Automating Dashboard Capabilities at Capital One

A Technical Report submitted to the Department of Computer Science

Presented to the Faculty of the School of Engineering and Applied Science

University of Virginia • Charlottesville, Virginia

In Partial Fulfillment of the Requirements for the Degree

Bachelor of Science, School of Engineering

Matthew Yang

Spring, 2022

Advisor

Daniel Graham, Computer Science

Technical Report

Automating Dashboard Capabilities at Capital One

CS4991 Capstone Report, 2021

Matthew Yang
Computer Science
University of Virginia
School of Engineering and Applied Science
Charlottesville VA USA
my9uu@virginia.edu

ABSTRACT

At Capital One, a major pain point involved across the tech division creating their own dashboards for their own purposes, causing redundancy, since most dashboards served the same functionality. My internship team was tasked with leveraging different languages and frameworks as well as company technologies to automate the creation of dashboards through both a web wizard and a command line utility. Our project, the dashboard automator, allows teams to automate the process of creating a dashboard and standing it up on the cloud without significant manual intervention. This effort not only substantially reduces the effort needed to stand up a dashboard, but also kick-starts the potential beginning of a "marketplace" with the purpose of reducing redundancy throughout the company. Some limitations that the dashboard automator currently faces includes limited customizability as well as lack of complete testing.

1 Introduction

Automation has been a key driving force in human innovation and ingenuity for centuries. In a sense, the idea of automating human labor is quite intuitive: less effort, but same result, at the cost of some initial investment. As humans and society evolved, automation has taken on many different forms. During the agricultural revolution, we saw humans automate food production in the form of farming instead of hunting and gathering. In the industrial age we saw the automation of mass production in the form of factories instead of heavy manual labor. And now, we are living in an age of large-scale technological evolution where we are seeing automation in almost every aspect of life.

Currently, society is primarily dominated by millions of corporations, each with their own markets. However, to really break through to the top of these markets, companies need to be efficient with the use of their resources. In a constantly evolving competitive technological world, many large businesses and companies are looking to automation in the workplace as a way to best use their business resources. More specifically, the idea of reducing redundancy and allocating business resources to other business endeavors is quickly becoming a top priority in order to stay competitive.

2 Background

Capital One is trying to break through to the top, especially in online and digital banking. There have been enormous efforts, especially in recent years, to increase the company's presence in the technology sector. Whether that be a massive surge in new technology department hires, or a commitment to be fully on the cloud, Capital One is committed to becoming a leader in the industry.

They are looking for any areas where they can redistribute and refocus business resources to stay competitive. In order to do that, there are internal non-customer facing teams responsible for developing tools for customer-facing teams to do their jobs more efficiently. One of the key goals of these teams is to help reduce redundancy; that is, to reduce common efforts across the company that have the same general goal. One of the biggest areas of redundancy is the creation of dashboards. Many teams have uses for dashboards and commit resources to creating and maintaining these dashboards. Every dashboard served a similar purpose but every team was responsible for creating them them from scratch. Our project, the dashboard automator, seeks to reduce these redundant efforts and automate the process of creating and standing up a dashboard on the cloud for teams to use in an effort to reduce this area of redundancy.

3 Related Work

Automation exists everywhere in the industry. The degree of automation extends from things like automatically sending out emails to a list of people ~~all the way~~ to advanced artificial intelligence making advanced decisions and predictions.

One prominent automation tool used by companies, including Capital One, is automated content analysis and automated report generation. These two functions, especially of those on media, generate important insights to help the company understand their users [1]. It helps reduce intensive human interaction of sifting through tons of data but also provides a more holistic and comprehensive analysis of the data as a whole, since a machine can compute and analyze large amounts of data a lot more efficiently than a human can.

The dashboard automator's role serves a similar purpose: to automate a process that a human *can* do but in a much more efficient manner. Although teams can spend a week or two to prepare and stand up a dashboard, the dashboard automator will help them jump start that effort in just a matter of a couple of days. However, where the dashboard automator differs from automated report generation and content analysis is the dashboard automator is a lot more static. It is less focused on automating the generation of insights; rather, it is focused more on automating the production of a tool or product

4 Project Design

The project given to our internship team was to create a tool called the Dashboard Automator. Its primary purpose is to generate and create the code needed to kickstart a dashboard application, as well as generate the necessary resources and infrastructure to get it running on Amazon Web Services (AWS). This is an effort intended primarily to streamline and reduce the redundancy of having teams across the company creating dashboards from scratch.

4.1 Review of System Architecture

The team split the architecture of the dashboard automator into two primary sections: the generation phase as well as the deployment and runtime phase.

4.1.1 Generation Phase

The generation phase is primarily concerned with the generation of the necessary code and resources needed to kickstart a dashboard. Users have the option of using either a command line interface (CLI) or a web-wizard. When the user runs one of the CLIs, it prompts them to fill in some details about team information, product information, etc., in order to fill out template files. Once all the information is received, the program then fills out several template files and creates a local repository that houses the relevant code. This code is then automatically pushed to GitHub too. There are several CLIs responsible for the front end, backend, and the infrastructure necessary in order to stand the dashboard up on the cloud.

Users who are less familiar with command prompts and running CLIs may choose the option to use our web-wizard application. The web-wizard is an internal cloud application that can be accessed by employees and essentially executes the same function as the CLIs, but with forms instead of prompts. Once all the forms are completed and submitted, the web-wizard actually has a back-end that will run the previously mentioned CLIs with the inputs in the forms.

4.1.2 Deployment and Runtime Phase

With the code and the repositories that are created in the generation phase, the deployment and runtime phases runs them through Capital One's internal continuous integration and continuous development (CI/CD) pipeline in order to get the dashboard running on the cloud. This phase primarily concerns itself with taking the infrastructure code that is generated and then running it through the CI/CD pipeline. The infrastructure code that is generated by the previous phase allows users to generate and create the necessary AWS credentials. Once the necessary cloud infrastructure is set up, the front-end application is then stood up on an internal cloud application being supported by either a backend EC2 server or a serverless Lambda function.

4.2 Key Components

The design of the dashboard automator is split up into several key components: the frontend, backend, frontend infrastructure, backend infrastructure, and finally the CLIs themselves. These are all the necessary components to not only get the application working, but also to get it on the cloud.

For our frontend component, users used an internal library that's used in Capital One. As a design decision, we wanted to utilize as many internal Capital One technologies as possible, so choosing an internal frontend library was an obvious choice. The frontend component primarily provides a frontend application for users to interact with. It includes a simple navbar with customizable links as well as a table display for database queries.

For the backend, we give users a choice in either using a Flask, NodeJS, or a serverless Lambda function. These frameworks and solutions were primarily selected as they are currently industry standards for backend applications. The backend component is primarily responsible for allowing our frontend component to be more dynamic. For the purposes of our project, the backend application was only responsible for interacting with a database.

Both the backend and frontend infrastructure components were primarily used to generate and create necessary AWS resources for cloud deployment. The files were mostly scripting files to feed into our internal CI/CD pipeline which would do most of the work of allocating necessary resources.

Finally, the backbone of the dashboard automator is our command-line interface tools. The previously mentioned components are all files that are templatized according to a NodeJS library that each CLI uses to fill out the templates. Each CLI prompts the user for different information such as database information, product information, and other things that are needed to correctly configure the template files. The CLI prompts the user for inputs then sends these to the template files to generate repositories that are then automatically pushed onto GitHub.

We were also able to reach our stretch goal, the web-wizard. This component is primarily used as a one-stop-shop for all of the previously mentioned CLIs. It compounds them all into a web-based application that has a NodeJS backend that runs all the CLIs. The web wizard, instead of using individual prompts, has several forms that the user can fill out instead and once submitted, will also generate and push the necessary repositories to stand up a dashboard. This also allows users not to install each individual CLI on their local machine since the web-wizard itself runs through AWS.

4.3 Challenges

For the team, the biggest challenges were learning how to use the internal Capital One resources, as well as learning new languages and frameworks. We spent the first couple weeks just doing our own research and getting more familiar with the tools that we used.

For every one of us, this was our first time coding a large-scale JavaScript application and our first time working with the NodeJS and Flask frameworks. We spent a significant time in the beginning experimenting with sample applications just to get familiar with everything.

Thereafter, we split into different "specializations." Team members involved with getting the CLIs working or working on the web-wizard had their own challenges. As the one responsible for making sure everything went through the CI/CD pipeline and getting everything on the cloud, I faced significant challenges in learning and understanding everything that goes on behind the scenes. I spent a good amount of effort learning about Docker containers, AWS resources, and scripting files in order to make sure that all of our tools could be successfully built and tested through the CI/CD pipeline. There is not a lot of course material that could have prepared me for this so it took a lot of digging through other team's codes and searching for references to make sure that I got everything right in the end.

5 Results

Since the project itself was generally just a proof of concept, the team was able to accomplish its goal. We were not only able to release a working product on its own, but we were also able to reach our stretch goal. Although it may not be fully customizable or exactly what every team needs from a dashboard, it does set up all the most time-consuming components such as configuring AWS resources and creating proper CI/CD scripting files.

Currently, several teams have actually chosen to use the dashboard automator to create a dashboard. These teams have reported success and with a bit of tweaking, were able to twist our skeleton dashboard into one that would be useful to them.

6 Conclusion

The result of our tool significantly helps teams reduce redundant efforts to create a dashboard. It can take one to two weeks or more for teams to create a dashboard from scratch, but with the use of the dashboard automator, that effort can be reduced to one to two days of configuring a working dashboard already running on the cloud. The dashboard automator, however, is only the tip of the iceberg, as the company plans to use it as the starting point for a plethora of other automated tools to reduce redundancy throughout the tech department.

7 Future Work

Although we were able to reach our stretch goal of implementing a web wizard along with the CLI tool, there is still much room for improvement for the dashboard automator. For one, due to our limited experience with industry technologies and CI/CD, there are still some that are not yet completely automated. Ideally, a user should be able to just fill out the forms in the web wizard and have a dashboard automatically spun up; however, there are several steps where the user has to manually intervene to generate things like AWS tokens or initiate CI/CD builds.

In order to further improve the dashboard automator, we would also have liked to integrate more functional tools or capabilities. The dashboard automator currently only sets up a basic navbar and establishes a connection to a database to pull data from. Automating the creation of things like search tools or the creation of figures and graphs would benefit certain teams greatly.

Additionally, due to the nature of our time on the internship, there is room for improvements in code structure. Given more time, we would have liked to properly go through and clean up and refactor redundant code (ironic given the purpose of the application itself) and overall make it a bit more readable for anyone trying to make any changes in the future. We would have also have liked to include a lot more documentation so that future contributors will be able to easily understand what all of the code is doing.

We also did not spend as much time on testing as we initially had planned to do. We had basic unit tests, but were not able to successfully complete a comprehensive test suite to test all the features of the dashboard automator. Given more time, we would like to have implemented such a test suite to ensure that all components worked correctly.

The end-goal of the dashboard automator was primarily to establish the first in a line of many shared tools among internal Capital One teams. As time goes on, this "marketplace" will be populated with similar tools that help automate processes for teams in order to further reduce redundancy throughout the company.

8 CS Evaluation

I was not as prepared for my internship as I thought I would have been. However, I do not believe that is an inherent flaw of UVA's CS department, but rather the nature of the work at Capital One. Although certain classes helped me feel more comfortable with certain aspects of the work, such as databases and the agile development cycle, my biggest struggle this past summer was really learning and understanding new technologies.

It is really hard for courses to properly prepare students to know how to pick new technologies and languages, since every company and every project and team utilize something different. For example, although a lot of the code that I wrote was in JavaScript, the majority of my time during the project was spent learning how to write Docker and Jenkinsfiles in order to feed our work into the CI/CD pipeline. Yes, classes provided me with the basic understanding of how to write good code, but did not really prepare me in terms of creating things like scripting files, which, to be fair, can be hard to integrate in a classroom setting.

The most significant room for improvement for the CS department would be to either make adjustments to the CS3240 curriculum or create another path for students seeking to pursue academia or a job in the industry. I think it is important to make that distinction because some classes such as Computer Architecture and Operating Systems, although important and interesting topics, do not generally have any practical applications to most entry-level positions in the industry.

Although CS3240 provides that experience to many students, it:

1) is far too short of a class to really properly simulate what the development cycle really feels like;
2) is a bit dated in terms of the technologies that are used (although Django is a popular web framework, from my own and other students' experience frameworks such as NodeJS are much more prevalent in the industry); and

3) does not spend nearly enough time on the CI/CD portion of software development.

Creating a separate pathway for students interested in jumping straight into the industry will help better prepare them for future jobs. A couple suggestions would be to:

1) make another class that is a continuation of CS3240. Similar to the new curriculum with classes like DSA1 and DSA2, we could see something similar that would help build on CS3240; and
2) update the languages and frameworks used in CS3240 and provide a larger focus on the CI/CD pipeline.

Other than those suggestions though, there are not many other complaints for the CS department. It can be a bit hard to tailor and fully prepare every student for every possible job in the industry because everything is always a bit nuanced. The biggest lesson that a student can really learn is how to problem-solve, and I think the courses provided in the CS department do properly prepare students for that. Learning how to learn and picking up new technologies quickly is something that our department does better than any other, in my opinion.

REFERENCES

[1] UNIVERSITY OF HELSINKI, M.AND M., UNIVERSITY OF HELSINKI, D.OF C.S., TOIVONEN, H., AND BOGGIA, M. 2021. Proceedings of the EACL hackashop on news media content analysis and automated report generation. *Proceedings of the EACL Hackashop on News Media Content Analysis and Automated Report Generation*. https://helda.helsinki.fi/handle/10138/329203.