study of 3,952 industry research papers from IEEE, ACM, ScienceDirect, Wiley, Scopus, and Springer, filtered to 69 of the most relevant papers, the findings show that 11 used or discussed Jenkins, six used or discussed GitHub, and six used or discussed Subversion. Subversion and GitHub are version control systems which allow for projects to be stored and versioned for use by multiple developers or a company [3]. Jenkins is an open-source pipeline service that promotes Continuous Integration.

Jenkins, the "leading open source automation server"[14] released in 2011, is one of the most popular CI services on GitHub with 18.7k stars at the time of this writing. It is built with Java and offers more than 1,700 community plugins to support the automation of a multitude of software [14]. Jenkins is often used in academics, being free and open-source, and has the highest number of questions asked on StackOverflow of Continuous Integration tools [17]. From the Jenkins Documentation, Jenkins allows users to create a "Jenkins Pipeline" which is a "suite of plugins which supports implementing and integrating continuous delivery pipelines into Jenkins" [18]. Jenkins utilizes what is known as a "Jenkinsfile" which is a text configuration file that lays the foundation of a

Jenkins Pipeline [18]. Thanks to the numerous community plugins, Jenkins currently integrates with Blue Ocean, Amazon AWS, GitHub, and many other services. Jenkins requires the creation of scripts for individual steps throughout the process of creating a pipeline, and doing so can be a complicated process [19]. Jenkins is often used in tandem with another tool, Ansible, to offer Continuous Delivery.

Ansible is an automation tool developed by RedHat, used to deploy applications and other types of software. Ansible deployments are described using YAML files, which are static configuration files that describe the build process much like a Jenkinsfile. Jenkins alone does offer CD as well as CI, but scalability and versioning are a concern when using this service alone [19].

Travis CI is another open-source CI service, founded in 2011, and the most widely used on GitHub. Out of 31 million users on GitHub, 158,446 utilized Travis CI in 2019, and out of 2.1 million organizations on GitHub, 106,013 utilized it [20]. These numbers are significant, as only approximately 40.27% of GitHub projects incorporate some form of pipeline into their project [16]. The most prominent subsection of users of Travis CI is corporate institutions, including Apache, Elastic Search, Mozilla Firefox, and Microsoft [20]. The two most used features of Travis CI are testing and building applications in a pipeline. Travis CI builds are described using YAML files and are typically set up utilizing GitHub Actions or webhooks (similar to Jenkins) [21].

As a newcomer to the CI services market, GitHub Actions is beginning to see some interesting usage. GitHub Actions was introduced in November 2019 by GitHub to allow the automation of tasks based on various triggers such as commits, pull requests, issues, comments, etc. [22]. GitHub Actions also use YAML files to describe the build process, known as a "Workflow". Developers can create Actions by writing custom code or by gathering existing ones from the GitHub Marketplace. The most used feature of GitHub Actions is Continuous Integration with 27.12% of usage being attributed to this category [22]. 12.29% was attributed to "Deployment" which is much higher than the use cases of the other tools reviewed. At 4.66% of the actions performed, "Testing" was a much smaller portion of actions than the other tools reviewed [22]. This decrease showcases a market movement towards CICD from the traditional use of pipelines: testing and building applications.

In these existing tools, developers find complications in employing CI practices such as deployment pipelines. In "Problems, causes, and solutions when adopting continuous delivery - a systematic literature review", 40 problems were identified with adopting delivery pipelines. The most egregious of these problems were in the category of "system design," meaning the pipeline development process. The study found seven main categories of problems with the adoption of CICD: build design, system design, integration, testing, release, human and organizational, and resources. In build design, build systems were often shown to be inflexible and could not be modified [23]. "Complex builds are difficult to modify and significant effort can be needed to maintain them. Complex builds can cause builds to be broken more often" [23]. In resource problems, "effort" was listed as the largest resource issue. This issue had two meanings. First, a weakly developed pipeline required constant effort to be fixed. Second, the initial effort required to set up the system and monitor the results was too high when adopting a pipeline. “Continually monitoring and nursing these builds has a severe impact on velocity early on in the process” [23].

Developers also feel that they are missing features from these existing tools. From a survey of 691 developers from over 30 countries, it was found that 52% of respondents felt that they needed “easier configuration of CI servers or services”, 38% needed “better tool integration”, and 37% needed “better container/virtualization support” [11]. Developers are seeking a tool that is easier to use, configurable, and customizable. The current needs of this market are not being met by existing tools.

1.4.2 Market Needs

As the Cloud Computing industry continues to grow into the future, the issues that developers and organizations experience with the current CICD tools will become magnified. Automation is becoming increasingly important to developers and organizations alike, forcing a review of the current technologies used to build automation for applications of all kinds. Likewise, as Software as a Service becomes more commonplace and additional models are built on top of SaaS, specific tooling developed for this model will become increasingly valuable. Customization and flexibility will be at the forefront of what the market desires in automation tools such as deployment pipelines. This need will require existing technologies to evolve substantially or new technologies to come into the market to fill this gap. Automation will continue to advance and developers will find or create the tools they need to succeed. In all size categories of the tech industry, better methods for creating deployment pipelines and automating software as a service will become more attractive over time.

For example, enterprise-level architecture is currently undergoing a culture shift towards Agile production and automation, which work efficiently jointly. “Adaptation of Agile practices enables flexibility… which is attracted by software development companies… Implementation of CICD pipeline on agile has enabled fast delivery of software and increase in productivity.” [24]. “Enterprise Architecture” is an established discipline that many software development professionals and academics closely monitor. It is defined as the “fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution” [25]. Increasing efficiency in telecommunication and aspects of globalization are forcing large companies to adapt rapidly and increase acceleration [25]. This shift means that social techniques such as Agile will continue to gain momentum, as well as technical ones such as Software as a Service (and other Cloud Computing models) and deployment pipelines.

Specifically within enterprise-level architecture, performance is more of a concern than it may be in other organizations. To achieve a higher level of performance and to test deployments, the typical system requires that changes roll out in two levels. The first level is to deploy to a test-bench environment. The second is to deploy to the completed production environment. These levels allow for performance to be monitored, outstanding bugs to be recorded, and, most importantly: zero downtime for users. There are many different types of this style of deployment. Three of the most common styles utilized by enterprise-level organizations are Blue-Green Deployment, Canary Deployment, and Rolling Deployment [26].

Blue-Green Deployment is a container-based deployment methodology that involves replacing an existing environment container with a newer image [26]. This type of deployment is notable to this thesis as the process of building Banquet included this methodology. Banquet can quickly create many copy containers, allowing for seamless switches between Blue-Green containers. This method is also efficient for load balancing, as various environments can be switched out in tandem.

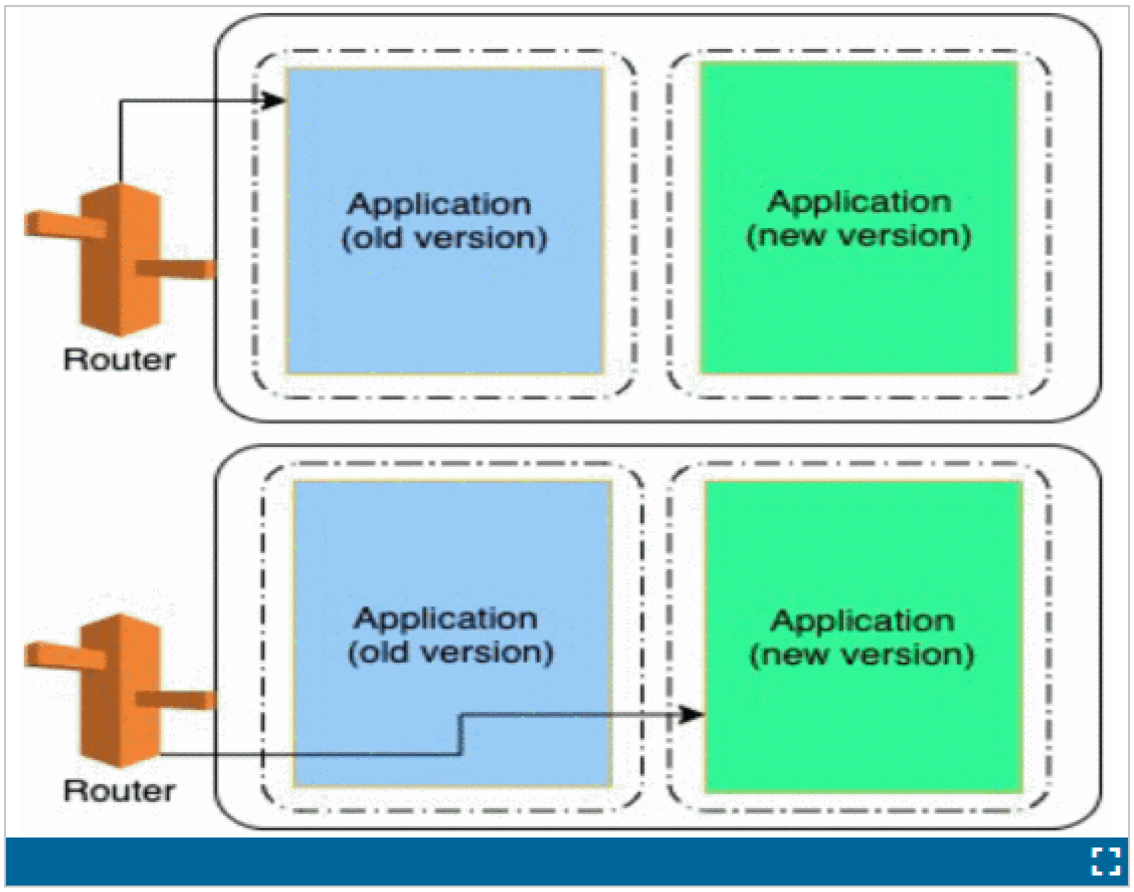


Fig 5. A Typical Blue-Green Deployment Diagram [26]

Likewise, Canary Deployments involve a switch between containers, but rather than all traffic moving to the new container at once, only specific populations of users are switched to new containers in phases until all users are on the new container [26]. Through this approach, there is additional time for version approval by users and for feedback to be received by developers before all users begin using the new container. Rolling Deployment, in contrast, involves the manual change of groups of users to the new environment before all users switch. Rolling Deployment typically does not involve containers [26]. In general, these zero downtime methods of deployment are highly advantageous to the efficiency of deployment pipelines. They also fulfill the increasing flexibility and speed requirements of enterprise Cloud Computing and Software as a Service.

Mid-level organizations are taking on many requirements identical to those experienced by enterprise-level organizations. However, unlike many of the larger organizations, mid-level companies are receiving pushback from decision-makers due to the knowledge gap created by the increasing complexity of Cloud Computing topics. “Since for firms, SaaS is a relatively new model of sourcing information and communication technology (ICT) services, they are struggling in their SaaS related decision-making concerning whether they should adopt SaaS and, if so, how they can gain from it more benefits and a positive impact on firm performance” [27]. As researchers perform more studies in this field and large companies continue to adopt these practices, the SaaS industry will likely grow in the mid-level organization size category as well. However, this process can be sped up by increasing the usability of existing cloud technologies and making Software as a Service easier to implement across the board.

Successful solo developers and smaller nascent (startup) organizations are often pioneers of innovation in software development and typically bring brand-new products and services to the market [28]. In corroboration, findings show that most startup organizations fail because the startup followed a traditional product development path rather than an innovative one [28]. From, "Early-Stage Software Startups: Main Challenges and Possible Answers," one of the critical solutions for success in nascent organizations is "Go Up to the Cloud". Cloud services such as Platform as a Service and Infrastructure as a Service are one path to success for startups, as they provide enterprise-level Cloud Computing and scalability without the traditional costs associated with considerable backend software [29]. Tools such as Google Cloud, Amazon AWS, Microsoft Azure, and Digital Ocean offer the resources for developers to build impactful backend services.

Furthermore, new tools such as Google's Firebase, AWS Amplify, Supabase, and more, offer complete prebuilt backends for developers to incorporate. The challenge with these tools is that pricing can often be a concern. Cloud pricing is one of the most critical topics in Cloud Computing, as it is impractical for Cloud Service Providers to offer personalized pricing to suit each customer's need [30].

Content Management Systems (CMS) are services that allow developers (and occasionally customers) to edit the deployed content of an application. Companies often use them to create Software as a Service applications. The issue with CMS is that they are rigid. They require applications use the CMS at the start during development or for significant refactoring to take place for an application to utilize the service properly. They also do not scale and as the product evolves, it becomes hard to adapt the tool to business needs. While this type of tool is efficient at creating Minimum Viable Products (MVPs), there does not currently exist any dynamic content management tool for SaaS apps. This content injection is the novel aspect of Banquet as a tool.

Both small and mid-level organizations are vital to the cloud market in its entirety. They account for 95% of businesses and approximately 60% of employment in the private sector [31]. Due to the importance of these types of companies in the market, it is equally essential that these companies adopt Cloud Computing. The benefits of Cloud Computing to small and mid-level organizations are numerous. Cloud Computing allows smaller organizations to compete with enterprise architecture without significant infrastructure investment [31]. It eliminates the need for an up-front investment in both hardware and software. It also offers burst opportunities for software where the need for resources will vary markedly over time [31]. These advantages make small to mid-level organizations not only the most typical creator of SaaS applications, but also potential customers of them as well. These types of SaaS creators will often look for the most cost-effective and easiest to use technologies available.

Based on existing technologies and market demands, the way is paved for innovation in pipeline development for SaaS applications. The problems of existing pipeline tools are becoming magnified due to the increasing demand for software automation. Developers within all levels of size category are in need of efficient, scalable, and relatively simple tools to meet these demands. The remainder of this thesis will detail a proposed solution in the form of the software Banquet, which offers the ability to create elaborate apps for many companies.

1.5 Thesis Organization

The organization of the remaining chapters in this thesis is as follows: Chapter two enumerates the development process of the Banquet application, describing the process of creation, the technologies used, unique features, licensing, and included tests. Chapter three gives example use cases for the software in every size category of software development and gives results of tests run on the Banquet software compared to existing CI tools. Chapter four describes the project's predicted future, including possible technology integrations, shortcomings of the existing software, and the conclusion of the thesis.

Chapter 2

APPLICATION DEVELOPMENT

Banquet began as a small hobby project meant to automate the often encountered intricate tasks involved with building and deploying a SaaS application by a solo developer for multiple companies. It began as a collection of Bash scripts that automatically answered prompts from a collection of existing CLI tools. Utilizing the Docker CLI, Google Cloud CLI, and NPM, the initial version of Banquet prompted users for information about their credentials and application. It automatically ran these existing tools to deploy an application to Google's Firebase. This structure was rigid, OS-dependent, and not very user-friendly. The redesign of the application was built with flexibility in mind to support customizable automation. Focusing on abstraction and test-driven development, the new version of Banquet was built with Go to solve the existing problems of CICD. The base process for typical CICD that Banquet performs is as follows: An application is pulled from GitHub, custom CSS is added to the project, custom TypeScript is added to the project, the application is built using NPM, the application is built using Capacitor, the application is containerized using Go-Docker, and finally the container is deployed locally on the host system. Each stage of this process is not tightly coupled with the others, allowing the possibility for any stage to be replaced with a different technology, method, or choice, without impacting the rest of the process.

The name “Banquet” comes from the idea of what the application provides to end-users. A banquet is defined as an elaborate and formal meal for many people; Likewise, this tool provides elaborate and formal applications for many companies. The name of the tool and the themed names of internal functions and classes are meant to simplify the idea of what the application does. For example, when a developer uses Banquet to deploy an application, they are serving a “dish” to a client. The containerization process is known as “plating”, to showcase that while the application may be different, the container it runs on is the same as others, much like different foods on similar plates. When a developer calls the REST API, they create “dishes” in the “kitchen” before they will be served to the client. While these metaphors may be silly or seen as indecorous, they play a part in one of the overarching goals of the application. Banquet seeks to lower the bar to entry for pipeline development and SaaS creation, which could be aided by providing a simple visual to users or developers as to what actions they are performing and what outcome these actions will have.

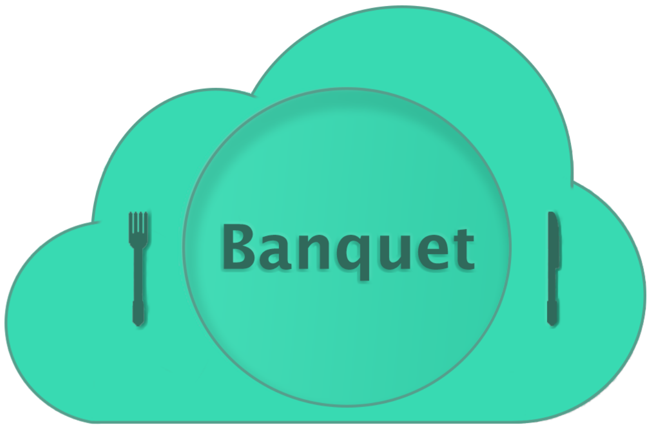
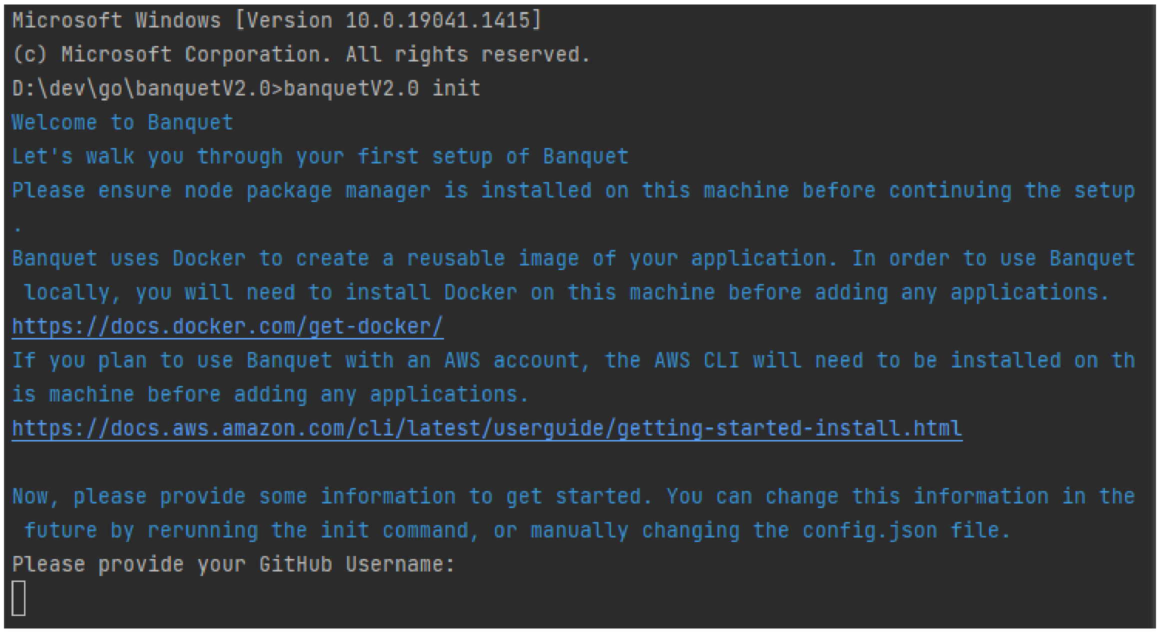


Fig 6. The Banquet Logo

2.1 Interface & Commands

There are two ways to interface with the Banquet software: the CLI or the REST API. Banquet provides an executable that can run on any Windows or Linux device. It can also be added to the user's system path to call the 'banquet' keyword from any command line on the host machine. The Command-Line Interface allows developers to run Banquet on their host machine and answer a series of prompts used to describe their credentials, application, styles, and deployment settings.



```
Microsoft Windows [Version 10.0.19041.1415]
(c) Microsoft Corporation. All rights reserved.
D:\dev\go\banquetV2.0>banquetV2.0 init
Welcome to Banquet
Let's walk you through your first setup of Banquet
Please ensure node package manager is installed on this machine before continuing the setup
.
Banquet uses Docker to create a reusable image of your application. In order to use Banquet
 locally, you will need to install Docker on this machine before adding any applications.
https://docs.docker.com/get-docker/
If you plan to use Banquet with an AWS account, the AWS CLI will need to be installed on th
is machine before adding any applications.
https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

Now, please provide some information to get started. You can change this information in the
 future by rerunning the init command, or manually changing the config.json file.
Please provide your GitHub Username:
```

Fig 7. Banquet Initialization Through CLI

The CLI uses a standard library explicitly created for Banquet that includes functions such as the UserInput function, which captures user input response to a prompt within the command line, the PrintPositve and PrintNegative functions, which display ANSI escape code color responses, and the CheckForError function, which logs errors that the program encounters. This library appears throughout the application, but it finds most of its use in the main program, which runs the CLI.

Following the ‘banquet’ command, users can enter two keywords to travel down two different operation paths. When the application is run for the first time or to change user credentials, the ‘init’ command has two options. Banquet begins the initialization process when no keyword follows ‘init’. This command informs the user of current dependencies that Banquet needs to function fully, then prompts the user for their GitHub username and preferred hosting location. This information is written locally to the ‘config.json’ file, also allowing users to change these credentials through a text editor or alternative program. Banquet currently has two tight dependencies. The first is Node Package Manager (NPM), which builds JavaScript applications through the NPM config. For the second requirement, the host machine must be running the Docker Daemon at the time of containerization. The daemon allows for Docker containers to be created. Banquet downloads the required container images at runtime if they do not previously exist.

Running the ‘kitchen’ keyword following ‘init’, will start an internal server on port 8080 of the host machine. This server will expose the REST API endpoints on this port. The server, known as the “kitchen”, provides access to the existing dishes that users can serve. REST stands for Representational State Transfer and is a software architectural style and messaging pattern commonly used across the web. The main features of REST are that it is stateless, uniform, and cacheable [32]. API stands for Application Programming Interface. A REST API is often used with HTTP to provide usable and flexible endpoints for users [33].

Currently, there are four endpoints that users can access when this server is running. The ‘/api’ GET endpoint provides basic information about the server. The ‘/api/returnMenu’ GET endpoint exposes the GetDishes function, which returns a JSON structure of all of the stored documents on existing applications. The ‘/api/addCourse’ POST endpoint receives a JSON structured ‘dish’ object that includes all needed information on application details, known as a courseRequest. It exposes the AddDish function, which will build, deploy, and write the details of the proposed application to a JSON file called ‘menu.json’. The ‘/api/removeCourse/{id}’ DELETE endpoint receives the ID of an existing dish, causing Banquet to attempt to destroy any existing containers with that ID, delete any build files associated with it, and finally remove that dish's information and ID from the 'menu.json' file. These endpoints provide the full functionality of Banquet, other than initialization, which must be done by the command line (as the kitchen has to be initialized first to begin).

Alternatively, providing the 'dish' keyword following 'banquet' will provide the user with four options to choose. Users can either add, remove, see all dishes, or see a specific dish with this command. The functions provided to the user here through the CLI are the same functions provided through the REST API, which is a direct benefit of test-driven development and abstraction. For these dish operations, the CLI will walk the user through each step of the retrieval from GitHub, the build with NPM, and the subsequent containerization and deployment. If the user is building a copy of an existing project built with Banquet, they can specify this when adding a dish. Doing so will drastically reduce the runtime of the 'banquet dish add' command, as package download and compiling can be skipped.

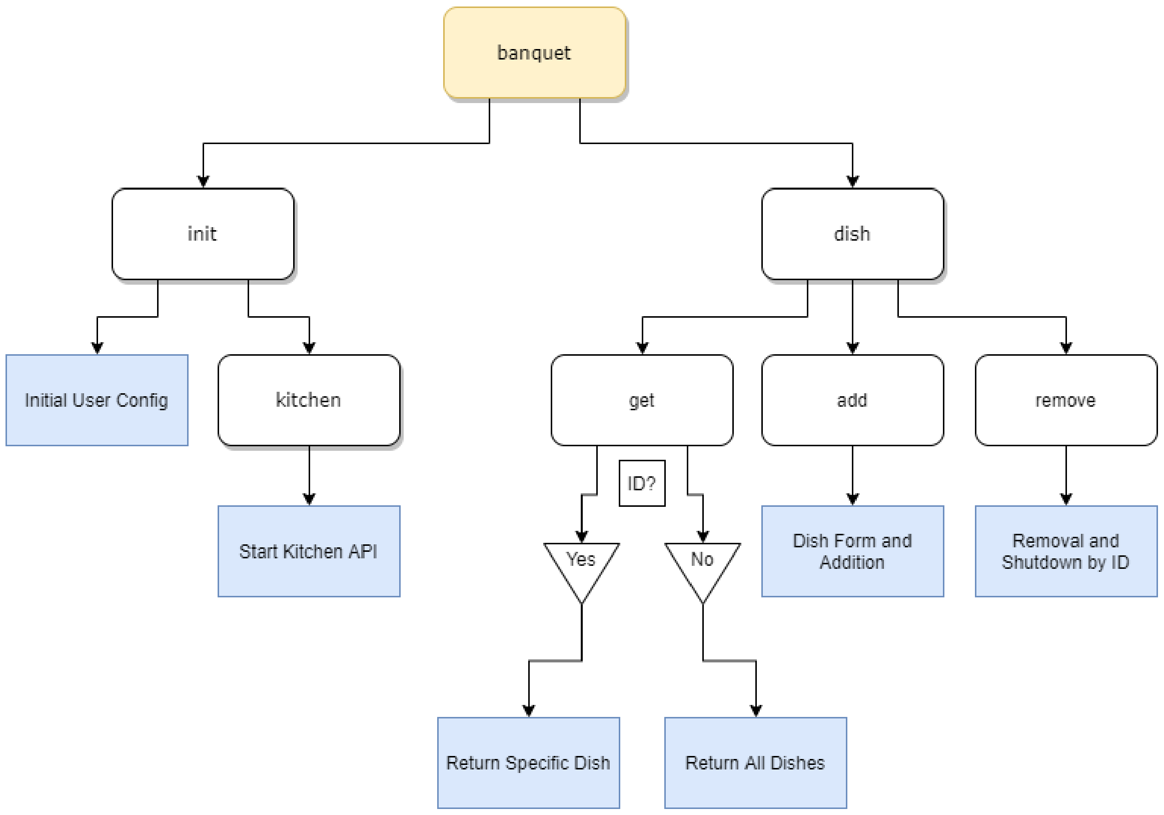
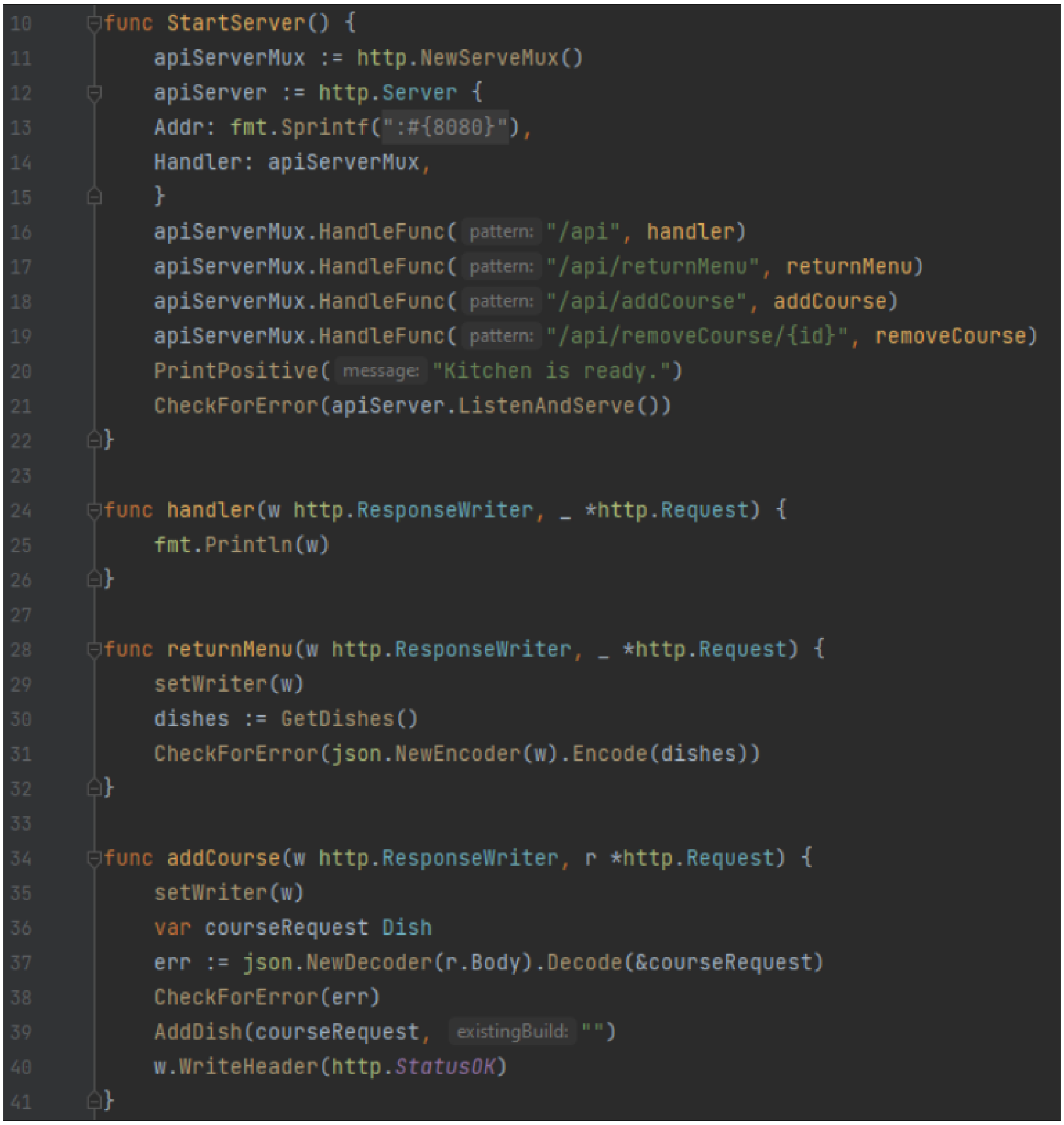


Fig 8. A Diagram of Banquet Command-Line Flow

If the user makes a mistake in entering commands to Banquet, or if they specifically request for it, a help menu will be provided that lists the available commands and some information on what each command does. If any command other than the ‘banquet init’ command runs before a ‘config.json’ file exists, Banquet will display the help menu. This action informs the user that ‘banquet init’ must run before other commands are available.

Banquet is integrated with Ionic, allowing users to change Ionic themes from the CLI to dynamically change an Ionic app. This was done to demonstrate a possible technology integration and show that the versatility of this software allows it to expand to contain hundreds of common tools. This integration is separate from the Capacitor integration described below. Ionic is a framework that provides prebuilt styles and themes for an app, while Capacitor is a tool that converts JavaScript code into native runtime code for Android and iOS applications.

A notable aspect of the “kitchen” is that it can be used to create an alternative to the CLI for less technically inclined users. The REST API endpoints can be called by a web application to provide a portal for managing deployed applications by project managers or clients. Initially, Banquet came with a prebuilt web portal that would show the status of deployed applications and allow management. However, deciding that this could increase confusion on what Banquet does, while also increasing development time, it was removed from the base project. Setting up a web portal in this way allows the automation of SaaS products on a large scale. For example, a solo developer that creates a generic food delivery SaaS application could create one of these web portals, running Banquet in the background. When a customer discovers the site, they could upload their colors, brands, logos, etc, make a payment in the portal, and the Banquet REST API would automatically deploy an instance of the app with their selected cosmetics. This aspect is likely one of the software’s most valuable use case scenarios, along with internal pipeline development.



```
func StartServer() {
	apiServerMux := http.NewServeMux()
	apiServer := http.Server {
	Addr: fmt.Sprintf(":#{8080}"),
	Handler: apiServerMux,
	}
	apiServerMux.HandleFunc( pattern: "/api", handler)
	apiServerMux.HandleFunc( pattern: "/api/returnMenu", returnMenu)
	apiServerMux.HandleFunc( pattern: "/api/addCourse", addCourse)
	apiServerMux.HandleFunc( pattern: "/api/removeCourse/{id}", removeCourse)
	PrintPositive( message: "Kitchen is ready.")
	CheckForError(apiServer.ListenAndServe())
}

func handler(w http.ResponseWriter, _ *http.Request) {
	fmt.Println(w)
}

func returnMenu(w http.ResponseWriter, _ *http.Request) {
	setWriter(w)
	dishes := GetDishes()
	CheckForError(json.NewEncoder(w).Encode(dishes))
}

func addCourse(w http.ResponseWriter, r *http.Request) {
	setWriter(w)
	var courseRequest Dish
	err := json.NewDecoder(r.Body).Decode(&courseRequest)
	CheckForError(err)
	AddDish(courseRequest, existingBuild: "")
	w.WriteHeader(http.StatusOK)
}
```

Fig 9. Server Initialization and Endpoint Options

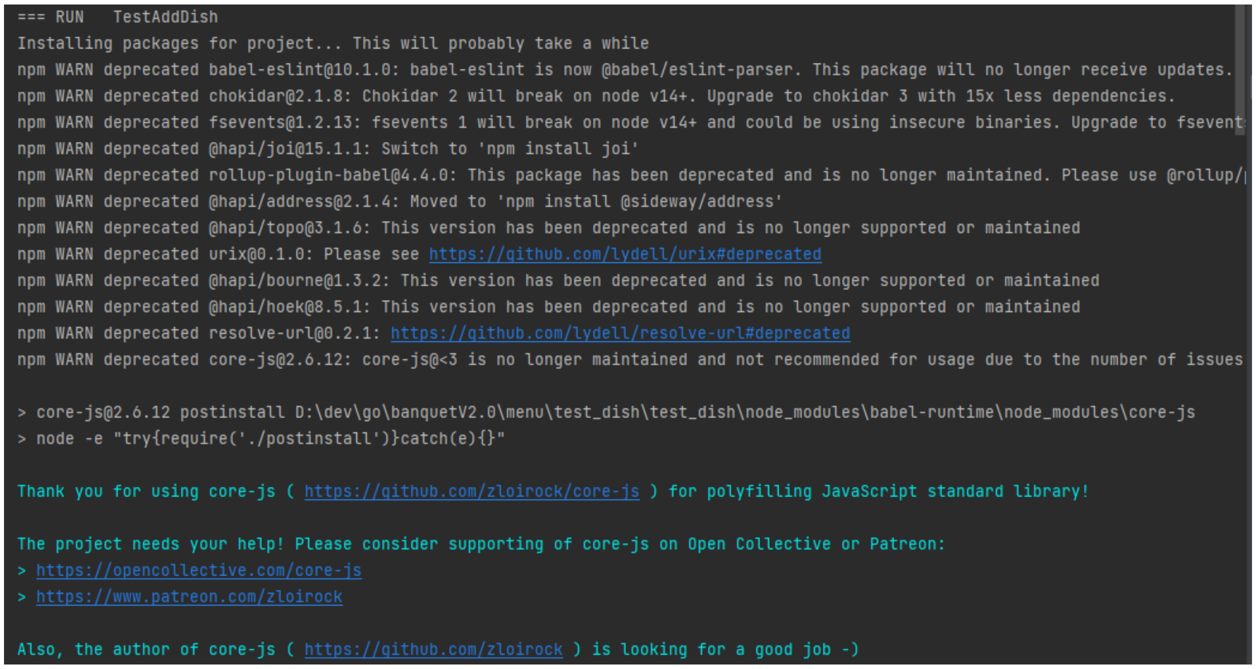
2.2 Package Management

Modern JavaScript applications have the benefit of integrating with thousands of packages, plugins, and libraries that have developed over the many years of the web. The number of packages used in modern JavaScript applications can be hundreds or even thousands. Due to this large number of packages, a tool is needed to manage, download, and compile these packages to be used by the project. The most popular tool is Node Package Manager or NPM.

NPM is the frontend of a large repository of JavaScript-based software packages for both server-side Node.js and client-side JavaScript applications [34]. NPM can refer to three separate entities: the organization, the repository of packages, or the command-line tool. The NPM CLI is often paired with JavaScript frameworks to automatically manage packages needed for these projects and is often vital to the usability of these frameworks. Banquet incorporates the NPM CLI to help download and manage packages for the apps pulled from GitHub. The packages included in modern JS applications can reach gigabytes in size, so these packages are often not stored within version control such as GitHub. To accommodate for this, NPM uses metadata files to store information about the packages needed, the build process, compilation needs, and other project information [34].

To Banquet, the most essential NPM metadata file is the ‘package.json’ file, which holds two notable fields. When Banquet pulls an application from GitHub, the first thing it does is utilize the NPM CLI to run the ‘npm install’ command, which looks at the ‘dependencies’ field of the ‘package.json’ to download all of the needed packages for the project to run successfully from the NPM repository. Banquet will then attempt to build the project, once again using the NPM CLI to run the ‘npm run build’ command, which builds an application for production use by reading the ‘scripts’ field of the ‘package.json’. This field describes how best to run the current application and changes across frameworks. By utilizing the NPM CLI, Banquet can be a one-size-fits-all for building applications, rather than having a different build process for every framework that a user could desire to build [35].

The build time of an application built with Banquet will vary greatly depending on the number of packages that the project has as a part of its dependencies, as package sizes can be substantial, and wait times for downloading will depend on internet connection - downloading and building the application with NPM accounts for approximately 80% of the build time of an application. Fortunately, this process is skippable for applications that have previously been built with Banquet. Banquet can store project files and packages to be used later for copies of projects, reducing the build time to a few seconds.



```
=== RUN   TestAddDish
Installing packages for project... This will probably take a while
npm WARN deprecated babel-eslint@10.1.0: babel-eslint is now @babel/eslint-parser. This package will no longer receive updates.
npm WARN deprecated chokidar@2.1.8: Chokidar 2 will break on node v14+. Upgrade to chokidar 3 with 15x less dependencies.
npm WARN deprecated fsevents@1.2.13: fsevents 1 will break on node v14+ and could be using insecure binaries. Upgrade to fsevent
npm WARN deprecated @hapi/joi@15.1.1: Switch to 'npm install joi'
npm WARN deprecated rollup-plugin-babel@4.4.0: This package has been deprecated and is no longer maintained. Please use @rollup/
npm WARN deprecated @hapi/address@2.1.4: Moved to 'npm install @sideway/address'
npm WARN deprecated @hapi/topo@3.1.6: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
npm WARN deprecated @hapi/bourne@1.3.2: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated @hapi/hoek@8.5.1: This version has been deprecated and is no longer supported or maintained
npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
npm WARN deprecated core-js@2.6.12: core-js@<3 is no longer maintained and not recommended for usage due to the number of issues

> core-js@2.6.12 postinstall D:\dev\go\banquetV2.0\menu\test_dish\test_dish\node_modules\babel-runtime\node_modules\core-js
> node -e "try{require('./postinstall')}catch(e){}"

Thank you for using core-js ( https://github.com/zloirock/core-js ) for polyfilling JavaScript standard library!

The project needs your help! Please consider supporting of core-js on Open Collective or Patreon:
> https://opencollective.com/core-js
> https://www.patreon.com/zloirock

Also, the author of core-js ( https://github.com/zloirock ) is looking for a good job -)
```

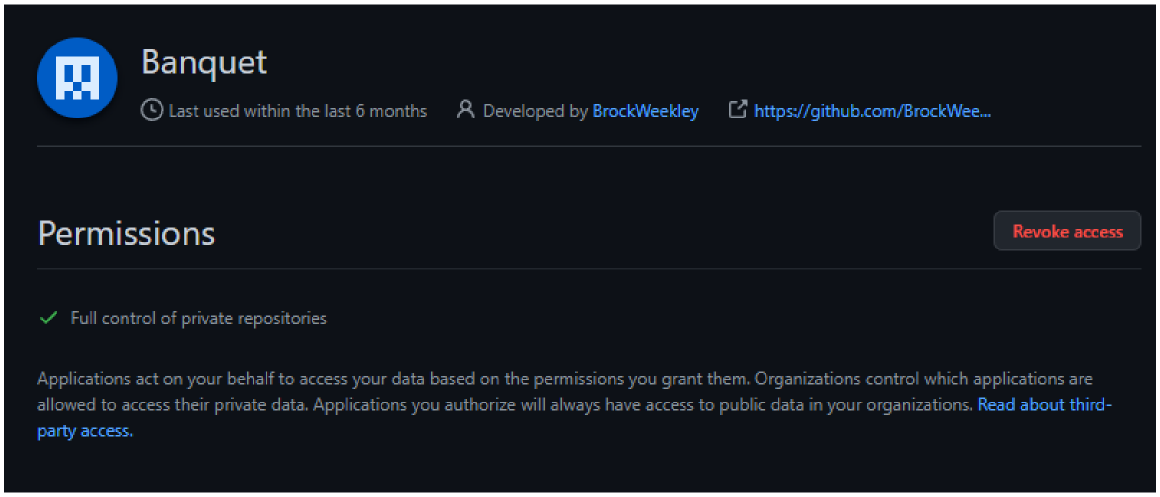
Fig 10. An Example of Packages Downloaded From NPM

NPM is a growing ecosystem that allows developers to build on the previous works of others by adding dependencies. NPM was selected for Banquet as it is the most common package management tool for JavaScript frameworks [34]. However, for applications written without package dependencies, the build time and subsequent pipeline efficiency could be drastically reduced if NPM was removed as the main driver for building applications with Banquet. Overall, the value of the NPM CLI is integral to allowing Banquet to provide enough flexibility to work across different JS platforms.

2.3 GitHub

GitHub is the world’s largest development platform, where millions of developers and companies store their application code in version control. GitHub provides collaborative coding, CICD, security monitoring, client apps, project management, administration, community, and more. It is utilized by over 73 million developers, as well as 84% of Fortune 100 companies [36]. GitHub integrates with git, a CLI code versioning tool that allows developers to check-in their code and push or pull it from/to a repository such as GitHub.

Banquet is a GitHub-approved OAuth application. This means that Banquet is registered with GitHub as an integration, allowing Banquet to access the GitHub API and allowing users to authorize Banquet to have access to private repositories and to integrate with GitHub Actions or Webhooks easily. In the first initialization of Banquet, the user is prompted for a GitHub profile username. This username is used to contact the GitHub API to ask the user profile for access to private repositories. If the user chooses to use Banquet with private repositories, Banquet will provide a link for the user to confirm this interaction with GitHub. Once the user has accepted this use on GitHub, GitHub will provide an access code to the user that they can then provide to Banquet. Banquet will use this code to generate an access key that can be used as long as Banquet continues to have permissions on GitHub, and is stored on the user's config.json file.


Banquet

Last used within the last 6 months · Developed by BrockWeekley · https://github.com/BrockWee...

Permissions

✓ Full control of private repositories

Applications act on your behalf to access your data based on the permissions you grant them. Organizations control which applications are allowed to access their private data. Applications you authorize will always have access to public data in your organizations. Read about third-party access.

Fig 11. Banquet Provided Access to Private Repositories on GitHub

To download a repository when a user adds a dish, Banquet begins by creating a directory to hold all future projects, known as the '/menu/' directory. Banquet then uses the GitHub URL generated by the prompts provided to the user requesting GitHub username, project name, and branch, to make a GET request to GitHub with the generated token. This will return data from GitHub that can be converted into a zip file of the repository code in that branch. This zip file is unzipped by Banquet, allowing access to build the project. This operation relies on the user's internet download speeds, but was found to be a quick operation in practice, even with suboptimal speeds.

Through GitHub Actions and Webhooks, Banquet can be configured to "watch" a specific repository for changes. When a change is made to a repository or specific branch, Banquet can be triggered to redownload, rebuild, and redeploy an application. This is the key feature of Banquet as a pipeline development tool. Continuing with the previous example of a solo developer that creates a generic food delivery SaaS application, when the developer makes a change to their generic application that they would like to deploy to all existing sites that utilize their app, a hook could be placed on that repository which automatically triggers Banquet to redownload, recustomize, and redeploy all of the apps which utilize this generic template.

GitHub is not the only version control system that exists. Unfortunately, to integrate with other version control systems, approval must be received for Banquet at each one individually, as well as integration created within Banquet to deliver to these platforms. This is not an incredibly complicated task, but due to time constraints, GitHub was chosen as the example version control system to show potential future developers how this can be achieved. Users could even utilize proprietary or internal version control systems, as integration with Banquet simply requires an HTTP endpoint to access and receive the project data.

2.4 Containerization

One of the main issues with existing CICD pipelines that Banquet seeks to solve is flexibility. Containerization with Docker is one of the biggest ways that Banquet achieves this goal. Lightweight virtualization technologies have recently been gaining particular notoriety due to the performance enhancements and effective scalability that they offer [37]. Docker is one of the most popular virtualization tools available and was chosen for Banquet due to its well-documented integration with Go. By utilizing Docker, Banquet is able to create standardized environments for any type of app that is built with it, which allows for deployments of every type of app, without the subtle differences in deployment methods [38].

Docker provides the ability to package and run applications in an isolated environment called a container. This isolation is especially important for a CICD tool such as Banquet, as it allows multiple containers to be run on a single host. Without containerization, specific measures would have to be taken to avoid the default running behavior of like-minded JavaScript frameworks that prefer to be run on the same ports. Containers also provide a high level of portability, which allows Banquet to be enjoyed on a developer's local machine, on cloud providers such as AWS, on a physical server, or over a virtual cluster. This provides a fast, cost-effective, and flexible alternative to traditional hosting or other virtualization methods such as hypervisor-based machines [38].

The containerization that Docker provides makes Banquet inherently able to scale very highly horizontally, meaning that a near-limitless number of Software as a Service app instances can be supported if resource requirements are met. Docker has default memory limits that may prevent vertical scaling when using Banquet if additional tooling is not performed. By using Banquet's REST API and a load monitoring tool, a developer could easily set Banquet to create and destroy additional copy containers to handle large traffic loads. Banquet works well with AWS in this way, for example, as Banquet could be run on an EC2 instance, deploying containers managed by Amazon ECS [38].

Docker also inherently provides a security upgrade for running applications. Using containers reduces the attack surface that an application may face and resulting attacks are often limited to a single container, rather than the entire system. Banquet utilizes trusted images and avoids unnecessary privileges within these containers, which overall provides an extra layer of security over running on base hardware [38]. These containers also run with the latest NGINX image, which provides secure proxying.

To begin containerization, Banquet communicates with the Docker daemon immediately after building the project with NPM. Banquet writes a custom Dockerfile that describes how the container image should be created. This includes references to the NGINX and Node Docker images, which are downloaded on the host machine if they are not already available. This image is then used to create a container running the requested app, which is deployed on the user-specified port.

Banquet uses NGINX within Docker images to serve the files from built JavaScript projects. NGINX is a web server that provides the ability to host, reverse proxy, and load balance apps. It is also open-source, highly performant, and quickly outpacing Apache as the most popular web server [39]. Banquet writes a custom 'nginx.conf' file which tells NGINX how to proxy requests, deal with errors, and where to find build files. By default, Banquet only opens ports 80 and 443 to allow HTTP and HTTPS communications to the served application. This prevents accidental openings of sensitive ports, meaning Banquet apps are at less risk of attacks, being accessible only from the web.

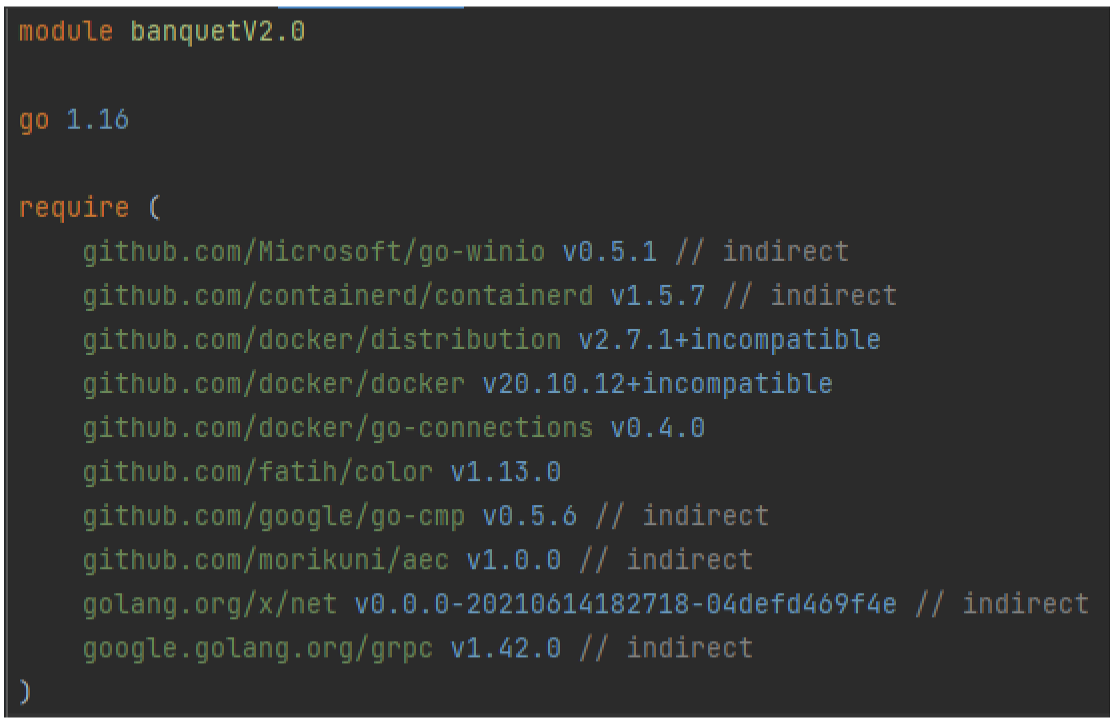
In contrast, when an app is deleted from Banquet, the container is deleted using the original application ID that was associated with the container. If this is the only or original version of the application, the original image is also deleted to free up remaining resources. Additionally, all "dangling" containers are deleted any time an image is removed, which assures that all copies of the image are removed as well, keeping the Docker daemon size low.

Docker as a tool was written in Go, similar to Banquet. This is especially important as the Go library for Docker, GO-Docker, is a fantastic open-source library and batch scheduling tool that provides API access to the Docker daemon. This allows for images and containers to be downloaded from the Docker registry and created on the host machine. Without this library, a custom one would need to be implemented or a batch file would need to be used to interact with the Docker CLI. A library similar to this would be incredibly beneficial to Banquet for working with NPM, as an API would offer more error resistance and efficiency in comparison to utilizing the NPM CLI.

2.5 Golang

The Go language, often called Golang, is a relatively new language at 12 years old, that is syntactically similar to the C language. It was released in 2009 and developed at Google by Robert Griesemer, Ken Thompson, and Rob Pike, with the goal to “Make programming fun again” [40]. Go itself is a compiled, concurrent, garbage-collected, and statically typed language, that is open-source by Google [41]. It was built to address many of the issues that developers had with older programming languages, including: slow builds, poor readability, poor automation, and lack of concurrency. The saying goes that Go was conceived during a 45-minute build of a C program. Go is strongly typed, fast, and includes many memory safety features. This, along with its efficient concurrency, is why it was chosen for Banquet.

Banquet runs on Go 1.16, which emphasizes a shift to Go modules, a dependency management system much like node modules in JavaScript. In fact, Go is very similar to the TypeScript/Node environment that Banquet focuses on. The strong typing of Go is used throughout Banquet to define application requirements, REST API JSON bodies, and to help manage Docker containers. The REST API that Banquet features would be a separate application if Banquet were not written in Go, as the concurrency features allow the kitchen server to run alongside the CLI. The Go community is growing steadily and updates to Go are released regularly. Go has updated to 1.18 since the development of Banquet began and continues to improve [42].



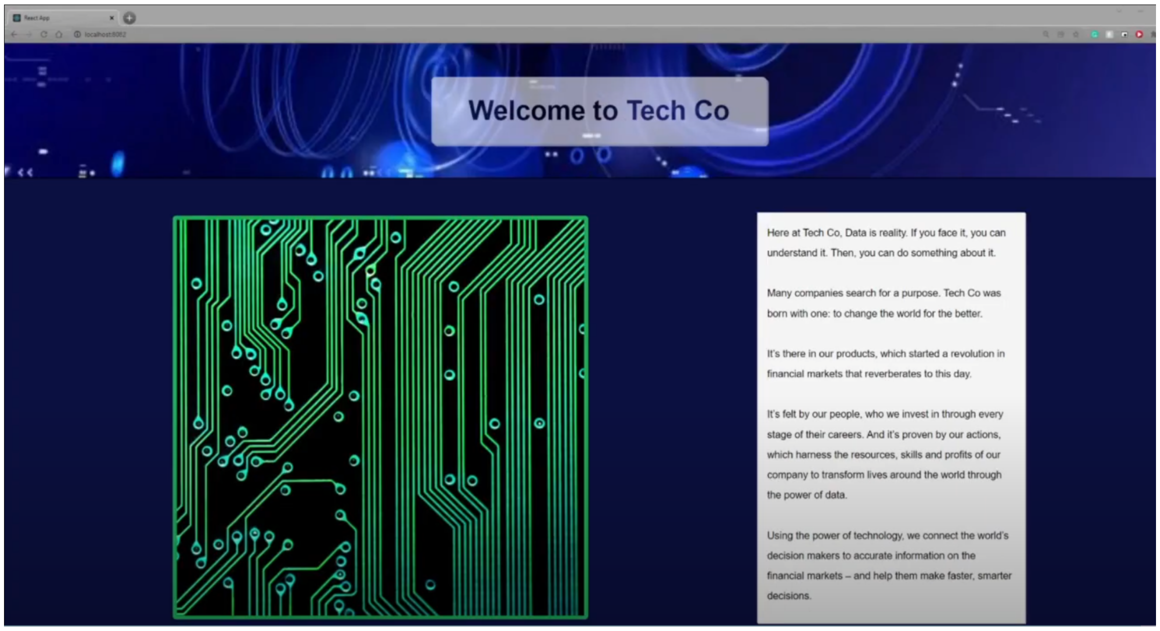
```
module banquetV2.0

go 1.16

require (
    github.com/Microsoft/go-winio v0.5.1 // indirect
    github.com/containerd/containerd v1.5.7 // indirect
    github.com/docker/distribution v2.7.1+incompatible
    github.com/docker/docker v20.10.12+incompatible
    github.com/docker/go-connections v0.4.0
    github.com/fatih/color v1.13.0
    github.com/google/go-cmp v0.5.6 // indirect
    github.com/morikuni/aec v1.0.0 // indirect
    golang.org/x/net v0.0.0-20210614182718-04defd469f4e // indirect
    google.golang.org/grpc v1.42.0 // indirect
)
```

Fig 12. Banquet’s Go Mod File and Dependencies

To run the Banquet application, developers can run the ‘go run’ command on the Banquet package, or they can build the application with the ‘go build’ command on the same package. Go build creates an executable much like a GCC compiler does with C, although it is much quicker. This executable can be run on any similar OS to the host machine that go build was run on. For users of Banquet, the built executable for their OS can simply be downloaded and added to the host machine’s PATH variable, allowing for the ‘banquet’ command to be run from any command line on the machine.

2.6 Code & Style Injection

One of the most novel and important parts of Banquet is its ability to inject style or custom code into any application. When using Banquet to serve a new app, a developer can define the path for a style sheet to be added to the build process of the application. The name of this style sheet will be one defined in the existing application or if the app was built to be used with Banquet, can be defaulted to ‘banquet.css’. This integration allows colors, images, text layout, and any other designs made with CSS to be changed on the fly with any provided style sheet. Furthermore, the default Banquet style sheet can be created directly from the command line, meaning users that wish to change images and colors throughout an app do not need to have any knowledge of CSS. This feature is very important to SaaS application development, as it allows for pipeline automation at the very base level of what SaaS requires: customization.



Here at Tech Co, Data is reality. If you face it, you can understand it. Then, you can do something about it.

Many companies search for a purpose. Tech Co was born with one: to change the world for the better.

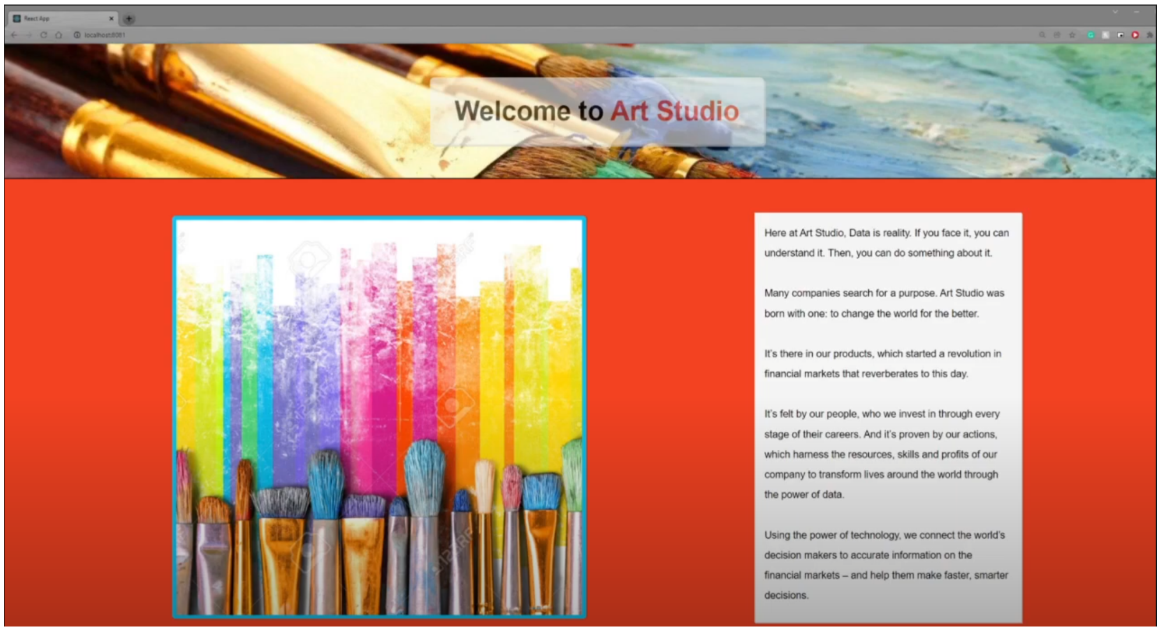
It's there in our products, which started a revolution in financial markets that reverberates to this day.

It's felt by our people, who we invest in through every stage of their careers. And it's proven by our actions, which harness the resources, skills and profits of our company to transform lives around the world through the power of data.

Using the power of technology, we connect the world's decision makers to accurate information on the financial markets – and help them make faster, smarter decisions.

Fig 13. Example 1 of Style Injection

On top of style changes, Banquet is able to inject TypeScript files as well, which allows developers to replace text, functions, or other custom code throughout an application. Users can define the path to a custom code file which will then be injected into the application before it is built. For example, if a static site has a block of text describing the company that is using the app, that text can be changed by a variable in the custom code file. This allows for applications to change information about a site across the app, as well as add custom features specific to one company out of the multiple that employ a SaaS application.


Welcome to Art Studio

Here at Art Studio, Data is reality. If you face it, you can understand it. Then, you can do something about it.

Many companies search for a purpose. Art Studio was born with one: to change the world for the better.

It's there in our products, which started a revolution in financial markets that reverberates to this day.

It's felt by our people, who we invest in through every stage of their careers. And it's proven by our actions, which harness the resources, skills and profits of our company to transform lives around the world through the power of data.

Using the power of technology, we connect the world's decision makers to accurate information on the financial markets – and help them make faster, smarter decisions.

Fig 14. Example 2 of Style Injection and Code Injection

2.7 Mobile Integration

Banquet allows for any application that can be built with the tool to be converted into a mobile application as well. Developers that have built apps with responsive design will be able to have functioning mobile applications created from their web apps. When the application is containerized, the same build code that was created for Docker containers is also used to create a web view of the JavaScript application that can be displayed on either Android or iOS. This is made possible by CapacitorJS being incorporated into the build process.

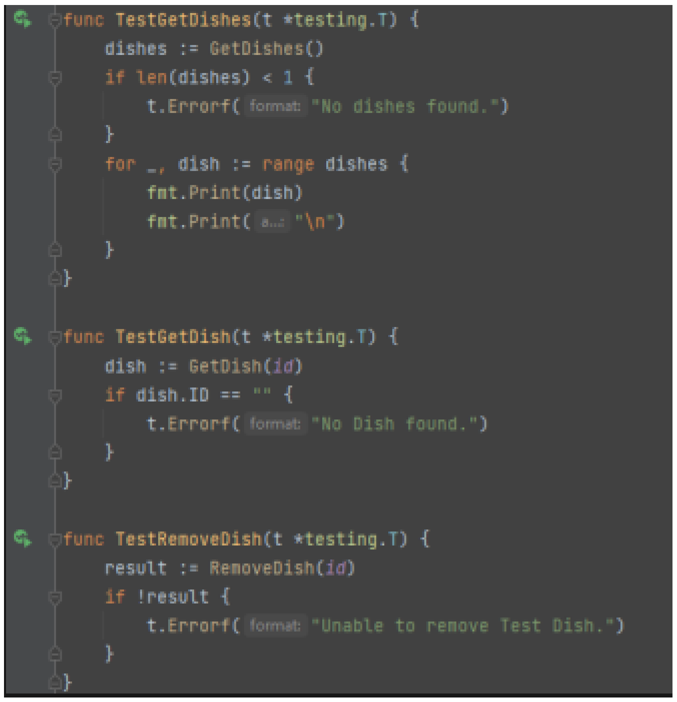
Capacitor is an open-source native runtime for building cross-platform applications. This means that a JavaScript application built for the web can be converted to a mobile app using this tool alone. If Capacitor is selected when building the application with Banquet, Capacitor will be installed locally on the host machine and a platform will be built for Android. iOS applications can currently only be built on MAC OS, which Banquet does not currently support. Users must have the Android SDK installed on the host machine to properly convert JS code into a native runtime for Android. Capacitor also allows for interaction with Native APIs, allowing for native functionality if the developer of the application has previously included it within the app. The Capacitor community is growing, previously called PhoneGap and built by the same team as Ionic [43].

2.8 Test Driven Development

Test Driven Development, also known as test-driven design, or TDD, essentially requires that for each functionality that a programmer creates, they first write unit tests describing and testing that functionality [44]. The code is then subsequently added to make those unit tests pass. This typically forces programmers to think about new aspects of the code that they may not have without TDD, before coding it. It also ensures that developers are testing their code fully. Statistically, it has been shown to drastically increase code quality. Industry studies have shown that programmers using TDD produced code that passed between 18 and 50 percent more external test cases than code created without TDD [44]. TDD was shown to initially reduce productivity among programmers, but as developers became more experienced, productivity increased [44].

Version Two of Banquet was developed with TDD and began with four unit tests. The unit tests that were developed described the key functionality of the tool that was desired. This included: TestAddDish, TestGetDishes, TestGetDish, and TestRemoveDish.

These simple create, read, and delete operations forced the issue of what information would be required from the user of the application in order to create an app, how applications would be identified and stored after creation, and what the most likely use cases for the app would be. It also created an abstraction of code functionality that became increasingly useful for the flexibility and future use of the application.



```go
func TestGetDishes(t *testing.T) {
	dishes := GetDishes()
	if len(dishes) < 1 {
		t.Errorf( format: "No dishes found.")
	}
	for _, dish := range dishes {
		fmt.Print(dish)
		fmt.Print( a...: "\n")
	}
}

func TestGetDish(t *testing.T) {
	dish := GetDish(id)
	if dish.ID == "" {
		t.Errorf( format: "No Dish found.")
	}
}

func TestRemoveDish(t *testing.T) {
	result := RemoveDish(id)
	if !result {
		t.Errorf( format: "Unable to remove Test Dish.")
	}
}
```

Fig 15. Some Banquet Unit Tests

TDD was made possible for this project in part due to Go. Go has its own testing library that is very intuitive and simple to use. Go’s testing package provides support for automated testing of Go packages. It requires a special syntax of included functions and is run with the ‘go test’ command. The library provides methods for passing, failing, or erroring out of functions, as well as benchmarking the functionality of the tool [45]. The syntax was simple and intuitive, leading to the incredible ease of use in developing unit tests for Banquet.

2.9 Open Source License

The full release of Banquet will be made under the open-source MIT license which states: “Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions: The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software” [46]. This license is an extensive use of open-source that allows Banquet to be used by anyone with attribution to the original. This allows Banquet to be used in other open-source projects, proprietary software, or other business ventures.

The reason behind this decision is twofold. One reason is that making the project open-source is the only way to guarantee trust in the program. Any developer or otherwise can review the code to ensure there is no malicious or erroneous code implemented. Banquet needs this increased trust to allow proprietary software to utilize the tool, as well as apps that rely on credentials or other secrets such as API keys.

Second, and most importantly, the growth of the open-source community is incredibly vital to both academia and industry development of software. Knowledge sharing and building upon previous works are integral to the success of software and technology advancements as a whole. Furthermore, the open-source community is growing drastically and usage of open-source projects has skyrocketed. The largest reason for this has been attributed to the ease of customization that is offered by open-source projects [47]. This is central to Banquet, as the flexibility and future integration of many technologies are vital to its success as a tool. The tools chosen for Banquet are just some of the standard choices for pipeline development, but many more exist. Being open-source, the way is open for developers to add more technologies, support for other systems, or integrations to truly make the app customizable for every use case of a company or a developer.

Chapter 3

USE CASES

As examined in previous chapters, issues exist with pipeline development in all size categories of development. Banquet seeks to solve many of the issues faced at each of these stages of organization. This is accomplished in many ways through Banquet, as well as through the possible customization of the tool to fit a specific use case. However, the examples given below are the most likely use cases based on the intentions of developing the tool. This will hopefully serve to show that not only does Banquet solve some of the existing issues of CICD pipelines, but that it also offers new opportunities for SaaS application developers to automate their development and deployment in new ways. Three example use cases are given, but many more exist, and additional work on the tool can go a long way in providing more use cases in the future.

3.1 Individual Developer

Individual developers may find use in Banquet in a variety of ways. Developers may find that Banquet helps increase productivity, build new tools to help manage customers, or introduce them to new technologies that they may be unfamiliar with using. Individual developers will most likely use the CLI portion of Banquet the most, as automation may not be as important when one is not working with other individuals.

Banquet can increase an individual's productivity by removing many of the menial tasks that developers face when creating, building, and deploying applications. By running the 'banquet dish add' command, developers have the possibility to remove the following steps from their workflow: git cloning an existing project, adding new styles or code to a project, writing Dockerfiles, writing NGINX files, building a project with NPM, creating new Docker images, deploying a new container, and building an application for mobile. A process that takes under a minute on average in Banquet would take hours to complete without the tool. For contractors or freelance developers, this increase in productivity directly means more business.

Individual developers may also use Banquet to help manage existing and future customers. Banquet automatically organizes applications that are built with the tool, allowing developers to spin up and take down containers with just a few commands. For a developer with multiple clients, running the 'banquet dish add' command would allow the addition of a new client in just a few minutes when utilizing a SaaS template. To remove an existing client, due to contract end, lack of payment, or other reasons, the developer need only run the 'banquet dish remove {id}' command to end service immediately. Furthermore, the ‘banquet dish get’ command allows a developer to see the status of all existing projects and the various information about each, putting greater control in the developer's hands.

Finally, Banquet uses many tools that new developers may be unfamiliar with using. Developers inexperienced with NPM, Docker, GitHub, REST, or Capacitor will likely find value in using Banquet to become familiar with these technologies. Using Banquet allows developers to use these technologies without fully understanding what they do. However, the documentation of the code, along with clear console outputs of the process, and the simplified description of each technology in the Banquet metaphor offer valuable learning tools to introduce more advanced concepts of the tools being used.

3.2 Startup Company

A startup company will likely find great value in Banquet. The innovation that Banquet provides on the CICD pipeline is integral for new or smaller companies that are looking to have cost-effective, simple tools for deployments. Banquet allows startup companies to integrate with the latest tools, build powerful pipelines, manage teams, and acquire more clients through increased flexibility. The kitchen REST API will likely become more valuable for this size category of developers.

Startup companies often seek an increased number of long-term clients in comparison with individual developers. The benefits of using Banquet are similar in this way to individual developers, allowing startups to manage existing customers and quickly add new ones from a SaaS template. However, Banquet’s ability to redeploy copy applications would also allow companies to update their applications regularly for security, bug fixes, and code improvements, which will likely be done more often with an efficient development team.

Startups will find high value in performant pipelines like those that can be built with Banquet, as this will free up development time to focus on feature requests, code improvements, and client acquisition. An example of pipeline development for a startup using Banquet could include an instance of Banquet running on an AWS EC2 instance. This instance would have an attached web portal for either developers or clients to manage existing instances of applications. This web portal would interact with the Banquet REST API to create and destroy various containers on the EC2 instance. As the company grew, the developers may seek to add load balancing and cluster management tools into Banquet.

Most importantly for startups, Banquet allows for cleaner code through efficient testing pipelines. A startup may set Banquet to watch a GitHub branch for code changes from an EC2, where it will automatically deploy updates to this branch. This branch may be protected to only allow reviewed code to be deployed, removing any need for developers to deploy applications manually. Banquet can also be tooled to run a test suite through NPM very easily. A failure to pass tests may stop Banquet from containerizing and deploying, alerting a development team of existing issues.

Small to medium-sized companies such as startups benefit disproportionately from cloud tools, as they allow them to compete with enterprise companies without the dedication to resources, investment in time, and advanced knowledge required to develop powerful pipelines and cloud management tools. Banquet especially could prove vital in this by decreasing the investment needed in developing a pipeline and allowing SaaS companies to thrive.

3.3 Enterprise Company

Enterprise Architecture often requires significant investment into cloud structures, deployment pipelines, and test automation. Many existing tools designed for building these services have a high learning curve and are inflexible or costly. Banquet solves these issues with intuitive-use functions, a modern build process, and low cost due to high efficiency and scalability. Banquet can be integrated with existing CICD pipeline services to provide an all-encompassing environment.

Previous chapters have shown that existing CICD pipelines are often considered rigid, too hard to integrate, or too costly by developers. The simplicity of Banquet as a tool is offered in the three main commands that perform creates, reads, and deletes of any JavaScript application. This may convince existing enterprise companies to begin developing a pipeline if they did not previously have one. Furthermore, the use of JSON structures and abstracted steps in the build process allows builds to be far more dynamic than those built with YAML build processes. In the Test Results and Comparison section, Banquet is compared with Jenkins and GitLab in terms of performance, outperforming both.

For an enterprise company that employs an existing cloud and pipeline, Banquet may be used to implement some form of testing automation or zero downtime deployments. If one of these companies has built a cloud, but has no pipeline, Banquet can then be used to quickly develop a deployment system that seamlessly integrates with the cloud resources, allowing for horizontal scaling.

3.4 Integration with Existing Clouds

In the process of creating Banquet, integrations for Google Play Store, Amazon AWS, Google CloudRun, and Google Firebase were started. Existing libraries in Go are lacking for all four existing cloud solutions and custom integrations would need to be made to employ the use of these services. Google Play Store is the most likely to see a finalized integration without community support, as the build requirements are sufficiently met by Capacitor, and the Google Play API is a viable option to develop this feature.

Utilizing the Amazon AWS SDK, authentication was implemented under the ‘aws’ branch of the GitHub repository. This feature establishes a connection with AWS through the use of a service account key that is created through the AWS console. With further work, and possibly increased features of the AWS SDK, this could prove to allow Docker containers created with Banquet to automatically be uploaded to a user’s AWS account under the ECS service, thereby avoiding the need for users to download and use the AWS CLI or console.

For both Google Firebase and Google CloudRun, which are essentially treated as the same service by Google, Banquet was successfully used to upload Docker containers to the service through the use of the Google Cloud CLI. This batch process, which is saved under the ‘gcloud’ branch of the GitHub repository, allows users to automatically upload containers to their Google Cloud account by authenticating with the user’s key.json, then deploying without the use of the Google dash. Implementation through an SDK or API was unsuccessful. Furthermore, integration with Google Play was unsuccessful initially due to time constraints. The androidpublisher package from Google was used to establish authentication with Google Play under the ‘playStore’ GitHub branch.

An existing proprietary cloud could be integrated with Banquet through the use of a REST API or other means of communicating Docker container data to the cloud app. It is also possible that the Docker container repository could be helpful in integrating with an existing cloud, as containers could be uploaded to this repository and then downloaded to the proprietary cloud. Further work is needed to test the viability of Banquet with other non-public cloud tools.

3.5 Test Results and Comparisons

In Comparison of Different CICD Tools Integrated with Cloud Platform by Singh et al., the performance of both Jenkins deployments and GitLab deployments were measured. Ten deployments were run on an AWS EC2 VM for Jenkins, GitLab Specific Runner, and GitLab Shared Runner [48]. The results of this are shown in Figure 14:

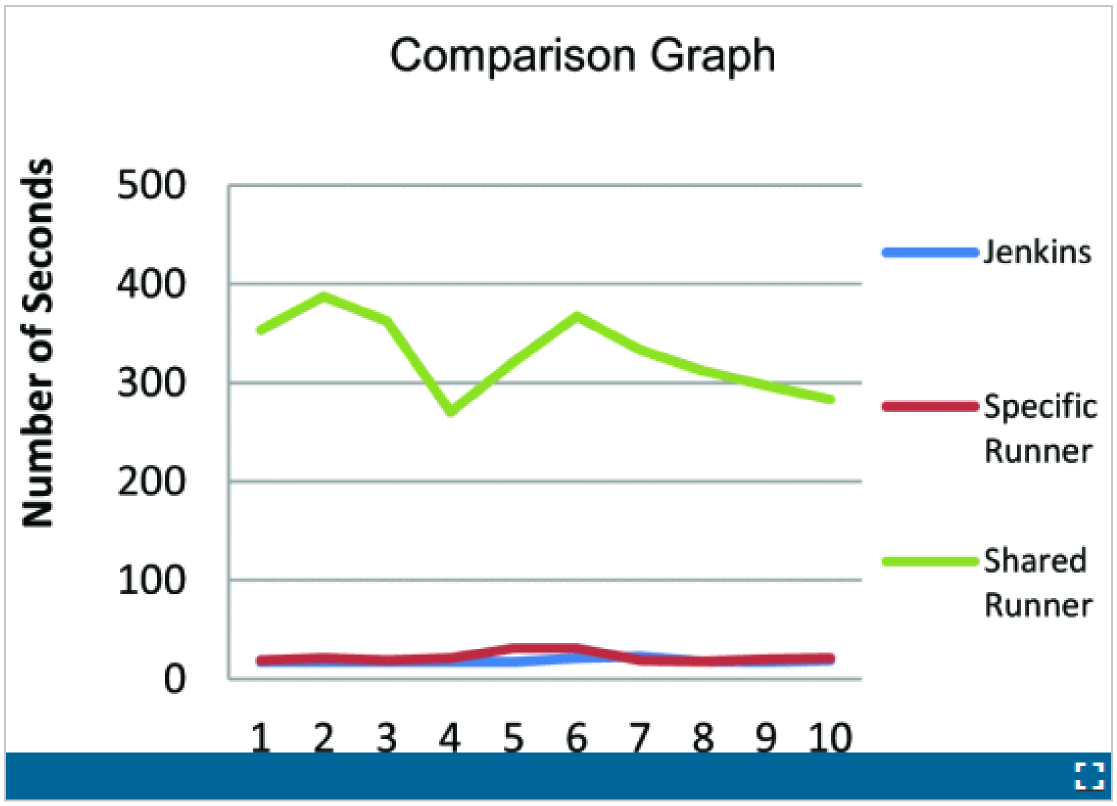


Figure 5.
Deployment time for Jenkins, shared runner and specific runner

Fig 16. Deployment Time for Jenkins, Shared Runner, and Specific Runner [48]

This same test was run with Banquet to compare with existing CICD tools. An instance of Banquet was started on an Amazon EC2 t2.micro, and ten deployments were created of one test application. Using the Linux ‘time’ command, the runtime of the application deployments were tested and recorded on the same Comparison Graph from Figure 14. 30 seconds was removed from each run time in the graph view to account for the exact time used to enter information to the CLI. This time would be negligible when utilizing the REST API instead. The results are shown below in Figure 17:

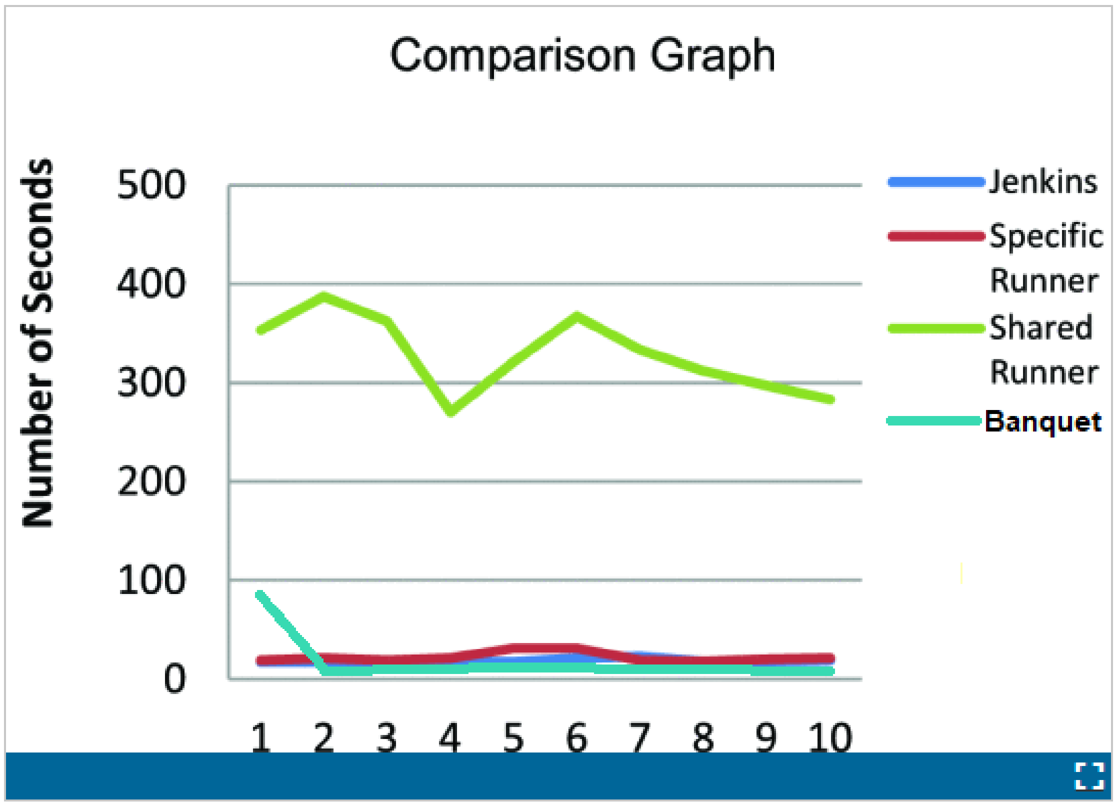
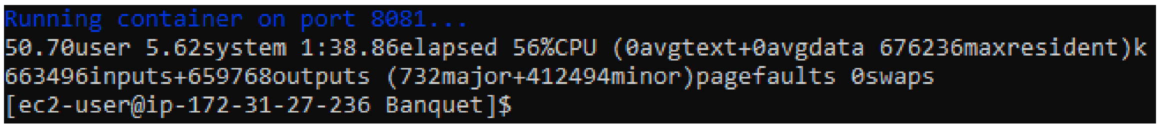


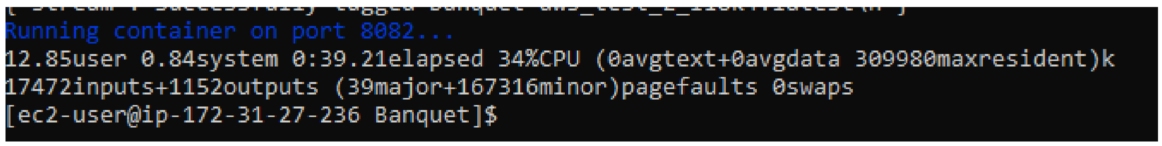
Figure 5.
Deployment time for Jenkins, shared runner, specific runner and Banquet

Fig 17. Deployment time for Jenkins, Shared Runner, Specific Runner, and Banquet



```
Running container on port 8081...
50.70user 5.62system 1:38.86elapsed 56%CPU (0avgtext+0avgdata 676236maxresident)k
663496inputs+659768outputs (732major+412494minor)pagefaults 0swaps
[ec2-user@ip-172-31-27-236 Banquet]$
```

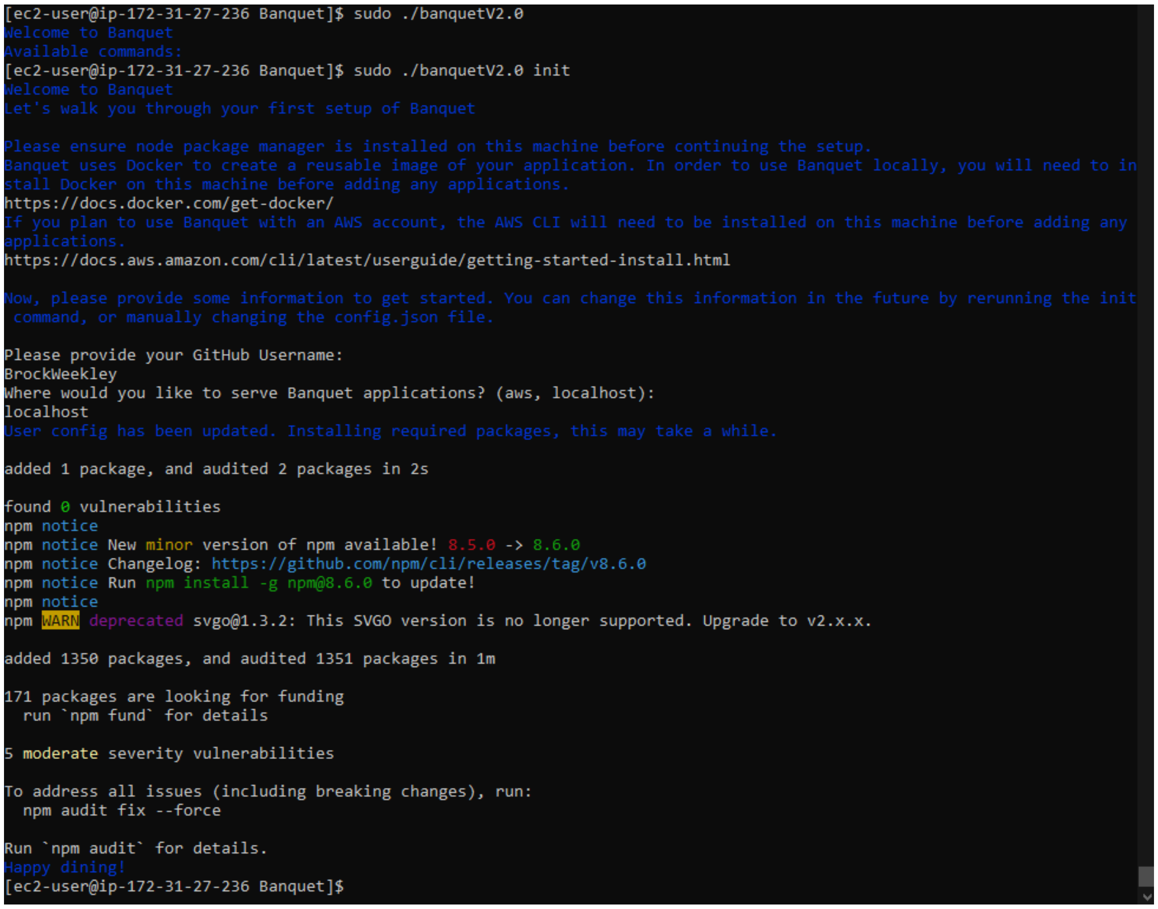
Fig 18. Test Results for Deployment 1



```
Running container on port 8082...
12.85user 0.84system 0:39.21elapsed 34%CPU (0avgtext+0avgdata 309980maxresident)k
17472inputs+1152outputs (39major+167316minor)pagefaults 0swaps
[ec2-user@ip-172-31-27-236 Banquet]$
```

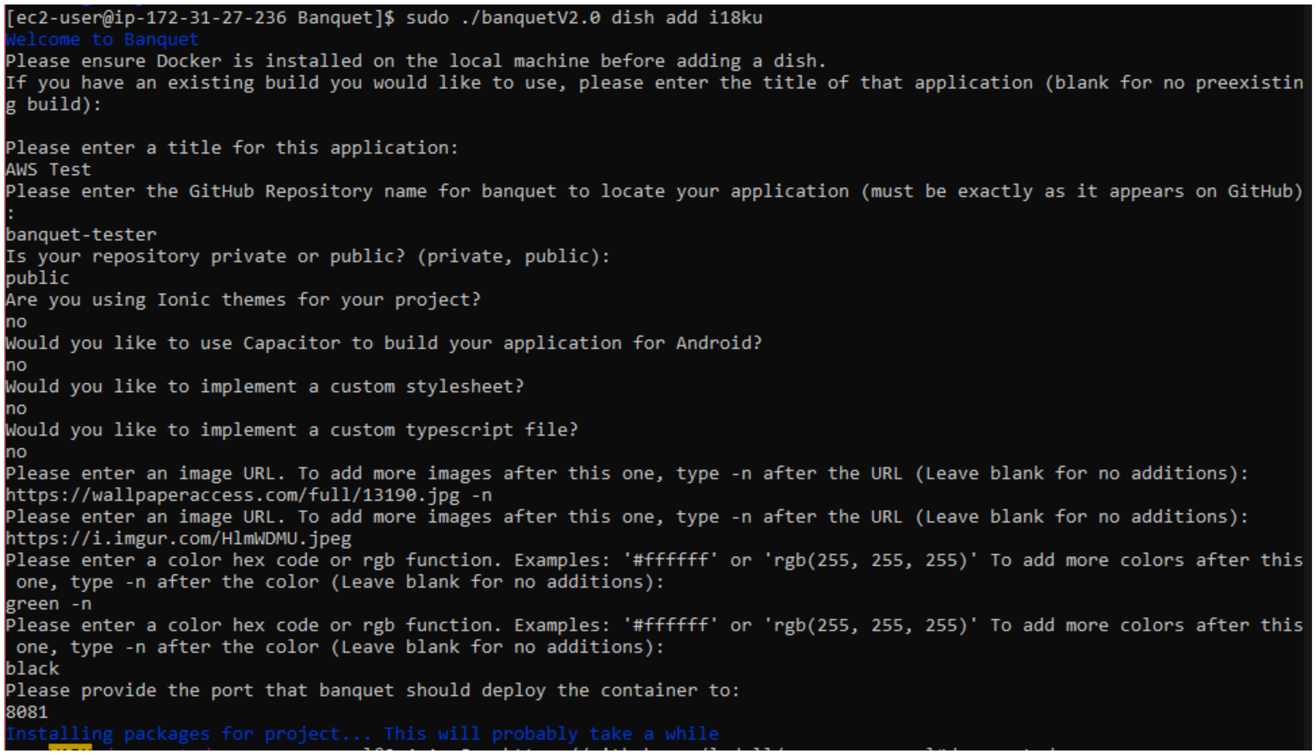
Fig 19. Test Results for Deployment 2

After the initial deployment, which requires building and downloading packages from NPM, Banquet outperformed Jenkins and GitLab in terms of time to deploy. The setup of Banquet for the first time on an AWS server proved to be surprisingly simple. NPM, Docker, and Go were installed on the host instance using Yum. Typically, Go would not be required to be installed, but as this was the first time the executable was built for Linux, the entire project had to be built. The Docker daemon was started using the Docker CLI. Then, Banquet was built using the 'go build' command. This command produced the executable used for all of the tests on this machine. The results of the initialization command are shown below in Figure 20, as well as the CLI usage of the tool and resulting output in Figures 21 and 22 respectively, with the deployed application in Figure 23:



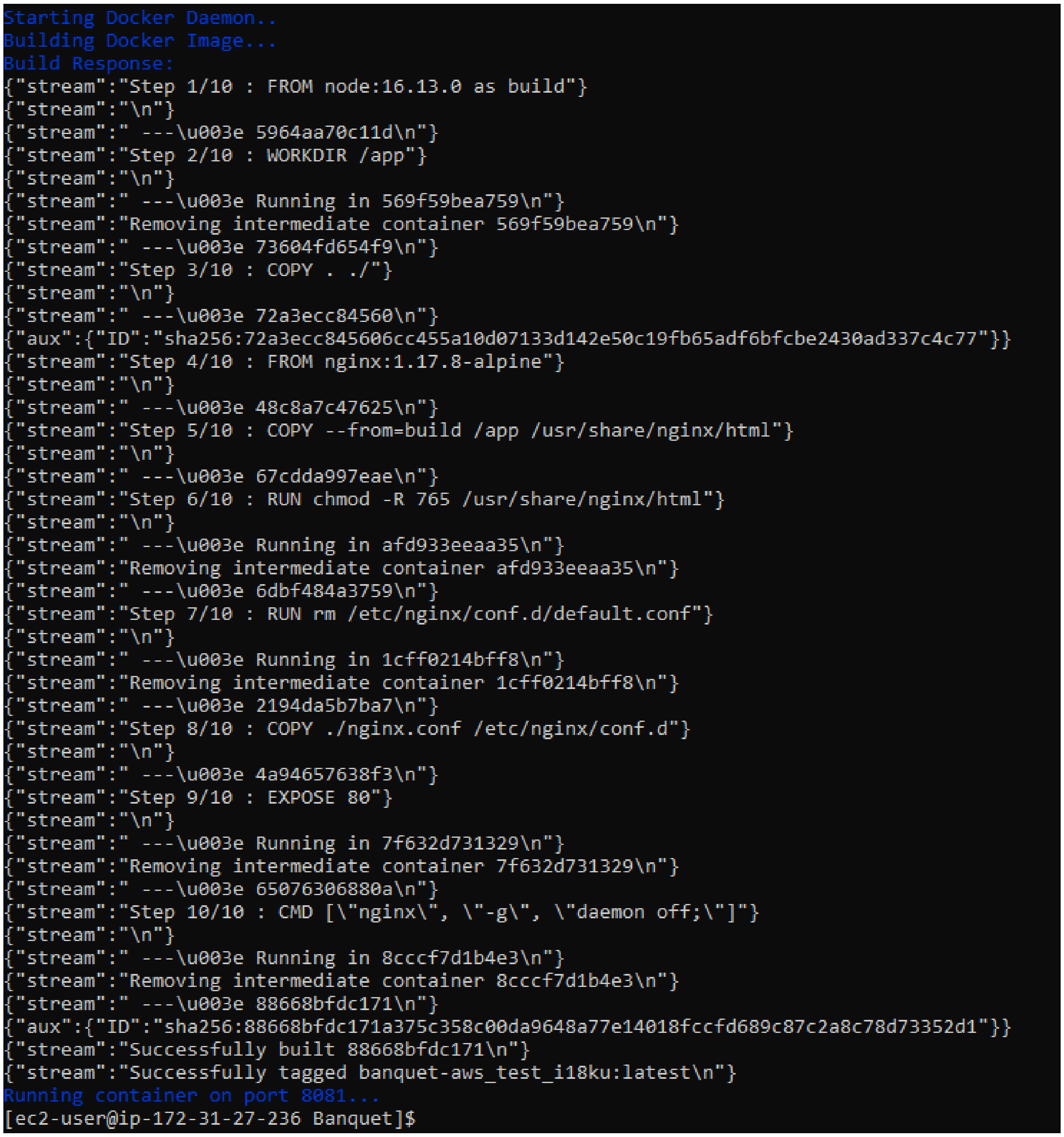
```
[ec2-user@ip-172-31-27-236 Banquet]$ sudo ./banquetV2.0
Welcome to Banquet
Available commands:
[ec2-user@ip-172-31-27-236 Banquet]$ sudo ./banquetV2.0 init
Welcome to Banquet
Let's walk you through your first setup of Banquet

Please ensure node package manager is installed on this machine before continuing the setup.
Banquet uses Docker to create a reusable image of your application. In order to use Banquet locally, you will need to in
stall Docker on this machine before adding any applications.
https://docs.docker.com/get-docker/
If you plan to use Banquet with an AWS account, the AWS CLI will need to be installed on this machine before adding any
applications.
https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html

Now, please provide some information to get started. You can change this information in the future by rerunning the init
 command, or manually changing the config.json file.

Please provide your GitHub Username:
BrockWeekley
Where would you like to serve Banquet applications? (aws, localhost):
localhost
User config has been updated. Installing required packages, this may take a while.

added 1 package, and audited 2 packages in 2s

found 0 vulnerabilities
npm notice
npm notice New minor version of npm available! 8.5.0 -> 8.6.0
npm notice Changelog: https://github.com/npm/cli/releases/tag/v8.6.0
npm notice Run npm install -g npm@8.6.0 to update!
npm notice
npm WARN deprecated svgo@1.3.2: This SVGO version is no longer supported. Upgrade to v2.x.x.

added 1350 packages, and audited 1351 packages in 1m

171 packages are looking for funding
  run `npm fund` for details

5 moderate severity vulnerabilities

To address all issues (including breaking changes), run:
  npm audit fix --force

Run `npm audit` for details.
Happy dining!
[ec2-user@ip-172-31-27-236 Banquet]$
```

Fig 20. Output of Banquet Init Command



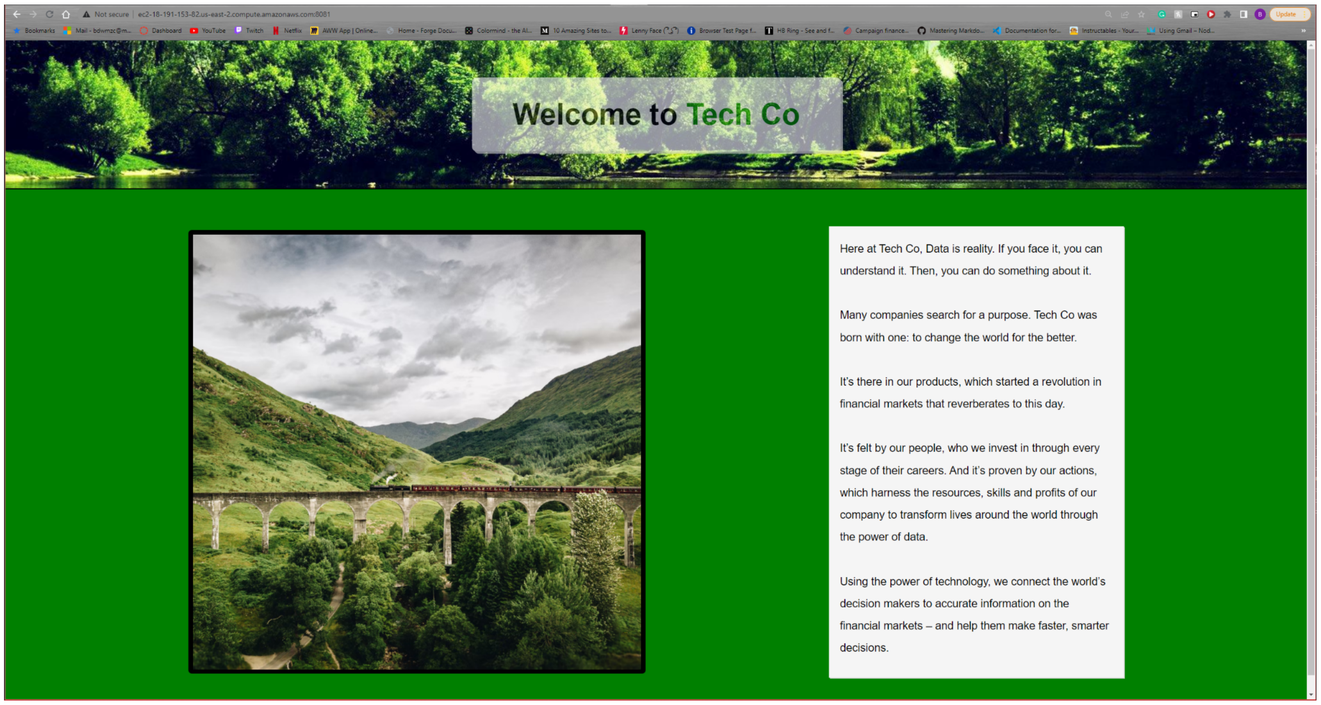
```
[ec2-user@ip-172-31-27-236 Banquet]$ sudo ./banquetV2.0 dish add i18ku
Welcome to Banquet
Please ensure Docker is installed on the local machine before adding a dish.
If you have an existing build you would like to use, please enter the title of that application (blank for no preexistin
g build):

Please enter a title for this application:
AWS Test
Please enter the GitHub Repository name for banquet to locate your application (must be exactly as it appears on GitHub)
:
banquet-tester
Is your repository private or public? (private, public):
public
Are you using Ionic themes for your project?
no
Would you like to use Capacitor to build your application for Android?
no
Would you like to implement a custom stylesheet?
no
Would you like to implement a custom typescript file?
no
Please enter an image URL. To add more images after this one, type -n after the URL (Leave blank for no additions):
https://wallpaperaccess.com/full/13190.jpg -n
Please enter an image URL. To add more images after this one, type -n after the URL (Leave blank for no additions):
https://i.imgur.com/HlmWDMU.jpeg
Please enter a color hex code or rgb function. Examples: '#ffffff' or 'rgb(255, 255, 255)' To add more colors after this
 one, type -n after the color (Leave blank for no additions):
green -n
Please enter a color hex code or rgb function. Examples: '#ffffff' or 'rgb(255, 255, 255)' To add more colors after this
 one, type -n after the color (Leave blank for no additions):
black
Please provide the port that banquet should deploy the container to:
8081
Installing packages for project... This will probably take a while
```

Fig 21. Prompt Answers to Banquet Dish Add Command



```
Starting Docker Daemon..
Building Docker Image...
Build Response:
{"stream":"Step 1/10 : FROM node:16.13.0 as build"}
{"stream":"\n"}
{"stream":" ---\u003e 5964aa70c11d\n"}
{"stream":"Step 2/10 : WORKDIR /app"}
{"stream":"\n"}
{"stream":" ---\u003e Running in 569f59bea759\n"}
{"stream":"Removing intermediate container 569f59bea759\n"}
{"stream":" ---\u003e 73604fd654f9\n"}
{"stream":"Step 3/10 : COPY . ./"}
{"stream":"\n"}
{"stream":" ---\u003e 72a3ecc84560\n"}
{"aux":{"ID":"sha256:72a3ecc845606cc455a10d07133d142e50c19fb65adf6bfcbe2430ad337c4c77"}}
{"stream":"Step 4/10 : FROM nginx:1.17.8-alpine"}
{"stream":"\n"}
{"stream":" ---\u003e 48c8a7c47625\n"}
{"stream":"Step 5/10 : COPY --from=build /app /usr/share/nginx/html"}
{"stream":"\n"}
{"stream":" ---\u003e 67cdda997eae\n"}
{"stream":"Step 6/10 : RUN chmod -R 765 /usr/share/nginx/html"}
{"stream":"\n"}
{"stream":" ---\u003e Running in afd933eeaa35\n"}
{"stream":"Removing intermediate container afd933eeaa35\n"}
{"stream":" ---\u003e 6dbf484a3759\n"}
{"stream":"Step 7/10 : RUN rm /etc/nginx/conf.d/default.conf"}
{"stream":"\n"}
{"stream":" ---\u003e Running in 1cff0214bff8\n"}
{"stream":"Removing intermediate container 1cff0214bff8\n"}
{"stream":" ---\u003e 2194da5b7ba7\n"}
{"stream":"Step 8/10 : COPY ./nginx.conf /etc/nginx/conf.d"}
{"stream":"\n"}
{"stream":" ---\u003e 4a94657638f3\n"}
{"stream":"Step 9/10 : EXPOSE 80"}
{"stream":"\n"}
{"stream":" ---\u003e Running in 7f632d731329\n"}
{"stream":"Removing intermediate container 7f632d731329\n"}
{"stream":" ---\u003e 65076306880a\n"}
{"stream":"Step 10/10 : CMD [\"nginx\", \"-g\", \"daemon off;\"]"}
{"stream":"\n"}
{"stream":" ---\u003e Running in 8cccf7d1b4e3\n"}
{"stream":"Removing intermediate container 8cccf7d1b4e3\n"}
{"stream":" ---\u003e 88668bfdc171\n"}
{"aux":{"ID":"sha256:88668bfdc171a375c358c00da9648a77e14018fccfd689c87c2a8c78d73352d1"}}
{"stream":"Successfully built 88668bfdc171\n"}
{"stream":"Successfully tagged banquet-aws_test_i18ku:latest\n"}
Running container on port 8081...
[ec2-user@ip-172-31-27-236 Banquet]$
```

Fig 22. Output of Banquet Dish Add Command


Not secure | ec2-18-191-153-82.us-east-2.compute.amazonaws.com:8081

Welcome to Tech Co

Here at Tech Co, Data is reality. If you face it, you can understand it. Then, you can do something about it.

Many companies search for a purpose. Tech Co was born with one: to change the world for the better.

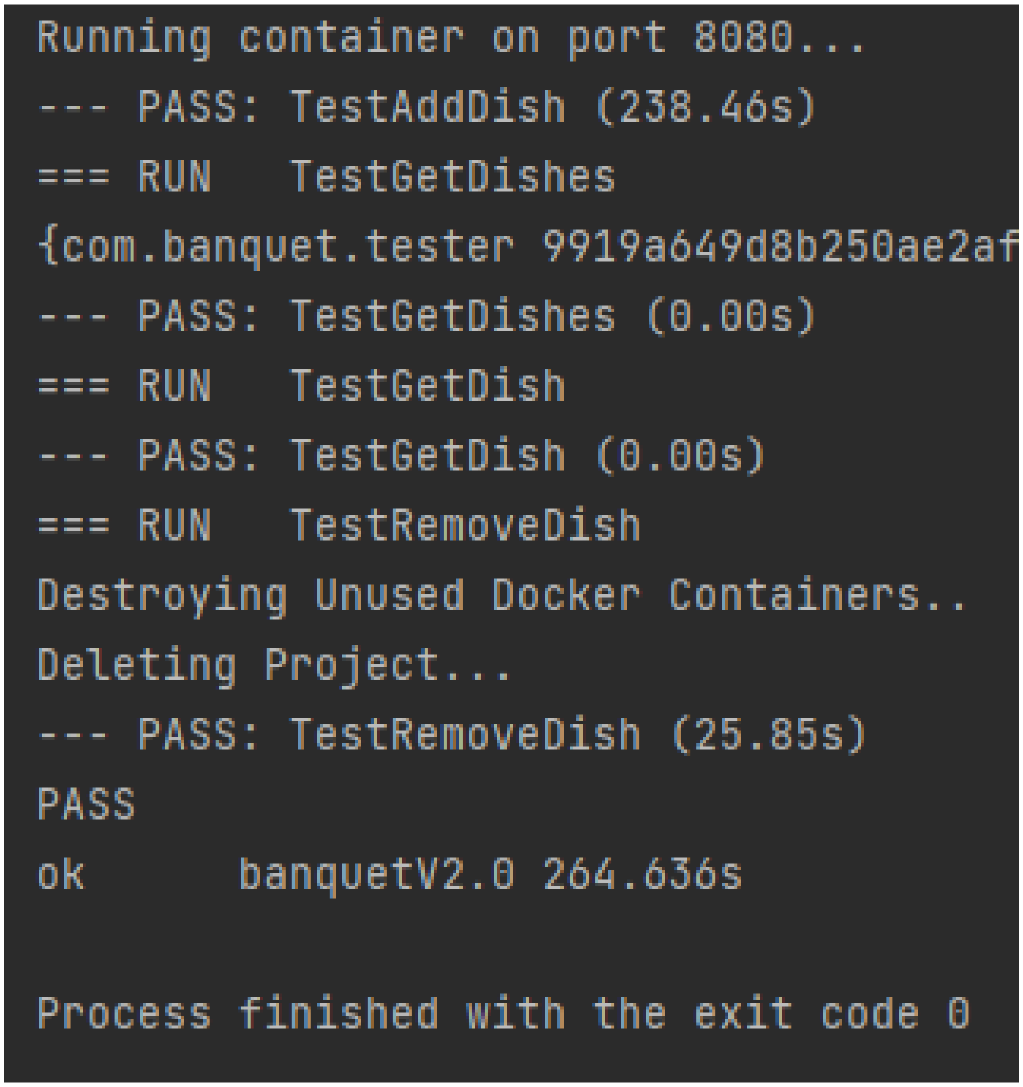
It's there in our products, which started a revolution in financial markets that reverberates to this day.

It's felt by our people, who we invest in through every stage of their careers. And it's proven by our actions, which harness the resources, skills and profits of our company to transform lives around the world through the power of data.

Using the power of technology, we connect the world's decision makers to accurate information on the financial markets – and help them make faster, smarter decisions.

Fig 23. Deployed Application Created by Deployment 1

Additionally, the unit tests and integration test included with the Banquet project were run, resulting in a pass for all four unit tests and a pass for the one integration test. The results of these tests are shown in Figure 22:



```
Running container on port 8080...
--- PASS: TestAddDish (238.46s)
=== RUN   TestGetDishes
{com.banquet.tester 9919a649d8b250ae2af
--- PASS: TestGetDishes (0.00s)
=== RUN   TestGetDish
--- PASS: TestGetDish (0.00s)
=== RUN   TestRemoveDish
Destroying Unused Docker Containers..
Deleting Project...
--- PASS: TestRemoveDish (25.85s)
PASS
ok      banquetV2.0 264.636s

Process finished with the exit code 0
```

Fig 24. Results of Banquet Unit Tests

These tests show that Banquet offers some specific advantages to traditional CICD tools, while also offering some novel features. Banquet has several similarities with existing services, but the differences are fundamental to the tool. Some of the key differences of Banquet are: Banquet's type of integration with GitHub, Banquet's abstraction, and Banquet's SaaS focus.

While Banquet can use GitHub actions/webhooks much like other services, it can also work in reverse, where Banquet calls GitHub using a developer's account. This option makes setup quicker and less complicated for the developer as no knowledge or skill with GitHub is required to pull and use apps. Banquet's abstraction also offers increased flexibility, allowing developers to add whatever integrations they need easily. 11.56% of Travis CI users were running off of a fork of Travis CI [20]. This percentage would likely be much higher for Banquet at the same popularity level, as users tooled Banquet for their specific needs.

These existing tools also lack a SaaS focus, with no customization options out of the box. Banquet's code injection, style injection, and mobile integration make it a tool specific to SaaS that is not available anywhere else. The model that offers features closest to this is the CMS, which offers dynamic content management of apps. However, as requirements change in complexity or scope, CMS tools prove to be too rigid and do not scale with a project's lifecycle [49]. Banquet offers the same features without the additional load times and poor scalability of a CMS. Banquet was built with both CI and CD in mind, as opposed to other tools which typically require more tooling for Continuous Deployment. Furthermore, Banquet can be run locally as a CLI or integrated with other pipelines through the REST API - a feature that typically only exists as one or the other in other tools. With all of this in mind, it is clear that Banquet offers new solutions to old problems. Regardless of the adoption level of Banquet in the future, the work presented in building this tool can be used to challenge existing pipeline implementations to simplify further and increase the efficiency of future tools.

Chapter 4

FUTURE WORK

In terms of limitations of Banquet, there is ample room for improvements and future work. Some examples include: increased testing methods, further technology integrations, more methods of deployment, and better container monitoring/orchestration. As developers are experiencing frustration with existing CICD tools and the SaaS model continues to grow, it is hopeful that the open-source community behind Banquet can grow to be a well-rounded and powerful tool that can be used by a much broader range of incoming developers.

The Banquet project currently has four unit tests and one integration test, created with the Go testing library. These tests were sufficient for the development of the initial features of Banquet and went a long way in helping the quality of the codebase. There are many options for expansions in testing, however. Unit tests could be created for the REST API endpoints to ensure working functionality and to ensure that a connection is established and viable for usage with HTTP protocol. Stress tests could evaluate any limitations in the upper performance of Banquet, including the horizontal scalability of the tool with cloud resources provided. End-to-end tests of the CLI could be valuable in finding any edge case responses or poor inputs that are unlikely to be caught with manual testing of the tool. Overall, performing more benchmark tests on the tool could also show that the tool is powerful enough to compete with existing tools and show Banquet’s value in other areas.

Further technology integrations are vital to the success of Banquet. New JavaScript frameworks are created every year, pushing the limits of the capabilities of JavaScript. Automation should be pushing the same limits. Integration with future frameworks is important, as well as ensuring that all existing frameworks work with Banquet. Currently, Banquet has been used to build the three most popular frameworks: Angular, Vue, and React. Likewise, integration with other forms of version control is important. Banquet only has existing integrations with GitHub for pulling projects. While it may not be difficult to integrate other version control systems, it may be time-consuming to add all systems that users would like to use. To push the limit on what automation can do, the Banquet project should also seek to integrate with brand new innovative technologies as they become available.

Currently, the only functioning form of deployment is locally on the host machine. While this works sufficiently for most needs, it would be ideal to allow users to deploy directly to their preferred cloud service provider. Integrations have begun for Amazon AWS, Google Cloud, and the Google Play Store, but there are many more cloud providers that could find a use case with Banquet - as such, providing this flexibility by adding more hosting options fits right in with the goals of the tool overall.

Banquet could greatly benefit from container monitoring and orchestration. Future work on the tool will certainly include container monitoring to ensure and alert developers that applications built with Banquet are being provided correctly at the desired location. Integration with a container orchestration tool such as Kubernetes would allow Banquet to manage cloud resources itself, rather than relying on external tools to scale the output of containers that Banquet creates. By providing monitoring returned in the get dish commands and handling resource allocation, Banquet would essentially become a one-stop tool for application deployments.

Adaptability is becoming increasingly important in the software industry. Software as a Service and Cloud Computing are on the rise, and the need for better, more performant CICD tools is rapidly developing. The current popular methods of deploying applications for multiple clients are sloppy and inefficient. This thesis proposed a new tool developed to allow the automatic style change, packaging, and deployment of web applications for multiple clients. The tool, titled Banquet, is a low-cost, low development time solution for building scalable Software as a Service pipelines and utilizes some of the newest technologies in web development. The future of Cloud Computing will require tools like Banquet to automate the way cloud deployments and customization are created, and the work presented in this thesis lays the groundwork for pursuing better automation and pipeline development.

Bibliography

[1] Gartner. (n.d.). *Gartner Forecasts Worldwide Public Cloud End-user spending to grow 23% in 2021*. Gartner. Retrieved March 13, 2022, from https://www.gartner.com/en/newsroom/press-releases/2021-04-21-gartner-forecasts-worldwide-public-cloud-end-user-spending-to-grow-23-percent-in-2021

[2] Dutta, P., & Dutta, P. (2019). Comparative study of cloud services offered by Amazon, Microsoft and Google. *International Journal of Trend in Scientific Research and Development*, *Volume-3*(Issue-3), 981–985. https://doi.org/10.31142/ijtsrd23170

[3] Shahin, M., Ali Babar, M., & Zhu, L. (2017). Continuous integration, delivery and deployment: A systematic review on approaches, tools, challenges and practices. *IEEE Access*, *5*, 3909–3943. https://doi.org/10.1109/access.2017.2685629

[4] Dillon, T., Wu, C., & Chang, E. (2010). Cloud computing: Issues and challenges. *2010 24th IEEE International Conference on Advanced Information Networking and Applications*. https://doi.org/10.1109/aina.2010.187

[5] Soni, M. (2015). END TO END automation on cloud with build pipeline: The case for DevOps in insurance industry, continuous integration, continuous testing, and continuous delivery. *2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*. https://doi.org/10.1109/ccem.2015.29

[6] Chen, H. (2016). Architecture strategies and data models of software as a service: A Review. *2016 3rd International Conference on Informative and Cybernetics for Computational Social Systems (ICCSS)*. https://doi.org/10.1109/iccss.2016.7586486

[7] IBM Cloud Team. (2021, May 14). *SOA vs. microservices: What's the difference?* IBM. Retrieved March 30, 2022, from https://www.ibm.com/cloud/blog/soa-vs-microservices

[8] Nassif, A. B., & Capretz, M. A. M. (2010, September 16). *Moving from SAAS applications towards SOA services*. IEEE Xplore. Retrieved March 30, 2022, from https://ieeexplore.ieee.org/document/5575822

[9] Ying, F., & Lei, G. (2014). Optimal scheduling simulation of software for Multi-tenant in cloud computing environment. *2014 Fifth International Conference on Intelligent Systems Design and Engineering Applications*. https://doi.org/10.1109/isdea.2014.158

[10] Zampetti, F., Geremia, S., Bavota, G., & Di Penta, M. (2021). CI/CD pipelines evolution and restructuring: A qualitative and quantitative study. *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. https://doi.org/10.1109/icsme52107.2021.00048

[11] Hilton, M., Nelson, N., Tunnell, T., Marinov, D., & Dig, D. (2017). Trade-offs in Continuous Integration: Assurance, security, and flexibility. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. https://doi.org/10.1145/3106237.3106270

[12] Nicolai, J. (2017, November 8). *GitHub welcomes all CI tools*. The GitHub Blog. Retrieved April 5, 2022, from https://github.blog/2017-11-07-github-welcomes-all-ci-tools/

[13] Evans, C. C., Ben-Kiki, O., Net, I. döt, Müller, T., Antoniou, P., Aro, E., & Smith, T. (2021, October 1). *Yaml Ain't markup language (YAML™) version 1.2*. YAML Ain't Markup Language (YAML™) revision 1.2.2. Retrieved March 31, 2022, from https://yaml.org/spec/1.2.2/

[14] Jenkinsci. (n.d.). *Jenkinsci/jenkins: Jenkins automation server*. GitHub. Retrieved April 5, 2022, from https://github.com/jenkinsci/jenkins

[15] Gargees, R. S. (2020). Multi-stage Cloud Framework based on agents for dynamic, scalable, and secure distributed computing. *University of Missouri--Columbia. Graduate School. Theses and Dissertations*, 112–125. https://doi.org/10.32469/10355/86473

[16] Hilton, M., Tunnell, T., Huang, K., Marinov, D., & Dig, D. (2016). Usage, costs, and benefits of continuous integration in open-source projects. *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. https://doi.org/10.1145/2970276.2970358

[17] Mysari, S., & Bejgam, V. (2020). Continuous integration and continuous deployment pipeline automation using Jenkins Ansible. *2020 International Conference on Emerging Trends in Information Technology and Engineering (Ic-ETITE)*. https://doi.org/10.1109/ic-etite47903.2020.239

[18] Jenkins Development Team. (n.d.). *Jenkins User Documentation*. Jenkins user documentation. Retrieved April 5, 2022, from https://www.jenkins.io/doc/

[19] Cepuc, A., Botez, R., Craciun, O., Ivanciu, I.-A., & Dobrota, V. (2020). Implementation of a continuous integration and deployment pipeline for containerized applications in Amazon Web Services using Jenkins, Ansible and Kubernetes. *2020 19th RoEduNet Conference: Networking in Education and Research (RoEduNet)*. https://doi.org/10.1109/roedunet51892.2020.9324857

[20] Durieux, T., Abreu, R., Monperrus, M., Bissyande, T. F., & Cruz, L. (2019). An analysis of 35+ million jobs of Travis Ci. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. https://doi.org/10.1109/icsme.2019.00044

[21] Travis CI Development Team. (n.d.). *Travis CI documentation*. Travis CI Docs. Retrieved April 5, 2022, from https://docs.travis-ci.com/

[22] Kinsman, T., Wessel, M., Gerosa, M. A., & Treude, C. (2021). How do software developers use github actions to automate their workflows? *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. https://doi.org/10.1109/msr52588.2021.00054

[23] Laukkanen, E., Itkonen, J., & Lassenius, C. (2017). Problems, causes and solutions when adopting continuous delivery—a systematic literature review. *Information and Software Technology*, *82*, 55–79. https://doi.org/10.1016/j.infsof.2016.10.001

[24] Arachchi, S. A. I. B. S., & Perera, I. (2018). Continuous integration and continuous delivery pipeline automation for Agile Software Project Management. *2018 Moratuwa Engineering Research Conference (MERCon)*. https://doi.org/10.1109/mercon.2018.8421965

[25] Sabau, A. R., Hacks, S., & Steffens, A. (2020). Implementation of a continuous delivery pipeline for Enterprise Architecture Model Evolution. *Software and Systems Modeling*, *20*(1), 117–145. https://doi.org/10.1007/s10270-020-00828-z

[26] Rudrabhatla, C. K. (2020). Comparison of zero downtime based deployment techniques in public cloud infrastructure. *2020 Fourth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. https://doi.org/10.1109/i-smac49090.2020.9243605

[27] Loukis, E., Janssen, M., & Mintchev, I. (2019). Determinants of software-as-a-service benefits and impact on firm performance. *Decision Support Systems*, *117*, 38–47. https://doi.org/10.1016/j.dss.2018.12.005

[28] Melegati, J., Goldman, A., Kon, F., & Wang, X. (2019). A model of requirements engineering in Software Startups. *Information and Software Technology*, *109*, 92–107. https://doi.org/10.1016/j.infsof.2019.02.001

[29] Melegati, J., & Kon, F. (2020). Early-stage software startups: Main challenges and possible answers. *Fundamentals of Software Startups*, 129–143. https://doi.org/10.1007/978-3-030-35983-6_8

[30] Wu, C., Buyya, R., & Ramamohanarao, K. (2018). Cloud computing market segmentation. *Proceedings of the 13th International Conference on Software Technologies*. https://doi.org/10.5220/0006928008880897

[31] Senarathna, I., Wilkin, C., Warren, M., Yeoh, W., & Salzman, S. (2018). Factors that influence adoption of cloud computing: An empirical study of Australian smes. *Australasian Journal of Information Systems*, *22*. https://doi.org/10.3127/ajis.v22i0.1603

[32] Li, L., & Chou, W. (2011). Design and describe rest API without violating rest: A petri net based approach. *2011 IEEE International Conference on Web Services*. https://doi.org/10.1109/icws.2011.54

[33] Severance, C. (2015). Roy T. Fielding: Understanding the rest style. *Computer*, *48*(6), 7–9. https://doi.org/10.1109/mc.2015.170

[34] Wittern, E., Suter, P., & Rajagopalan, S. (2016). A look at the dynamics of the JavaScript package ecosystem. *Proceedings of the 13th International Conference on Mining Software Repositories*. https://doi.org/10.1145/2901739.2901743

[35] Thomson, E. (n.d.). *NPM docs*. npm Docs. Retrieved April 10, 2022, from https://docs.npmjs.com/

[36] GitHub. (n.d.). *Build software better, together*. GitHub. Retrieved April 10, 2022, from https://github.com/about

[37] Sallou, O., & Monjeud, C. (2015, October 29). *Go-docker: A batch scheduling system with Docker containers*. IEEE Xplore. Retrieved February 24, 2022, from https://ieeexplore.ieee.org/document/7307636

[38] Stijn, S. van, Villele, G. de, Suda, A., Linville, M., & Cano, F. (2022, February 24). *Docker Security*. Docker Documentation. Retrieved February 25, 2022, from https://docs.docker.com/engine/security/

[39] Hutchings, A. (n.d.). *Welcome to NGINX Wiki!* NGINX Wiki. Retrieved April 11, 2022, from https://www.nginx.com/resources/wiki/

[40] Pike, R. (2009, October 30). *The Go Programming Language*. Google. Retrieved April 11, 2022, from https://9p.io/sources/contrib/ericvh/go-plan9/doc/go_talk-20091030.pdf

[41] Pike, R. (2012, October 25). Go at Google: Language Design in the Service of Software Engineering. Retrieved April 11, 2022, from https://talks.golang.org/2012/splash.article

[42] Bui-Palsulich, T., & Compton, E. (2019, March 19). *Build fast, reliable, and efficient software at scale*. Go. Retrieved April 11, 2022, from https://go.dev/

[43] Capacitor. (n.d.). *Cross-platform native runtime for web apps*. Capacitor. Retrieved February 25, 2022, from https://capacitorjs.com/

[44] Crispin, L. (2006). Driving software quality: How test-driven development impacts software quality. *IEEE Software*, *23*(6), 70–71. https://doi.org/10.1109/ms.2006.157

[45] Go Development Team. (n.d.). *Go Testing*. testing package - testing. Retrieved April 12, 2022, from https://pkg.go.dev/testing

[46] *The MIT License*. The MIT License | Open Source Initiative. (n.d.). Retrieved April 12, 2022, from https://opensource.org/licenses/MIT

[47] Lenarduzzi, V., Tosi, D., Lavazza, L., & Morasca, S. (2019). Why do developers adopt open source software? past, present and future. *IFIP Advances in Information and Communication Technology*, 104–115. https://doi.org/10.1007/978-3-030-20883-7_10

[48] Singh, C., Gaba, N. S., Kaur, M., & Kaur, B. (2019). Comparison of different CI/CD tools integrated with cloud platform. *2019 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. https://doi.org/10.1109/confluence.2019.8776985

[49] Srivastav, M., & NAth, A. (2016). Web Content Management System. *International Journal of Innovative Research*, *3*, 51–56.