7Factor AWS Cost Analysis Tool

A Major Qualifying Report:
Submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science
In cooperation with 7Factor, LLC
Submitted March 21st, 2024

By:

Cameron Goodrich

Joey Rozman

Nathaniel Sadlier

Oliver Shulman

Dov Ushman

Project Advisor:

Professor Joshua Cuneo

Abstract

7Factor is a consulting company for clients seeking solutions for their software development. Since 2021, groups of WPI students have been working to develop a tool that can analyze AWS ECS clusters and find the most cost effective setup. The groups before us built a solid foundation and we used that to identify what works well and what could be improved. Over the course of the 2023-2024 academic year, we worked with 7Factor to further optimize the tool and revamp the user interface.

Acknowledgements

We would like to thank the following people for their help throughout the entirety of this project:

Joshua Cuneo, Computer Science Professor at WPI - Project Advisor
Jeremy Duvall - CEO of 7Factor
Allen Brooks - Engineering Manager at 7Factor

Their help and guidance made this project possible and we could not have completed it without them.

Table of Contents

Introduction

Amazon Web Services (AWS) is an easy and effective set of products used for cloud computing and storage. However, the products that AWS offers are not cheap, especially when used for large-scale computations. As a result, it is pertinent to any company using AWS that they use no more of the service than necessary to minimize costs. Finding the optimal configuration for an Amazon Elastic Container Service (ECS) cluster that produces the best results compared to the costs is not a simple task, so the best way to approach the problem is with an optimization tool.

This project is a continuation of the work from two previous teams. The past two years of work provided a basic tool that runs different ECS cluster configurations through an assortment of algorithms and can then be visually represented. Both the algorithms used and the frontend display function as intended, but there are improvements that can be made in regards to efficiency and usability.

1 Background

In this section, we will discuss some important background topics for our project. With a project focused on analyzing Amazon Web Services (AWS), we will discuss what AWS is and some of the important features of AWS that we will be working with. In addition to this, we will create an application utilizing a JavaScript front end through the React framework, as well as a back end utilizing Python. We will also discuss Python libraries that we are working with in our project from previous works. Some key topics that will be discussed are the EC2 Systems, the ECS, SQLAlchemy, Alembic, Tenacity, FastAPI, and Pydantic. We will also develop this application using VSCode, as well as GitHub for our version control.

1.1 Amazon Web Services (AWS)

1.1.1 Introduction to AWS

Amazon Web Services is a service that Amazon supplies to people looking for “compute power, database storage, content delivery, or other functionality” (https://aws.amazon.com/). AWS is often used by software engineers to help support the needs of a web page or app they are working on by helping create a web domain with a database and their Amazon EC2 system.

1.1.2 Amazon EC2 System

The Amazon EC2 is short for Amazon Elastic Compute Cloud. It is a cloud-based platform that offers a wide range of compute platforms with more than 700 instances and the choice of any of the latest processor, storage, networking, operating system and purchase model to help the user find the best fit for their needs (https://aws.amazon.com/ec2/). These EC2 clouds act like virtual machines that can be used to run software and complete tasks. The EC2 system is supported by the Amazon Elastic Container Service (ECS).

1.1.3 Amazon Elastic Container Service (ECS)

The Amazon Elastic Container Service was designed to help manage containerized applications.
This allows for easy deployment and use of apps and web pages, as it helps developers manage the resources like storage and processor for the web page or app in a much easier and simple way (https://aws.amazon.com/ecs/features/). It can also be used to help manage larger workloads with multiple tasks by helping understand what and where each task is being done by supporting multiple containers, each containing task(s).

1.2 Python Libraries

The previous team used an assortment of different Ppython libraries for this project to create and test their algorithms. We will discuss the libraries they used in this section. These libraries include SQLAlchemy, Alembic, Tenacity, FastAPI, and Pydantic.

1.2.1 SQLAlchemy

SQLAlchemy is a Python SQL toolkit and Object Relational mapper. It is primarily used for its persistence patterns, and its efficient and high-performing database access. Its features include no Object Relational Mapper (ORM) required, high performing architecture, DBA approved, non-opinionated, unit of work, function-based query construction, modular and expendable, separate mapping and class design, eager-loading and caching of related objects and collections, composite primary keys, self-referential, inheritance mapping, raw SQL statement mapping, pre and post processing of data, supported platforms, and supported databases. This package will be used for any queries for our back end to our database. The SQLAlchemy documentation can be found with this link.

1.2.2 Alembic

Alembic is a database migration tool used with the SQLAlchemy toolkit. Its features include open ended and transparent configuration and operation, full support of transactional Data Definition Language (DDL), minimalist script construction, auto generation of migrations, full support for migrations generated as SQL scripts, non-linear dependency-graphed versioning, and provide a library of ALTER constructs that can be used by any SQLAlchemy applications (https://github.com/sqlalchemy/alembic). This library will be used for accessing any configuration data that will be used. The Alembic documentation can be found with this link.

1.2.3 Tenacity

Tenacity is a general-purpose retry library. Its features include Generic Decor API, specify stop condition, specify wait condition, customize retrying on Exceptions, customize retrying on expected return result, retry on coroutines, and retry code blocks with context manager. The Tenacity documentation can be found with this link.

1.2.4 FastAPI

FastAPI is a web framework for building APIs with Python based on standard Python type hints. Its features include fast performance, fast to code, fewer bugs, intuition, easy, short, robust, and standard-based. The FastAPI documentation can be found with this link.

1.2.5 Pydantic

Pydantic is a data validation library for python. Its features type hints powering schema validation, high performance, serialization, JSON schema, strict mode and data coercion, dataclasses, customization, ecosystem, and organizations using Pydantic. The Pydantic documentation can be found with this link.

2 Methodology

This section will go over the procedures we went through during our project. Since we had two major goals to reach, a new front end and developing the back end, this section will be split into two parts. The first part will follow our process in building a new front end. The second part will discuss the methods we used in developing the back end from the existing code.

2.1 Front End Development

This section discusses the different steps in our development of the front end. Most of these steps follow common Human Computer Interaction methods. The first step is to review the existing design. Then we design a new look based on our review. Once this look is complete, it's taken into a user study to get outside feedback. The last step is to take this feedback and apply it to the design, developing a fully functional front end. Each subsection covers one of these steps in more detail.

2.1.1 Study Previous Front End

When studying the current/previous front end for development, there are a few questions you want to ask yourself:

- What do we like about it?
- What do we not like about it?
- What do we want to keep?

- What can we remove?
- Is there anything missing?

By thinking about these questions, it helps to narrow down what the new model should look like. By asking what we like about it, we create a list of positives about the current/previous front end. Then by asking what we don't like, we get a list of negatives about the current/previous front end. Once these questions have been answered, the remaining three help distribute the positive and negative lists into keeps, remove, add, and change.

Developing the keeps, remove, add, and change lists from these questions sets up the next step. Make sure that the questions are answered with the goals of the new front end in mind. A feature that you like may not be a feature that remains relevant in the new front end, rather it can help be an influence on how new or changing features are designed. Similarly, a feature that may not be liked may be vital to the success of the new front end.

2.1.2 Design a New Front End Look

Now that the current/previous front end has been reviewed, now it's time to start designing the new front end. To complete this step we used Figma, a platform that can be used to design a front end that can be interacted with similarly to a fully coded front end. Using this, we could design a front end without fully coding it and be able to get some feedback on the design before finalizing.

This is where our lists from reviewing the current/previous front end comes in. With these lists, we start by creating a functional design and a design for the look we want to keep. The design for the look may have multiple variations that one is chosen from. The functional design lays out the functionality the front end will have. Once both of these are finalized, it's time to apply the look to the functional design. Once this step is complete, it's ready for the next step, a user study.

2.1.3 User Study

Now that the new front end design is complete, it's time to test it out. To test it, start by figuring out who your users will be. Once this is known, it's time to decide what form of user study to use. In our case, since the users will be 7Factor, we aimed to have them complete the user test in their own time, completing an online survey to get feedback from them on the design. If possible, we would've done it in-person to get a better understanding of where and when the users might struggle or succeed.

Once you've decided on the format of your user study, it's time to develop the study. The first thing is to decide what your main goals are for your study. In our case, our main goal was for the user to be able to navigate to and receive an AWS setup effectively. We also wanted the user to be able to run tests between various algorithms for further development of this tool. Lastly, we wanted to get a feel for how the look of the tool was received. This led to our final survey which can be seen in appendix A. These questions with the tasks they reflect on are aimed to get unbiased and undirected feedback about the users' experiences to give us more accurate feedback since we cannot directly observe their experience.

2.1.4 Develop New Front End

With a new front-end design being finished, we must convert it from the Figma mock-up to functional React code. Initially, our idea was to code all of our mockups into proper React code manually, but we were hopeful that there was a more efficient way of doing this. Our first thought was to use a front-end framework (such as Bootstrap) to make it so we would have an easier time. Through our research, we found software that can essentially take a Figma mock-up and make the general outline in proper code. For example, if we have a Figma page with a title, buttons, and images, after being put into the software to convert it to code, we would have the page we designed in React code with non-operational buttons. Fixing the buttons was very easy as the software (in most cases) knew they were buttons, and we just had to change the functionality so that when pressed, they would lead to a new page. In the instances where it didn't realize it was a button, our coding was still easier since it would use the wrong javascript tag but still provide helpful information such as spacing and colors.

While this way of creating our front end was incredibly useful, it did have its faults. As a result, we used a combination of this software and Bootstrap to create our front-end. The creation of the pages themselves were coded using the Bootstrap framework, but by using this information provided by the software, the pages were easier to make. Using Bootstrap allowed us to have a consistent format using their components to help make pages easy to replicate and would allow for future additions to be added in an easy and clean manner. For more complex scenarios on our front-end, such as graphs being displayed and updated as the AWS analysis occurred, we had to code components that would reflect the data we produced manually. While this wasn't too hard, it was a realization we had that we had to work around.

Another realization was ensuring that our front end was appealing on various screens. While we assume that our page will most likely be viewed on laptops and desktop monitors, we do not necessarily know this and want to ensure that our program won't break if a user accesses our page via a phone or other device. Ensuring that our code would adjust via screen was a significant discovery we made and something we ensured to check while developing our application.

2.2 Back End Development

This next section details our development process for the back end. There are three main components to the back end that produce the desired result of an optimal AWS EC2 cluster configuration. The first part is creating and running API calls to pull the data on each EC2 instance. Next is converting the data into a usable structure for the following step. The last step is running the converted data through an algorithm to determine if the setup is optimal. If the setup is non-optimal, the optimal setup is suggested to minimize costs.

2.2.1 API Calls

The first step in our process was to create a generalized form of an API call to AWS CloudWatch that allows the app to take in AWS credentials for the account that has the clusters

to optimize. Allowing user input for the credentials is critical to allow the application to be run on any machine whilst still being secure. The most impactful data points to pull from the API call are RAM, CPU, and CPU utilization. How we pull the data from what is returned by the API is discussed in the next section.

2.2.2 Data Conversion

The raw data that AWS CloudWatch returns is in a JSON format which needs to be parsed before any meaningful data points can be retrieved. For passing the data into an algorithm, we decided to follow what the previous year's group did and make a CSV file out of the data. To do this, we first parse the JSON data and put it into a Python list. Then we utilize the Python library "pandas" to convert the list into a CSV file.

2.2.3 Optimization

Our initial method of optimization is to pass in all of the CSV files and calculate based on average CPU utilization whether or not the EC2 cluster's size should increase, decrease, or stay the same. This is not the best logic to test the data with since it only considers the average utilization when in reality the utilization may spike at different times meaning the cluster shouldn't decrease in size when the logic tells it to. To remedy this, we worked on more algorithms to optimize the clusters so that we can test them all and work towards finding the best one for optimizing EC2 cluster costs.

2.2.4 Alternate Algorithm Testing

Since our initial optimization algorithm is meant to serve as a baseline, we needed to try out many algorithms to discover which one performed the best. The first algorithms we tested are the ones used by the previous years group. These algorithms were all different types of knapsack algorithms. One was a standard knapsack, another was a recursive knapsack, and the last was a dynamic knapsack. Testing with these gave us a better understanding of how to work with the

data we received and with that we designed a new algorithm that takes some elements from what worked well in each of the other algorithms.

3 Results

In this section, we will discuss the results of our work. First we'll talk about our overall software and what we accomplished with it. Then we will get more detailed in talking specifically about the front and back end.

3.1 Application

The application we built successfully completes the methods we wanted it to be capable of. The first method is for future work on this project, allowing for the software to be easily improved through testing. The second method we implemented was a tool that takes in documents that includes information about tasks and returns a cost effective AWS setup. This is the practical use of the software for 7Factor to use when creating or updating software.

3.2 Front End

We decided to redesign the front end of the software to make it easier for a user to use and navigate. This meant improving the navigation throughout the front end. The previous front end was done using one page and used scrolling to find most of the utility. To help fix this we implemented multiple pages with a navigation bar that can be used to return to the Home and About Us pages. This helps to make each page readable without needing to scroll in order to find different tools of the software. The navigation bar was built to help the users navigate back to the homepage quickly and easily to start using a different tool.

3.3 Back End

Our back end code takes in AWS credentials as user input and returns a list of the optimal EC2 cluster configurations. Our initial algorithm along with the algorithms from the previous year's group served as a good baseline but fell short of any meaningful optimizations on their own. By learning from those algorithms we design a new algorithm that provides some better results, but there is still a lot of room for refining the algorithms. We ultimately wanted to create a strong, modifiable baseline so that regardless of what algorithm we decided to start with, any future teams will be able to jump start the process without needing to deal with pulling and parsing the data. Our application does just that and with more work could prove to be very useful for optimizing AWS EC2 cluster costs.

4 Discussion

This section will discuss more about the way that we worked as a team and reflect on things we wished to do differently and the things we thought were really helpful. First we will start by talking about something that was more unique to our project in that we spent almost an entire term without being able to access the previous team's work on this project. With this in mind, we spent most of that term thinking about what we might be able to do based on speculation of the other team's work through reading their report. With this in mind, we plan to make it easier for the next team by ensuring that our code repository is in the project advisor and 7Factor's access to give to the team next year.

Once we started working with the previous team's code, we decided to set up weekly meetings as a group outside of meeting with the project advisor. With these weekly meetings, we were able to plan out who would work on different timeline goals each week to keep the work spread out. It was also a time for us to share our progress from the previous meeting and discuss any changes that we needed to make from our original plan. This kept us on track and in good communication, especially when our team split into front and back end teams. We were able to

talk about the progress on the front and back ends as well as communicate what each needed from the other to complete the application.

One issue we ran into was setting up meeting times. To help with this we used when2meet to find out each other's schedules in order to find the time to set up meetings. With each of us having schedules that sometimes changed week to week, it became harder to ensure everyone could make it to the meeting. This issue also made it difficult to set the times we were available for the coming week(s) as some things changed with our responsibilities outside of this project, which meant planning meetings much later than we preferred.

Another issue we ran into was when we made our survey for the front end user, we found out we needed to complete an IRB form. This form ended up slowing down our process on the survey tremendously and we didn't really get a chance to use the feedback to make adjustments because of it. Due to this experience, I would caution future teams to think about starting their work on any survey as soon as possible to make sure they can improve the front end based on the feedback they get.

5 Future Work

With the work that we've completed, there is still much more that can be done to improve this software. Here we are going to list some of the improvements we think could be made to our software.

5.1 Further Algorithm Analysis

The current software successfully runs and completes what we aimed for. There is room to build and test other algorithms performance to try and achieve a faster and more efficient execution of the tool. We built the algorithm analysis in order to help complete this future work.

5.2 Front End Device Compatibility

For this future work, we are proposing to look into different devices that the user may want to use. With these devices in mind, test and build out a front end that works well for these devices. The current front end was built with only desktop use in mind and that may make users with handheld devices have a harder time navigating and using the software.

6 Conclusion

Throughout our work on this project, we have faced challenges. Despite these challenges, we have succeeded in building a piece of software that is capable of running the tools we planned for. This tool has improved from last year in both the front and back end. With the front end, the implementation of multiple pages has made user navigation simple and easy. We also created a navigation bar to help the users navigate through the software even easier. On the back end, we tested out algorithms with new testing data and built a foundation for future teams to work off of. Overall, we built software that was improved from the previous team on both the front and back end.