CloudLab: A Cloud-based Development Environment for Introductory Programming Courses

Daniel Vijayakumar, Michael Delong, Qusay H. Mahmoud (UOIT)
School of Computer Science
University of Guelph, Guelph, ON, N1G 2W1 Canada

Abstract

Both students and instructors of introductory university computer science courses are faced with unique challenges for programming components due to the many problems encountered as a result of the obligatory use of specific hardware and software resources, and thus the inevitable installation and configuration thereof. In this paper we present a simple, unified and platform-independent integrated development environment (IDE) based in the cloud for use in such academic courses.

Keywords: CloudLab, Cloud, CS, Lab, IDE, Development, Amazon Web Services, 3-Tier Model, EC2, S3, RDS, NodeJS, Websockets.

1. Introduction

Computer science students and instructors at universities such as the University of Guelph face a number of routine technical issues during their course of study that tend to inhibit learning at varying levels of significance. These problems are primarily encountered during applied practices for courses that involve programming components such as labs, tutorials and assignments. The most significant of the issues can be summed up as follows: the unique challenge of applying computer science concepts through programming assignments/projects and at the same time ensuring an application development process that is consistent among all students for academic purposes.

In order to address this problem, instructors of courses today, usually supported by one or many teaching assistants (TA), develop protocols that must be followed by all students. These generally entail important details regarding the working of the assignments as well as their submission. For example students may have to submit C programs that compile successfully without warnings under the ANSI C89 standard. Such rules are a literal solution to the problem of ensuring consistency and should be very effective in theory. In order to assist students in following the protocol, two valid options are generally provided: 1) provision of computer labs consisting of many computer systems that are setup and configured homogeneously; and 2) granting of freedom for students to use their personal computer systems at the cost of ensuring that program submissions correctly compile and run under the targeted system as specified by the instructor. But these solutions bring about their very own problems.

If development environments are provided for students by the school, they run into significant performance issues. School-provided computer labs are generally not suited for large numbers of concurrent users performing CPU-intensive tasks such as compilation and execution of code. Even worse, user disk space is generally implemented remotely as network filesystems, which are certainly not appropriate for the large number of I/O operations that are performed by many programs running concurrently. Both of the student authors can attest to this due to their severely negative experiences of major network interruptions and latency issues during the final assignments of one of their core programming courses (CIS*2750). Such issues are faults at the infrastructure level and are thus near impossible to address during the limited time period of an academic semester, let alone a few weeks.

On the other hand, many students may prefer to work on their own machines for reasons of convenience and familiarity. This generally means having to manually setup and configure their system according to the target environment, which in most cases is a distribution of Linux. While this may not be an issue for a few experienced students, the majority of students face a myriad of technical issues that are prevalent when migrating to a different computing platform entirely. Even if such a feat is accomplished in a reasonable amount of time, students still struggle to submit programs that compile cleanly as per the specifications of the instructor. The obligatory debugging of these quirks takes up a lot of limited and valuable time.

When it comes to assignment submission, we have a different breed of issues. The three primary methods used are: 1) e-mailing a compressed archive of the deliverables; 2) uploading a compressed archive to a specified location, usually the course website; and 3) submitting through the mechanics of a version control system, usually Subversion. Using e-mail for assignment submission is generally frowned upon due to the various quirks encountered such as filtered e-mail, bounce-back mail, etc. It is also quite inefficient to download every single e-mail attachment separately, especially in large-sized classes. Uploading compressed archive is usually employed with minimal concerns for students, however instructions regarding

specific compression file formats can be missed. Also, retrieving the submissions of all students can be a hassle similar to downloading e-mail attachments unless the particular LMS or website provides a means to perform batch downloads. Version control systems offer the greatest challenge since most students are not familiar with the involved concepts. However, they are generally much easier for instructors since a simple shell script can generally be written quickly in order to perform batch checkouts.

The common theme from the above discussion is frustration among both students and instructors that ultimately only inhibits progress in learning. It therefore stands to reason that both students and instructors would benefit from a solution that addresses the issues of concurrency and scalability while simultaneously providing availability as well as accessibility from a unified interface.

2. Goals

The main purpose of this project is to develop an integrated development environment (IDE) tailored towards students and instructors of university computer science courses. Emphasis must be placed on **five** of the most important features that are necessary for an academic setting. These features are now discussed below.

The most important feature is **platform-independence**. Users of the system must not be constrained by any platform so that the need for manually consolidating to a specific submission protocol (as described in section 1) can be eliminated. In order for this to be effective, the solution must strive to provide an interface that is unified and desirable to all possible users. Secondly, the system must be very capable of handling large numbers of concurrent processes with ease. Although the user traffic in such an environment is not heavy at all times during the semester, such **scalability** is required to accommodate for the predictable bursts that occur due to known events such as approaching deadlines.

As a consequence of scalability, the solution must be highly **reliable** and **available**. It is not acceptable for users to be inhibited from access to their tools and resources at any time during the semester. This allows students and instructors to follow through with their rigid schedules and deadlines that are commonplace in academia. Finally, the solution must adhere to strict standards of **security** and implement a strong access control system. This is crucial for minimizing academic misconduct, particularly plagiarism.

Ultimately, we desire to enhance the interactions and experiences of both students and instructors by abstracting away from them the problems of uniform development processes, controlled environments and submission logistics.

3. Proposed Solution

After much thought and consideration of existing and related solutions (discussed in detail in section 4), we decided to implement our own solution using a composition of a variety of existing tools and services that were customized to meet our needs.

Addressing the goal of platform-independence, our solution is a browser-based academic development environment powered by Amazon's cloud-based web services (AWS) [1]. Despite the large number of possible computing platforms (not to mention the vast number of Linux distributions), web-browser clients are ubiquitous and most of the popular ones provide compiled binaries (or at least stable source code with instructions for compiling) for the major operating systems. Instead of having students and instructors install and configure various software on their own systems in order to comply with the course requirements as mentioned in section 1, they would simply have to access a web URL using a browser for all their programming and development resources. As a consequence of the ease of access offered by a uniform browser-based environment, the issues of reliability and availability are also satisfied.

In order to tackle issues of scale, portions of the system will be built on the recently popular NodeJS framework [26]. Finally, the solution must be highly secure and must respect high standards of privacy in order to address issues regarding academic misconduct.

The above goals are achieved through the following list of features that are ideally supported by our system:

- Secure and private access controls for students and instructors
- Organize all programming assignments by course or topic
- Allow users to conveniently manage their files and folders
- A high-fidelity code editor
- Full support for compiling and executing (currently) C and C++ files
- Optional support for building entire projects
- Full access to administrative controls from within the application (by special administrative users with such privileges), including creating course/topic areas and enrolling students to them

Our solution was only briefly outlined in this section and carries a more specific set of implementation design points as well as issues. These are discussed in detail from section 5 onwards.

4. Related Work

The idea of an integrated development environment in the cloud is hardly a new one. A good number of such services exist including open-source projects that have been well-received. In our initial iteration, some even posed as candidates for use as a base foundation upon which our system could be built. However, as we developed our vision, we found that none of the existing solutions were able to cater for all of the needs and use cases of our system. A few of the most interesting ones will now be discussed.

4.1 Cloud9

Perhaps the most viable and visually appealing development environment in the cloud was Cloud9 IDE. It is a completely browser-based development environment primarily for developing web applications that utilize the standard web stack technologies, such as HTML, CSS, JavaScript, PHP, Python and Ruby [14]. Although it supports syntax highlighting and other neat language-specific features for editing the source code of a wide array of languages (both interpreted and compiled), it is these web technologies that are quite emphasized from all stages of the development process. This is because Cloud9 is currently in a partnership with Windows Azure, Cloud Foundry and the popular Heroku platforms in order to facilitate seamless application deployment straight from the development environment, and this list of partners is expected to grow as adoption of the IDE increases. Among other features, Cloud9 boasts a very feature-rich code editor complete with a project file browser tree, native support for Git and Mercurial version control systems, a custom command-line for basic UNIX shell commands, and a rich set of themes [13].

But two intriguing features are the integrated Terminal and the Cloud9 concept of Workspaces. According to the web site, a Cloud9 workspace is a virtual space assigned to a created project where all files, dependencies and libraries are stored. Within this runtime environment, various platforms can be run such as NodeJS, a traditional web stack (Apache, MySQL and PHP), Ruby, or Python [12]. Integrated with Workspaces is the recently added Cloud9 Terminal [27], which is a browser-based implementation of a UNIX terminal that runs a standard bash shell. With a premium account, users have complete access to all executables within a Workspace along with the standard shell commands for interacting with the file-system.

So far, users would only have Terminal access to the secure sandbox environment for their project and this is clearly very limiting. However, with the Remote SSH feature, users can log into a pre-existing server or virtual machine via SSH and integrate that platform with the Cloud9 Workspace. The entire file-system (as per user privileges on the server) are exposed to the Cloud9 Terminal and thus it acts as a normal shell where users can perform whatever operations that they could normally perform on that machine [15]. This leads to many interesting use cases for the Cloud9 IDE.

Despite its plethora of promising features, Cloud9 was unable to suit our needs in the given amount of time. Firstly, the Workspace and Remote SSH features successfully address our goal of platform-independence but do not truly eliminate the steps of environment setup and configuration for users. Secondly, the IDE is focused on web technologies and does not provide seamless build support of compiled languages like C/C++ and Java, which are used ubiquitously in CS courses. Lastly, although the software is open source and the source code is freely available for modification at its home in github [19], it is still a complex codebase that would be far too challenging to modify for our academic setting due to its very design and implementation. It is also worth mentioning that the IDE is built with collaboration with mind (this is also a highly promoted feature) and does not provide an intuitive means to integrate Workspaces with the access control and accounts system that we desire.

4.2 Compilr

The Compilr project [18] is a complete learning environment that allows users to write and run code on the browser and share it as well. It sports a highly cross-platform (even across mobile devices) interface due to its strong and obvious leverage of the Twitter Bootstrap front-end framework [11]. It also supports many compiled and interpreted languages and features an interactive console that functions like a standard UNIX terminal running a shell. The terminal is quite powerful and even supports user input via a standard blocking input stream such as stdin by ensuring users enter all inputs prior to executing the application.

It is quite evident that Compilr was built with open collaboration in mind (just like Cloud9) and although it features a highly active integrated forums, it once again did not satisfy our needs of privacy and access control.

4.3 Others

Apart from the above major solutions, there exist several small applications that were rather quickly decided to be unsuitable for our purposes but nevertheless had some interesting elements. The most impressive of these was **Coderun** [17], which provides a complete desktop-grade IDE along with unparalleled native compilation and debugging features [32] for use in the browser but is quite heavy-weight and currently only supports C# applications, Javascript web applications and PHP web applications. **Codenode [**16**]** is a browser-based interface to Python and Sage applications and provides an interactive REPL.

Ideone [21] is a promising browser-based debugging tool for more than 40 languages, including compiled languages. Although these solutions clearly were not usable by us, we were able to guage the scope of the existing solutions with regards to browser-based compiling and executing of native applications.

4.4 Combating Academic Misconduct

There are various existing solutions such as Moss [4] that have proven to be very effective in detecting similarities in software and consequently reducing plagiarism. However, it is well-known among all practices and industries that prevention is better than cure. Thus, we hope that preventing students from access to each other's work would prove significantly beneficial by attempting to prevent events of academic misconduct from occurring in the first place.

4.5 Verdict

It should be evident that there was no solution from the above that we could have leveraged as a base for our system. While most solutions had fantastic front-end features like project/file management and code editing (which we regret having to let go) and also had interesting implementations of native compilation and executing via custom terminals and shells, they simply lacked in simplifying the setup and configuration process for academic users. Also, most of the IDEs emphasized support for web-based programming and technologies. This is the general trend in curren web-based IDEs and is also noted by the authors of a paper (in section 2.3.2) that proposes an architecture for a contemporary web-based IDE [29]. Additionally, most of them provided a solution for preventing academic misconduct as they mostly encouraged open collaboration. In other words, we found useful features that were unfortunately spread across multiple solutions.

Although we did compose existing tools and technologies together to create our service, we still had to implement most of our features from the ground up and tailor them towards the needs of academia.

5. System Design

In this section we present in detail the design and implementation of our proposed solution. The architecture of our system is shown in Figure 1. The system is composed of three layers according to the 3-Tier model [24] and the three layers of the architecture are briefly expanded upon in the following sub-sections.

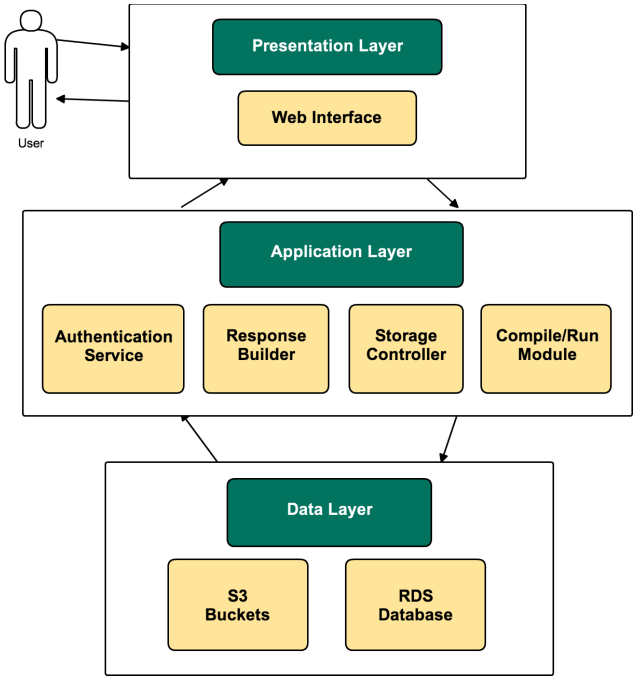

Figure 1: Architecture of CloudLab

5.1 Presentation Tier

The presentation tier of our system is designed to allow users to view and interact with data. This component consists of a web interface that is hosted on an Amazon EC2 instance (see section 6). It is implemented using HTML5, CSS3 and JavaScript technologies along with the assistance of the popular frameworks Twitter Bootstrap [11] and jQuery [23] that provide a more aesthetic look-and-feel and intuitive DOM manipulation respectively.

The Presentation tier is implemented as a web interface with access control in mind. There are three types of users for this system: **students**, **instructors** and **administrators**.

The following is a list of the main features that are supported for students:

- View and access all courses (or topics) that the student is enrolled in
- View and access all assignments (or projects) that exist in the course
- View, edit (and save changes) and delete all files that are associated with the assignment that are owned by the student
- Create new files for the assignment
- Upload any files to the assignment
- Compile source files (currently *.c and *.cpp files) and execute them
- View all compilation and execution output
- Submit an assignment and its files to the instructors of the course

The following are some of the main features that are supported for instructors:

- View and access all courses that the instructor is enrolled in
- View, edit, delete and access all assignments that exist in the course
- Create new assignments for the course
- View, edit (and save changes) and delete all files associated with the assignment that are owned by the student
- View (but not edit or delete) all files that are owned by other students in the same course
- Upload any files to the assignment
- Distribute an assignment and its files to all students in the course

Finally, the following are the features that are supported for administrators:

- View a list of all existing student and instructor accounts
- Edit or delete any existing student or instructor account
- View a list of all existing courses
- Edit or delete any existing course
- Create new users
- Create new courses
- Enroll existing users in existing courses

The web interface consists of a total of five pages: for students and instructors, we have the login/authentication page, the courses page (displays a list of all enrolled courses), the projects page (displays a list of all assignments/projects for the course) and the file editor page (displays a list of all files/source code and allows the editing of the same); for administrators, we have the control panel page which allows administrators to perform all admin tasks as mentioned in the features list above. Figure 3 shows an example of creating a new source file.

When instructors create assignments for courses, they are able to create files (usually demo files or skeletons) for the assignment or upload any similar files to the same. Ideally, the instructor can also restrict the types of file that can be created for the project if desired. The distribute feature can then be used to send the content created by the instructor to the students who will be able to view the same from their own accounts. Assignment submission works using the very same mechanism. When students use the submit feature, all the contents of their assignment are sent to the instructors' accounts who can then access the various submissions from their account. The submission feature is slightly different from the distribution feature in that instructors must have a more organized view of the assignments of the various users.

Each of the pages, except for the authentication page, contains elements that must be dynamically populated by the data from the Data tier. This is briefly expanded on in the Application and Data tiers subsections (sections 5.2 and 5.3).

5.2 Application Tier

The Application tier of our system primarily consists of code that controls the data flow between the Presentation and Data tiers and thus serves as an intermediary between the two. The software components are implemented as an API that receives POST requests from the web interface of the Presentation Tier that specify the data that needs to be accessed (or manipulated) from the Data tier. The information from the Data tier is retrieved, the necessary actions are carried out and a response (that follows a specific format) is constructed and sent back to the Presentation tier (which is aware of the format of the response). These components are implemented in a server-side scripting language (described more in section 6) so they can easily be invoked from the web interface. The requests processed by the Application tier can include saving a file, opening a file, executing a compiled file, retrieving a list of courses that a user is enrolled in, deleting a project, and so on and so forth.

5.3 Data Tier

The Data tier of our system consists of the model of the data involved in the system as well as the physical data. The data model is implemented as a relational database and functions as the interface between the Application tier and the physical data. A more detailed discussion is provided in sections 6.3 and 6.4.

6. Implementation Technology Overview

We now discuss the major technologies and tools that we used in the implementation of our system.

6.1 Cloud hosting and Amazon EC2

All the 3 tiers of the system are hosted on Amazon's various cloud computing platforms. AWS provides the scalability, reliability and availability that we need for our system. AWS provides developers with the flexibility of being able to programmatically or manually provision compute instances on a per-needs basis. Compared to competitors Heroku and Google App Engine, AWS possesses a better infrastructure and architecture for scalability and overall performance [30]. From the financial side of things, Amazon charges on a per-use basis (not unlike many providers) and practically eliminates the fixed costs that come with private clouds (maintenance, configuration, installation, updates and improvements, provisioning, etc.) while transferring them to variable costs

by providing the aforementioned as a service [9]. But the icing on the cake is that Amazon has been consistently reducing their prices more than any other provider and plan to continue doing so due to their innovative R&D and business model that focuses on low-profit margins as a result of economies of scale [9]. This proves to be a win-win scenario for academic institutions that do not possess the relatively larger funds and resources that for-profit corporations and businesses usually do.

The primary service that we used is the Elastic Compute Cloud (EC2) service that offers virtual instances with resizable compute capacity in the cloud from which services can be delivered [5]. These instances can run various platforms via an AMI (Amazon Machine Image), which is a special type of pre-configured operating system or virtual application software from which a virtual machine on an instance can be created [6]. We use EC2 instances to host the various components of the system. Certain instances can be accessed via SSH (securely using a public/private keypair) and the required tools can be installed on the system and the adequate configurations applied.

Ideally, we would use three or more separate instances for the three tiers of our system but we currently host almost all of our components on a single Ubuntu 12.04 Server instance, which was chosen due to familiarity with the Ubuntu Linux distribution. All the required tools (as described in the following sub-sections) were installed and configured as mentioned in the previous paragraph. For web-hosting, we went for a traditional LAMP stack and installed the latest versions of Apache HTTP Server [10] and PHP while our MySQL server was hosted using Amazon RDS (described in section 6.4). For compiling and executing, we installed the latest versions of NodeJS and Socket.IO (via a Ruby Gem-like package manager, NPM [25]). Furthermore, our Ubuntu instance was configured with the latest GCC and G++ compilers for C and C++ programs. All our actual data (files) are stored using Amazon S3 (described in section 6.3).

6.2 Web Interface and Code Editing

Apart from the useful features mentioned in section 5.1, the most exciting component of our system is the source code file (or plaintext) editor of the web interface. It is provided by the open-source Ace [3], which is built using JavaScript and is thus embeddable as a HTML document element. Among the myriad of features that it offers (see [3]), the following are the most important ones for our system:

- Text-entry modes for all application types targeted by our service (currently C, C++)
- Syntax highlighting and some auto-completion of code
- Auto-indenting and code organization
- Extensive support for key-shortcuts and special key-bindings for various systems and environments that can also be programmatically edited or defined
- Full browser support for all 4 major web browsers (Firefox, Chrome, Internet Explorer, and Google Chrome)
- Handles huge documents (the current maximum is quoted as 4 million lines)

Additionally, it is being used as the primary code editor for the Cloud9 web-based IDE [20] and is being actively maintained. Hence, any improvements in Ace will be directly reflected in our system.

A final note must be made on the compile and execute commands that can be run from the respective buttons of the interface. For security purposes, users will not be permitted to enter their own commands as if they were using a shell. Instead, they will have the option of choosing parameters through a drop-down box (such as program arguments, compiler flags, etc.) which ensures that safe commands are passed to the compilation/execution components.

6.3 Data Model and Amazon RDS

Our system contains a very simple but effective model that represents the typical usage of the system. An ER diagram depicting the entities and relationships between them is shown in Figure 2.

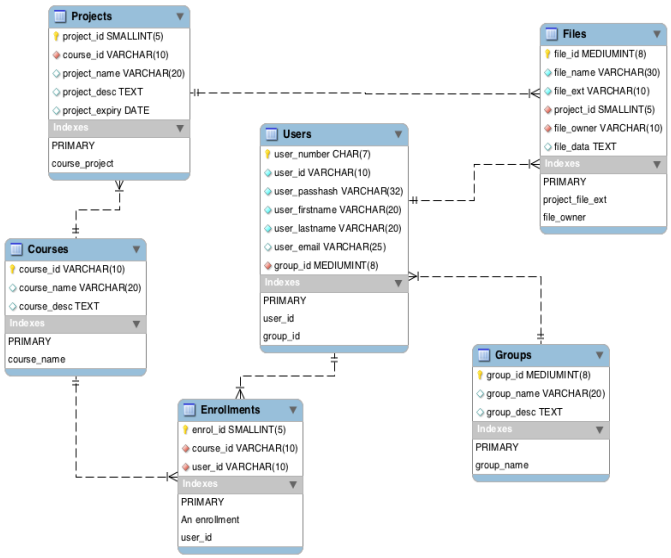

Figure 2: Entity-Relationship Diagram of the Cloudlab Data Model

The model is implemented in the form of a relational database hosted on an Amazon RDS (Relational Database

Service) [7] instance using the popular MySQL engine [33]. MySQL was chosen because of its excellent integration with the PHP language (described in the next sub-section). Additionally, familiar tools such as phpMyAdmin [28] have really simplified the database management process in previous projects and this also factored as a motivation for our choice.

Access control natively exists within the model through the database relationships and constraints. Files each have a single unique owner (User) and are indexed by a unique auto-incremental integer as well as a unique pair of Course and Project. Projects (or assignments) are unique within a Course and since users can only create/upload files from within a project (as mentioned in section 5.1), Users can only own files that are associated with a Course and a Project. As a consequence of this relationship and also the implementation of the interface, users can only query for files that they own. Exceptions to this rule can easily be made through PHP scripts to accommodate for the desired functionality.

Amazon RDS instances provide dedicated hosting of relation databases and offer complete handling of all back-end maintenance including database software updates and back-ups. In the case of any failure, RDS instances also offer full control via the web-based console (or programmatically through API calls) to carry out point-in-time recoveries [7] and rebooting of instances. Additionally, we gain the benefits of fast compute performance and scalability due to Amazon's implementation and design (see section 6.1). As mentioned before, Amazon Web Services provide the ability to programmatically or manually (via the web console) scale applications by leveraging extra instances when needed.

6.4 Web Storage and Amazon S3

Amazon's S3 web storage service is used to physically store all files of users. The courses created by administrators are saved as S3 'buckets' [8], into which assignments and the respective source files of various users can be stored. Assignments/projects are represented as folders and the files are trivially stored as the actual physical files.

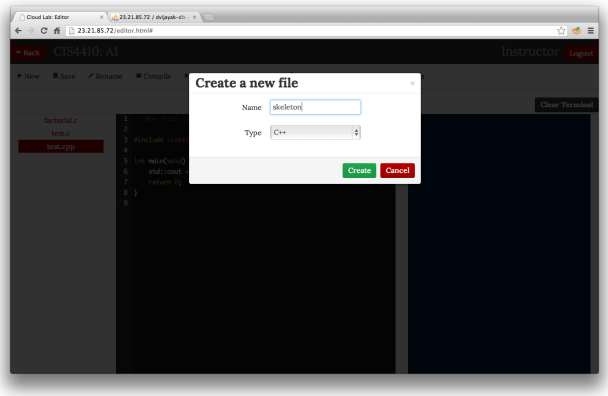

Figure 3: Creating a new source file in CloudLab

Although S3 does not support creation of real folders, folders can be simulated by appending the username to the front of the S3 object, directory-style. For example, the file "main.c" belong to user "bsmith" of assignment "Assignment 1" would be named "Assignment1/bsmith/main.c".

In order to perform such file operations and database queries (as outlined in section 6.1), there is a need for controller components to manage data between the web interface and the RDS and S3 services. These are implemented as PHP scripts on the web server side. PHP was chosen because it is the only familiar Web scripting language which has a mature API provided by Amazon. As expected, the S3 buckets and their contents represent the relationships described in the RDS database and can be accessed via PHP calls to the AWS SDK using the unique identifiers received from MySQL queries.

6.5 Server-Side Compilation and Execution

Compilation and execution services are provided by a NodeJS HTTP server that can be hosted on a separate EC2 instance. Most importantly, we used the Socket.IO framework [22] for NodeJS that provides easy real-time full-duplex bi-directional communication between a server and a client over the HTTP protocol by abstracting away all transport mechanisms [31]. In most modern browsers, it uses the very new WebSocket protocol [2] to accomplish this feat but can also fall back to previous mechanisms such as AJAX polling if needed.

Our NodeJS (henceforth referred to as "Node") server is implemented as a very basic HTTP server that listens for HTTP connections from clients (the browser viewing the web interface) on an unreserved port (we use 8080). The server also contains a Socket.IO socket that listens for established connections. Once the connection is established, the client-side JavaScript library for Socket.IO that enables the WebSocket protocol is served to the browser client and the connection on the server is then

routed to the Socket.IO socket. A full-duplex communication has now been established. Note that we do not use port 80 as our Node server lies on the same instance that runs our Apache HTTP server. A better infrastructure would contain the Node server on a separate instance and could then use port 80 if desired.

Message-passing between the browser client socket and the Socket.IO server socket works very much like how standard sockets do. A protocol of events to be handled is established and either (or both) sides listen for or generate these events. In our system, the browser client (which would be on the code editor page in the web interface) generates (or emits) compile and execution events and passes the contents of the source code being edited as well as unique identifier information to the server. The server listens for these events and performs the compilation or execution operations accordingly. Any output from compilation or execution is emitted back to the client via another event. This event is handled by the client and the output is displayed to the user on the terminal widget on the interface.

When the Node server carries out a compile operation, it creates a reserved directory within the instance file-system, and the contents of the received message (which contains the source code) are written to a file that is uniquely named with a combination of the file owner's name, the assignment id and the course name. The appropriate compiler is determined via the file extension and compilation is performed, the bytecode file is created (within the same directory) and the output is sent to the client, as mentioned before. In a similar fashion, when an execute operation is performed, the Node server checks to see if the reserved directory exists, looks for the respective bytecode file that was generated in the compilation process and then executes it. All the output is once again sent to the client in the same manner described above. If desired, the temporary file containing the source code can be deleted but the executable bytecode must not be deleted in order to allow users the feature of executing a previously compiled program file without having to re-compile it all over again.

We used NodeJS for compilation and execution due to its support for server-side push (via Socket.IO) capabilities which are required for real-time compilation that provides a bash shell-like experience. Additionally, NodeJS is known for its immense improvements in application scalability due to its implementation of asynchronous non-blocking I/O operations [34, 35]. The majority of time spent on our system is most likely to be by students compiling and executing their various programs, and the compilation and execution process is highly indeterminate. Thus, Node's elimination of waiting time between I/O operations significantly benefits our system and allows it to scale significantly despite large amounts of concurrent users.

7. Prototype Implementation

Our efforts in realizing our vision have been successful so far through the development of an initial prototype model of the system called the Playground. The purpose of the Playground system is to allow instructors to create a demonstration environment for students where they can demonstrate sample programs and compile them while students can access the very same demo files and play around with them without having to save or create new files.

Instructors can access a course and create assignments or projects for it. Each assignment can then be populated with some files by using the file editing page. After the files for the assignments have been created, they will become accessible to students in their **own** accounts. A single student user account is sufficient since students cannot create or save files. Administrator user features have been fully implemented and do not change in the Playground system. The Playground mode helps instructors convey programming concepts during a lecture or a lab through a live demonstration, while providing a separate space for students to also interact with the program. An interesting benefit of this mode is that a single student account is enough since students cannot perform file management operations.

Although the entire system is currently running in Playground mode, most of the features for the full-fledged environment have already been implemented. These include project and file operations such as creating, editing and deleting projects and files. Access control is also fully implemented and this is demonstrated in the fact that only instructors can see the toolbar on the projects page and that only instructors receive the full toolbar on the file editing page, while students only receive options to compile and execute the file and change the editor preferences. However, a small number of interesting features are yet to be implemented. Uploading files has not been implemented yet. The assignment submission and distribution features have been skipped for now. Uploading files to an assignment is currently not implemented. Also, there is a planned feature where instructors can add a description of the assignment via a markdown README file or other similar formats which will then be rendered to HTML and displayed on the projects page. This is currently unimplemented.

As briefly mentioned in earlier sections, only C and C++ projects/files have been supported for now. This was done in order to greatly simplify the development of the system and will continue to remain this way until significant

improvements are achieved in the compilation and execution features.

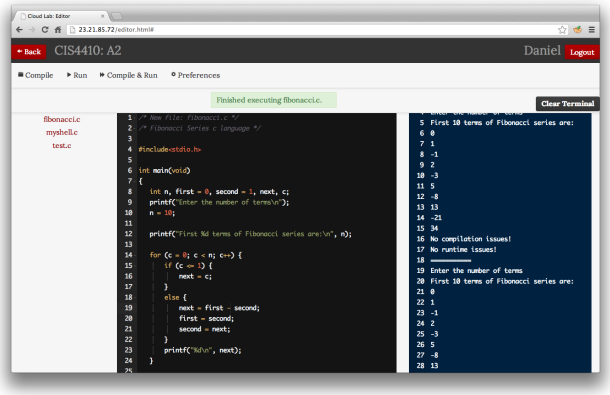

Figure 4: Compiling and executing a C program

8. Limitations and Future Work

Although our prototype Playground implementation is quite functional, it has allowed us to gauge the various limitations and challenges of our proposed solution as a whole. Some of the major ones are listed below and possible future work is also briefly discussed.

- Although there are a decent number of file operations, there is a lack of supported project operations. We would like to implement more project management features, such as build project, submit project and distribute project (mentioned in earlier sections, including section 7).
- The file/code editing interface is currently quite inconvenient to use despite its vast number of predefined shortcuts. This is because the keybindings for these shortcuts are quite inconvenient as they are designed not to interfere the user's browser's keybindings. More interesting commands and shortcuts should be implemented. Creating an effective, slick IDE is a very time-consuming and challenging task, and is easily enough work to become a project of its own. Our current plan is to continue improving our existing Ace implementation.
- As mentioned in section 6, the entire system (except for the Data tier) is currently hosted on a single Micro EC2 instance. This is a violation of the decoupling of the system components in the 3-tier model, and is also inhibiting the overall compute performance of the system. Although this was done just for the sake of simplicity, the Presentation and Application tiers must be truly decoupled in the future by hosting them on separate EC2 instances.
- The system is currently unable to process large numbers of requests at a reasonable speed. This is in part due to the limited compute capacity of the Micro EC2 instances that host the system. Performance can be improved by moving the system to a better instance type and also programmatically provisioning extra instances as needed. This back-end feature is a huge undertaking in itself but is necessary to truly leverage Amazon's cloud computing platform.
- On a similar note to the above limitation, only the compiling and executing components are implemented in Node while the rest of the Application tier was developed using the standard server-side scripting language PHP. Significant gains in scalability and concurrency will be garnered if the entire application was served by a Node server.
- In the current implementation of compiling, the Node server writes the contents to a file that is provided a so-called unique name, which is a concatenation of the file's owner and the assignment to which the file is associated. This causes issues for the Playground prototype as for multiple students from the same course working on the same assignment, the server would frequently write to the same file. This violates the principle of isolation and users cannot be sure that the compiled file is truly the bytecode of the program that they were editing. We need to improve our unique identifier for the file that is created during compile time. One valid and efficient solution is to simply concatenate the PID of the process that is running the compile command to the existing file name. According to the UNIX man pages of the fork() function, the PID is guaranteed to be unique. More processor intensive solutions would involve generating hashes for the file names.
- No support for handling functions that invoke the standard blocking input stream (stdin) such as scanf() and getchar(). Some existing solutions (mentioned in section 3) support this feature by requesting users to enter any input prior to running the compiled program. While this certainly is a possible solution, we can do better. A plausible idea worth investigating would be to synchronize the entire executing process step-by-step on the Node server with the client thereby informing the client when an input is required.

- Although it was mentioned in section 7 that the current use of the MySQL database on the RDS instance for physical storage of file contents is an inefficient implementation and that the files must be moved to S3 and the database used to interface with S3, there is no need to completely eliminate storage of file contents in the MySQL database. To improve performance, active files can be temporarily cached in the database. When the file is closed, the latest saved data can then be transferred to the S3 bucket. This is expected to be more efficient in theory since POST responses should be more lightweight than the SimpleXML responses sent by the AWS API. However, further investigation will need to be conducted to verify this theory.
- An issue that can arise with our browser-based solution is cross-browser compatibility. The Playground is currently tested and fully supported in the latest versions of Chrome and Firefox. In order to cater to a wide range of browser preferences among students and instructors, the web interface must be made to be standards-compliant.

Our prototype system is currently being used by Dr. Mahmoud (one of the authors) in an introductory programming course ENGR1200 that he is teaching at the University of Ontario Institute of Technology (UOIT). The results of this semester-long deployment will provide further guidance and insight towards the improvement of our solution.

9. Conclusion

CloudLab is intended to simplify the many issues that arise with course lab and assignment development for both computing students and instructors. The various problems and issues outlined truly echo the same obstacles that we had faced during our academic pursuits in our undergraduate studies. It was also intended to explore a possible useful application of cloud computing and web services. Our hope is that the proposed cloud computing service will significantly enhance the user experience of both students and instructors and facilitate increased learning by reducing the impact of the problems mentioned above.

10. References

[1] About AWS. (2013). Retrieved January 29, 2013, from http://aws.amazon.com/what-is-aws/

[2] About HTML5 WebSockets. (2012). Retrieved January 29, 2013, from http://www.websocket.org/aboutwebsocket.html

[3] Ace - the high performance code editor for the web. Retrieved January 29, 2013, from http://ace.ajax.org/#nav=about

[4] Aiken, A. (June 9, 2011). Moss - A system for detecting software plagiarism. Retrieved January 29, 2013, from http://theory.stanford.edu/~aiken/moss/

[5] Amazon EC2 functionality. Retrieved January 29, 2013, from http://aws.amazon.com/ec2/#functionality

[6] Amazon machine images (AMIs). Retrieved January 29, 2013, from https://aws.amazon.com/amis

[7] Amazon relational database service (amazon RDS). Retrieved January 29, 2013, from http://aws.amazon.com/rds/

[8] Amazon simple storage service (amazon S3). Retrieved January 29, 2013, from http://aws.amazon.com/s3/

[9] AmazonWebServices (Producer), & AmazonWebServices (Director). (2012, Novemebr 28, 2012). *2012 re: Invent dat 1 keynote: Andy jassy.* [Video/DVD] YouTube: YouTube.

[10] Apache HTTP SERVER PROJECT - what is the apache HTTP server project? Retrieved January 29, 2013, from http://httpd.apache.org/ABOUT_APACHE.html

[11] Bootstrap. Retrieved January 29, 2013, from http://twitter.github.com/bootstrap/

[12] Cloud0 IDE - sign up for standard or premium! Retrieved January 29, 2013, from https://c9.io/site/pricing/

[13] Cloud9 IDE - features. Retrieved January 29, 2013, from https://c9.io/site/features/

[14] Cloud9 IDE - your code anywhere, anytime. Retrieved January 29, 2013, from https://c9.io/

[15] Cloud9 user documentation - running your own SSH workspace. (). Message posted to https://docs.c9.io/run_your_own_workspace.html

[16] Codenode - interactive online programming notebook. Retrieved January 29, 2013, from http://codenode.org/

[17] CodeRun - online IDE. Retrieved January 29, 2013, from http://coderun.com/ide/

[18] Compilr - online compiler & IDE for C, java, C# and C++. Retrieved January 29, 2013, from https://compilr.com/

[19] GitHub - ajaxorg/cloud9. Retrieved January 29, 2013, from https://github.com/ajaxorg/cloud9/

[20] gjtorikian. (2013). Readme.md - ace (ajax.org Cloud9 editor). Retrieved January 29, 2013, from https://github.com/ajaxorg/ace/blob/master/Readme.md

[21] Ideone.com - online IDE & debugging tool >> C/C++, java, PHP, python, perl and 40+ compilers and interpreters. (2013). Retrieved January 29, 2013, from http://ideone.com/

[22] Introducing SOCKET IO v.9. Retrieved January 29, 2013, from http://socket.io/

[23] jQuery - what is jQuery? (2013). Retrieved January 29, 2013, from http://jquery.com/

[24] Marston, T. (2012). What is the 3-tier architecture - the 3-tier architecture. Retrieved January 29, 2013, from http://www.tonymarston.net/php-mysql/3-tier-architecture.html#3-tier

[25] Node packaged modules. Retrieved January 29, 2013, from https://npmjs.org/

[26] Node.js - node's goal is to provide an easy way to build scalable network programs. Retrieved January 29, 2013, from http://nodejs.org/about/

[27] Pardee, M. (2012, November 21, 2012). Cloud9 IDE blog - the terminal. Message posted to https://c9.io/site/blog/2012/11/the-terminal/

[28] phpMyAdmin - about. Retrieved January 29, 2013, from http://www.phpmyadmin.net/home_page/index.php

[29] Richelle Charmaine, G., Marc Anthony, M., & Audrey Elaine, G. (2010). *An architecture for a web-based IDE*. (Unpublished Bachelor of Science, Computer Science). De La Salle University Manila, Manila.

[30] Silverberg, E. (2011, February, 2011). The unofficial guide to migrating off of google app engine. Message posted to http://www-cs-students.stanford.edu/~silver/gae.html

[31] SocketIO - FAQ. Retrieved January 29, 2013, from http://socket.io/#faq

[32] Tasha, N. (2013,). Message posted to http://www.hongkiat.com/blog/cloud-ide-developers/

[33] Why MySQL? Retrieved January 29, 2013, from http://www.mysql.com/why-mysql/

[34] Dahl, R. node. js. 2009. *Online: http://nodejs. org/jsconf. pdf.*

[35] Dahl, R. Node. js: Evented I/O for V8 JavaScript.