Algorithm Deployment Platform

by

Tylor Hess

B.S. Mechanical Engineering, Massachusetts Institute of Technology (2010)

Submitted to the Department of Mechanical Engineering in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering at the MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2016

© Massachusetts Institute of Technology 2016. All rights reserved.

Signature redacted

Certified by ............

Signature redacted

Accepted by .......................................................
Algorithm Deployment Platform
by
Tylor Hess
Submitted to the Department of Mechanical Engineering on May 20, 2016, in partial fulfillment of the requirements for the degree of Master of Science in Mechanical Engineering

Abstract

Algorithm users, such as researchers, clinicians, engineers, and scientists, want to run advanced, custom, new research algorithms. For example, doctors want to run algorithms developed by researchers for clinical applications. These algorithm users see an algorithm as a black box. They want to input data and get results without having to understand the intricacies of algorithm implementation and without having to download, install, configure, and debug complex software. We refer to these algorithm users as black-box users. Researchers and developers create the algorithms; therefore they understand the algorithms' inner workings. We refer to these algorithm developers as glass-box users. There is a need for a platform or technology that allows algorithm developers to efficiently deploy algorithms. We propose the best way to do this is as a web application. Therefore, there is a need to deploy algorithms as web applications without having to learn web development. We developed a web application that enables algorithm users to run developers' algorithms on data stored locally or in cloud storage services. To deploy algorithms as web applications, developers upload their algorithms to cloud computing services. The developer has the option to create an object native to the language in which the algorithm was developed. The platform turns this object into HTML displayed to the algorithm users, so developers can deploy algorithms as web applications without having to learn web development, which is beneficial, since algorithms are often not developed in web-friendly languages. In addition, our platform allows developers to turn the computers that they developed their algorithms on into cloud computing resources, instead of leveraging existing cloud computing services. Using the developer's computer instead of existing cloud computing services is beneficial because their computers were already configured with the appropriate operating system, installed programs, licensed software, etc. to run the algorithms. We evaluated our design with three in-depth interviews, a twenty-one-person focus group, and a survey of six users, who estimated that our platform would significantly reduce deployment time.

Thesis Supervisor: Brian W. Anthony
Title: Principal Research Scientist, Department of Mechanical Engineering

1 Cloud Storage Services: services that store files remotely or “in the cloud,” such as in Dropbox
2 Cloud Computing Services: computational processing power run “in the cloud” and accessed remotely, such as using Amazon Lambda or IronWorker
3 “in the cloud”: roughly “over the Internet”
Contents

Chapter 1   Introduction.................................................................................................................9

1.1  Motivation ..........................................................................................................................10

1.2  Needs........................................................................................................................................14

1.3  Thesis Overview ..................................................................................................................20

Chapter 2   Background and Related Work ..................................................................................22

2.1  Microservices ........................................................................................................................22

2.2  Virtual Machines (VMs) ........................................................................................................23

2.3  Containers................................................................................................................................24

   2.3.1  Linux Containers (LXC and LXD) ..................................................................................24

   2.3.2  Docker Containers ............................................................................................................25

   2.3.3  Open Container Initiative (OCI) Specification and runC ...............................................26

   2.3.4  Summary of Containers ...................................................................................................27

2.4  Container Orchestration and Management.........................................................................28

2.5  Related Work and Our Contribution ..................................................................................28

Chapter 3   Architectural Design..................................................................................................33

3.1  Architectural Overview .......................................................................................................33

3.2  Design Decisions for the Web Application ..........................................................................36

   3.2.1  Web Application ................................................................................................................37

   3.2.2  Single-page Web Application ............................................................................................37

   3.2.3  Angular Web Application Framework .............................................................................38

   3.2.4  Player Directive ................................................................................................................40

   3.2.5  Layering Data over Images ...............................................................................................40

   3.2.6  Scalable Vector Graphics (SVG) ......................................................................................41
List of Figures

Figure 1-1: A figure from Edwin Schouten’s book, IBM SmartCloud Essentials [6], compares Traditional On-Premise computing to the three service models of cloud computing: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). It illustrates which parts of “the stack” the client manages and which parts the vendor manages. .................................................. 10

Figure 1-2: A screenshot of Matlab. The “Package App” button creates a Matlab App, the “Application Compiler” button creates a standalone executable for Windows (.exe), and the “Library Compiler” creates standalone executables for non-Windows operating systems. ............................................. 12

Figure 1-3: A screenshot of Excel. The Function Wizard enables Matlab packages to be run in Excel. .... 13

Figure 2-1: Companies supporting the Open Container Initiative (OCI). Figure reproduced from the OCI Website [33]. ........................................................................................................................................ 27

Figure 2-2: Predominant container orchestration and management offerings and their utilization/evaluation rates. Figure reproduced from The Docker Survey 2016 [46]. ....................... 28

Figure 2-3: A screenshot of Icy’s GUI. Figure reproduced from Icy’s Introduction Tutorial [50]. ........ 30

Figure 2-4: A timeline of the “Flavors of ImageJ” and related software. Figure reproduced from ImageJ website. .............................................................................................................................................. 31

Figure 3-1: The four major components of our platform (i.e., web application, web server, algorithm server, and database) and Dropbox. Arrows [A]-[K] indicate communication between components. ........................................................................................................................................ 33

Figure 3-2: The web server acts as a message broker between multiple users (left) and servers (right). The web server seeks an available server that is configured to run the algorithm, which is Matlab in this case ............................................................................................................................................. 35

Figure 3-3: JavaScript mouse click event callback function for Internet Explorer 8 or less. ............... 37
Figure 3-4: JavaScript mouse click event callback function according to the W3C standard. A mature
JavaScript library, like jQuery [61], allows developers to use the standardized callback function,
even with Internet Explorer 8. .............................................................................................................. 37

Figure 3-5: Notice how raster (i.e., bitmap) image gets pixelated, when scaled, and a vector image does
not. Image reproduced from Wikimedia [64]. .............................................................................................................. 42

Figure 3-6: The “Dropbox Chooser” popup window used to select files. ............................................................ 44

Figure 3-7: An example of a file as JSON. The “Dropbox Chooser” returns an array of these objects, one
for each file chosen. .................................................................................................................................................. 45

Figure 3-8: A static file server receives a request from a url (e.g., http://url.com/path/to/file.html) and
responds with the file located at the path (i.e., /path/to/file.html). ........................................................................ 46

Figure 3-9: A simple, static file server in JavaScript, using Node.js’s Express. .................................................. 46

Figure 3-10: The web server acts as a message broker between multiple users (left) and servers (right).
The web server seeks an available server that is configured to run the algorithm, which is Matlab in
this case. .................................................................................................................................................. 47

Figure 3-11: Only sending run_id forces the web server and algorithm server to download the rest of the
information from Firebase, which enforces security rules. This architecture ensures that only users
with appropriate permissions can execute algorithms on a server and prevents savvy users from
sending potentially-malicious commands directly to a server. ........................................................................ 48

Figure 3-12: A user object represented in JavaScript Object Notation (JSON). .................................................. 52

Figure 3-13: A JSON example of a hierarchical data structure. Notice how one user cannot be in multiple
groups.................................................................................................................................................. 54

Figure 3-14: A JSON example of a flattened data structure. It also allows for many-to-many relationships
(i.e., a user can belong to multiple groups, and a group can have many users), and users or groups
can be downloaded, as needed, without downloading the entire list of users or groups. .................... 54
Figure 3-15: A “best practice” example of a flattened data structure. Every user object directly references its groups and every group object directly references its users, which prevents security issues from arising.  

Figure 4-1: Translation file, runDemoTemplateTracking.m, translates information provided by the platform.  

Figure 4-3: Edit Algorithm screen.  

Figure 4-4: Edit Algorithm screen with Dropbox Chooser popup window.  

Figure 4-5: Edit Algorithm screen with main algorithm file and supporting files.  

Figure 4-6: Run Algorithm screen with data uploaded as frames and viewed in a video player.  

Figure 4-7: Run screen with data uploaded as frames, viewed on a video player, and a rectangle drawn on the first frame, which serves as input into the algorithm.  

Figure 5-1: Six participants’ (Part1-Part6) survey responses and average response (Average).  

Figure 5-2: All six participants’ (Part1-Part6) estimated time to deploy an algorithm without this platform (blue), the first time using the platform (red), subsequent times using the platform (green) on a linear vertical axis (left) and a logarithmic vertical axis (right).
List of Tables

Table 2-1: Icy compared to other noncommercial open-source software packages. Table reproduced from Nature article [48] ................................................................. 31

Table 5-1: Average and standard deviation (STD) of survey results on a scale from 1 to 10. .................. 68

Table 6-1: Survey results, average, and standard deviation on a scale from 1 to 10. .......................... 73

Table 6-2: Estimated time to deploy an algorithm, converted to hours .................................................. 74
Chapter 1  Introduction

Like any buzzword, the term “cloud” lacks a clear, specific, consistent definition. The watering-down of its definition is often attributed to it being used to describe a variety of very different technologies\(^4\) [1], and its rapid adoption by non-technical groups, such as media, marketing, and management teams\(^5\) [2]. Nonetheless, many have attempted to define this impalpable term\(^3\) [3] [4] because definitions are useful and provide clarity.\(^6\) Generally, if a resource is “in the cloud” or “on the cloud,” it is available over the Internet; and if a resource is moved “to the cloud,” it is outsourced to a vendor and accessed by a client over the Internet (Figure 1-1).

According to the National Institute of Standards and Technology (NIST) [4], “cloud computing” is scalable, on-demand, shared computational resources, including applications, data, runtime, operating systems, servers, storage, or networking. A vendor maintains, monitors, or measures these resources, and a client accesses these resources over the Internet (Figure 1-1). Cloud computing can be deployed privately, publically, to a community, or as a hybrid. The service models of cloud computing are as follows:

1. **Software as a Service (SaaS):** Web applications, such as Gmail, ran on cloud infrastructure and accessed via the Internet, instead of desktop or mobile applications, such as Outlook Desktop, which are downloaded, installed, and ran on a device.\(^5\)

2. **Platform as a Service (PaaS):** An environment, such as PHP web page hosting, for running application code, such as programming languages, libraries, services, or tools supported by the

\(^4\)“The variety of technologies in the Cloud makes the overall picture confusing” [3].

\(^5\)“The hype around Cloud Computing further muddies [its meaning]” [3].

\(^6\)“It’s true that any attempt to draw sharp boundaries around abstract terms involves some arbitrary choices. But unless we’re willing to draw the line somewhere, these concepts will remain confusing and difficult to use. Definition brings clarity. And when it comes to concepts that are so fundamental to performance, no organization can afford fuzzy thinking” [89].
platform, on cloud infrastructure without managing the underlying infrastructure, such as runtime, operating systems, servers, storage, or networking. [4] [5]

3. **Infrastructure as a Service (IaaS):** Physical or virtualized hardware, such as servers or storage, or networking, such as a firewall, hosted over the Internet and maintained by a third party, such as Amazon Web Services. [4]

![Diagram comparing Traditional On-Premise computing to the three service models of cloud computing: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). It illustrates which parts of “the stack” the client manages and which parts the vendor manages.]

**Figure 1-1:** A figure from Edwin Schouten’s book, IBM SmartCloud Essentials [6], compares Traditional On-Premise computing to the three service models of cloud computing: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS). It illustrates which parts of “the stack” the client manages and which parts the vendor manages.

### 1.1 Motivation

When a potential user wants to run a developer’s algorithm, the user invests time and effort to install, configure, debug, and run the software stack required for the algorithm. Potential users may need to be quite savvy. They may need to know, or be willing to learn, specific terminal commands, programming languages, development environments, operating systems, etc. Even if they already have the requisite knowledge and skill, the process itself can be fairly involved, requiring time to configure and debug,
money for software licenses and hardware, or other resources that a potential user does not have. These requirements become barriers, which can be too high to surmount, preventing potential users from running the developer's algorithm. This concern is illustrated by the following Stack Overflow user's request for a solution to run his algorithm “without downloading the code, installing Matlab, etc.”

“A colleague and I have spent a few years developing a really cool Matlab application, MDLcompress. Within Matlab, I can type MDLcompress('filename.txt') and it will tell me all sorts of really cool stuff about the contents of filename.txt. We'd like to allow other people to use MDLcompress without downloading the code, installing Matlab, etc. Ideally, we'd have a simple web page where they selected a file from their machine, it got uploaded to my workstation (which is already running tomcat for other purposes, if that makes things easier), kicked off a process along the lines of matlab < MDLcompress.m filename.txt > results.txt and then displayed results.txt in their browser or showed a link to let them download it. Trouble is, my Matlab skills far exceed my web skills. Google has 100 generic tutorials, but nothing as simple as I want, at least not specific to Matlab.” [7]

- Stack Overflow user

All five responses to this question recommend that the original poster (OP) learn some form of web technology, including the top response, which recommends a CGI tutorial. However, the OP professes his weak web development skills and explicitly asks for something simpler than the tutorials available online, implying little interest in learning web development. The tools and approach to desktop, mobile, and web development are specific and distinct from those of algorithm development, which creates a barrier for algorithm developers, who want to deploy their algorithms.

---

7 Original Poster (OP): a user who originally asked a question or submitted a post to start an online discussion
8 Common Gateway Interface (CGI): a web standard for accessing executable programs hosted on a server
In a video entitled “Application Deployment with MATLAB” recorded on April 23, 2015, Mathworks Senior Application Engineer, Bonita Vormawor, suggests five ways to deploy Matlab applications and components [8]:

1. Matlab App
2. Standalone Executable
3. Macro-enabled Excel Spreadsheet
4. Desktop or Mobile Application
5. Web Application

The first three deployment methods Ms. Vormawor suggests, Matlab App, Standalone Executable, and Macro-enabled Excel Spreadsheet, have easy-to-use, point-and-click solutions, Package App, Application Compiler, Library Compiler, and Excel Function Wizard, respectively. Most importantly, these point-and-click interfaces do not require the developer to know any language other than Matlab.

Figure 1-2: A screenshot of Matlab. The “Package App” button creates a Matlab App, the “Application Compiler” button creates a standalone executable for Windows (.exe), and the “Library Compiler” creates standalone executables for non-Windows operating systems.
If a developer wants to deploy to desktop, mobile, or web applications, the first step is to use the Application Compiler or Library Compiler (Figure 1-2) to create a standalone executable. The second step is to develop a desktop, mobile, or web applications. As illustrated with the Stack Overflow question and discussed above, this second step, which Ms. Vormawor glazes over, can be fairly involved. In a half-hour-long Mathworks video entitled “Matlab for the Masses: Deploying to External Clients,” Peter Orr tells the story of how he started Intuitive Analytics [9]. In his story, deploying a Matlab application is not a trivial task.

“I saw people were making decisions without very good information. So on a few nights and weekends… I got dangerous enough to hack out some Matlab code, and I could just tell there was a ton I could get done in a very short period of time… I coded up some Monte Carlo simulation stuff, some optimization code, and that got rolled out to 200 desktops… But what we’re really going to do today is look at the architecture we’ve set up and build over the past 7 years… How do you use Matlab code as a core component of a solution outside the organization? … We beat our head against the wall and tried a
bunch of things... It's messy. It's a long process to figure out a good solution to various problems, whether you're talking about pulling data from various sources, or crunching that data in memory-efficient ways. These are hard problems... Eventually, over time, we came up with what I'm about to show you... Excel is what the user sees. It's the interface they use. We don't have much business logic in Excel. We do a lot of data logic and conditional formatting. We do some basic error checking. For the actual business logic... we go into the .NET layer for the presentation layer, formatting, array abstraction, forms, Windows Communication Foundation (WCF), function signature abstraction, [and] object relational mapping... The actual calculations happen in the Matlab Component Runtime (MCR)... but we wrap the MCR into a separate process, which lives outside Excel... in our application suite called SmartModels... Then, of course, there are bits of Java for custom GUIs, web services communication, https connections, [and] report generation.” [9]

- Peter Orr, Intuitive Analytics Founder

These struggles are not limited only to Matlab developers. Similar challenges are found in deploying almost any other language, especially languages that are not web-friendly, such as Matlab, Julia, R, etc. In this thesis, we explore the details of these challenges and offer a potential solution.

1.2 Needs

Our needs discovery process began with an in-depth exploration of the needs of three potential users. This exploration occurred over the course of three sessions. Each session was approximately two-hours long for a total of about six interview hours per person. The first session was a pre-observation interview. We entered the interview with a pre-observation questionnaire but willingly strayed from the script, in accordance with the third principle of customer development interviews: “Go with the Flow.” During the second session, we observed the participants use the platform to deploy their own custom algorithm. The

---

9 Graphical User Interface (GUI)
third and final session was a follow-up interview. Again, we arrived prepared with a post-observation questionnaire but strayed from the script, when appropriate. Over the course of approximately 18 hours of interviews, we collected 142 raw user insights. These insights came in the form of raw user statements, observations, and feature requests, which we translated into six primary interpreted needs.

Our interview process follows Steven Blank’s Customer Development and Eric Ries’ Lean Startup models. Customer Development Interviews were first introduced in Blank’s book The Four Steps to the Epiphany [10]. The Four Steps to the Epiphany inspired Ries [11] to launch the Lean Startup movement [12] [13]. The Lean Startup community continued developing these interview strategies [14]-[15]. Our approach to interviewing borrows heavily from this work and can be summarized with the following ten guiding principles:

1. **Find Users**: “Who you are talking to is as important as what they are saying” [14]. Do not expect people to find you [16]. “Go where the fishes swim.” [16]

2. **“Have a Plan”** [16] [17]: “Know your goals and questions ahead of time” [18]. Prepare three to five questions [19]. “Have specific hypothesis you’re trying to validate” [20]. Prepare a script [17]. “Bring a partner” [19] [17] [14] to take care of note-taking and the peripherals, so the interviewee can have your full-and-undivided attention [17]. If solo, record the meeting [17] or take quick notes [18] [14] [17]. Meet face-to-face, one-person-at-a-time [17] [18]. “Work to their schedule” [21], but choose a location you are comfortable with [17].

3. **“Go with the Flow”** [22]: When the focus of the conversation wanders, gently guide it back to relevant topics [16]. Sometimes it is hard to plan in advance where a conversation will go. If it goes somewhere unexpected, but relevant, be willing to stray from the script. “In preparing for battle I have always found that plans are useless, but planning is indispensable.” Dwight D. Eisenhower

Minimize yes or no questions [18] [16]. Make it hard for them to answer with one-word responses or simple phrases. Reflect. “Parrot” their answers back to them [18] [17]. Occasionally, misrepresent their answer to see if they correct you [18].

5. “Smile” [17] and “Listen” [18] [19]: “Try to shut up as much as possible” [18]. “Listening is uncomfortable” [16]. Learn to love the silence [16] [17]. Aim to be listening 90% of the time [17]. “Don’t just listen to what they say, listen to how they say it” [14]. Show that you are listening with non-verbal cues, encouragement, and short reflections, but be careful not to influence [21].

6. “Get Psyched to Hear Things You Don’t Want to Hear” [18]: Defer Judgment [17]. It is easy to get people to say what they think you want to hear. Avoid leading questions, confirmation bias, or hypothesis bias [23]. It is hard to get people to say what they think you do not want to hear. It is easier to agree with other people than to disagree with you, so reference anonymous people’s opposing opinions [17]. “Encourage Complaints” [17]. “Follow emotions” [17]. “Disarm ‘Politeness’ Training” [18]. People have been taught to be polite. Ask them to be brutally honest. Explain that this is the best way to help. To build something people do not care about is a huge waste and the worst thing that could happen [18].

7. “Separate Behavior and Feedback” [18]: “Start with behavior” [21]. “Ask about facts” [19]. Focus on past behavior, actual situations, and actual events [19]. Avoid hypothetical situations, speculation, abstract feelings, or creative descriptions [18] [16]. Then, show (do not explain) an idea [16] and solicit opinions and feedback. Showing the idea beforehand is “leading the witness.”

8. Be the Architect: Have the courage and “vision to look deep into a problem and come up with the right solution” [15]. Do not ask the user to define the solution [15].

9. Wrap Up: Be as quick, organized, and professional as possible. Finish on time [19]. Unless the interviewee is enthusiastic or insistent, “don’t overstay your welcome” [19]. Provide a short summary of what you heard them say [19]. “Ask for introductions” [18]. “Thank them” [21].
10. **Follow Up:** That same day, make time to reflect alone [19]. Identify pain/pressure points [23].

“Look for patterns” [18]. “Apply judgment” [18]. After individual reflection, debrief with any partners [19]. Within a few days, send a follow-up email summarizing what they said, highlighting how they helped, soliciting any final thoughts, asking for any additional introductions, and most importantly, thanking them [19].

The following is additional advice for interviewing: “Listening is uncomfortable” [16]; “Talking to strangers is unnatural” [16]; “Interviewing is a skill” [23]; “It takes practice” [14]; and it gets easier [14]. In the meantime, have fun, be compassionate, and forgive mistakes.

After 18 hours of customer development interviews, we gathered 142 insights, which we translated into six primary interpreted needs. According to Ulrich and Eppinger [22], “each [raw user] statement or observation may be translated into any number of [interpreted] needs... Needs are the result of interpreting the need underlying the raw data gathered from the customers... and are expressed as written statements.” Ulrich and Eppinger provide five guidelines for writing need statements: “Express the need in terms of what the product has to do, not in terms of how it might do it; Express the need as specifically as the raw data; Use positive, not negative, phrasing; Express the need as an attribute of the product; Avoid the words ‘must’ and ‘should.’” Using these principles, we translated the 142 insights into six primary interpreted needs.

We borrow the terms “black-box user” and “glass-box user” from Intuitive Analytics Founder, Peter Orr, who defines them as follows:

“I looked around and I started putting people into two different groups... The first type of person, I call a ‘black-box’ person... A black-box person needs to take data; crunch numbers; put things into a box, push a button, and expect things to come out the other side. And they better come out, ideally, very quickly and very accurately. And if they put something wrong into a box, ideally, it behaves nicely... [They] are not Matlab [developers]... [You tend to see more of them, the farther up the chain you go], particularly in large organizations... I’m sure you know a ton of [these people]... And there are far more of them.
frankly, than the other kind of person. The other type of person is what I call a 'glass-box' person. The glass-box person has the desire, sometimes even a need, to get into whatever is happening in that box. They want to understand what makes the gears turn, what makes them go faster, what makes them grind to a halt, and in my own biased opinion, those people are more valuable than black-box people.” [9]

The six primary needs we identified are as follows:

1. Glass-box users can deploy algorithms quickly and with minimal effort.
2. Black-box users can run algorithms without specialized knowledge.
3. Researchers increase the visibility and impact of their work.
4. Glass-box users can deploy algorithms as desktop, mobile, or web applications.
5. Glass-box users can deploy algorithms without having to learn desktop, mobile, or web application development.
6. Glass-box users have the option to turn the computer they developed the algorithm on into cloud computing resources.

We interpreted these six primary needs predominantly from the following user statements, which are quotations from the customer development interviews we conducted. Examples of representative user statements follow:

“I first learned to program Java in high school. I thought the programs I made were really cool, and I wanted to show them to people, but I didn’t know how. I only knew how to run Java in CodeWarrior, the IDE installed on my school’s Macintosh computers, which wasn’t available for Windows computers like mine at home. So I couldn’t show anyone what I was working on, except at school. Years later, when I learned JavaScript, it was very liberating. JavaScript is the language of the Internet, so it’s easily run on

---

10 Integrated Development Environment (IDE)
any computer with an Internet browser, and it’s the easiest language to turn into a web page. It was so much more fun learning JavaScript because it was so simple show everyone what I was working on.”

“There are so many people sitting in labs like this, who are gurus at Matlab, Python, C, Haskell, … but deployment is still a big question mark. Imagine if you really, really lowered the barrier for that. They can share using email, github, and the web, but very rarely do they deploy it with an interface, unless they also happen to know all the stuff you need to know for web development.”

“I need my doctor to annotate my images, and it’s really tedious, and I feel bad making him come here for three hours, when he can do it in the comfort of his own home with his two-year-old running around.”

“If you make it really easy to deploy, you can help advanced, academic algorithms to actually see the light of day.”

“It would be nice if I could just send [my collaborator] a link and say, ‘When you have a chance, can you do this?’ It would make my life easier.”

 “[My labmate] wrote an iPad app to do this kind of stuff, so the guy could do it from home, and I don’t know how to do that, so I can’t do that… I don’t have months to learn iPad development and develop it.”

“I wanted to compare the features detected by different computer vision algorithms run on the same set of images. It took weeks to install, configure, debug, and run these algorithms… Everything was different. They required different operating systems, languages, dependencies… everything.”
1.3 Thesis Overview

We translated raw user statements, observations, feature requests, and other insights into interpreted needs. Based on these interpreted needs, we propose that there is a need for a platform that allows glass-box users to deploy algorithms as web applications without having to learn web development and a web application that enables black-box users to run glass-box users’ algorithms on data stored locally or in cloud storage services. To deploy algorithms as web applications, glass-box users upload their algorithms to cloud computing services.

We developed such a platform. The developed platform could help researchers demonstrate their work, while presenting a keynote, giving a talk, speaking at a conference, giving a lecture, presenting a poster, or meeting with a research sponsor. It may serve as a teaching aid in a classroom setting. Students could explore algorithms without having to download, install, configure, and debug. It could serve as an archiving tool for the major releases of a researcher’s algorithm. It may help with knowledge transfer. According to our interviews, researchers often first look at the black-box input and output of an algorithm, before delving deeper into its inner workings, so our platform could be helpful, whether onboarding a new lab group member or sharing previous work with a labmate, who is working on a similar problem. Our platform may be particularly valuable to researchers collaborating with non-developers, such as doctors. In Section 1.1, Peter Orr describes black-box and glass-box users. His experience was in the financial sector, but his insights can be applied to almost any sector, where glass-box users want to deploy algorithms to black-box users: research, education, healthcare, manufacturing, finance, energy, entertainment, transportation, IT, management, services, and other sectors. [24]

In Chapter 2, Background and Related Work, we describe microservices as small, independent cloud-computing resources. We explain how virtual machines (VMs) powered microservices and how

---

11 Cloud Storage Services: services that store files remotely or “in the cloud,” such as in Dropbox
12 Cloud Computing Services: computational processing power run “in the cloud” and accessed remotely, such as using Amazon Lambda or IronWorker
13 “in the cloud”: roughly “over the Internet”
virtual environments, called containers, are now replacing VMs in many applications. Linux Containers (LXC) evolved into Docker Containers, which are the industry de facto standard. Docker donated approximately five percent of its codebase and joined with other cloud computing vendors to form Open Container Initiative (OCI). We discuss projects similar to our platform and outline how our contributions differ from these related works.

In Chapter 3, Architectural Design, we outline the platform's architecture and go into detail about the design decisions that went into creating the four major components: web application, web server, algorithm server, and database.

In Chapter 4, User Experience (UX), we walk through the user experience of deploying a Normalized Cross-Correlation Algorithm.

In Chapter 5, Evaluation, we evaluate our design with a twenty-one-person focus group and survey six users, who estimated that our platform would significantly reduce deployment time.

In Chapter 6, Conclusion, we summarize the thesis and make suggestions for future work.
Chapter 2  Background and Related Work

In this chapter, we describe microservices as small, independent cloud-computing resources. We explain how virtual machines (VMs) powered microservices, and how virtual environments, called containers, are now replacing VMs in many applications. Linux Containers (LXC) evolved into Docker Containers, which are the industry de facto standard. Docker donated approximately five percent of its codebase and joined with other cloud computing vendors to form Open Container Initiative (OCI) [25]. We discuss projects similar to our platform and outline how our contributions differ from these related works.

2.1 Microservices

Microservices, also known as compute services, runtime-as-a-service, and serverless computing,¹⁴ are small, independent cloud-computing resources that often abide by Robert Martin’s Single Responsibility Principle, “Gather together those things that change for the same reason, and separate those things that change for different reasons” [26]. This principle encourages discrete, decoupled services that can be updated and re-deployed without changing anything else [27].

Microservice-based architectures often adopt the Unix philosophy, “Do one thing and do it well” [28]. When architected this way, microservices form small building blocks for larger, more-complex applications. Microservices use APIs¹⁵ to communicate with internal components, such as databases, data storage, etc., external applications, such as desktop, mobile, or web applications, and other microservices.

Microservices are first discussed in 2007 with Griffin and Pesch’s “A Survey on Web Services in Telecommunications” [29]. As early as 2011, companies like Iron.io and Serverless.com began offering

¹⁴ “Serverless computing” is a bit of a misnomer because the prefix “-less” implies “without” [90], and serverless computing is not “computing without servers.” It is, however, “computing without having to manage servers,” which is presumably where the term comes from.
¹⁵ Application Programming Interface (API)
microservice solutions [30]. It was not until November 2014, when Amazon made a soft "preview release" of AWS Lambda, that the first major cloud service provider offered a microservice solution. Since then, all major cloud service providers, such as Amazon Web Services (AWS), Microsoft, Google, and IBM, and many smaller companies, began offering microservices that: "run code for virtually any type of application or backend service with zero administration... without provisioning or managing servers" [31], "efficiently isolate code and dependencies of individual tasks so they can be processed on demand" [32], "automatically trigger from AWS services or from any web or mobile app" [31], "give you the flexibility to power any task in parallel at massive scale" [32], "you pay only for the compute time you consume [without] charging when your code is not running" [31], etc. Many of these microservices were released in the past few months. In February 2016, IBM and Google released their alpha versions of OpenWhisk\(^\text{16}\) and Google Cloud Functions,\(^\text{17}\) respectively, and in April, Microsoft released its alpha of Azure Functions. Of these microservice offerings, IronWorker\(^\text{18}\) is the oldest, most mature, and most robust microservice, and AWS Lambda\(^\text{19}\) is the oldest, most mature, and most robust microservice from a major cloud service provider. Most, if not all, of these microservice offerings are based off of, or at least utilize, containers (Section 2.3), even if they did not originally.

2.2 Virtual Machines (VMs)

Prior to containers (Section 2.3), microservice-based architecture relied heavily on virtual machines (VMs). VMs take minutes to start, are heavy in that they are allocated a lot of resources, and take more time to configure and use, as compared to containers, which take seconds to start, are lightweight in that they share resources, and have fast and easy command-line tools. The advantages of VMs are that: VMs are fully isolated, whereas containers are only partially isolated; VMs can virtualize any operating system,

\(^\text{16}\) IBM OpenWhisk: Microservice that runs JavaScript, Swift, or Dockerfile with Triggers, Rules, and Actions.
\(^\text{17}\) Google Cloud Functions: JavaScript-only microservice with Cloud Events, Triggers, and Cloud Functions.
\(^\text{18}\) IronWorker: The oldest, most mature microservice that runs "every popular language – Ruby, Python, PHP, Java, .Net, Node.js, Go, and more" [32].
\(^\text{19}\) AWS Lambda: The oldest, most mature microservice available from a major cloud-computing vendor, but is not as open as others, integrating only with other Amazon services.
whereas containers are currently only available with Linux operating systems; and VMs guarantee resources at a hardware level, whereas containers share resources, although sharing resources is an advantage, when running many lightweight processes, processes that are not resource intensive, on one machine.

2.3 Containers

“In the past two years, there has been rapid growth in both interest in and usage of container-based solutions” [33]. All major cloud providers, such as Amazon Web Services (AWS), Microsoft, Google, and IBM, offer container-based microservice solutions, and most non-container-based microservice solutions accept a container image format. “The rapid growth of the Docker project has served to make the Docker image format a de facto standard for many purposes” [33]. Most of the major players in the container ecosystem have joined with Docker to form the Open Container Initiative (OCI) to establish standards around a container format and runtime. repeat of above?

2.3.1 Linux Containers (LXC and LXD)

Linux Container (LxC) or LinuX Container (LXC) was the first container technology. “Containers offer an environment as close to possible as the one you would get from a VM but without the overhead that comes with running a separate kernel and simulating all the hardware. This is achieved through a combination of kernel security features such as namespaces, mandatory access control, and control groups” [34]. LXC v1.0, the first production version, was released November 2014 [35], but Linux Containers served as the foundation for Docker Containers (Section 2.3.2) since at least March 2013, well before LXC’s production release.

---

20 Non-Linux operating systems (e.g., Windows and OS X) can run containers inside a Linux VM, but not natively, although Windows and OS X may run containers natively in the near future. [91]
LXC has since been superseded by LXD.\textsuperscript{21} “LXD is the new LXC experience. It offers a completely fresh and intuitive user experience with a single command line tool to manage your containers. Containers can be managed over the network in a transparent way through a REST API. It also works with large-scale deployments by integrating with OpenStack. LXD was announced in early November 2014 and is still under very active development” [34].

“LinuxContainers.org is the umbrella project behind LXC, LXD, [and others]. The goal is to offer a distro and vendor neutral environment for the development of Linux container technologies. Our main focus is system containers” [34].

2.3.2 Docker Containers

Docker containers mainstreamed virtual environment technology. Much like Linux Containers, "Docker containers wrap up a piece of software in a complete filesystem that contains everything it needs to run: code, runtime, system tools, system libraries – anything you can install on a server. This guarantees that it will always run the same, regardless of the environment it is running in” [36]. However, Docker extends Linux Containers (LXCs) by adding: GitHub-like distributed version control, hosting, and sharing ecosystem; portability across machines; optimization for the deployment of applications; automatic assembly of a container from source code; component reusability; etc. [37] [38] [39].

In March 2013, platform-as-a-service provider dotCloud open sourced Docker [40]. “Docker allows you to package an application with all of its dependencies into a standardized unit for software development” [36]. Until March 2014, Docker used LXC as its default driver. In March 2014, Docker replaced LXC with libcontainer [41]. In June 2015, Docker donated libcontainer [42] to Open Container Initiative (OCI) to serve as the foundation for OCI’s new effort to create an “open industry standards around container formats and runtime” [33]. Libcontainer was Docker’s container format and runtime

\textsuperscript{21} LXD is not an abbreviation. It is simply the name of the Linux Container project that came after LXC.
was the “heart of Docker” [43], and constituted approximately five percent of Docker’s codebase [25]. In 2015, OCI merged libcontainer and appc to form runC [33].

2.3.3 Open Container Initiative (OCI) Specification and runC

“While the rapid growth of the Docker project has served to make the Docker image format a de facto standard for many purposes, there is widespread interest in a more formal, open, industry specification, which is: not bound to higher level constructs such as a particular client or orchestration stack; not tightly associated with any particular commercial vendor or project; portable across a wide variety of operating systems, hardware, CPU architectures, public clouds, etc.” [33]

“The Open Container Initiative is a lightweight, open governance structure, formed under the auspices of the Linux Foundation, for the express purpose of creating open industry standards around container formats and runtime” [33]. The OCI has two main projects: The OCI Specification [44] and runC [45], the implementation of this specification, which “is a CLI\(^{22}\) tool for spawning and running containers according to the OCI specification” [33].

“The OCI was launched on June 22nd 2015” [33]. In June 2016, the contributors plan on releasing the first draft of the specification and the first production version of runC, v1.0.0.

\(^{22}\) Command-Line Interface (CLI)
2.3.4 Summary of Containers

It seems we are at the beginning of a new era in cloud computing, where faster, more lightweight containers will replace heavier virtual machines, whenever projects do not need resources guaranteed at a hardware level or full isolation. In June 2016, the Open Container Initiative (OCI) will release the first production-ready version of runC, which is an implementation of the OCI Specification of an international, industry standard for container formats and runtime.
2.4 Container Orchestration and Management

According to The Docker Survey 2016 [46], there are three predominant container orchestration and management offerings: Docker Swarm, Google Kubernetes, and Amazon EC2 Container Service. Docker Swarm is “five times faster than Kubernetes to spin up a new container” and “seven times faster than Kubernetes to list all the running containers,” and “is the only container orchestration and management provider that is cloud agnostic.”

![Diagram showing usage/evaluation rates of container orchestration and management solutions.](image)

Figure 2-2: Predominant container orchestration and management offerings and their utilization/evaluation rates. Figure reproduced from The Docker Survey 2016 [46].

2.5 Related Work and Our Contribution

Of the projects we found during our research, Algorithmia.com [47] is the closest to our platform. Algorithmia is a marketplace for algorithms. Developers can upload their algorithms, and users can integrate these algorithms into their desktop, mobile, or web application “with less than 5 lines of code” [47]. Unfortunately, this still requires algorithm users to know desktop, mobile, or web app development, so doctors and other non-developers still cannot use the algorithms, unless they partner with a developer, which was always an option, even without Algorithmia.

With our platform, developers can deploy algorithms as web applications, so black-box users, such as doctors, can use the algorithms without any specialized knowledge, such as programming skills. The
developer has the option to create an object native to the language in which the algorithm was developed. The platform turns this object into HTML displayed to the algorithm users, so developers can deploy algorithms as web applications without having to learn web development, which is beneficial, since algorithms are often not developed in web-friendly languages. To make it even easier for the algorithm developer, our platform allows them to turn the computers that they developed their algorithms on into cloud computing resources. Using the developer’s computer instead of existing cloud computing services saves them time and effort because their computers were already configured to run the algorithms with the appropriate operating system, installed programs, licensed software, etc.

As a prototype, not all of our platform’s features are fully developed. The most prominent of all features is security, which will have to be greatly improved before deploying, even in limited or controlled circumstances. Part of this security involves isolating the environment in which the algorithm is run to prevent user-uploaded code from executing malicious attacks. Virtual machines (VMs) can provide guaranteed isolation at a hardware level, but the partial isolation of containers will likely be sufficient.

We aspire to develop many of capabilities and features similar to Icy [48] [49], except that Icy is a desktop application and we are developing a web application. Icy is “an open community platform for bioimage informatics” [50] and “provides the software resources to visualize, annotate and quantify bioimaging data” [50]. Icy uses a GUI based on well-known components, such as ribbon (Figure 2-3 [1]), navigator (Figure 2-3 [2]), and look-up table (Figure 2-3 [3]).
Icy has native ImageJ integration. ImageJ is “an open source Java image processing program” [51].

Table 2-3 compares Icy to ImageJ, Fiji [52], and other noncommercial open-source software packages.

NIH Image evolved into ImageJ, Fiji, and many other flavors [53] [54] (Figure 2-4).
<table>
<thead>
<tr>
<th>First release</th>
<th>Funding</th>
<th>People maintaining the software</th>
<th>Language</th>
<th>Multi-operating system bundle</th>
<th>Image formats</th>
<th>Data types</th>
<th>Available plug-ins</th>
<th>Native 3D visualization</th>
<th>Plug-in versioning</th>
<th>Native software interoperability</th>
<th>Community connection</th>
<th>Plug-in error report</th>
</tr>
</thead>
<tbody>
<tr>
<td>Icy</td>
<td>April 2021</td>
<td>Institut Pasteur, France BioImaging</td>
<td>Java</td>
<td>Yes</td>
<td>&gt;100 (bio-formats)</td>
<td>3D and time, 3 x 8-bit color and 8,16,32-bit grayscale</td>
<td>-120</td>
<td>VTK</td>
<td>In-app updater rollback via CMS</td>
<td>ImageJ, VTK, and Matlab (.dat)</td>
<td>Web forum and in-app chat</td>
<td>NA</td>
</tr>
<tr>
<td>ImageJ</td>
<td>September 1997</td>
<td>US National Institutes of Health, Max Planck Institute ETH and University of Zurich, EMBL and University of Wisconsin-Madison</td>
<td>Java</td>
<td>Yes</td>
<td>7 (native)</td>
<td>nD (imglib); any data type</td>
<td>-500</td>
<td>NA</td>
<td>In-app updater</td>
<td>NA</td>
<td>Web mailing list</td>
<td>NA</td>
</tr>
<tr>
<td>Fiji</td>
<td>November 2008</td>
<td>Max Planck Institute ETH and University of Zurich, EMBL and University of Wisconsin-Madison</td>
<td>Java</td>
<td>Yes</td>
<td>&gt;100 (bio-formats)</td>
<td>nD (imglib); any data type</td>
<td>-150</td>
<td>NA</td>
<td>Not found</td>
<td>ImageJ and Fiji and OMERO</td>
<td>Web mailing list, forum and wiki</td>
<td>NA</td>
</tr>
<tr>
<td>ImageJ2</td>
<td>In development</td>
<td>US National Institute of General Medical Sciences, Max Planck Institute ETH and University of Zurich, EMBL, and University of Wisconsin-Madison</td>
<td>Java</td>
<td>Yes</td>
<td>&gt;100 (bio-formats)</td>
<td>nD (imglib); any data type</td>
<td>Not found</td>
<td>NA</td>
<td>Not found</td>
<td>ImageJ, Fiji and OMERO</td>
<td>Web mailing list, forum and wiki</td>
<td>NA</td>
</tr>
<tr>
<td>BioImageXD</td>
<td>February 2006</td>
<td>University Jyväskylä, University Turku</td>
<td>C++ or Python</td>
<td>No</td>
<td>Not found</td>
<td>&gt;100 (bio-formats)</td>
<td>12 (native)</td>
<td>VTK</td>
<td>Not found</td>
<td>ImageJ, Matlab, R and SQL</td>
<td>Web forum and wiki</td>
<td>NA</td>
</tr>
<tr>
<td>Cell Profiler</td>
<td>December 2005</td>
<td>Broad Institute</td>
<td>Python</td>
<td>No</td>
<td>Not found</td>
<td>&gt;100 (bio-formats)</td>
<td>2D and time; not found</td>
<td>NA</td>
<td>NA</td>
<td>Not found</td>
<td>ImageJ, Matlab, R and SQL</td>
<td>Web forum</td>
</tr>
</tbody>
</table>

Table 2-1: Icy compared to other noncommercial open-source software packages. Table reproduced from Nature article [48].

**History of ImageJ**

Flavors of ImageJ

- NIH Image
- ImageJ
- ImageJ1
- ImageJ2
- ImageSXM
- WCIF ImageJ
- ImageJX
- MBF ImageJ
- Fiji
- Bio-Formats
- BioImageXD
- CellProfiler
- OMERO
- Bio7
- Endrow
- Micro-Manager
- BoneJ
- Aida
- MitoBio
- Icy

Related software

- ITK
- VisBio
- KNIME
- Bio-Formats
- BioImageXD
- CellProfiler
- OMERO
- Bio7
- Endrow
- Micro-Manager
- BoneJ
- Aida
- MitoBio
- Icy

Figure 2-4: A timeline of the “Flavors of ImageJ” and related software. Figure reproduced from ImageJ website.
Another project that our research uncovered is Cloud’N’Sci.fi [55]. Cloud’N’Sci is a people-in-the-loop service, whereas with Algorithmia, developers can upload their algorithms themselves. Cloud’N’Sci “is a Finnish company founded in 2010 with a single goal in mind: to turn algorithms into business” [55]. “No easy way exists to commercialize algorithms, so they are forgotten into research chambers or published for free. Rarely does any new algorithm end up into real world applications due to integration challenges and even when it does, the original developer gets nothing. Cloud’N’Sci aims to fix this grievance and become the middleman between those who can [glass-box users] and those who need [black-box users]” [55]. “The Cloud’N’Sci.fi marketplace offers algorithm developers a shortcut to global markets and makes their algorithms available to a wide range of business applications” [55]. “Algorithmic solutions can be offered and evaluated at low risk and expenses. When a perfect match is found between a use case and a solution, the customer and the algorithm provider may enter into a long-term mutual commitment. The solid legal framework behind Cloud’N’Sci agreements guarantees that further negotiations can go as deep as necessary. In short, it's a win-win for all parties: developers, users and Cloud’N’Sci” [55].

Other solutions exist that allow algorithm users to access algorithms, but does not allow developers to deploy them, such as BigML [56], Google Cloud Prediction API [57], Abdullah Qasem’s Master’s Thesis “A Framework for Provisioning Algorithms as a Service” [58], and others.
Chapter 3  Architectural Design

In this chapter, we outline our platform's architecture and describe in detail the design decisions that we made while creating the four major components of our platform: the web application, web server, algorithm server, and database.

3.1 Architectural Overview

This section provides an overview of the major components of our platform and how they communicate with each other. Figure 3-1 provides illustrates the communication between the components of the architecture. This illustration structures our discussion for Section 3.1 “Architectural Overview” of this thesis.

![Diagram of the four major components of the platform and Dropbox with arrows indicating communication between components.](image-url)

Figure 3-1: The four major components of our platform (i.e., web application, web server, algorithm server, and database) and Dropbox. Arrows [A]-[K] indicate communication between components.
When users navigate to the appropriate URI, they download the web application (web app) onto their personal computer from the web server (Figure 3-1 [A]). The web app helps users create, manage, and share our two main objects: algorithms and data sets. It is built using Angular, D3.js, and standard web technologies, HTML, CSS, and JavaScript. The web app is hosted by our web server, which is a static server powered by Node.js [59], a server-side JavaScript runtime built on Google's open-source, high-performance V8 JavaScript engine [60].

Once the web app loads on their personal computer, users log in to the web app and authenticate with Firebase (Figure 3-1 [B]). We chose Firebase as our database solution because our backend needs were simple and Firebase offers authentication, non-relational database storage, data encoding, and security, so we were able to use this standard Database as a Service (DBaaS). Firebase stores all user-specific content that personalizes users’ experiences. Another reason why we chose Firebase was because it offers AngularFire, which is a JavaScript library that makes integrating with our Angular web app easy.

After logging in, AngularFire creates a two-way connection between Angular in the web app and Firebase (Figure 3-1 [C]). This way, any changes the user makes in the web app are immediately and automatically saved to the database, and any changes that are made to the database by other users are immediately and automatically downloaded by the web app and displayed to the user.

Users select files using the Dropbox File Chooser (Figure 3-1 [D]). Since files are stored on Dropbox, saving the files separately to our system would be redundant. Instead, our platform saves the files’ metadata, such as the filenames and URIs, to Firebase (Figure 3-1 [C]). We decided against adding file hosting, version control, and other features provided by digital asset management (DAM), version control (VC), and file systems. Instead, we integrate with systems that potential users already use to share and store data and algorithms, specifically Dropbox. This keeps our backend needs simple and allows us to use a standard DBaaS, such as Firebase.

---

23 Uniform Resource Identifier (URI), previously known as Uniform Resource Locator (URL), also known as Web Address: The name, identifier, address, or location of a web page, site, app, or object on the Internet. It is the text that users type into browsers to go somewhere on the Internet, usually beginning with http://www…
Users run algorithms on datasets by pressing a "Run" button, which sends a HTTP request to the web server (Figure 3-1 [E]). The HTTP request only contains \texttt{run\_id}, which is the unique identifier for a \texttt{run} object. A \texttt{run} object is an JavaScript object we created to contain the necessary information to execute an algorithm on a dataset. With this \texttt{run\_id}, the web server downloads the corresponding \texttt{run} object from Firebase (Figure 3-1 [F]), which enforces security rules. By forcing the web server to download the \texttt{run} object instead of passing it directly via the HTTP request, this architecture ensures that only users with appropriate permissions can run algorithms on algorithm servers and prevents savvy users from sending potentially-malicious commands directly to a server. After downloading the \texttt{run} object from Firebase, the web server acts as a message broker between users' requests to run an algorithm and algorithm servers that are available and configured to run users' requests (Figure 3-2).

![Diagram of the process](image)

Figure 3-2: The web server acts as a message broker between multiple users (left) and servers (right). The web server seeks an available server that is configured to run the algorithm, which is Matlab in this case.
After finding an algorithm server that is available and configured to run the algorithm, the web server forwards the web app’s HTTP request to the algorithm server (Figure 3-1 [G]). Algorithm servers are actually algorithm developers’ computers, which are already configured to run the algorithms that users upload. Our platform enables users to convert their computers into algorithm servers using a small Node.js script. Currently, we can only run Matlab algorithms (.m files), but we can easily enable other languages. Like the web server, the algorithm server downloads the run object from Firebase (Figure 3-1 [H]), further enforcing security. Then the algorithm server responds to the web server (Figure 3-1 [I]), which forwards it to the web app (Figure 3-1 [J]), confirming receipt of its initial HTTP request containing run_id. Simultaneously, the algorithm servers download the necessary files and run the algorithm on the dataset. The algorithm server responds to the HTTP request before the algorithm finishes running because the HTTP request would probably timeout, if the algorithm server waited for the algorithm to complete. After the initial confirmation that the algorithm server received the HTTP request, the algorithm server communicates progress and results via Firebase (Figure 3-1 [K]), which is immediately displayed in the web app due to the two-way AngularFire connection between Angular and Firebase (Figure 3-1 [C]).

Above is a broad overview of our platform’s architecture. The remainder of the chapter describes the detailed design decisions that guided the development of each of the four major components of our platform: the web application, web server, algorithm server, and database.

3.2 Design Decisions for the Web Application

This section discusses the detailed design decisions that went into making the web application component of our platform. It explains why we chose to design a web application over a desktop or mobile app; why we chose a single-page web app over a traditional, multi-page web app; why we chose Angular as our web application framework over Backbone or Ember; how we designed our video player directive with D3.js; why we chose NVD3 as our charting library; why we chose Bootstrap to enhance
our HTML, CSS, and JavaScript; and why we chose Dropbox File Chooser over other implementations of Dropbox.

3.2.1 Web Application

We considered developing our platform as a desktop or mobile application, but ultimately we decided to build a web application. Desktop and mobile applications need to be ported\textsuperscript{24} for each operating system and device. Compared to a desktop or mobile apps, web applications have increased interoperability because they are opened in Internet browsers, such as Google Chrome, Mozilla Firefox, Internet Explorer, Safari, which abstract away differences in operating systems and devices. Developers still need to be aware of subtle differences between each browser and other cross-browser compatibility issues. These differences will decrease as browsers standardize and web application libraries mature. A mature web library can completely abstract away all cross-browser compatibility issues.

```javascript
function callback() {
    return window.event.srcElement.type; // returns: 'onclick'
}
```

Figure 3-3: JavaScript mouse click event callback function for Internet Explorer 8 or less.

```javascript
function callback(event) {
    return event.target.type; // returns: 'click'
}
```

Figure 3-4: JavaScript mouse click event callback function according to the W3C standard. A mature JavaScript library, like jQuery [61], allows developers to use the standardized callback function, even with Internet Explorer 8.

3.2.2 Single-page Web Application

Traditional web pages are not commonly referred to as “multi-page web applications,” except when compared to single-page web applications. With traditional web apps, each link requests a separate file

\textsuperscript{24} \textbf{to port:} to modify software to run in a new environment
from the server. For example, navigating to a traditional web app located at www.url.com/users.html will request the file users.html from the server. Whereas single-page web apps download the entire application the first time a user visits any link, and they download partial web pages, called templates, when necessary. Most visitors do not notice the difference between the traditional link www.url.com/users.html and the single-page link www.url.com/index.html#users because the user experience is the same. Features, such as browser history, back button, and bookmarks, behave exactly the way they expect them to. Single-page apps use hashtags (e.g., #users) instead of filenames (e.g., users.html) to track which page the user is on because a single-page app is all one file (e.g., index.html). Since visitors are downloading the entire web app, the page takes longer to download the first time it is viewed. After that, it loads much faster because it only downloads what it needs, when it needs it. Traditional web apps often reload the same content, such as headers, footers, side panels, JavaScript files, and CSS files, over and over again, every time a visitor clicks a link. This works fine for static web pages that do not change, but users have grown to expect features like constantly-updating new feeds, messages that pop up once received, and other always-up-to-date features, called asynchronous JavaScript events. Such events are easy to implement with single-page web apps because they are natively built into the framework. These frameworks inherently need to handle asynchronous JavaScript events, even just to fetch the HTML templates mentioned above.

3.2.3 Angular Web Application Framework

We chose Angular as our web application framework over other, such as Backbone or Ember. Web application frameworks are code repositories that allow developers to “stand on the shoulders of giants” [62], while they program web applications. Frameworks provide objects, functions, and paradigms that resolve common challenges web developers face, like abstracting away cross-browser compatibility issues, as described in Section 3.2.1.
Angular is opinionated and prescriptive, meaning that if a desired outcome can be achieved many ways, Angular will choose one way and that way becomes The Angular Way. The advantage is that Angular makes a lot of decisions that the system architect would otherwise have to make him/herself, which protects against decision fatigue. Also, when developers follow The Angular Way, they can be confident they are following the web development industry’s best practices as decided by intelligent, experienced web developers, who are aware of common pitfalls. The disadvantage is that if the system architect has a good reason for not implementing The Angular Way, he/she will find it harder to do so.

Other frameworks, such as Backbone or Ember, are more lightweight and open-ended, allowing for more freedom in implementation. Angular is also heavier, meaning that it comes with more tools, whether developers use them or not, at the cost of an increase in the time it takes for the web page to load.

Angular directives are one of the main reasons why Angular claims “AngularJS is what HTML would have been, had it been designed for building web-apps” [63]. Angular directives allow developers to easily create custom HTML features. Developers can create custom HTML elements, such as the <player/></player> element discussed in Section 3.2.4, and attach custom attributes to new or existing HTML elements, like the config attribute on the <player config="..."> tag. Angular comes with dozens of commonly-used, pre-made attribute directives that can modify new or existing HTML elements. For example, adding ng-repeat="item in array" duplicates an element once for each item in the array, and adding ng-show="statement" shows the element if statement is true and hides the element if statement is false. In this manner, Angular augments traditional HTML and makes it easy for developers to further expand on HTML’s limited vocabulary. Outside of Angular, it is common for developers to continue working with HTML’s limited vocabulary, and if they need a custom element, they usually try to repurpose the standard <div> tag and modify its functionality with CSS and JavaScript.
3.2.4 Player Directive

Since we are building a web application for video and image processing researchers, one of the primary features of our web application is a video player. Our player is unique because it is built for video and image analysts, who want to be able to step through videos frame-by-frame and zoom in to inspect any pixel. Researchers analyzing video and images tend to think of a video as a sequence of discrete images in time or space. This is slightly different from the way typical Internet users think about videos, which tends to be as more of an audiovisual stream. Informed by these differences, our player takes a sequence of image files as input, instead of a video file. This makes it more of an image sequence player than a video player. There are many pre-made video players available on the Internet. Video players are so common that the W3C integrates a `<video>` element into the HTML5 standard, so browsers can natively play videos without developers having to create their own video player tools and libraries. Unfortunately, we could not find any tools or libraries that played a sequence of image files as a video. This became an excellent opportunity to use Angular directives to create a custom, reusable HTML element that can play an image sequence. We called this directive the `<player>` directive.

3.2.5 Layering Data over Images

Video and image processing researchers sometimes run algorithms that change many pixels in many images, such as converting color images to grayscale images. In these cases, it makes sense to save the output as an image, which developers can easily display with a simple image sequence player. Often, the algorithm does not modify the images much, if at all, for example, when an algorithm tracks a circle’s motion through each frame. The algorithm may draw a circle on each frame and save each new image. Again, these images can then be viewed using a simple image sequence player. Alternatively, the algorithm can simply save and send the circle’s center coordinate and radius. The advantage to this latter approach is that it conserves storage space and downloads faster. However, it also requires a more-advanced image sequence player that can layer pixels, shapes, and other data over each image. This more-
advanced approach is the image sequence player we chose to implement. More generally, our image
sequence player aims to be able to pair annotations and other information to each frame.

3.2.6 Scalable Vector Graphics (SVG)

As with most software, there are many ways to achieve the same goal in HTML. The \texttt{<player>}
directive could have extended any HTML tag. In this section, we consider which of the following tags are
most appropriate: \texttt{<div>}, \texttt{<canvas>}, and \texttt{<svg>}.

A \texttt{<div>} tag is the default HTML element for containing other elements and modifying to achieve
desired functionality. \texttt{<div>} tags are used, unless another HTML element is specifically designed for the
task. Other HTML tags can be used, but \texttt{<div>} tags are seen as the default behavior expected from an
HTML element, so they require less modification and formatting to achieve new behaviors.

A \texttt{<canvas>} tag is an appealing alternative to a \texttt{<div>} for exactly the reason one might expect. A
\texttt{<canvas>} is like a blank artist's canvas and comes with the ability to add shapes and colors, features that
a developer would have to manually add to the more-generic \texttt{<div>}. These features are desirable for
layering images and annotations described above.

An \texttt{<svg>} tag is like a \texttt{<canvas>} tag, except that it uses scalable vector graphics (SVG) instead of a
bitmap. Those familiar with Adobe Creative Suite know that Adobe Photoshop creates bitmaps and that
Adobe Illustrator creates vector graphics. A bitmap or raster image, such as Portable Network Graphics
(PNG)\textsuperscript{25} file, is effectively a two-dimensional matrix of pixels. If a developer draws a black line on a
white \texttt{<canvas>}, the bitmap fills in black pixels. If a user zooms in, he/she sees a pixelated line.
Alternatively, an \texttt{<svg>} uses scalable vector graphics, which represent lines as two points. This way, a
user can zoom in infinitely and still see a smooth line. Scalable vector graphics, or \texttt{<svg>} HTML tags,
offer smoother, cleaner annotations for our image sequence player (i.e., \texttt{<player>} directive), as
compared with bitmaps, or \texttt{<canvas>} tags.

\textsuperscript{25} \textit{Portable Network Graphics (PNG)}: bitmap image files that end with the .png extension

41
It is worth noting that `<canvas>` tags may perform slightly more efficiently than `<svg>` tags, but both perform significantly more efficiently than `<div>` tags [65].

3.2.7 Data-Driven Documents (D3)

The D3.js JavaScript library [66] is the primary reason why we ultimately settled on building the `<player>` directive off of an `<svg>`. D3, Data-Driven Documents [67], is a powerful, free, open-source JavaScript library for the web that uses standard web technologies, such as HTML, CSS, JavaScript, SVG [68], to visualize almost any data [69] [70]. It is "arguably the most dominant and important programming library in the field [of data visualization]" [71]. Since we planned on leveraging the power of D3, we chose to use `<svg>` tags because many of D3’s features are specific to `<svg>` elements and are not available for `<canvas>` elements. The fact that the authors of D3 decided to build their advanced visualization library based on scalable vector graphics supports our decision to use `<svg>`.

Angular and D3 offer two different paradigms for JavaScript data binding. Angular explicitly binds JavaScript objects, called models, to HTML elements in a view. These bindings can be one-way or two-way. D3 creates a one-way data binding directly to the HTML Document Object Model (DOM). We tried it both ways and found that The Angular Way works best. Since most of the web application is developed using the Angular paradigm, we found it the easiest to stay consistent with the existing paradigm, do
things *The Angular Way* as much as possible, and the D3 way as little as possible. D3 also enables this approach because D3 is modular, while Angular is not, making it is easy to include only the D3 features used and exclude features not used. Angular does not offer similar modularity.

### 3.2.8 Novus D3 (NVD3)

The `<player>` directive was a complex and unique feature for our web applications, and since nothing quite like it exists, we had to built it from the ground up, using HTML, SVG, CSS, JavaScript, Angular, and D3. Other features, such as 2D charts, are much more common. We could build these features from scratch with the same technologies we used to create the `<player>` directive, but it was not worth the time and effort, since excellent charting libraries already exist. We considered the following charting options: NVD3, Vega, FusionCharts, HighCharts, Google Charts, Sencha ExtJS Charts, Chart.js, Flot, jqPlot, gRaphael, Canvas.js, and many others. Ultimately, we decided to go with NVD3 for a variety of reasons. It has all of the chart types we expect to use: line, filled line, stacked line, scatter, bubble, bar, grouped bar, stacked bar, pie, etc. It implements D3, which we are already using for the `<player>` directive. *The Angular Way* to integrate NVD3 into our web app is to turn the NVD3 charts into Angular directives. Fortunately for us, NVD3 Angular directives already exist, such as angularjs-nvd3-directives [72] or angular-nvd3 [73]. In addition to powering our `<player>` directive, D3 and NVD3 can serve as the foundation for all the Matlab-like plots, figures, charts, graphs, and visualizations we plan on implementing in the future.

### 3.2.9 Bootstrap

Bootstrap is a popular CSS and JavaScript library that enhances HTML in ways that are useful for almost any webpage. Bootstrap offers boilerplate templates, which makes getting started easy. These templates adjust to any screen size and are optimized for mobile devices with limited processing power and bandwidth. Its grid systems makes page layout easy. It enhances traditional HTML and CSS, such as
tables, forms, buttons, images, fonts, colors, etc. It adds new HTML components, such as dropdowns, button groups, navigation bars, pagination, labels, badges, jumbotron, page headers, thumbnails, alerts, progress bars, media objects, panels, wells, etc.; CSS formatting, such as CSS helper classes; and JavaScript actions, such as transitions, modal boxes, scrollspy, tabs, tooltips, popovers, carousel, etc.

3.2.10 Dropbox File Chooser

![Dropbox File Chooser](image)

Figure 3-6: The “Dropbox Chooser” popup window used to select files.

Dropbox Developer API provides a Dropbox chooser popup window, which is perfect for our application to help users select which Dropbox files they want to use for their algorithms and datasets.
Currently, the web app only saves a link to the file, and not the actual file. The advantages to this approach are that it is simpler, easier to implement, requires less storage, and allows for rapid iteration because developers can edit an algorithm without having to re-upload the file. The disadvantage is that if the user edits the algorithm, our platform does not save previous versions of the algorithm used to generate saved results.

3.3 Design Decisions for Web Server

This section discusses the detailed design decisions that went into making the web server component of our platform. It explains how we chose to use a static server to host the web application and how the web server also acts as a message broker.

3.3.1 Static Web Server

The server for the web application has one, simple, primary requirement. It primarily acts as a static file server to host the web application. Static file servers receive a request from a url and respond with the file located at the path. For example, if a user navigates to http://url.com/path/to/file.html, the user is effectively requesting the file located at ./path/to/file.html on the server. A simple, static file server can serve the Angular web application.
Figure 3-8: A static file server receives a request from a url (e.g., http://url.com/path/to/file.html) and responds with the file located at the path (i.e., /path/to/file.html).

Any static file server will do. Since we are already familiar with and working with JavaScript, we decided to use Node.js to implement a static file server in a few lines of code.

```javascript
var express = require('express');
var app = express();
app.use('/', express.static(__dirname)).listen(port);
```

Figure 3-9: A simple, static file server in JavaScript, using Node.js's Express.

### 3.3.2 Message Broker Between Users and Algorithm Servers

The web application server has another role, and that is to serve as a message broker between users and the algorithm servers. In our case, the message broker is a workload queue, or a message queue, where a user requests that an algorithm is run on a specific data set. The message broker queues this task, while it queries for an available algorithm server that is properly configured. Once it finds an appropriately-configured, available server, it forwards the request to that server.
Figure 3-10: The web server acts as a message broker between multiple users (left) and servers (right). The web server seeks an available server that is configured to run the algorithm, which is Matlab in this case.

At each step along the way, the front end, the message broker, and the algorithm server sync with the database. By syncing with the database, they ensure that the information being sent is valid because only verified users with appropriate credentials and permissions can modify the database, while any sufficiently-savvy user can send a potentially-malicious message.
3.4 Design Decisions for the Algorithm Servers

This section discusses the detailed design decisions that went into making the algorithm server component of our platform. It explains why we chose self-hosted servers over distributed servers; how we communicate with HTTP requests; why we chose to read files instead of parsing standard streams; and how we use the database to display results.

3.4.1 Self-hosted vs. Distributed Servers

Initially, we assumed a distributed, cloud platform, such as Amazon Web Services (AWS), Google Cloud Platform (GCP), or Microsoft Azure, would host our algorithm servers. There are many benefits to hosting our instances on a distributed cloud platform, such as flexibility, scalability, reliability, consistency, which is why we hope to offer this option in the future.

Another option is to allow users to host their own servers. Self-hosted servers are the first option we decided to implement. The main benefit is that users' personal computers are already configured to run their algorithms. By taking the same computers that the algorithms were developed in and repurposing
them as servers, it minimizes or eliminates server configuration and debugging. Users’ personal computers are already configured to run the correct programming language, language version, installed packages, and licensed software. We used a Node.js script to convert users’ personal computers into servers.

3.4.2 HTTP Requests

When the user clicks the “Run” button, our Angular frontend sends an HTTP request to the web server. The request contains the unique identifier for that run. The web server acts as message broker, deciding which available algorithm server to send the request to. The algorithm server downloads all the necessary files locally and executes the algorithm. When the algorithm finishes executing, it updates Firebase with the results, instead of responding to the HTTP request.

3.4.3 Algorithm Input/Output (I/O): Standard Streams vs. Read/Write to File

Even though the algorithm could be written in any one of many languages, the options for communicating with the algorithm are fairly standard and limited. We could use standard streams, such as standard input (stdin), standard output (stdout), and standard error (stderr), which are standardized input/output communication channels that computer programs use to interact with their environment. Alternatively, we could read and write to files, using any number of file formats. However, reading from or writing files to hard disk is generally much slower than parsing standard streams, which reads and writes to memory. The difference is typically five or six orders of magnitude, which is 10,000 or 100,000 times faster [74] [75] [76]. Ultimately, we went with this slower option because it was easier to implement. At this early stage of development, we deemed it premature optimization to save ourselves a fraction of a second, when executing the algorithm likely takes much longer to run.

---

26 Donald Knuth: “Hoare’s dictum that ‘premature optimization is the root of all evil in programming.’” [92] [93]
3.4.4 Displaying Algorithm Results to the Frontend

Our first implementation worked with exactly one algorithm\textsuperscript{27}. After the algorithm finished executing, the algorithm server responded to the HTTP request with the result. Once this result was received by our Angular frontend, a JavaScript callback function transformed the algorithm output, so that it could be displayed graphically. As previously mentioned, this was a hard-coded, one-off solution.

Our second iteration attempted to make this translation process more generic. Our solution was to require the algorithm developer to include a JavaScript file along with the algorithm file, the data, and any supporting files. This JavaScript file was executed in addition to the algorithm file to parse the algorithm’s output into something that the frontend could easily display. The main problem with this approach is that it required the algorithm developer to learn JavaScript and program in this unfamiliar language. It was also became a new source of bugs and was difficult to debug.

Our third and final iteration eliminated this translation process by requiring the algorithm to parse input and output directly from and to the frontend. To accomplish this, it uses JavaScript Object Notation (JSON), a popular web standard for transmitting data using human-readable, key-value pairs. Due to the pervasiveness of the web and JSON’s near-ubiquitous use throughout the web, most, if not all, programming languages have libraries for reading and writing JSON\textsuperscript{28}. For example, MATLAB is not a particularly web-friendly language. Still, MATLAB has JSONlab \cite{JSONlab} for reading and writing MATLAB objects to JSON. In most cases, these libraries convert JSON into objects native to the language the developer is currently using, and vice versa. In this manner, algorithms can output their results simply by appending the results to a native object and outputting that object as JSON. This way, developers do not have to upload a JavaScript file and can continue using the language that the algorithm was written in, which they are likely more comfortable and productive with. This is the cleanest, most-elegant solution we have discovered thus far for displaying algorithm results to the frontend.

\textsuperscript{27} The first algorithm we implemented was a template tracking algorithm.

\textsuperscript{28} If a library does not exist for a language, JSON can be constructed by concatenating text, which is a fundamental capability of most programming languages.
3.5 Design Decisions for the Database

This section discusses the detailed design decisions that went into making the web application component of our platform. It explains why with integrate with existing systems instead of adding features from those systems; why we chose database as a service over a self-managed solution; how we used AngularFire to elegantly communicate between our Angular frontend and Firebase backend; and why we chose Firebase over other database-as-a-service solutions. We also discuss the costs and benefits of different methods for saving the results of an algorithm.

3.5.1 Integrating with DAM System vs. Being a DAM System

At first we thought our platform would have to serve as a Digital Asset Management (DAM) system and/or a Version Control (VC) system, but we quickly realized that there are many excellent DAM and VC systems that users are already using, such as Dropbox and GitHub. We decided that it would be much easier to integrate with these pre-existing DAM and VC systems than to re-implement many of their features. The benefit of integrating with pre-existing DAM and VC systems is that our platform integrates into developers’ current workflow without having to learn a new tool and offer solutions to problems that they already have solutions to. Not having to add these features helped us to get to a functioning prototype faster.

3.5.2 Database-as-a-Service (DBaaS) vs. Self-managed Backend (No Services)

Database-as-a-Service (DBaaS) is a service that replaces all or part of the backend for web, mobile, and desktop applications. When developers only need typical backend features, such as data persistence, user management, developers can build applications without having to write a single line of server-side code. This can save approximately six developer-months\(^{29}\) of work. For example, if a developer wants to

\(^{29}\) A “developer-month” is the amount of work a typical developer can get done in a month, so six developer-months takes one developer six months, two developers three months, and three developers two months. Beware of “The
save a user object (Figure 3.12) to the backend, he/she simply sends it to the URI corresponding to the API (e.g., dbaas.com/users). If the API does not exist, the DBaaS can automatically create it. Since we decided against implementing DAM and VC system features, our backend requirements are pretty minimal and standard, which makes our platform an excellent candidate for DBaaS, which is what we decided to go with.

```
{
    "name": "Tylor Hess",
    "age": 27,
    ...
}
```

Figure 3-12: A user object represented in JavaScript Object Notation (JSON).

There are many DBaaS providers, such as AnyPresence, Appcelerator, Appery, Backendless, Built.io, CloudMine, FeedHenry, Firebase, Kii, Kinvey, Kony, Kumulos, moBack, Syncano, and Telerik, to name a few.

### 3.5.3 AngularFire: Angular, Firebase, and Google

Ultimately, we settled on Firebase for a variety of reasons. Firebase appears to be the only DBaaS marketed at frontend JavaScript applications. Firebase offers AngularFire, a version of Firebase that easily integrates with Angular, our frontend JavaScript framework. Like Angular, Firebase was acquired by Google, which probably means that Angular and Firebase will continue to integrate well together and will be adopted by more-conservative big-name companies because the Google brand offers an assurance of stability in this ever-changing web technology landscape, and that Angular and Firebase will become a more enduring platform because of the support of all the aforementioned parties. However, these assumptions are not necessarily true. Another option that we were seriously considering but ultimately

---

Mythical Man-Month” [94]. Just because it takes one woman nine months to make a baby, does not mean three women can make one baby in three months. The same is true with developers. Managers need to account for the time it takes a new team member to ramp-up, the time it takes an existing team member to onboard a new team member, and decreased productivity due to increased communication. As each new team member is added, communication increases combinatorially, or quadratically.

30 Technically, Angular was not acquired by Google. It is maintained by Google [63].
31 Google acquired Firebase in October 2014, and Firebase is now part of Google Cloud Platform [95].
decided against was Parse, which was acquired by Facebook in April 2013 [78] [79] and shut down in January 2016 [80].

3.5.4 Using Firebase

For the most part, APIs abstracts away backends’ inner workings, like whether they are relational databases, such as MySQL, or non-relational databases, such as MongoDB. However, the fact that Firebase is a non-relational database does have some consequences, especially when it comes to structuring our data [81]. Since Firebase data can be nested[12] 32 levels deep[81], it is tempting to nest our data, but a flattened data structure is better practice because it is hard to enforce proper permissions and security using a hierarchical data structure, since the entire object is downloaded from Firebase to the Angular, and since Firebase Security and Rules do not offer the ability to filter out child objects. Also, flattened data can be downloaded in parts, as needed, and allows for many-to-many relationships.33

```json
{
    "groups": {
        "group1": {
            "name": "Alpha",
            "members": {
                "user2": {
                    "name": "Brian Anthony",
                    "age": 45
                },
                ... // more group members
            }
        },
        "group2": {
            "name": "Beta",
            "members": {
                "user1": {
                    "name": "Tylor Hess",
                    "age": 27
                },
                ... // more group members
            }
        },
        ... // more groups
    }
}
```

32 to nest: to store objects inside of objects inside of objects
33 many-to-many relationships: a user belonging to multiple groups, and a groups having many users
Figure 3-13: A JSON example of a hierarchical data structure. Notice how one user cannot be in multiple groups.

```json
{
    "users": {
        "user1": {
            "name": "Tylor Hess",
            "age": 27
        },
        "user2": {
            "name": "Brian Anthony",
            "age": 45
        },
        ... // more users
    },
    "groups": {
        "group1": {
            "name": "Alpha",
            "members": {
                "member1": "user1",
                "member2": "user2",
                ... // more group members
            }
        },
        "group2": {
            "name": "Beta",
            "members": {
                "member1": "user2",
                ... // more group members
            }
        },
        ... // more groups
    }
}
```

Figure 3-14: A JSON example of a flattened data structure. It also allows for many-to-many relationships (i.e., a user can belong to multiple groups, and a group can have many users), and users or groups can be downloaded, as needed, without downloading the entire list of users or groups.

```json
{
    "users": {
        "user1": {
            "name": "Tylor Hess",
            "age": 27
        },
        "user2": {
            "name": "Brian Anthony",
            "age": 45
        },
        ... // more users
    },
    "groups": {
        "alpha": true,
        "beta": true
    }
}
```
"groups": {
  "group1": {
    "name": "Alpha",
    "members": {
      "user1": true,
      "user2": true
    ... // more group members
  }
  },
  "group2": {
    "name": "Beta",
    "members": {
      "user2": true
    ... // more group members
  }
  },
  ... // more groups
}

Figure 3-15: A "best practice" example of a flattened data structure. Every user object directly references its groups and every group object directly references its users, which prevents security issues from arising.

3.5.5 Saving Algorithm Output

As described in Section 3.2.5, we do not always need to save the output as an image. Sometimes we can layer pixels, shapes, and other data over each image. In these cases, we can drastically reduce the storage space required to save the algorithm's output because saving a shape requires much less space than saving a whole new image. It also changes the way we save the output. We can easily and simply save shapes and other annotations to our database, while images require file storage in addition to database storage.

Another option that we considered was whether to save the output at all, as opposed to re-running the algorithm every time the results are needed. Traditional web development best practices save everything [ref] because storage space is relatively cheap and because data needs to be saved, at least temporarily, in order to be transmitted over the Internet. However, there may be a point where the cost of re-running a fast algorithm is cheaper than the cost of storing the results. Large data sets tend to yield large results, and algorithm execution time tends to scale at least linearly with the size of the data, so it is tempting to assume that storage costs will always be cheaper than the costs of re-execution. It is, however, easy to imagine a fast algorithm that runs on a relatively small data set that yields disproportionately large results.
In these cases, it might make sense to re-run the algorithm every time, instead of storing the data. A design decision that includes these features should also consider the cost of building this added layer of complexity to the system. For our current system, these features make even less sense because re-running an algorithm carries with it the additional inconvenience of consuming resources on people’s personal or lab computers.
Chapter 4 Evaluation: Focus Group

In this chapter, we evaluate our design with a twenty-one-person focus group. Over the course of an hour, the participants followed a tutorial on how to deploy a normalized cross-correlation algorithm [82] for video object tracking [83] [84] that we prepared in advance written in Matlab.

Before logging in and using the platform, users had to create a translation file, runDemoTemplateTracking.m (Figure 4-1). This file translates the information provided by the platform, such as a user-drawn rectangle (Figure 4-7, white box), into the information expected by the algorithm, such as a Matlab matrix containing the rectangle’s x position, y position, width, and height. Then it calls the algorithm and translates the function’s output into information that the platform can interpret. There are five steps to this translation process. The first step (Figure 4-1, lines 4-5) is to load runConfig_input.json, which is a JSON file containing the runConfig object, which contains all of the information that the platform can provide to the algorithm developer. In this case, the useful information that runConfig contains is file names of the image frames (Figure 4-1, line 10) and the dimensions of the rectangle drawn on the first frame (Figure 4-1, lines 13-14). The second step (Figure 4-1, lines 7-14) translates this useful information into the format that the algorithm function expects. In this case, the algorithm function, demoTemplateTracking (Figure 4-1, line 17), expects a Matlab matrix of image filenames and a Matlab matrix of the rectangle’s x position, y position, width, and height. The third step (Figure 4-1, lines 16-17) runs the algorithm function, demoTemplateTracking (Figure 4-1, line 17). The fourth step (Figure 4-1, lines 20-36) takes the algorithm’s output, rects (Figure 4-1, line 17), and translates it into runConfig, which the platform can interpret. The fifth step (Figure 4-1, lines 38-39) saves runConfig as runConfig_input.json, which immediately gets uploaded to Firebase and displayed to the user in the Angular frontend.
Figure 4-1: Translation file, runDemoTemplateTracking.m, translates information provided by the platform.

After the users create these translation files, the process is fairly straightforward and simple (Figure 4-2). The users log in, upload the algorithm files, upload the dataset files, and run the algorithm on the dataset. If necessary, users may need to debug and re-run the algorithm.

![Block diagram representation of user experience (UX) flow.](image)
Users start by navigating to the URI, where the web application is hosted. At the login screen, pre-existing users can log in and new users can create an account. Once a user has logged in, a new algorithm is created, and the user is immediately redirected to the page where he/she can edit the algorithm (Figure 4-3).

**Edit Algorithm**

![Algorithm Unique Identifier (Id): -KFo479x8KmsuAwrs4cf](image)

**Algorithm Name:** Normalized Cross-Correlation Algorithm

**Normalized Cross-Correlation Algorithm**

**Algorithm Description:**

This template tracking algorithm uses normalized cross-correlation to...

![Algorithm File:](image)

![Supporting Files:](image)

![Run](image)

Figure 4-3: Edit Algorithm screen.

As the user types, changes are saved to the database immediately and automatically (Figure 4-3, blue box). Then, the user selects the main algorithm file to be executed, `runDemoTemplateTracking.m` (Figure 4-5, red box), and any supporting files, `demoTemplateTracking.m` (Figure 4-5, green box), using a Dropbox Chooser popup window (Figure 4-4, popup window).
Edit Algorithm

Algorithm Unique identifier (id): -KFo479x8KmsuAwf
Algorithm Name: Normalized Cross-Correlation Algorithm
Algorithm Description:
This template tracking algorithm uses normalized cross correlation.

Algorithm File:
Choose Algorithm File

Supporting Files:
Add Supporting Files
Run

Figure 4-4: Edit Algorithm screen with Dropbox Chooser popup window.

After using the Dropbox Chooser to select the main algorithm file to be executed, `runDemoTemplateTracking.m` (Figure 4-5, red box), and any supporting files, `demoTemplateTracking.m` (Figure 4-5, green box), the Edit Algorithm screen is populated with these files (Figure 4-5).
Edit Algorithm

Algorithm Unique Identifier (id): -KFo479x8KmsuAwrs4cf

Algorithm Name: Normalized Cross-Correlation Algorithm

Algorithm Description:
This template tracking algorithm uses normalized cross-correlation to...

Algorithm File: runDemoTemplateTracking.m
runDemoTemplateTracking.m (1.69 KB)
https://www.dropbox.com/s/vp4pl9x09web8inn/runDemoTemplateTracking.m?dl=1

Choose Algorithm File

Supporting Files:

demoTemplateTracking.m (1.87 KB)
https://www.dropbox.com/s/3j61zh61y1e66el/demoTemplateTracking.m?dl=1

Add Supporting Files

Run

Figure 4-5: Edit Algorithm screen with main algorithm file and supporting files.

Once the files are selected, the user clicks the “Run” button, which creates a new, unique Run Algorithm page and redirects the user to this newly-created page (Figure 4-6).
Algorithm Name: Normalized Cross-Correlation Algorithm

Algorithm Description: This template tracking algorithm uses normalized cross-correlation to...

Algorithm Input Type: player

Player Configuration:
- penZoom
- drawDot
- drawRect
- drawLine
- drawPath

Frames:
- image_000060_cropped.jpg (15.53 KB)
- image_000061_cropped.jpg (15.52 KB)
- image_000062_cropped.jpg (15.45 KB)
- image_000063_cropped.jpg (15.45 KB)
- image_000064_cropped.jpg (15.4 KB)

Data Files:
- Framerate: 10
- No additional files selected.

Figure 4-6: Run Algorithm screen with data uploaded as frames and viewed in a video player.

The algorithm developer, or glass-box user, can send this Run Algorithm page to any algorithm user, or black-box user. From this Run Algorithm page, any user can upload data files (Figure 4-6, red box) using the Dropbox Chooser and run the algorithm (Figure 4-6, green box) without any specialized knowledge or skill. If necessary, the user can configure how the input and output are viewed (Figure 4-6, orange box). For example, if the input is a video, the user may select to view the uploaded data on a video player (Figure 4-6, purple box). If necessary, the user can also annotate the data or add other forms of user
input. In this example of a Normalized Cross-Correlation Algorithm, the algorithm requires the user to draw a rectangle on the first frame (Figure 4-7, white box).

Figure 4-7: Run screen with data uploaded as frames, viewed on a video player, and a rectangle drawn on the first frame, which serves as input into the algorithm.

Once the data is uploaded, input is configured, and user input is collected, the user clicks the Run button, which begins execution of the algorithm. The terminal in the upper-right-hand corner of the screen displays progress, and any output is displayed under the input (Figure 4-8, red box). If the user obtained the results they were hoping for, they can preview the results in the browser or download and share the results. If they did not, they can update the files and re-run the algorithm.
Figure 4-8: Run screen in its final state with output displayed.
Approximately half of the participants were so interested and intrigued by the platform that they deployed their own custom algorithms without prompting. One participant became familiar enough with the system that he exploited a minor vulnerability in good faith with hilarious results. As discussed in Chapter 2, security is already a known issue and ideas for improving security are discussed in Chapter 6. The focus group was valuable, but yielded comparably fewer insights than the in-depth interviews.
Chapter 5  Evaluation Survey

In this chapter, we evaluate our design with a survey of six users, who estimated that our platform would significantly reduce deployment time. From the focus group participants, six people volunteered to deploy their own custom algorithm.

These six participants completed survey before and after deploying their own custom algorithms. When deploying their own custom algorithms, they followed the same process outlined in Chapter 4, expect all of their files were custom, instead of the tutorial files that we prepared in advance. From this survey, we discovered that there are two types of algorithms: algorithms that require a translation file and algorithms that do not. The platform was originally designed for running algorithms on video datasets. During our survey, we learned that running generic algorithms non-video datasets is a simpler problem, which we solved simultaneously. Our platform can run algorithms that take files as input and output files. Additionally, if they do not require user input and they do not display anything visually to the user on the Angular frontend, then they do not need to create the laborious translation file, and can follow the simple, straightforward process of uploading their algorithm and dataset files, clicking “Run,” and viewing their results. As a result, only two of our users deployed algorithms on video datasets and tested the full features of our platform. One deployment was an implementation of the Star-Kalman Filter, which segments out the carotid artery from a sequence of ultrasound images. The user clicked a point near the center of the artery on the first frame, which served as input to the algorithm. The results displayed a plot of the minor axis of the artery versus frame number and a file listing the artery contour location for each frame [85] [86] [87]. Another deployment was template tracking of anatomic features in ultrasound images. The user clicked a point near a feature on the first frame of a video, and the algorithm propagates that information through the rest of the video using template tracking [88].
Before and after deploying their own custom algorithms, our six participants complete a survey (Appendix A) and estimated the time it would take them to deploy an algorithm with and without the platform (Appendix B). It was no surprise that the platform scored the lowest on user interface (UI) design. This platform was a “works-like” prototype and not a “looks-like” prototype, so aesthetic design was not a priority. We were happy that the user experience (UX) design scored higher than the user interface (UI) design. While we had not prioritized the UX much either, we had put some effort into it. We were surprised that the ease of use scored higher than the user experience because we were expecting them to score about the same. We think that the user experience lost some points for lack of features, where the ease of use gained points for simplicity of design. The last two questions really validated our interpreted needs. For the most part, users said that they would recommend this platform to a friend or colleague. We received an even more enthusiastic response, when we asked if they would use the platform themselves or with a collaborator. Five of the six participants scored an average 8.4 (±1.1) with one participant, who currently has no use for the platform, giving the lowest score, 1.
<table>
<thead>
<tr>
<th>Question</th>
<th>Average (STD)</th>
</tr>
</thead>
<tbody>
<tr>
<td>How would you rate the user interface (UI)? (1 = poor, 10 = excellent)</td>
<td>4.3 (±2.1)</td>
</tr>
<tr>
<td>How would you rate the user experience (UX)? (1 = poor, 10 = excellent)</td>
<td>5.2 (±1.8)</td>
</tr>
<tr>
<td>How easy was the platform to use? (1 = difficult, 10 = easy)</td>
<td>6.8 (±1.1)</td>
</tr>
<tr>
<td>How likely are you to recommend this platform to a friend or colleague? (1 = very unlikely, 10 = very likely)</td>
<td>6.3 (±0.8)</td>
</tr>
<tr>
<td>How likely are you to use this platform yourself or with a collaborator? (1 = very unlikely, 10 = very likely)</td>
<td>7.2 (±3.2)</td>
</tr>
</tbody>
</table>

Table 5-1: Average and standard deviation (STD) of survey results on a scale from 1 to 10.

Figure 5-1: Six participants' (Part1-Part6) survey responses and average response (Average).
The most extraordinary survey findings were an expected 90% to 99.9% reduction in deployment time, which is a ten-fold to one-thousand-fold increase in deployment rate, even for first-time users (Appendix B). Overall, the survey seemed to affirm our belief in a need for a platform such as this.

Figure 5-2: All six participants’ (Part1-Part6) estimated time to deploy an algorithm without this platform (blue), the first time using the platform (red), subsequent times using the platform (green) on a linear vertical axis (left) and a logarithmic vertical axis (right).
Chapter 6 Conclusion

In this chapter, we summarize the thesis and make suggestions for future work. This thesis presented a platform that enables black-box users to run glass-box users' algorithms on data stored locally or in cloud storage services like Dropbox. This platform allows glass-box users to deploy algorithms as web applications without having to learn web development. Glass-box users upload their algorithms to cloud computing services and have the option to create an object native to the language in which the algorithm was developed. The platform turns this object into HTML displayed to the black-box users, so glass-box users can deploy algorithms as web applications without having to learn web development, which is beneficial, since algorithms are often not developed in web-friendly languages. We turned the computers the algorithms were developed on into cloud computing resources, instead of leveraging existing cloud computing services, such as Amazon Lambda or iron.io, because the local computers were already configured to run the algorithms with the appropriate operating system, system configuration, licensed software, etc. We evaluated our design with three in-depth interviews, a twenty-one-person focus group, and a survey of six users, who estimated that our platform would reduce deployment time 90% to 99.9%, which is an increase of one to three orders of magnitude in deployment rate. Other survey results on a 1-10 scale are as follows: 4.3 ±2.1 user interface (UI), 5.2 ±1.8 user experience (UX), 6.8 ±1.1 ease of use, 6.3 ±0.8 likelihood of recommending platform, and 7.2 ±3.2 likelihood of using platform or 8.4 ±1.1 likelihood of using platform if we exclude the outlier. Overall, the survey seemed to affirm our belief in a need for a platform such as this.

Going forward, it will be important to continue discovering potential users through customer development interviews, especially black-box users, whose needs thus far have been relatively unexplored. Through these interviews, we should explore the importance of turning glass-box users' computers into cloud computing resources. One interview participant stated, "Most institutions are behind a firewall, unlike MIT, so lab computers cannot be converted into servers without heavy overhead."
need to explore how true this is, what the implications are for us, and possible solutions, such as deploying as a VM on Amazon AWS. Technologies discussed in the Chapter 2 of this thesis may be an important source of solutions for these problems. These technologies may also address security issues, which need to be considered before this platform can be rolled out, even as an alpha prototype.

While conducting our interviews, observations, focus group, and survey, we received a lot of requests for features, summarized below in no particular order. These suggestions may become the new needs for future work on our platform: integrate with GitHub and other version control solutions; provide the option to choose to follow the newest or current version of a file; integrate with other digital asset management systems, in addition to Dropbox; provide more documentation, tutorials, templates, and example code; display the contents of the folder on the server that the files are downloaded to and the algorithm is run from; use icons or other visual cues to show when files are downloading, downloaded, from Dropbox, GitHub, or other sources, added, modified, or removed by the algorithm, etc.; consider saving these files somewhere permanently because currently they are only temporarily saved to the server that runs the algorithm for the duration of the algorithm; allow the user to download these files or folders locally, to Dropbox, etc.; consider adding a checkbox that makes these downloads automatic; create a library, which provides commonly-used functions specific to this platform, such as converting .mat files into images, and vice versa; override native functions, such as Matlab’s plot function, so they display on this platform; add the ability to select Dropbox folder, in addition to the pre-existing ability to select Dropbox files; modify the <player> to accept one or many .mat file(s) in addition to, or instead of, images; provide a suite of tools for managing, sanitizing, validating, and pre-processing datasets; add the ability to chain algorithms; create two separate views, one for black-box users and another for glass-box users, like how Airbnb creates one view for guests and a separate view with more features for hosts; provide a suite of debugging tools for glass-box users; “When you work in a wet lab, you keep a lab journal. And every single experiment you do, you write down: ‘added 0.1 mL of whatever,’ ‘added 0.3 mL of whatever,’ … We don’t do that in computer science. If I ran an experiment 2 months ago, I can go back and look at the data. But I generally don’t know: Did I normalize it? Was my threshold 0.20 or 0.25? Maybe there’s a
system and I just don’t use it, but I just don’t keep lab notebooks in computer science.”; add version control features to the platform, such as push or pull with commits; be able to provide a version number, when uploading an algorithm; create an API for each algorithm, so algorithms can be accessed and run programmatically, in addition to being run through the GUI; make sure the program that turns lab computers into cloud computing resources is interoperable across operating systems; allow for a GitHub-style README.md\textsuperscript{34} or another way to design the algorithm’s run page; track which files are derived from which files, such as a grayscale image being derived from an RGB image; switch from nvd3 to plotly charts; be able to view and compare multiple outputs or datasets simultaneously; automatically save and be able to download data displayed in the GUI; and add more GUI visualizations.

\textsuperscript{34} README markdown file
## Appendix A Survey Results

<table>
<thead>
<tr>
<th></th>
<th>User Interface</th>
<th>User Experience</th>
<th>Ease of Use</th>
<th>Likelihood of Recommending</th>
<th>Likelihood of Using</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant 1</td>
<td>4</td>
<td>6</td>
<td>7.5</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>Participant 2</td>
<td>1</td>
<td>4</td>
<td>7</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Participant 3</td>
<td>7</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>Participant 4</td>
<td>5</td>
<td>3</td>
<td>6</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>Participant 5</td>
<td>5.5</td>
<td>6</td>
<td>8</td>
<td>5</td>
<td>8</td>
</tr>
<tr>
<td>Participant 6</td>
<td>3</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>10</td>
</tr>
<tr>
<td>Average</td>
<td>4.25</td>
<td>5.16...</td>
<td>6.75</td>
<td>6.33...</td>
<td>7.16...</td>
</tr>
<tr>
<td>Standard Dev.</td>
<td>2.09...</td>
<td>1.83...</td>
<td>1.08...</td>
<td>0.82...</td>
<td>3.19...</td>
</tr>
</tbody>
</table>

Table 6-1: Survey results, average, and standard deviation on a scale from 1 to 10.
## Appendix B  Estimated Time to Deploy an Algorithm

<table>
<thead>
<tr>
<th>Participant</th>
<th>Without Platform</th>
<th>With Platform (First Time)</th>
<th>With Platform (Subsequent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Participant 1</td>
<td>40</td>
<td>4.0</td>
<td>0.75</td>
</tr>
<tr>
<td>Participant 2</td>
<td>960</td>
<td>1.0</td>
<td>0.25</td>
</tr>
<tr>
<td>Participant 3</td>
<td>80</td>
<td>8.0</td>
<td>4.00</td>
</tr>
<tr>
<td>Participant 4</td>
<td>480</td>
<td>2.5</td>
<td>0.75</td>
</tr>
<tr>
<td>Participant 5</td>
<td>40</td>
<td>3.0</td>
<td>0.50</td>
</tr>
<tr>
<td>Participant 6</td>
<td>960</td>
<td>1.0</td>
<td>0.25</td>
</tr>
</tbody>
</table>

Table 6-2: Estimated time to deploy an algorithm, converted to hours.
Chapter 7 Bibliography

Bibliography


Apr-2010.


