

CloudSuite

Ad Hoc Laboratories Using Cloud Resources

A Thesis Presented to

The Faculty of the Computer Science Program

California State University Channel Islands

In (Partial) Fulfillment

of the Requirements for the Degree

Masters of Science in Computer Science

by

Drew Alex Clinkenbeard

March 2013

© 2012, 2013

Drew Alex Clinkenbeard

ALL RIGHTS RESERVED

All images and code generated by author unless otherwise indicated

APPROVED FOR THE COMPUTER SCIENCE PROGRAM

Andrzej Bieszcza 4/29/13

Advisor: Dr. Andrzej Bieszcza Date

Peter D Smith 4/29/13

Dr Peter Smith Date

Michael Berman 4/29/13

Dr Michael Berman Date

APPROVED FOR THE UNIVERSITY

Gary A. Berg 4-29-13

Dr. Gary A. Berg Date

Non-Exclusive Distribution License

In order for California State University Channel Islands (CSUCI) to reproduce, translate and distribute your submission worldwide through the CSUCI Institutional Repository, your agreement to the following terms is necessary. The author(s) retain any copyright currently on the item as well as the ability to submit the item to publishers or other repositories.

By signing and submitting this license, you (the author(s) or copyright owner) grants to CSUCI the nonexclusive right to reproduce, translate (as defined below), and/or distribute your submission (including the abstract) worldwide in print and electronic format and in any medium, including but not limited to audio or video.

You agree that CSUCI may, without changing the content, translate the submission to any medium or format for the purpose of preservation.

You also agree that CSUCI may keep more than one copy of this submission for purposes of security, backup and preservation.

You represent that the submission is your original work, and that you have the right to grant the rights contained in this license. You also represent that your submission does not, to the best of your knowledge, infringe upon anyone's copyright. You also represent and warrant that the submission contains no libelous or other unlawful matter and makes no improper invasion of the privacy of any other person.

If the submission contains material for which you do not hold copyright, you represent that you have obtained the unrestricted permission of the copyright owner to grant CSUCI the rights required by this license, and that such third party owned material is clearly identified and acknowledged within the text or content of the submission. You take full responsibility to obtain permission to use any material that is not your own. This permission must be granted to you before you sign this form.

IF THE SUBMISSION IS BASED UPON WORK THAT HAS BEEN SPONSORED OR SUPPORTED BY AN AGENCY OR ORGANIZATION OTHER THAN CSUCI, YOU REPRESENT THAT YOU HAVE FULFILLED ANY RIGHT OF REVIEW OR OTHER OBLIGATIONS REQUIRED BY SUCH CONTRACT OR AGREEMENT.

The CSUCI Institutional Repository will clearly identify your name(s) as the author(s) or owner(s) of the submission, and will not make any alteration, other than as allowed by this license, to your submission.

CloudSuite, Ad Hoc Laboratories Using Cloud Resources
Title of Item

Masters Thesis, Cloud Computing, Virtualization, Scientific Computing
3 to 5 keywords or phrases to describe the item

DREW A. CLINKENBEARD
Author(s) Name (Print)

Drew A. Clinkenbeard
Author(s) Signature

30-APR-2013
Date

CloudSuite : Ad Hoc Laboratories Using Cloud Resources

by

Drew Alex Clinkenbeard

Computer Science Program

California State University Channel Islands

Abstract

CloudSuite presents a proof of concept for cloud based ad hoc virtual laboratories. Using Amazon Web Services and a persistent web server, CloudSuite allows students and instructors to explore ideas in computer science. CloudSuite provides a web based interface to configure labs consisting of advanced algorithms and, using an Amazon Elastic Cloud Compute instance, execute those labs. The results of the execution are then made available for analysis. This thesis discusses the technology necessary for implementing this system as well as providing the code necessary to demonstrate the feasibility of such a system. Finally this thesis also presents a roadmap for future development of CloudSuite.

Acknowledgements

I would like to thank Dr. Andrzej Bieszczad for his patience, guidance, and encouragement on the long journey towards the completion of this project. My meetings with Dr Bieszczad helped keep me sane and on task.

I would not have been able to complete this project without the support of my wife. The hard work she put in on her own thesis inspired me to keep working, her encouragement kept me writing, and her proofreading made it all presentable. Thank you Jenny!

I would also like to thank my family, my in-laws, and my ‘Camarillo family’ whose support, encouragement, and home cooked meals sustained me throughout.

Finally I have to thank Shure, Thomas Leeb, and Daft Punk. Without headphones and music this never would have happened.

TABLE OF CONTENT

1. INTRODUCTION.....	9
USE CASES	9
UNDERGRADUATE STUDENTS	9
UPPER DIVISION OR GRADUATE STUDENTS.....	10
PROFESSORS	10
FUNCTIONAL REQUIREMENTS	10
CREATING LABS	10
SAVING LABS.....	10
LOADING LABS	11
DISTRIBUTING LABS.....	11
MODIFYING LABS.....	11
CONFIGURING MODULES.....	12
RUNNING A LAB.....	12
REMAINING CHAPTERS	13
KEY TERMS	14
2. FIELD OVERVIEW	16
WHAT IS CLOUD COMPUTING	16
HISTORY OF CLOUD COMPUTING	17
VIRTUALIZATION	18
HYPERVISORS	18
PARAVIRTUALIZATION	19
AMAZON.COM ELASTIC COMPUTE CLOUD.....	20
OTHER CLOUD BASED SERVICES.....	21
HEROKU	21
GOOGLE APP ENGINE.....	22
EUCALYPTUS AND APPSCALE.....	22
LANGUAGES	22
CODE FRAMEWORKS, LIBRARIES AND APIS.....	23
PHP SLIM.....	23
BOTO	23
ELEMENTTREE	23
JQUERY	23
BACKBONE.JS	24
UNDERSCORE.JS	24
TEMPLATING SYSTEMS	25
HIGHCHARTS	25
ONLINE EDUCATION	25
OTHER VIRTUAL LABS	25
KHAN ACADEMY.....	26
UNIVERSITY OF HAWAII VIRTUAL LAB.....	26
NAVY VIRTUAL LAB	26
GRIDS AND CLUSTERS	26
3. IMPLEMENTATION	28
WEB SERVER	29
PHP FRAMEWORK.....	29
USER	30
GROUP	30
MODULE.....	31
LAB	34
COLLECTION	36
XML MODELS.....	36
VIEW LAYER	36
AMAZON MACHINE INSTANCE	37

4. ANALYSIS OF CLOUDSUITE.....	38
STANDARD USER	38
LOGGING IN	38
CREATING A LAB	39
ADDING MODULES TO A LAB.....	40
REMOVING MODULES FROM A LAB	42
EDITING MODULES	42
SAVING A LAB.....	42
LOADING A LAB.....	43
QUEUEING LABS.....	44
DELETING LABS	44
RESULTS FROM A LAB	44
SUPERUSERS	45
DISTRIBUTING LABS.....	45
STARTING / STOPPING THE SERVER.....	46
5. CLOUDSUITE OUTPUT.....	47
LAB RESULTS	47
EXAMPLE EXPERIMENT	51
6. THESIS RESULTS	53
USE CASES	53
UNDERGRADUATE STUDENTS	53
UPPER DIVISION OR GRADUATE STUDENTS.....	53
PROFESSORS	54
FUNCTIONAL REQUIREMENTS	54
CREATING LABS	54
SAVING LABS.....	54
LOADING LABS	55
DISTRIBUTING LABS.....	55
MODIFYING LABS.....	55
CONFIGURING MODULES.....	55
RUNNING A LAB	55
ACCESSING STUDENT DATA	56
7. FUTURE WORK.....	57
USER MANAGEMENT	57
SECURITY	58
SESSIONS	58
LAB MANAGEMENT AND DISTRIBUTION	59
EDITING MODULES	59
COLLECTIONS.....	59
VIEW LAYER	60
DATA IMPORT AND USE	60
AMI / DATA PROCESSING	60
MODULE UPLOADING / PACKAGE MANAGER.....	61
API.....	61
8. CONCLUSION	62
Appendix	67
A. PHP CLASSES	67
B. SERVER MODULES	104
C. VIRTUAL HOST FILE	105
D. API REFERENCE.....	106
E. XML SCHEMAS.....	109
F. PYTHON WRAPPERS	119

TABLE OF FIGURES

Figure 1.	Preliminary design of the CloudSuite lab interface.....	11
Figure 2.	Preliminary design for adjusting modules.....	12
Figure 3.	Model View Controller diagram [2].....	15
Figure 4.	The two most common types of hyoervisor [20].....	19
Figure 5.	Control flow diagram of CloudSuite.....	28
Figure 6.	Control and data flow in CloudSuite.....	30
Figure 7.	Module used for encrypting and decrypting files.....	31
Figure 8.	XML representation of a lan showing a configured module.....	32
Figure 9.	Partial example of an XML representation of the RSA module.....	33
Figure 10.	Partial example of an XML schema that defines a lab.....	35
Figure 11.	Login button.....	38
Figure 12.	Username and password fields.....	38
Figure 13.	Failed login alert.....	38
Figure 14.	The username is displayed after successful login.....	39
Figure 15.	Logout options.....	39
Figure 16.	The task bar showing the current user options.....	39
Figure 17.	New lab dialog.....	39
Figure 18.	Warning when trying to use whitespace in a lab name.....	40
Figure 19.	Lab name and description displayed in the main lab area.....	40
Figure 20.	Modules displayed in CloudSuite.....	41
Figure 21.	Configuration options for the module 'neuro'.....	41
Figure 22.	Modules displayed in a lab.....	42
Figure 23.	Confirmation upon removing a module.....	42
Figure 24.	Lab saved success alert.....	43
Figure 25.	Display of labs available to load.....	43
Figure 26.	Queued lab alert.....	44
Figure 27.	Confirm delete lab dialog.....	44
Figure 28.	Listing of users data modules.....	45
Figure 29.	Data dialog.....	45
Figure 30.	Example lab distribution page.....	46
Figure 31.	Example server status dialog when a server is running.....	46
Figure 32.	The 'graph' module Configured with existing data.....	47
Figure 33.	Results from running the 'graph' module.....	48
Figure 34.	The data used to generate Figure 33.....	48
Figure 35.	A configured 'ga' module.....	49
Figure 36.	A CSV file showing the minimum and maximum gene values from the 'ga' module...50	
Figure 37.	The configured genetic algorithm lab.....	51
Figure 38.	The configured 'result_graphs' lab.....	51
Figure 39.	Graph showing single point crossover.....	52
Figure 40.	Graph showing two point crossover.....	52

Chapter 1:Introduction

This thesis presents a proof of concept for CloudSuite, Ad hoc cloud based laboratories. CloudSuite provides a cloud based framework to create virtual labs for the demonstration of scientific computing techniques. The framework allows instructors and students to create and share laboratories to demonstrate computer based simulations and processes. CloudSuite is intended to be as platform agnostic as possible. As such the framework for CloudSuite was developed using PHP and JavaScript. This allows CloudSuite to run in all modern web browsers, allowing the largest possible user base.

After reviewing the available literature, and the available cloud resources, we concluded that a cloud based method for creating and running virtual labs is a necessary component of higher education. Although there are several cloud based solutions for running arbitrary software in the cloud, our research did not find any examples that were tailored for the demonstration of scientific computation. Services like Heroku [33], Google AppEngine [41], and other “platform-as-a-service” providers do not offer a front end that addresses the many use cases that arise when demonstrating techniques in scientific computation. In this thesis we present a cloud based framework for this express purpose.

I. Use Cases

We developed several use cases by visualizing how CloudSuite could be used in an academic setting. The actors identified for these use cases are undergraduate students, upper division or graduate students, and professors. The primary concern for undergraduate students is running labs that have been created by their professors and configuring new labs for their own use. Graduate students will also create and run labs, but the primary focus for graduate students is on creating and configuring software for use in CloudSuite. Professors use CloudSuite to create and distribute labs, view the results from labs which have been run by others, and create new processes to run in CloudSuite. In the following section we explore the needs of each concerned party and how CloudSuite addresses those needs.

a. Undergraduate Students

Undergraduate students primarily use CloudSuite to experiment with preconfigured labs. Students must also be able to create and modify their own labs. Undergraduate students are limited in that they are not able to distribute labs to other users. Undergraduates are also only capable of viewing or modifying labs that have been shared with them or that they have created. An undergraduate student would receive a lab from a professor, modify the parameters contained within the lab, save the lab, then queue the lab for execution. Once the lab has run to completion, the student would analyze the results of the lab e.g., examine the generations created in a genetic algorithm, download an image produced by a graphing algorithm, or attempt to view the contents of an encrypted file.

b. Upper Division or Graduate Students

In addition to creating and modifying labs, graduate students will be expected to use CloudSuite in a more technical way than undergraduates. Graduate students will be given assignments to create processes and data sets for use in CloudSuite labs.

c. Professors

Professors require all of the abilities available to students: creating labs, modifying labs, creating new data, and new processes to operate on data. Professors must also be able to distribute labs to students either singly or in groups. Finally professors will use CloudSuite to view and edit student labs. This will allow professors to assist students with homework as well as allowing the professor to evaluate the results of a student lab.

II. Functional Requirements

Visualizing a typical classroom setting led us to develop several scenarios that define CloudSuite. The initial metaphor we addressed was that of an in-class lab assignment. We visualized all the tasks that are necessary to demonstrate an algorithm or teach a new technique. We concluded that creating, saving, loading, distributing, modifying, and running a lab in CloudSuite are necessary for CloudSuite to be functional. We expand on this set of criteria in the following sections.

a. Creating Labs

Labs will be created using a simple point and click interface. Each lab will possess a unique name to identify it to the user that created the lab. Creating a lab writes a unique file to a persistent server. The lab file will adhere to predefined criteria that will ensure new labs are compatible. Each lab will consist of zero or more objects that represent either data or operations upon data.

b. Saving Labs

A lab will be saved by clicking on a dedicated save button. Saving a lab will write a unique file to a persistent server for later retrieval or modification. The user will be notified with the status of the save, either success or fail. Saving a lab will not overwrite the labs of other users. A user may have an unlimited number of labs though it will be possible to limit the number of labs available to each user.

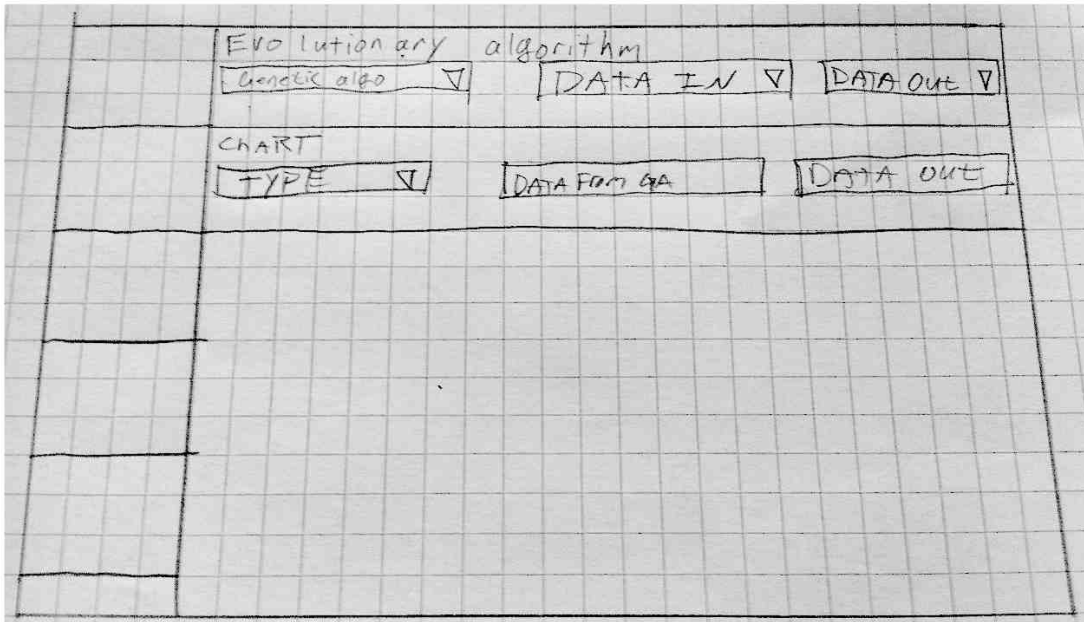


Figure 1. Preliminary design of the CloudSuite lab interface.

c. Loading Labs

Loading a lab is accomplished by selecting the name of the lab from a list of labs to which the user has access. Additionally, when a user first access CloudSuite, they will be presented with a list of labs that may be loaded. Only those labs belonging to the user, or that have been distributed to the user, will be available to load.

d. Distributing Labs

Running a lab as an in-class or homework assignment is one of the primary goals of CloudSuite. To enable this functionality a professor must be able to make configured labs available to students. This is accomplished by selecting a user, or group of users, that will gain access to the lab. This will then make the lab selectable for those users. When a user loads a lab that has been made available in this way the associated file representing that lab is copied to the users lab directory. Once a lab has been distributed to a user, that user may modify the lab without changing the original lab file. Distributing a lab, by necessity, distributes all of components associated with that lab. Components that are distributed with a lab may also be used in the creation of new labs, or the modification of existing labs.

e. Modifying Labs

In CloudSuite a lab is composed of 'modules' that represent either data or operations upon data. Additionally if a module is to produce data then that data will immediately be operable by other modules. Modules that operate upon data will be idempotent, when data is operated upon a copy of that data is produced, the original data is not modified. Each CloudSuite module represents a specific data set or operation. Users will only have access to modify their own labs unless they are of high enough privilege to edit the labs of others.

f. Configuring Modules

Modules are added to the lab through a simple point and click interface. Upon adding a module to the lab the user is presented with the configuration options for that module. A module may possess multiple configuration options that are dependent upon the module e.g., a graphing module may have options for the type of graph to be created, which data should be used to create the graph, or what to label the axes of the graph. Modules may be added or removed from a lab without affecting the operation of the other modules. If a module produces data that will be operated upon by one or more other modules, removing the module that produces the data will prompt the user regarding the missing module.

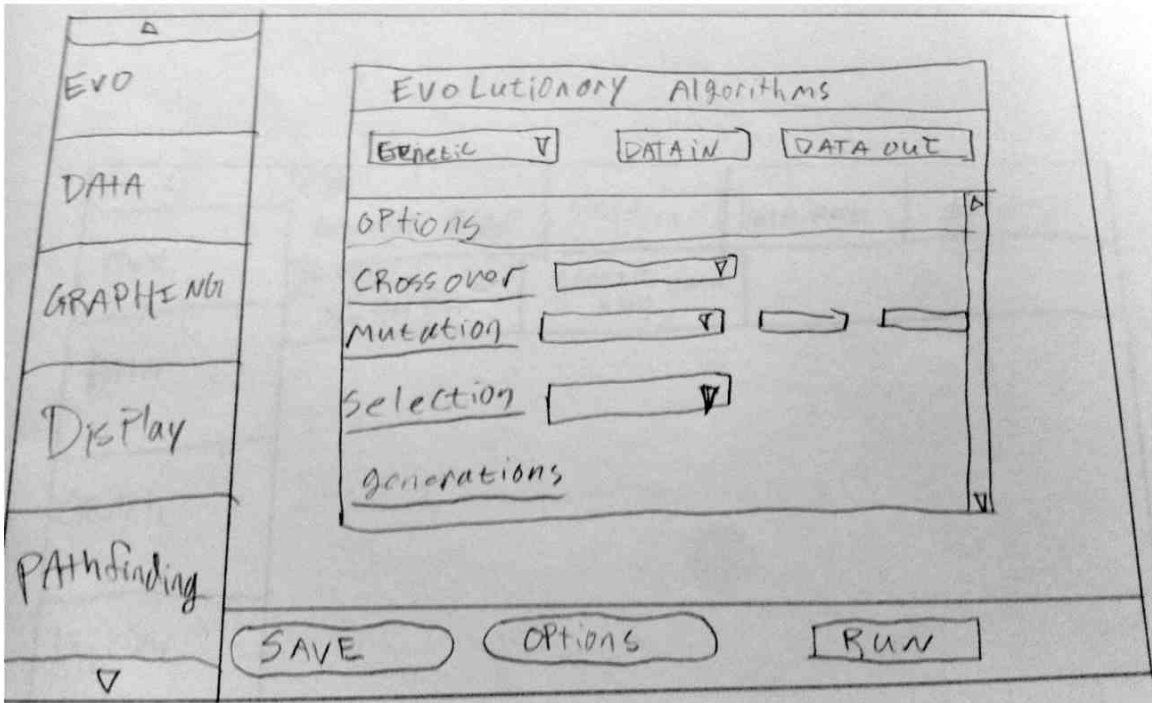


Figure 2. Preliminary concept for adjusting modules

g. Running A Lab

Lab processing is handled by Amazon Web Services and as such requires an active Amazon Machine Instance (AMI). Activating an AMI to process CloudSuite labs must be limited to users with high level of access, such as professors or administrators, or to users that are responsible for their own Amazon Web Services (AWS) account. Administrative users supply the AWS credentials that will be used when processing labs. This administrative user is also responsible for activating and deactivating the AMI. When a lab is selected to be run it is added to a 'first-in-first-out' queue that is processed when the AMI is activated. Administrative users may decide to allow certain users to supply their own AWS credentials. If this option has been exercised then any user that supplies their own credentials will be able to process their own labs whenever they wish. Users supplying their own AWS credentials will not be able to run labs for other users.

III. Remaining Chapters

In the second chapter we discuss our research into the popular platforms, infrastructure, and software as a service providers available today. We also discuss code frameworks and libraries, virtual lab solutions, and other relevant technology.

In the third chapter we discuss the implementation process for CloudSuite. We cover the setup of the necessary web server and the associated software as well as discuss the data objects that were created for CloudSuite. Finally we review the use of Amazon Web Services to process CloudSuite requests.

The fourth chapter analyzes the results of the use cases established in chapter one. We compare our results with the expected outcome. In the fifth chapter we examine the output from CloudSuite and in the sixth chapter we addresses how well our expectations established in chapter one relate to our final outcome.

In the seventh chapter we discuss the future of CloudSuite. We will cover how the user interface might be improved, user experience enhanced, and new functionality that might be added to CloudSuite.

In the eighth chapter we draw our final conclusions and reflect on how CloudSuite might advance the field of computer science. We also discuss the potential of CloudSuite as an educational tool.

IV. Key Terms

Amazon Machine Instance (AMI) – These are snapshots of computer systems used by Amazon EC2. An AMI may be one of the many public offerings or the end user may customize an AMI to suit their own needs.

Asynchronous JavaScript and XML (AJAX) – A group of techniques and technologies using client-side resources to create interactive web applications.

Application Programming Interface (API) – A collection of functions, classes, and objects which allow software components to communicate.

Elastic Compute Cloud (EC2) – Amazon Elastic Compute Cloud is an infrastructure as a service offering from Amazon.com that allows the end user to launch virtual machine instances “on the fly”.

Cluster – A cluster is a collection of loosely coupled computing resources. Typically it is used to mean a collection of commercially available hardware assembled to work as a single cohesive unit.

Code Library – A collection of code for performing similar tasks.

Document Object Model (DOM) – a platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents [74].

Dumb Terminal – A dumb terminal is a computer system with little, or no, capability for independent computation. Dumb terminals function by connecting to a main frame system.

Framework – A software framework provides a generic platform that may be customized by the end user to perform specific tasks.

Grid Computing – Similar to clusters, Grid Computing is the term applied when geographically distant systems work together to solve a single problem. Often this involves breaking the problem into discrete segments for each node of the grid to process.

Hypervisor – Also known as a Virtual Machine Manager, a hypervisor is used to control virtual computing instances on a host system.

Infrastructure as a Service (IaaS)– A business model wherein an organization provides equipment to support another businesses operation, such as web servers, file hosting, and networking components. The resource provider is responsible for the care and maintenance of the hardware.

Instruction set architecture (ISA) – This refers to the native data types, processes, and I/O operations implemented by a particular processor.

Model View Controller (MVC) – A software design pattern that seeks to abstract the users interaction with information. The controller changes the model, which updates, the view, which is seen by the user.

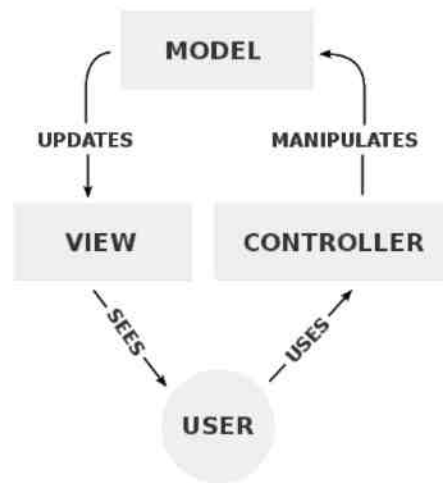


Figure 3. Model View Controller diagram [2]

Packet Switching - A networking method that seeks to group all data into similarly sized blocks.

Platform as a Service (PaaS) - A business model that allows the end user to deploy software applications to the public.

Representational State Transfer (REST) - is a style of software architecture for distributed systems such as the World Wide Web. REST has emerged as a predominant web API design model [71].

Scientific Computing - The science of constructing simulations and performing quantitative analysis using computer based resources.

Software Development Kit (SDK) - A set of tools used in the creation of software for a specified platform.

Software as a Service (SaaS) - A business model concerned with creating, hosting, and maintaining software for third part clients.

Thick Client - A computer system that provides self contained processing power and resources. Most modern workstations are, in essence, thick clients, in that they may operate independent of a central server. Also known as a fat, heavy, or rich client,

Thin Client - A hardware or software system that is dependent upon a remote system for data processing and/or data persistence. Also known as a lean or slim client.

Universal Resource Identifier (URI) - Characters used to identify a name or resource on the internet.

Virtual Private Servers (VPS) - Physical hardware that has been compartmentalized to offer multiple clients discrete web servers. Many web hosts offer a service wherein the end user has complete access to their own, and only their own, VPS instance.

Virtualization - Typically a software technique by which one piece of hardware imitates one, or more, different types of hardware.

Chapter 2:Field Overview

Creating a cloud based ad hoc laboratory system, such as CloudSuite, requires a variety of different technologies. The primary technology that is necessary for CloudSuite is hardware virtualization. Fortunately there are numerous examples available to us. We will perform an examination of hardware virtualization services in general and a specific exploration of the services offered by Amazon.com. In addition to virtualization we will also explore: existing virtual lab environments, templating systems, remote storage solutions, and finally, APIs and Frame Works.

I. What is Cloud Computing

The National Institute of Standards and Technology defines cloud computing as :

[...] a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [3].

Types of cloud resources are broken into three models : Infrastructure as a Service, Platform as a Service and, Software as a Service.

Infrastructure as a Service, or IaaS, is one of the more common cloud offerings. Although many providers offer a variety of services, this is most often thought of as website hosting. Two examples of typical IaaS providers are RackSpace.com and DreamHost.com. A standard IaaS service allows the user to install and maintain a suite of software on a remote server. These servers can either be discrete hardware or, more commonly, a virtual or shared environment. This allows a consumer to establish a web presence without purchasing, setting up, or maintaining physical hardware. Additionally many IaaS providers have Platform as a Service functionality available as well.

Platform as a Service, or PaaS, is similar to IaaS except the end user is not responsible for installing or maintaining software. PaaS allows a user to deploy their own software to an external server. Two examples of PaaS are Heroku and Google App Engine. Platform as a Service allows a user to create and deploy software without maintaining their own servers. The end user is then able to access the functionality of the software as long as there is an active internet connection. With the prevalence of smart phones, tablets, and other mobile computing platforms, PaaS is quickly becoming a popular choice for new app developers.

The third application of cloud computing is the “Software as a Service”, or SaaS model. Unlike traditional enterprise software SaaS does not install on the consumer’s local hardware. Instead the consumer accesses the software through a so called ‘thin client’ or web browser. The maintenance of the software is left up to the SaaS provider. Whereas typical enterprise software requires the end user to periodically buy new licenses, a SaaS model relies upon user subscriptions. Typical advantages of SaaS over traditional enterprise software include: customization such that the end user is able to change the look and feel of the software to create a seamless interface for their clients;

rapid feature delivery, since the software is centrally located it is easier for the developer to implement and release new features; and pricing, as SaaS typically has a lower initial price point compared to similar enterprise software offerings. Examples of Software as a Service include Blackboard [4], OmniUpdate [5], and Cloud9 Analytics [6].

II. History of Cloud Computing

Cloud computing can be described as the culmination of four technologies: distributed systems, some form of virtualization, thin clients, and ubiquitous high speed internet access. Even though the majority of these technologies were developed in the 1960s, it was not until high speed connectivity became common place that the ‘cloud’ as we know it was possible.

Traditional distributed systems include cluster, grid, and distributed computing. All of these refer to a collection of discrete hardware nodes combined to perform a common task. The hardware may be combined in different ways or locations; however, the end results are always similar: processing more instructions per cycle then would otherwise be possible. One aspect that makes cloud computing different from traditional distributed systems is that, in cloud computing, distributed resources are frequently made available to multiple end users through virtualized parallel computing.

Virtualization has been pursued by computer scientists since the early 1960s [7]. Virtualization is used to “[subdivide the] ample resources of a modern computer” [12]. Primary examples of network virtualization is the Apache HTTP server. The Apache web server is used to make one resource, a distinct hardware server, appear to be multiple servers. This process is similar to other types of virtualization provided by software such as VMWare® [13], Xen® [14], and VirtualBox [15]. Virtualization of this type allows a single resource, either discrete or composite, to be subdivided into appropriately sized slices. This allows the end user to access a much more powerful system when they need it and only pay for the services they require.

Accessing a system with more computational power than the local machine was a common paradigm in early computing. Such a system is referred to as a “dumb terminal”. These so called “dumb terminals” formed the backbone of early data entry and processing. Modern systems are powerful and inexpensive enough that this model is no longer widely used. However, when a modern system accesses distributed resources to perform complex or process intensive calculations it’s actions are nearly identical to the “dumb terminal” model. The systems that are available to end users are powerful enough that it would be more accurate to call them ‘smart terminals’ or ‘thick clients’. Though the terminals accessing the resources may be significantly more powerful than their ancestors, the basic usage is the same: a terminal submits data to a remote location to be processed and receives the results from the computation. This kind of remote architecture would not be possible with today’s large data sets if it were not for the advent of high-speed internet access.

The modern internet started life as the first packet switched network known as ‘ARPANET’. Despite commonly held beliefs, the original goal of ARPANET was not to build a communications system that was resistant to nuclear attack, but rather to make

computational resources more readily available. The internet has now become a nearly ubiquitous tool that is known the world over, capable of transmitting more data per second than the original systems that created it could store. This universal access allows universities, laboratories, and companies to share their computational resources worldwide.

Cloud Computing, as we know it today, is the culmination of these resources into a widely available system, that allows the end user access to levels of computing that would otherwise be impossible. Accessing these resources allows the end user to perform tasks and provides services that are a driving force in technology today. This thesis leverages these resources to provide a platform for the development of academic programs that demonstrate the scientific computing techniques to better understand and develop future technology.

III. Virtualization

Though this thesis does not work directly with virtualization, CloudSuite relies heavily on virtualized hardware. Because of this, we carefully reviewed existing virtual hardware services in order to both understand how the process works and to find the service that is best suited to CloudSuite.

Virtualization is, in many ways, the main focus of computer science. Each step away from the physical wires and transistors that make up a computer is an abstraction from what is really happening. To quote J. Stanley Warford “Once designed, hardware is difficult and expensive to change” [16]. Software, on the other hand, can be changed with relative ease. This is what makes hardware virtualization such a desirable goal.

Virtualization is one of the oldest areas of interest in computer science. One of the most successful examples of virtualization is the Java Virtual Machine (JVM) [17]. Using the JVM allows the Java programming language to produce identical results regardless of the instruction set architecture (ISA), that is used by the hardware platform. This is because, for all intents and purposes, the compiled Java code is running on the same machine – albeit in a virtual setting. As long as a system has a JVM installed that can translate Java byte code into the appropriate ISA, then the results will be identical.

The Java virtual machine allows one piece of virtual hardware to be present on multiple discrete systems. Software like VMWare allows multiple virtual hardware systems to be present on a single piece of physical hardware. Creating multiple virtualized hardware images on a single piece of physical hardware is one of the cornerstones of cloud computing. These concurrent virtual images are made possible by a hypervisor.

IV. Hypervisors

A hypervisor translates the instruction set architecture used by the guest system to the ISA used by the host system. Hypervisors are also commonly known as virtual machine managers (VMMs) because they are used to manage the resources allocated to the virtual machine guest instances being run on the physical host. Hypervisors are also

frequently used in load balancing and, as Bressoud discusses, can even provide fault tolerance [18]. Hypervisors are typically classified into one of two types[19]: Type 1, also known as native or bare metal, and Type 2, hosted hypervisors [20][21].

Type 1 hypervisors run directly on the hardware. They are not contained in an operating system. This type of hypervisor is a direct descendant of the first experiments with virtualization. As seen in Figure 4 a Type 1 hypervisor is the lowest level of software on the system. The guest operating systems run on top of the hypervisor and appear to be completely independent machines. Modern examples of this are Oracle VM Server for SPARC, the Citrix XenServer, KVM, VMware ESX/ESXi, and Microsoft Hyper-V hypervisor [20]. This type of hypervisor is what makes IaaS possible.

Type 2 hypervisors, again seen in Figure 4, are hosted by an underlying operating system. This allows for multiple operating systems to be tested and used within the confines of a parent operating system. This type of virtualization is especially helpful when testing cross platform compatibility as well as heterogeneous network connectivity. Examples of Type 2 hypervisors include: BHyVe, VMware Workstation and VirtualBox [20].

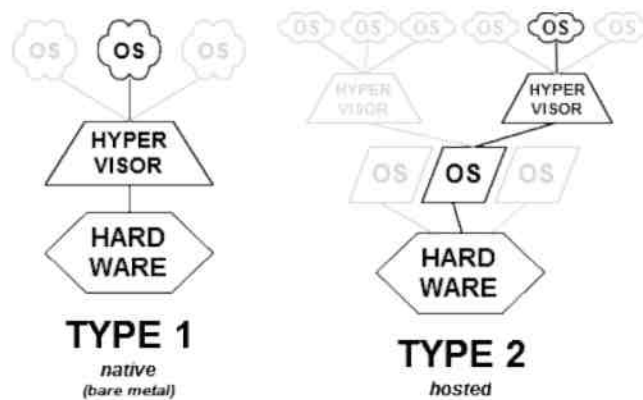


Figure 4. Diagram illustrating the two most common types of hypervisor [20].

Hypervisors originated with some of the earliest mainframe systems. Most notable were SIMMON [22] and IBM's CP-40 [11][23]. These two systems are cited as the first systems providing full virtualization. That is to say these systems were the first to make the entirety of the underlying system available to the guest system. An additional area of interest is the concept of paravirtualization, which will be discussed in the next section.

IV. Paravirtualization

Paravirtualization refers to the ability of a hosted operating system to make calls directly to the underlying hypervisor. This feature has been present as far back as 1972 [24], in IBM's VM systems. Calls to the underlying hypervisor were not called "paravirtualization" rather, these calls were referred to as "DIAGNOSE code". Modern instances of paravirtualization require the guest OS to be specifically tailored for use in a virtualized environment. These custom OS are designed to run in a different privilege level and make calls directly to the hypervisor, rather than in the native OS, when such calls would be difficult or costly in a virtualized environment. It is possible to do this by

changing the privilege level at which the virtualized OS runs. According to Smith and Nair [25], this leads to a virtual system that performs at 90+% efficiency compared to a native Linux install. One of the most well known examples of paravirtualization is the Xen hypervisor [21]. Xen is one of the leading virtualization solutions currently in use today, and is the primary hypervisor used by Amazon.com.

V. Amazon.com Elastic Compute Cloud

On August 25, 2006, Jeff Bezos announced the limited beta of Amazon Elastic Compute Cloud [26], referred to as Amazon EC2. Amazon EC2 is a service that allows users to, essentially, rent computational resources on an as needed basis [27]. This type of service is made possible by using the Xen hypervisor as discussed in the previous section, Xen allows Amazon EC2 to create Virtual Private Servers as demand requires. These virtual servers make use of operating systems that have been optimized for paravirtualization. Once the end user has customized the virtual machine to their liking they are then able to save it as an Amazon Machine Image, or AMI.

An AMI is a system image that has been created to be used with EC2. Users may customize these AMIs with their own software packages allowing an AMI to be tailored for any task. Once an AMI has been configured it can be saved and re-deployed at will. This allows the end user the ability to quickly and easily set up a machine that is configured exactly as needed, when it is needed. Using an AMI, it is possible to configure a powerful computing machine, use it to run a simulation, and then deactivate it. If the machine is needed again, it is a simple matter to restart the machine and re-run the simulation. This allows the end user to only pay for the computational resources they need when they need them. It also allows the user to distribute their AMI to allow others to reproduce or expand upon their experiments.

The Amazon Simple Storage Service (S3) [28] is used extensively in the code written for CloudSuite. S3 allows a user to upload and download files using HTTP requests. Amazon Simple Storage Service, like the rest of Amazon Web Services, is not free; however at the time of writing, Amazon offers a free tier of service available for the one year[28].

Amazon, and various 3rd parties, provide SDKs that allow developers to interact with Amazon Web Services. The framework for CloudSuite is written using SDKs for PHP, Python (Boto) [29], and Unix/Linux command line [30]. However SDKs exists for multiple platforms and languages [31]. The provided SDKs greatly ease interaction with the cloud services provided by Amazon.

The Platform as a Service model (PaaS) has become more prevalent due in part to Amazon EC2 as well as other IaaS providers. The PaaS model allows an end user to “rent” time on a fully realized high power system, regardless of their own hardware. Amazon EC2 allows third parties access to the computational resources of Amazon.com’s data centers. Services such as Heroku make use of this space by providing a platform to execute end user code. Many PaaS providers make use of Amazon EC2, or similar, services.

Amazon is far from the only available option for cloud services. Google recently announced ‘Google Compute Engine’ [32], a service similar to Amazon EC2, in addition to the long standing Google App Engine. In the next section we will take a closer look at cloud services offered by providers other than Amazon.com.

VI. Other Cloud Based Services

The PaaS business model is, at first glance, very similar to the previously mentioned Infrastructure as a Service. Where these services differ is that PaaS does not require the end user to setup real or virtual hardware. One of the better known PaaS providers that we will examine is Heroku [33].

In addition to Platform as a Service offerings there are also many of the more traditional persistent web hosting services, like Rackspace [34] or Dreamhost [35]. These services are typically limited to a pre-configured server to which the end user will upload web content. Many of these traditional web hosts also offer Virtual Private Servers that allow the end user complete control over the software that is used in their web hosting solution. Often, cases combining a traditional web host with a Platform as a Service provider allows a developer to present a fully realized product with very little investment in terms of hardware. To illustrate this point we will now examine the PaaS provider Heroku.

a. Heroku

Heroku [36] deserves special consideration because it is one of the best examples of a platform for executing arbitrary programs in the cloud. Heroku allows the end user to develop an application and leave the running of that application up to Heroku. The developer pushes code to Heroku using the Git [37] version control system. Once the code is present, Heroku relies upon HTTP requests to interface with the client application.

Heroku can be broken down into three main components: dynos, railgun servers, and slugs. A dyno is a Unix based “container” that is used to process requests [38]. Each application can have a number of dynos assigned to handle requests. These requests are processed by the railgun servers. Each railgun server is an Amazon EC2 instance configured specifically by Heroku to process these requests. Once the developer pushes their code to Heroku, it is compiled into a ‘slug’ [39]. A slug is a pre-compiled package containing the developer’s code. Each package is designed to be as fast as possible. These so called slugs are then called by the railgun server when the dynos receive a request. The entire process is encapsulated to ensure each slug that is run is processed in such a way as to prevent users from contaminating the code of others.

Heroku was put to great use in the Coursera [40] course ‘Software Engineering for SaaS’. In this course the students were asked to create software that was automatically deployed and made available on the internet with little to no wait time. Similar services are currently offered by Google App Engine, Engine Yard, and Microsoft Azure Services Platform.

b. Google App Engine

Google App Engine (GAE) [41] is another service similar to Heroku; however, Google App Engine places a number of restrictions on what code developers may use. Unlike an Infrastructure as a Service provider, GAE places limits on what code may be executed as well as only allowing the proprietary Google Query Language as a data store.

c. Eucalyptus and AppScale

Extensive research on cloud technology has been performed at University of California, Santa Barbara (UCSB). Two of the most well known products of that research are Eucalyptus [78] and AppScale [43].

Eucalyptus is an open source emulator for Amazon EC2 that can be run on local clusters [42]. Since the initial release of Eucalyptus, it has grown to become a commercial success, and is included in the Ubuntu operating system [69].

AppScale is an open source implementation of several cloud APIs, including Google App Engine. AppScale can be deployed over Amazon EC2, Eucalyptus, or an Ubuntu image. AppScale uses virtualization to provide a uniform experience across multiple cloud platforms.

One factor that separates cloud services are the languages that may be used to develop applications for the various platforms. Some services, like GAE and Heroku, limit the languages that may be used for development by only allowing certain types of code to run on their platform. By granting full access to an operating system, EC2, and other IaaS services, allow much greater freedom when developing software.

VII. Languages

The development of CloudSuite required the use of multiple computer languages. The initial development involved writing the model/controller layer in PHP, which stands for PHP : Hypertext Preprocessor. PHP 'is an HTML-embedded scripting language' [44] and is well suited for the processing needs of CloudSuite because it provides numerous functions and libraries for XML manipulation. Additionally there is a well documented API for accessing Amazon Compute Cloud web services.

The view/controller layer for CloudSuite is written primarily in HTML, CSS, and JavaScript. JavaScript was an obvious choice for the view/control layer because of the numerous libraries available to assist in the creation of web applications. Using modern code frameworks and design patterns allows the front end to present an accessible and modern look and feel. When possible HTML5 and CSS3 were used to increase the flexibility of the front end display.

The processing of CloudSuite labs makes use of Python and C. Python 2.7 is used for accessing the XML files stored in Amazon S3 buckets and for calling compiled code stored on the AML. Python was chosen because of the libraries available for accessing Amazon Cloud resources and the ease of accessing system level commands. C is used for a custom daemon that processes CloudSuite labs. Several CloudSuite modules also call

software that has been written in C. C was the obvious choice for the speed and efficiency it offers when accessing system level commands and processes.

Finally XML is used to transfer and store data about CloudSuite. The ability to define and validate custom data structures makes XML a good fit for the creation of a data persistence layer. This coupled with the PHP support libraries, makes using XML an excellent choice.

VIII.Code Frameworks, Libraries and APIs

Frameworks, libraries, and APIs allow software developers to use pre-written code for their own development work. This allows developers to quickly and easily create new software offerings with little initial setup. This section will examine some of the libraries that are available and how they relate to CloudSuite. Many of the code frameworks discussed here served as an inspiration for the design of CloudSuite and would be useful in creating a successful product. For this thesis we evaluate and discuss the most relevant of those products.

a. PHP Slim

PHP Slim [45] is a small footprint framework which provides RESTful, [70][71], utilities in a simple to use package. PHP Slim makes use of anonymous functions to allow the user to specify actions when a request is made to a specific URI [72]. Additionally PHP slim filters these requests based upon the HTTP request method used to access the URI. This framework acts as a communication layer between the Model, View, and Control layers of a web application.

b. Boto

Boto [29] is a Python API for accessing Amazon cloud resources. It allows the user to, among other things, create and access S3 buckets and instantiate and control EC2 instances. We make use of Boto in our AMI used for processing CloudSuite labs.

c. ElementTree

ElementTree [46] is a Python library for working with XML data. ElementTree allows the user to load XML data and access it similarly to a Python list or dictionary. We make use of ElementTree when parsing CloudSuite labs.

d. jQuery

jQuery is a popular javascript library that was initially created by John Resig in 2006 as a way to bind CSS selectors to JavaScript functions [47]. According to builtwith.com [73], jQuery has become one of the most used JavaScript libraries in use today as well as one of the most forked repositories on popular coding site github [48]. Additionally jQuery also boasts a number of large technology companies among it's users including: Google, IBM, and Amazon [49].

jQuery is free, open source software, dual-licensed under the MIT License or the GNU General Public License, Version 2. jQuery's syntax is designed to make it easier to navigate a document, select DOM elements, create animations, handle events, and develop Ajax applications [50].

jQuery simplifies many of the common tasks used in creating a modern website. It provides tools and shortcuts to address DOM elements and send AJAX requests. jQuery allows the developer to create simple calls to DOM objects using CSS identifiers. These calls are made using the jQuery mnemonic '\$'. A good illustration of how this is helpful is the simple act of changing the contents of an HTML DIV tag. Prior to jQuery the code necessary to change the contents of a known div would be:

```
document.getElementById('NodeID').innerHTML = 'new Value';
```

jQuery allows the same operation to be performed with this simple command:

```
$("#NodeID").html('new Value');
```

In addition to simplifying code, jQuery also allows the user to access DOM elements by using CSS selectors which makes the creation of dynamic web sites much simpler. In addition to providing tools to access rendered DOM objects, jQuery also simplifies accessing DOM elements that are created dynamically through AJAX calls. By providing the tools necessary to simplify querying DOM elements and making AJAX calls jQuery has become a standard for web development.

e. Backbone.js

Backbone.js is another popular JavaScript library that allows a web based application to create a custom models to represent data. These models can then be grouped into collections and operated on either singly or as a collective group. Additionally, Backbone can be used to trigger events when data is manipulated.

By connecting web events to the direct manipulation of data, Backbone creates a mechanism to keep data synchronized while minimizing the impact upon a database server, as well as decoupling data from the DOM.

Backbone allows data to be collected in a meaningful way while minimizing the amount of JavaScript that is necessary to track and manipulate that data. Separating the applications data from the HTML DOM object also allows for data persistence beyond what the DOM can offer. Backbone relies upon the Underscore.js library to manipulate and iterate over the collected data.

i. Underscore.js

The Underscore.js library offers several features that are not natively available in the JavaScript language. Underscore provides methods that allow aspects of functional programming to be used when dealing with web based data. The majority of the methods included in Underscore deal with creation and manipulation of lists. Underscore.js also offers a minimalist templating system (though there are many others available).

j. Templating Systems

Templating systems in JavaScript allow the developer to create an HTML based template for displaying JSON objects in a predictable way. This gives the developer full control over the display of dynamic content.

JavaScript templating systems are either standard or logic-less. Standard templates allow arbitrary code to be mixed in with the template. This can lead to large templates that can be difficult to follow. Standard templates allow the developer to control the data as it is presented to the template, resulting in greater flexibility within the templates. Standard templating libraries include : Underscore.js, Jade, EJS.

Logic-less templates do not allow arbitrary code to be present within the template itself. Logic-less templates forces any data manipulation to be performed prior to the data being presented to the template. These templates are designed with common use cases in mind and as such do not offer the same level of flexibility as a standard template. Logic-less templates are typically less complicated and easier to follow. Some popular logic-less templating systems include: Mustache.js, Handlebars.js, dust.js and Transparency.

k. Highcharts

Highcharts JS [75] is a charting library written in JavaScript. It allows the user to create interactive web based charts and graphs. The library offers the functionality and flexibility to generate charts dynamically from numerous data sources. We use the Highcharts library to create visual representations of data within CloudSuite. Using a JavaScript based language maximizes the computability of these visualizations.

IX. Online Education

With the growth of high speed internet access to distance learning has become increasingly more accessible. In addition to universities supplementing their courses with online offerings, many universities are also making classes available online for free. Coursera is a central location where those seeking to improve their education may participate in free online classes from sixteen universities including: Caltech, Stanford, and Princeton [40]. Additionally many universities have begun offering free classes online for no credit. One such offering from University of California, Berkley made use of Heroku to allow students to publish a web based application and see results in real time. The Massachusetts Institute of Technology also offers free online courses. It is important to note that these online courses typically do not confer any sort of degree or certificate, Coursera being notable in that some of it's courses do offer an electronic certificate of completion [51]. Courses offered online could make great use of a service like CloudSuite to provide their students with an interactive lab component.

X. Other virtual labs

The idea of virtual laboratories is far from new. Just as virtualization is one of the oldest areas in computer science, virtual labs have been pursued by educators for many years. In this section we examine some of the more prominent virtual lab offerings.

a. Khan Academy

Khan Academy is a not for profit online educational resource. Khan Academy offers videos explaining a variety of topics. In addition to videos Khan Academy also offers a web based set of tools for developing mathematical proficiency. In late 2012 Khan Academy introduced Computer Science curriculum that offers programming instruction and demonstration. The programming curriculum is based on the JavaScript language and is overseen by John Resig [52]. The program is designed to demonstrate the fundamentals of computer science to someone with little to no experience.

b. University of Hawaii Virtual Lab

In 2007 the Department of Educational Technology at the University of Hawaii at Mano [53] published a paper detailing an experiment where an online biology class made use of a CD-ROM based virtual wet lab. The study compared the use of a virtual lab to what the study referred to as a ‘face to face’ lab. The data analysis showed that, while students found the virtual labs to be useful, face-to-face lab time was more valuable to the overall learning experience than a purely computer-based lab. The study was primarily focused on the effectiveness of the virtual lab and did not give great detail on the technology [54].

c. Navy Virtual Lab

The Naval Postgraduate School (NPS) [55][56] has developed a distance learning solution that allows non-resident students to perform signal processing laboratory assignments. The Electrical and Computer Engineering department has developed an innovative mix of hardware and software to allow students to access lab equipment regardless of their location. The collection of signal generators and field programmable gate arrays allow students to perform experiments in real time.

XI. Grids and Clusters

No discussion of cloud computing is complete without mentioning grids and clusters. Grid and cluster computing share a problem domain and similar purpose with cloud computing; however, each differs in nuance. Grid computing typically consists of a loosely coupled collection of computers working together to achieve a common goal [57]. One of the most widely known examples of grid computing is the SETI@home project from the University of California, Berkeley [58].

The SETI@home, or the Search for Extra Terrestrial Intelligence at home, project was designed to use the resources of idle computers to process large amounts of signal data from radio telescopes. The project has since been expanded into the ‘Berkeley Open Infrastructure for Network Computing’, known as BOINC [59], and allows users to participate in multiple distributed computing projects.

Other examples of grid computing include the Worldwide LHC Computing Grid (WLCG) [60] used to process the data collected from the Cern Large Hadron Collider and the European Grid Infrastructure (EGI). EGI is, perhaps, the more interesting of the

two as the mission of EGI ‘is to allow researchers of all fields to make the most out of the latest computing technologies for the benefit of their research’ [61]. EGI maintains a list of grid based tools that registered user may use for their own experiments. EGI was of particular interest to this thesis as it makes virtual tools available to numerous users.

Computer clusters differ from grids in that, typically, clusters are often used for complex simulations and may be more tightly coupled than a grid system. Usually a cluster will be treated a single computational resource whereas grid systems frequently have multiple users focusing on many problems.

A well known example of a high powered computer cluster is the IBM Blue Gene system. As of June 2012 the BlueGene\Q system was listed as the highest performing computing in the world [62]. The BlueGene\Q system is composed of multiple processor nodes controlled by a central linux based system.

Chapter 3: Implementation

CloudSuite is composed of three distinct parts: a lightweight backend, a web based front end, and a set of tools based on Amazon Web Services. The backend uses PHP 5 and an Ubuntu based web server. The web site is written in HTML and JavaScript and makes use of several JavaScript libraries. The third part consists of Amazon S3 and EC2 components, Python scripts, and code written in C. This layer provides the data persistence and data processing for CloudSuite. We chose these tools based on analysis of available options, see chapter two, and our experience with the technologies involved.

The structure of CloudSuite is inspired by the model-view-controller (MVC) style architecture. The model layer is primarily composed of XML. The view layer is handled by the web server and the HTML front end. The controller is made up of the PHP backend, AWS tools, and Python code used for processing. Code examples are included in appendices A,E, and F.

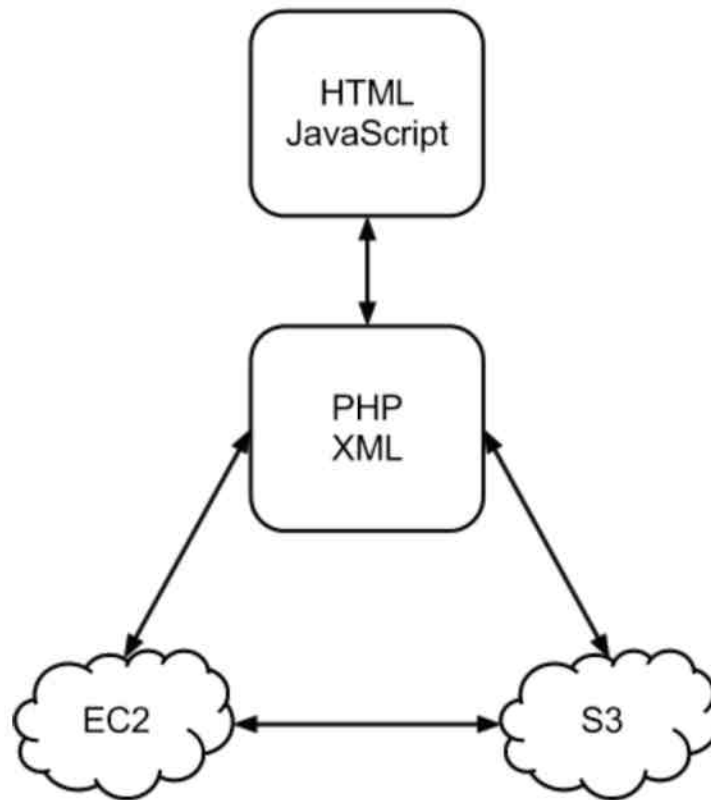


Figure 5. Control flow diagram of CloudSuite.

I. Web Server

Any cloud based application requires access to the internet. This requirement, along with the desire for CloudSuite to be platform agnostic, makes using a website for front end access the obvious choice. The first step toward creating cloud suite was registering the top level domain ‘cloudsuite.info’ using rackspace.com.

RackSpace provides several server options through their automatic provisioning system. We chose Ubuntu 10.04 LTS as it provides many features to assist in the creation and maintenance of a persistent web site.

Using the apt-get [66] packet manager we installed Apache 2.2.14, PHP 5.3.2 and the necessary modules to support serving PHP content through Apache. For a complete listing of all the modules installed please see Appendix B. Apache is configured using virtual hosts to provide DNS and routing information. We configured the virtual hosts file and used the built in ‘a2ensite’ script to establish the server. The provided ‘a2ensite’ script populates the ‘enabled sites’ folder with the symbolic links needed to properly serve content at the desired address [63]. The full text of the virtual hosts file is located in appendix C. The server was tested by creating a simple php file which contained the function ‘phpinfo()’ [64]. This was chosen because it demonstrates both the ability of Apache to serve content, as well as the successful installation of PHP. The phpinfo() file was removed after testing in accordance with best practices.

II. PHP Framework

PHP was chosen as the primary language for CloudSuite because it provides a convenient connection between the model layer and the view layer as well as between the view layer and the control layer. Additionally there are a number of useful PHP libraries for accessing and manipulating XML data structures.

CloudSuite is designed to be accessed in a RESTful fashion. This allows CloudSuite to be easily extensible as well as allowing end users to develop their own interface to CloudSuite if they so desire. CloudSuite provides API calls for each class that makes up the control layer of CloudSuite. In this section we discuss the class structure and data flow between CloudSuite classes. An API reference is included in appendix D.

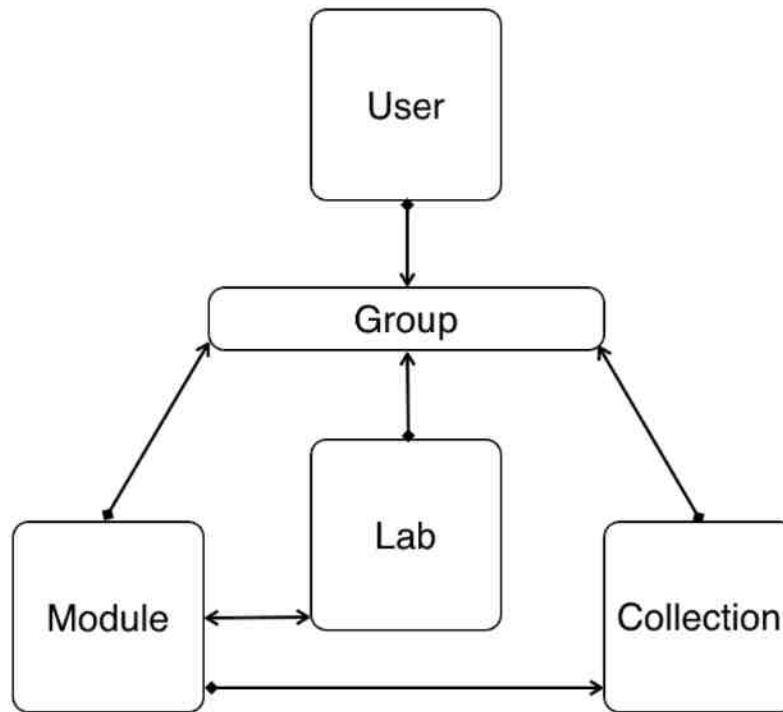


Figure 6. Control and data flow in CloudSuite.

a. User

The purpose of the user class is to create, load, and modify CloudSuite users. Additionally the user class contains helper functions for processing login information and user settings. The user class is also responsible for storing and processing the users personal Amazon Web Services settings, if applicable.

The user class is also used when determining access rights to resources within CloudSuite. User access is modeled after typical UNIX/POSIX access i.e., access permissions are determined at the owner, group, or global level. The data structures for CloudSuite currently have the fields necessary to accommodate access permissions however much of the implementation and integration of the user class has been left for future work.

b. Group

Following Unix style file permissions, each object in CloudSuite belongs to a group. Groups are used to control user access to objects within CloudSuite. Group affiliation will be evaluated when users first access the system.

Because groups function with respect to the user class, full implementation of groups has been left for future work. Despite not being fully actualized much of the work for groups is complete. The following describes the intended functionality of groups based on existing work.

When a group object is instantiated all of the groups that are applicable to the current user are loaded into memory. This allows access permissions to be verified quickly and

easily. The integrity of the loaded groups is verified by storing the last modified date from the file along with the group affiliations. When a group affiliation must be verified the last modified date from the inode of the group file on disk is accessed to ensure that no changes have been made. If the last modified date in memory and the date from the file are not the same, the group affiliations are refreshed from disk before the comparison is made.

c. Module

A CloudSuite module is a discrete element that represents either a data source or a function that is capable of producing and/or modifying data. Each processing module will have zero or more configuration options, as seen in Figure 7. These configuration options are stored in a lab object. Figure 8 illustrates the XML representation of a configured module within a lab. This allows the users choices to be stored for later retrieval without changing the module itself.

Figure 7. Module used for encrypting and decrypting files

Modules are defined by an XML schema, a complete listing of which are included in appendix E. This schema ensures that all CloudSuite modules are compatible. Data modules represent a data set that is contained in an Amazon S3 bucket. Processing modules make use of Python scripts that help create data modules and call the code necessary to perform the requisite action. Currently, any process that can be called from a command line can be called with CloudSuite. This includes custom programs, built in command line processes, and, through cURL [65], data from websites.

When operating on data, integrity is preserved by making a temporary copy of each data module selected. This prevents race conditions and allows CloudSuite to exactly reproduce results when applicable. The copies are discarded upon successful completion of the lab. It should be noted that the functionality of each module, and thus the lab that makes use of the module, is determined by the creator of the module. To this end the overall functionality of modules may, in time, differ from the initial implementation.


```

<?xml version="1.0"?>
<lab id="28050" labName="Neuro_test">
  <owner>Drew</owner>
  <description>run three neural networks</description>
  <permissions>
    <owner>7</owner>
    <group>4</group>
    <everyone>4</everyone>
  </permissions>
  <lastRunDate>0</lastRunDate>
  <lastRunUser>0</lastRunUser>
  <module id="28055" moduleName="neuro">
    <seqNumber>1</seqNumber>
    <method>neuro.py</method>
    <xmlrpcString>--Layers h3 </xmlrpcString>
    <filename>8.neuro.xml</filename>
    <description>Open, train, and use a neural network</description>
  </module>
</lab>

```

Figure 8. XML representation of a lab showing a configured module

A CloudSuite Module does not perform the operations itself, rather it provides an interface to execute command line programs and processes. This illustrates the power of CloudSuite as any program or operation that is callable by a UNIX operating system has the potential to be used in CloudSuite. Figure 9 illustrates a partial example of a CloudSuite module.

When a lab is executed the 'method' element and the 'xmlrpcString' elements are parsed from the lab file and are used to call the appropriate executable on the AMI. Currently, this makes use of the 'subprocess' Python library to execute the command using the system shell. There is a slight security concern in allowing access to the system shell, however, since administrator access to the AMI is necessary to upload modules, there is very little danger of malicious code.

Modules reside in one of two locations, either the persistent web server or an S3 bucket. The processing modules, which are modules used to create or modify data, by necessity reside upon the AMI. Modules that represent data are stored in an S3 bucket. This allows the data created by the modules to be accessed even when the AMI is not currently running.

```
<?xml version="1.0" encoding="UTF-8"?>

<!--
Document : module1.xml
Created on : May 8, 2012, 6:59 PM
Author : drew
Description: Module for calling the RSA utility
-->

<module id="21" name="rsa">
  <moduleType>method</moduleType>
  <description>Encrypt or decrypt a file using RSA.</description>
  <systemRequirement>
    <product>gcc</product>
    <version>2.4</version>
  </systemRequirement>
  <fieldset>
    <legend>Encrypt or Decrypt</legend>
    <element id="300">
      <type>radio</type>
      <name>--EnDe</name>
      <value>encrypt</value>
      <description>Encrypt file using RSA with provided key.</description>
      <input></input>
      <output>TextFile</output>
      <required>0</required>
      <default>0</default>
      <selected>true</selected>
    </element>
    .
    .
    .
  </fieldset>
  <permissions clearance="10">
    <owner>7</owner>
    <everyone>4</everyone>
  </permissions>
</module>
```

Figure 9. Partial example of an XML representation of the RSA module.

d. Lab

Labs represent a series of operations to be performed in sequence. Labs are constructed by selecting and configuring CloudSuite modules from available collections. When users first log into the system they are presented with a list of any existing labs and the option to create new labs. When a lab is loaded, the XML data associated with the lab is read from a persistent server, verified against a schema (pictured in Figure 10), and parsed into PHP objects. These objects are then used to restore the configuration choices made by the user who created the lab.

Administrators may distribute labs for others to use. In future versions of CloudSuite it will be possible to allow users other than administrators or professors to distribute labs. This option will be configurable and is discussed in chapter seven.

When a user chooses to run a lab, the associated configuration data is uploaded to an Amazon S3 bucket that serves as a processing queue. If the CloudSuite EC2 instance is ready, then the CloudSuite data processor reads each lab from the S3 bucket and processes them in order. Currently processing a lab is dependent upon an administrator making an EC2 instance available to execute the queued labs. The details of the EC2 instance are discussed in section V of this chapter.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!-- define simple elements -->
  <xsd:element name="owner" type="xsd:string"/>
  <xsd:element name="group" type="xsd:string"/>
  <xsd:element name="everyone" type="xsd:string"/>
  <xsd:element name="seqNumber" type="xsd:string"/>
  <xsd:element name="method" type="xsd:string"/>
  <xsd:element name="xmlrpcString" type="xsd:string"/>
  <xsd:element name="lastRunDate" type="xsd:string"/>
  <xsd:element name="lastRunUser" type="xsd:string"/>
  <xsd:element name="filename" type="xsd:string"/>
  <xsd:element name="type" type="xsd:string"/>
  <xsd:element name="name" type="xsd:string" />
  <xsd:element name="value" type="xsd:string" />
  <xsd:element name="dataType" type="xsd:string" />
  <xsd:element name="default" type="xsd:string" default="0" />
  <xsd:element name="location" type="xsd:string"/>
  <xsd:element name="description" type="xsd:string"/>
  <xsd:element name="required" type="xsd:boolean" />
  <xsd:element name="legend" type="xsd:string" />

  <!-- define attributes-->
  <xsd:attribute name="id" type="xsd:string" />
  <xsd:attribute name="moduleName" type="xsd:string"/>
  <xsd:attribute name="labName" type="xsd:string"/>
  <xsd:element name="selected" type="xsd:string" />

  <!-- define complex elements -->
  <xsd:element name="input">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="type" />
        <xsd:element ref="filename" />
        <xsd:element ref="location" />
      </xsd:sequence>
      <!--xsd:attribute ref="id" use="required"/-->
    </xsd:complexType>
  </xsd:element>
```

Figure 10. Partial example of the XML schema that defines a lab

e. Collection

Collections are used to organize modules, in much the same way folders organize files in a traditional operating system. A collection consists of a grouping of like modules that are currently available to the user. A module may be present in multiple collections; however, a collection will only be listed once per user. Currently collections do not feature prominently in the functionality of CloudSuite. Collections are envisioned as a tool for organizing and distributing modules. Further discussion of the future of collections may be found in chapter seven.

III.XML Models

The model layer of CloudSuite is in the form of XML files that define the data structures of CloudSuite. XML was chosen for the model layer of CloudSuite for several reasons. The primary reason was to eliminate the need for database management software (DBMS). Not requiring a DBMS makes CloudSuite very “light weight” and easy to install. Another reason for using XML is that providing XML schemas to the end user facilitates the creation of custom modules and collections. Verifying the structure of data using XML schemas is an easy process and ensures that user created data will always be compatible with the rest of CloudSuite. See figures 8, 9, and 10 for examples of the XML structures used in CloudSuite. Appendix E, XML Schemas, has a complete listing of the schemas used in CloudSuite.

The schemas that define CloudSuite are stored on a persistent server, eliminating the need to make additional network calls to validate the XML models. These schemas are used to verify the XML files that define the modules, labs, users, and other components of CloudSuite. Each time a CloudSuite element is accessed the structure of the element is compared against the stored schemas to verify the integrity of the file. This process is performed by using the DOMDocument PHP library. The functions used to perform this verification are located in the `utils.class.php` file, the full text of which is located in appendix A, section VII.

IV. View Layer

The the front end (or View layer of the MVC model) of CloudSuite is primarily composed of HTML and JavaScript. The JavaScript library jQuery, discussed earlier in this thesis, is relied upon heavily for front end processing. JavaScript is used to make CloudSuite portable and generally platform agnostic. By following JavaScript best practices, CloudSuite is accessible on the majority of modern platforms.

The front end of CloudSuite primarily consists of an ‘`index.php`’ file that makes calls to the other components of CloudSuite. This includes instantiating the necessary PHP classes and loading the PHP configuration files. This file also establishes a user session however, currently, user session data is not widely used in CloudSuite. The majority of the REST requests made by CloudSuite originate from this file.

REST requests are parsed using a custom handler as well as the SLIM PHP library discussed in chapter two. Initially CloudSuite processed each request by parsing the URL

variables in the request URI and calling the necessary functions. Through our research we discovered the PHP SLIM library and transitioned from our original approach to one based on the PHP SLIM library. This allows for greater flexibility in REST commands as well as a more consistent API.

Additionally future versions of CloudSuite will make use of more JavaScript libraries, such as Backbone.js and Bootstrap, both previously discussed in chapter two, to further increase compatibility and functionality with a variety of systems. For more information please see chapter seven.

V. Amazon Machine Instance

The ‘heavy lifting’ of CloudSuite is performed using an Amazon Machine Image running on the EC2 platform. This AMI has been configured by installing Python 2.7 with Elementtree [46], and Boto [29], both of which are described in chapter two. We have also included our own set of CloudSuite tools namely: a Python script for parsing labs, Python wrappers used for processing modules, and a daemon used to control the lab parsing process.

When the AMI is initialized a cron task starts the daemon that calls the Python file responsible for parsing labs. The daemon runs continually as long as the AMI is active. This structure allows us to update the files used in processing labs with little to no interruption in service. Errors with either process are written to a log file.

CloudSuite labs are processed through a ‘first-in-first-out’ (FIFO) queue. The Python lab parser first reads all the labs queued in the designated Amazon S3 bucket. The controller then loads the lab into memory and begins processing the modules contained therein.

Each module is associated with a Python wrapper. These wrappers are used to create data modules and to call the command line resources represented by the module. Each module is executed in turn and the results of the execution are stored in a log file which will be made available to the end user. If any data is created by executing the module those data are placed in the users data bucket. When the lab has been processed completely it is then removed from the active queue.

Currently the AMI must be activated and deactivated manually, either by logging in to CloudSuite or by logging in to the AWS management console. As discussed in chapter six: future work, in the future it will be possible to start and stop the AMI on a predetermined schedule. However, since this feature was not discussed in the initial description of CloudSuite it was determined to be outside the scope of this thesis and is not implemented in the present iteration of CloudSuite.

Chapter 4: Analysis of CloudSuite

The following chapter analyzes the efficacy of CloudSuite by working through the use cases presented in chapter one. This chapter analyzes the efficacy of our implementation and identifies areas of improvement.

I. Standard User

Each use case requires the user to have valid credentials for accessing CloudSuite. The login button in the upper right corner is used to access the username and password form fields.

a. Logging in



Figure 11. Login button

When clicked, the login button triggers a JavaScript function that displays the username and password fields. The user then supplies their authentication credentials to enter CloudSuite.

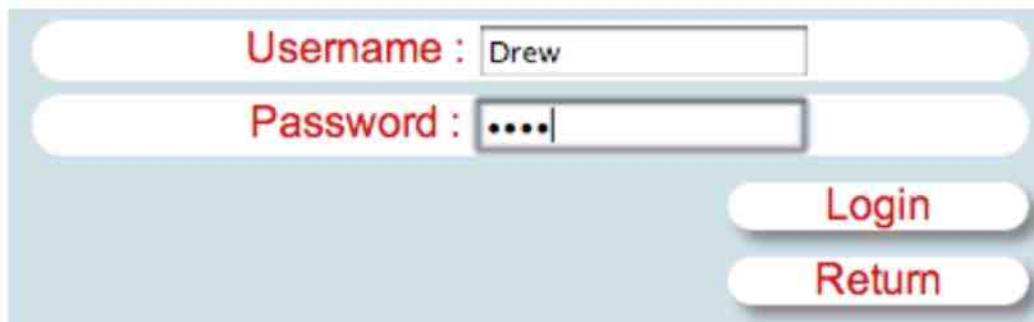


Figure 12. Username and password fields

If the user enters an incorrect username or password they are presented with an error message alerting them to the issue. Currently there are no provisions for processing repeated failed login attempts. This will be discussed in chapter seven: Future Work.



Figure 13. Failed login alert

Once the user has successfully authenticated the 'Login' button changes to show the username as shown in Figure 14. This serves to notify the user that they have successfully logged in as well as serving as a logout button.



Figure 14. The username is displayed after a successful login

Clicking the username button provides the user with the option to Log out of CloudSuite or return to their session, shown in Figure 15. Additional user options, such as changing their email address or password, are discussed in chapter seven.

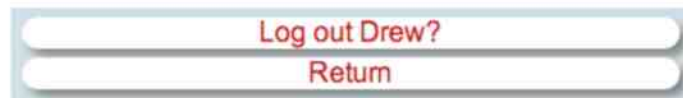


Figure 15. Logout options

Once the user has successfully authenticated they are then able to access the features of CloudSuite. As seen in Figure 16, all users will be able to save, load, queue, and create new labs. We will discuss each of these operations in depth in the following sections.



Figure 16. The task bar showing the current user options

b. Creating a Lab

When the user clicks the 'New Lab' button they are presented with the 'new lab dialog' shown in Figure 17. Here the user will specify the name and description of the lab they wish to create.

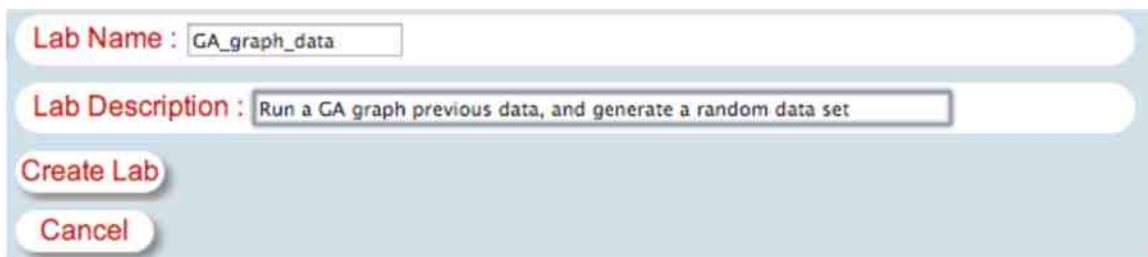


Figure 17. New lab dialog



Figure 18. Warning when trying to use whitespace in a lab name

Currently CloudSuite lab names may not contain white space characters. If a white space character is entered a warning is displayed and the lab name is highlighted. The lab description may contain whitespace.

Once a lab has been created the name and description are displayed above the main lab area as seen in Figure 19. Additionally the “Delete labname” button is also present which allows the user to delete the lab if they so desire.



Figure 19. Lab name and description displayed in the main lab area

c. Adding Modules to a Lab

After creating a lab, the next step is to add modules. Modules are listed by collection on the left side of the main screen, shown in Figure 20. A module is selected by clicking the button beneath the module description. When a user selects a module a number of configuration options are presented, an example of which may be seen in Figure 21. Once the user has selected their desired options the module is placed in the lab by clicking the “Add to Lab” button. The module is then listed in the main lab area, as shown in Figure 22. Currently the name and description of the module is listed as depicted in Figure 22. As discussed in chapter seven, in future versions of CloudSuite the configured options will be reflected in the module listing as well as the name and description.

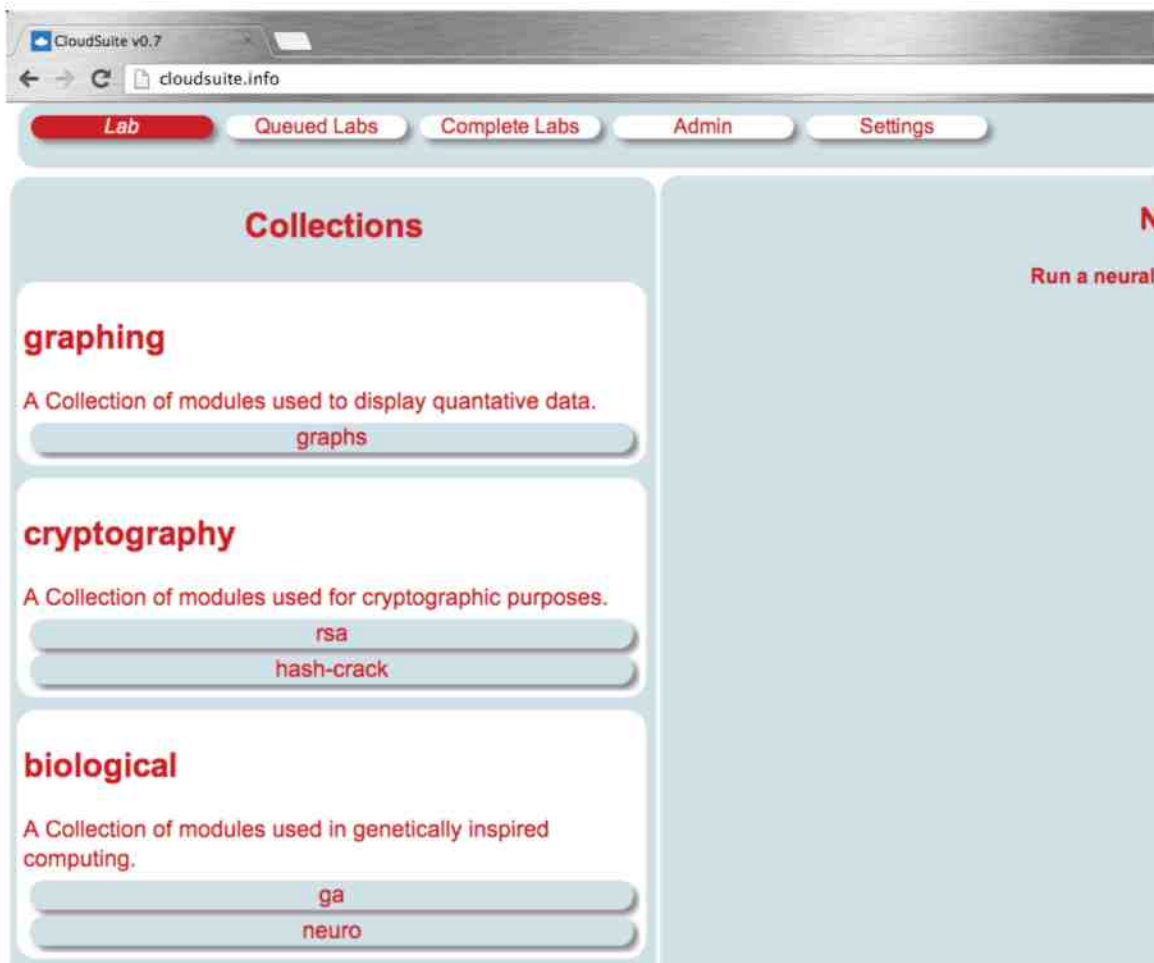


Figure 20. Modules displayed in CloudSuite

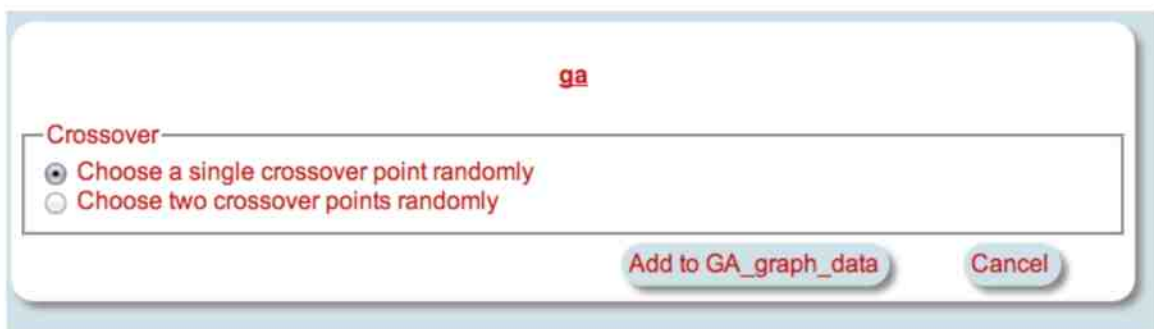


Figure 21. Configuration options for the module 'ga'

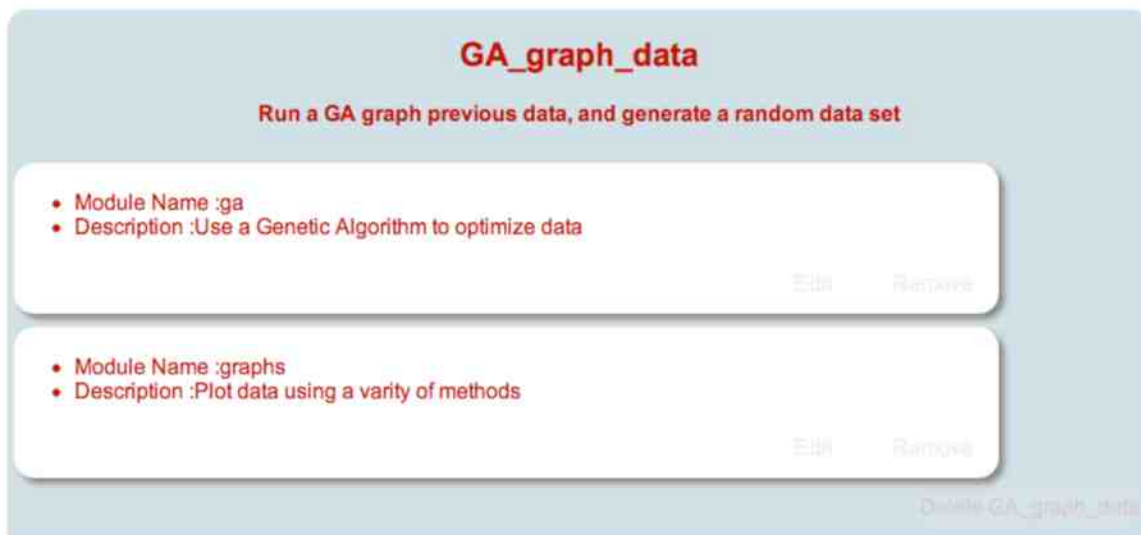


Figure 22. Modules displayed in a lab

d. Removing modules from a lab.

Modules may be removed by clicking the "Remove" button located in the lower right of the module listing within the lab. The user is prompted if they wish to remove the lab. Clicking the "Remove" button will remove the module and return to the lab, clicking "Cancel" will return to the lab without removing the module.

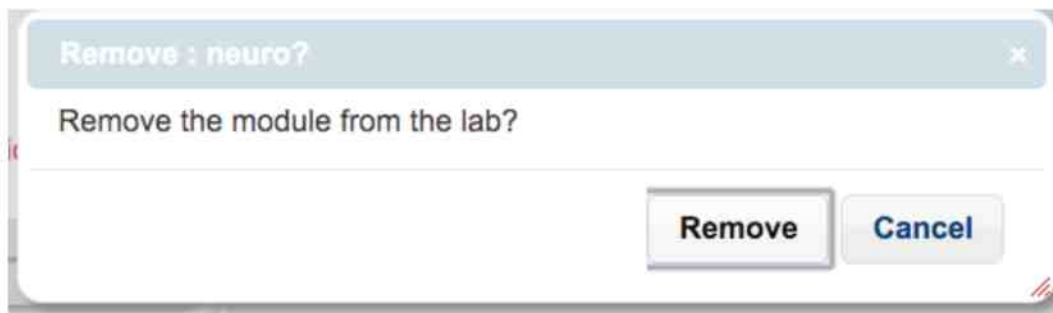


Figure 23. Confirmation upon removing a module

e. Editing Modules

Currently there is no mechanism for editing a module in place to change a module it must be removed from the lab and added with the new configuration options.

f. Saving a Lab

Saving a lab is accomplished by clicking the "Save Lab Button" illustrated in Figure 16. When the lab is successfully saved a dialog is presented to the user as presented in Figure 24.

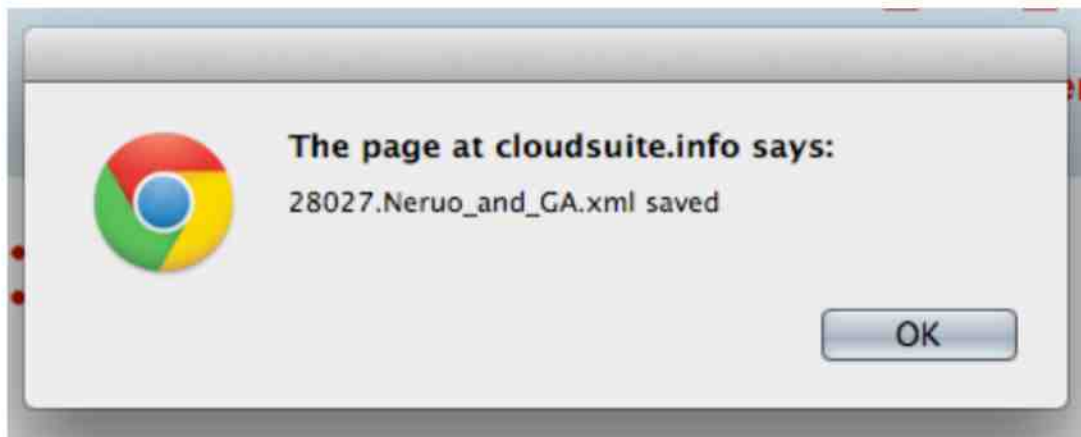


Figure 24. Lab saved success alert

g. Loading a Lab

When the user initially logs in to CloudSuite they are presented with a list of their available labs. Clicking on a lab will load it in the main lab area of CloudSuite. Additionally when working in CloudSuite, labs may be loaded at any time by clicking the "Load Lab" button shown in Figure 16. Clicking the "Load Lab" button presents a list of available labs. Clicking on the desired lab will load it in the main lab area. Any unsaved work done on the currently loaded lab will be lost. This is true even when the currently open lab is re-loaded. This allows the user an option to "undo" any unsaved changes.

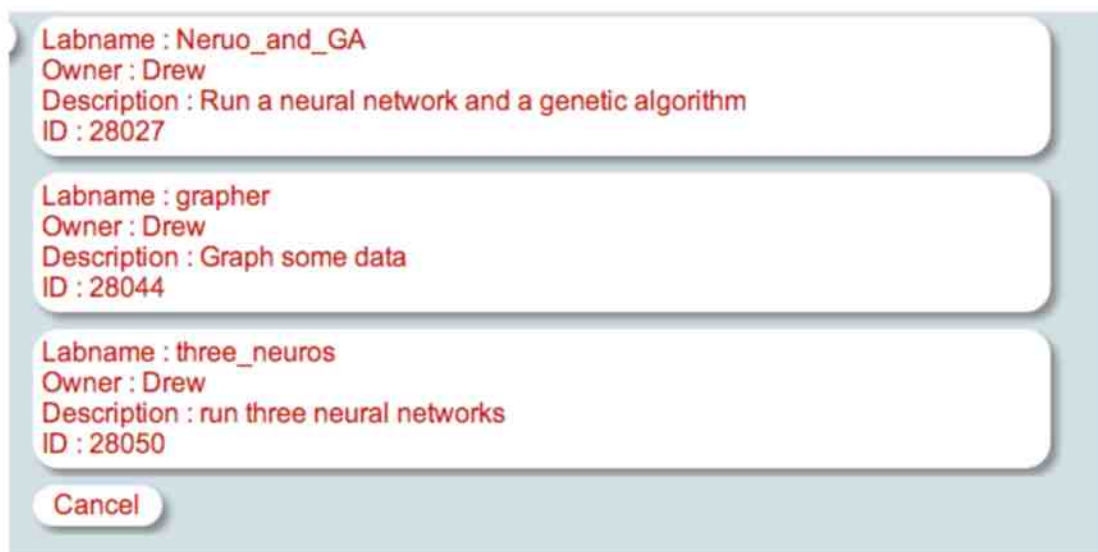


Figure 25. Display of labs available to load

h. Queueing Labs

Queueing a lab is accomplished similarly to saving a lab. By clicking the 'Queue Lab' button located on the status bar, pictured in Figure 16 the lab is placed in the queue for processing. Users may click on the 'Queued Labs' button to view the labs currently in the queue.

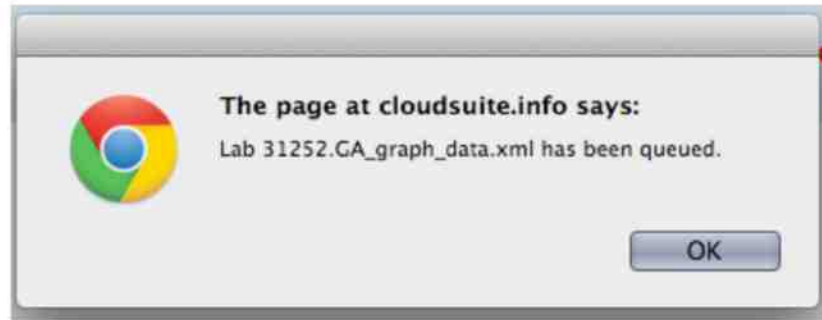


Figure 26. Queued lab alert

i. Deleting Labs

To delete a lab it must first be loaded as outlined in the previous section. Once the lab has been loaded the user then clicks on the 'delete lab' button located at the bottom of the list of modules. The user is presented with an alert indicating the success or failure of the delete action.

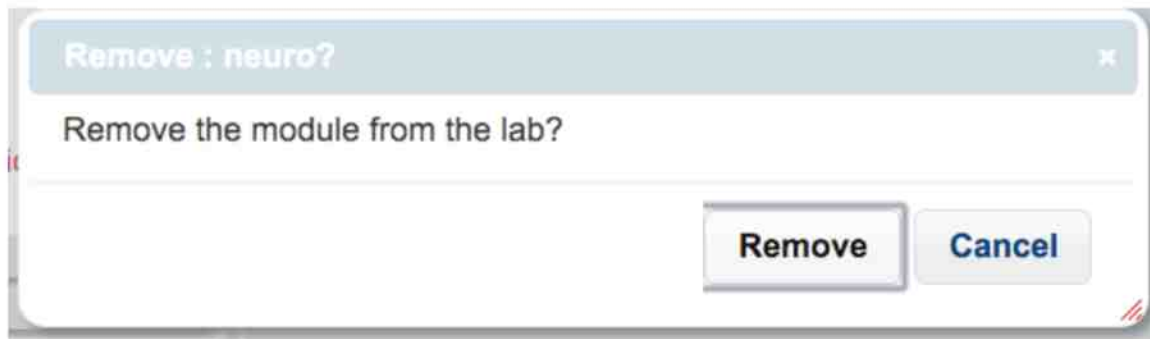


Figure 27. Confirm delete lab dialog

j. Results from a lab

Once a lab has been run the user is able to view the data from the individual modules as shown in Figure 28. When the user clicks on the data module then a data dialog screen is displayed, Figure 29, from this dialog the user is able to download or delete their data. To view the output from this lab, and a sample experiment, please see chapter five.

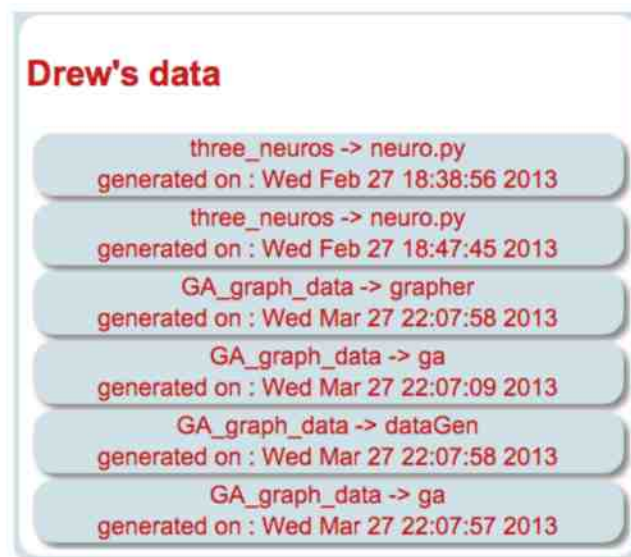


Figure 28. Listing of users data modules

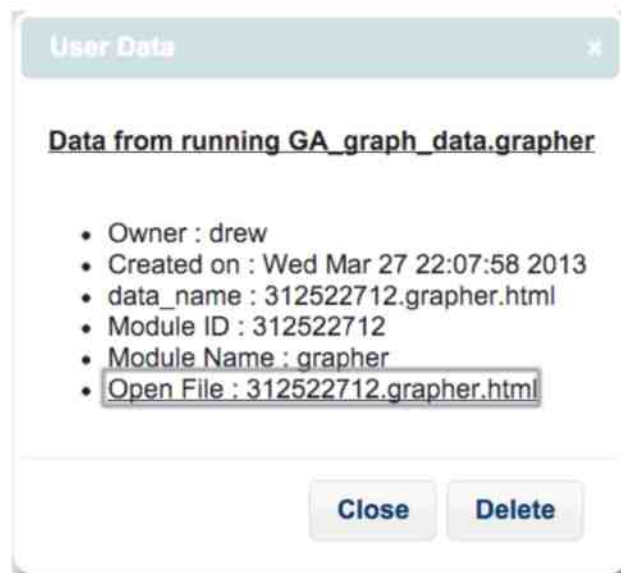


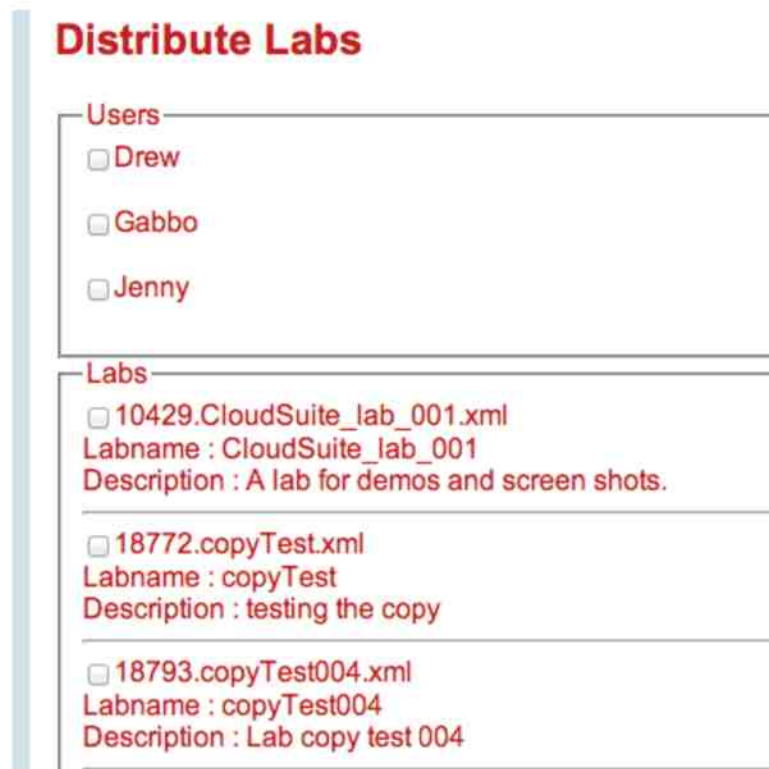
Figure 29. Data dialog

II. Superusers

Currently only CloudSuite administrators or professors may start and stop CloudSuite AMI instances. Additionally only CloudSuite administrators may distribute labs to other users. In the future normal, or "student", users will be able to submit labs to their instructors for review.

a. Distributing Labs

Labs are distributed by selecting the lab, or labs, to be distributed as well as the users who will receive the lab(s). Clicking the "Distribute Lab" button will make the selected labs available for the chosen users to load, modify, and queue.



Distribute Labs

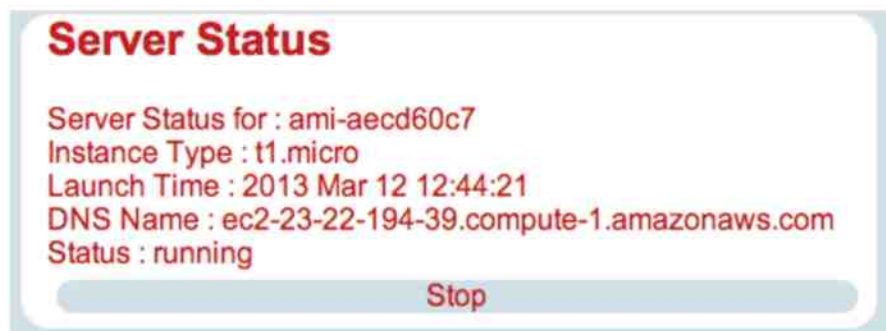
Users
☐ Drew
☐ Gabbo
☐ Jenny

Labs
☐ 10429.CloudSuite_lab_001.xml
Labname : CloudSuite_lab_001
Description : A lab for demos and screen shots.
☐ 18772.copyTest.xml
Labname : copyTest
Description : testing the copy
☐ 18793.copyTest004.xml
Labname : copyTest004
Description : Lab copy test 004

Figure 30. Example lab distribution page

b. Starting / Stopping the Server

The AMI used for lab processing is started and stopped with a simple push button interface pictured in Figure 29. The interface updates to inform the user of the current state of the AMI. The interface also provides the current DNS value for the AMI if the administrator requires SSH access to the server.



Server Status

Server Status for : ami-aecd60c7
Instance Type : t1.micro
Launch Time : 2013 Mar 12 12:44:21
DNS Name : ec2-23-22-194-39.compute-1.amazonaws.com
Status : running

Stop

Figure 31. Example server status dialog when a server is running

Chapter 5: CloudSuite Output

In this chapter we examine the practical output from CloudSuite. We first look at the results from the lab which we configured in chapter four. Then in the second section we examine an experiment performed using CloudSuite.

I. Lab Results

In chapter four we discussed the steps necessary to configure and run a lab using CloudSuite. This section will discuss, in detail, the configuration of the modules as well as the output from the execution of those modules.

First we look at the configured 'graphs' module shown in Figure 32. The 'graphs' module illustrates the data visualization potential of CloudSuite. The module requires the user to supply a title and a data file. Currently the module requires the data to be stored in a specific AWS S3 bucket. The format of the data is a CSV file as illustrated in Figure 34. The 'graphs' module is capable of parsing numerous columns of data though in our case we are using two columns. It is worthwhile to note that the data being used is the output from a previously executed 'ga' module.

Figure 32. 'graph' module Configured with existing data

Figure 33 shows the chart that was generated from running the 'graphs' module. By storing the results in a publicly accessible S3 bucket, the graph may be accessed by anyone. The chart shown in Figure 33 is available at <http://cloudsuite.datawarehouse.s3.amazonaws.com/312522712/grapher.html>.

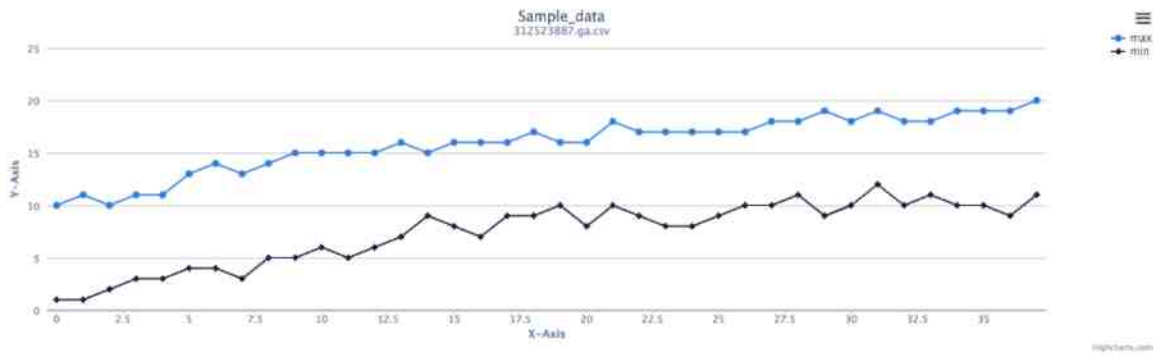


Figure 33.Results from running the graph module.

```
"max", "min"
"10", "1"
"11", "1"
"10", "2"
"11", "3"
"11", "3"
"13", "4"
"14", "4"
"13", "3"
"14", "5"
"15", "5"
"15", "6"
"15", "5"
"15", "6"
"16", "7"
"15", "9"
"16", "8"
"16", "7"
"16", "9"
"17", "9"
"16", "10"
"16", "8"
"18", "10"
"17", "9"
"17", "8"
"17", "8"
"17", "9"
"17", "10"
"18", "10"
"18", "11"
"19", "9"
"18", "10"
"19", "12"
"18", "10"
"18", "11"
"19", "10"
"19", "10"
"19", "9"
"20", "11"
```

Figure 34. The data used to generate Figure 33

The next module used in chapter four is the 'ga' module. This module runs a genetic algorithm written in C. This particular implementation of a genetic algorithm was originally written by John LeFlohic [76] and modified for our use. As shown in Figure 35 the user selects which type of crossover to use when running the genetic algorithm. The type of crossover determines which parts of a pair alleles are used to generate a new organism.

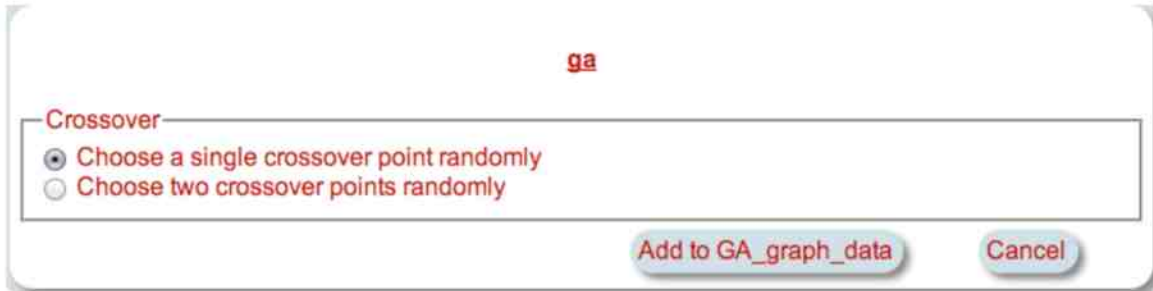


Figure 35. A configured ga module

Once the user has selected the desired type of crossover the algorithm will output a CSV file listing the maximum and minimum gene fitness value per generation. This data is shown in Figure 36. By allowing the user to select the type of crossover it is possible to examine the effects of single and two point crossover on the eventual outcome of a genetic algorithm.

```
"max","min"  
"11","1"  
"9","2"  
"10","1"  
"11","2"  
"12","2"  
"11","4"  
"11","4"  
"11","4"  
"13","4"  
"13","6"  
"13","6"  
"13","6"  
"13","6"  
"14","6"  
"13","6"  
"13","6"  
"14","7"  
"15","6"  
"15","4"  
"16","4"  
"15","7"  
"15","8"  
"15","7"  
"15","7"  
"15","8"  
"16","7"  
"16","7"  
"16","8"  
"16","7"  
"16","8"  
"16","8"  
"16","8"  
"16","8"  
"16","7"  
"16","7"  
"17","8"  
"17","9"  
"17","8"
```

Figure 36.A CSV file showing the minimum and maximum gene values from the 'ga' module

II. Example Experiment

This section examines an experiment that was run with the modules currently available in CloudSuite. The experiment compares the output from single and two point crossover in a genetic algorithm. We present this example experiment to show a practical application of the results from section I of this chapter. To perform this experiment we first configure a lab with both single and two point crossover. Figure 37 illustrates the configured 'ga' lab.

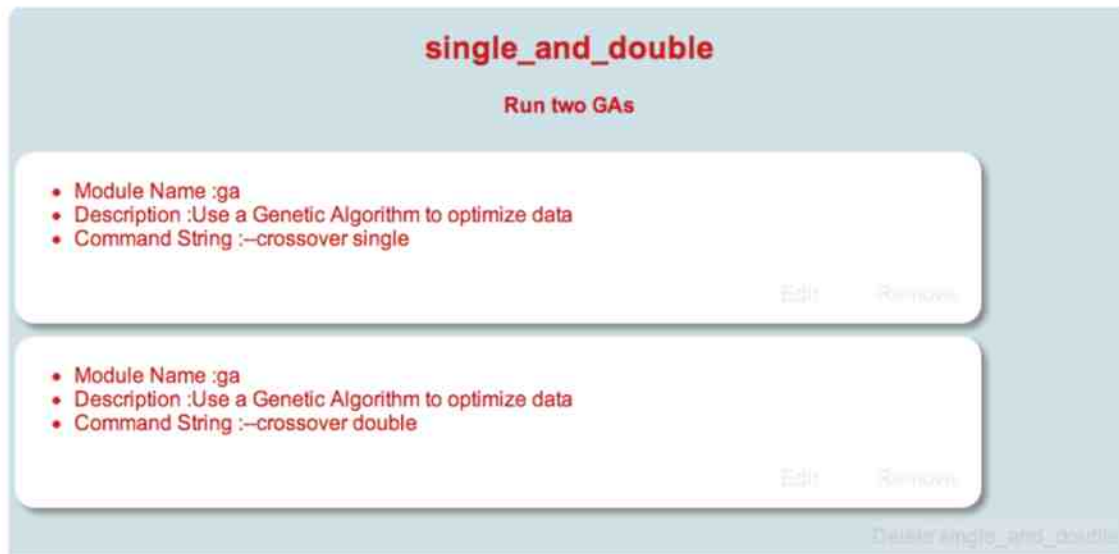


Figure 37. The configured genetic algorithm lab

Once we have results from running both of these labs, we then configure another lab containing the 'grapher' module as shown in Figure 38. This lab will produce two charts which will demonstrate how the process of crossover can affect the eventual outcome of a genetic algorithm.

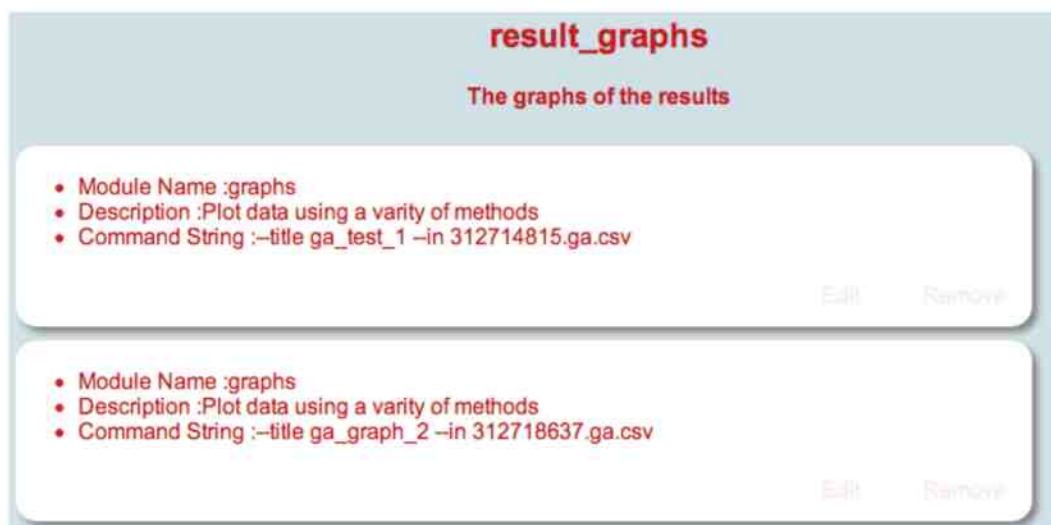


Figure 38. The configured 'result_graphs' lab

The results from the single point crossover module are shown in Figure 39. When looking at the graph online (http://cloudsuite.data.warehouse.s3.amazonaws.com/312803154_grapher.html) it is possible to see that the genetic algorithm found an optimal gene in 75 generations. When examining the two point crossover graph shown in Figure 40 we see that 473 generations were needed to find a solution. The graph is available at http://cloudsuite.data.warehouse.s3.amazonaws.com/312804608_grapher.html. From this we can determine that single point crossover may be more desirable over two point crossover depending on the intended results.

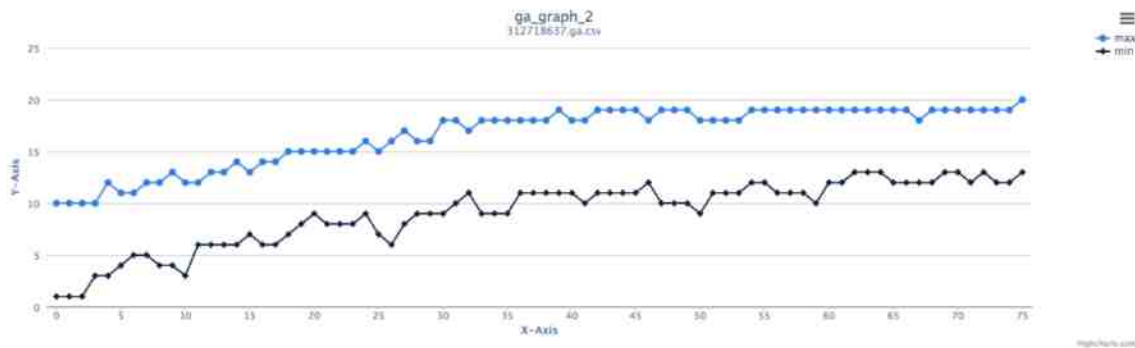


Figure 39.Graph showing single point crossover

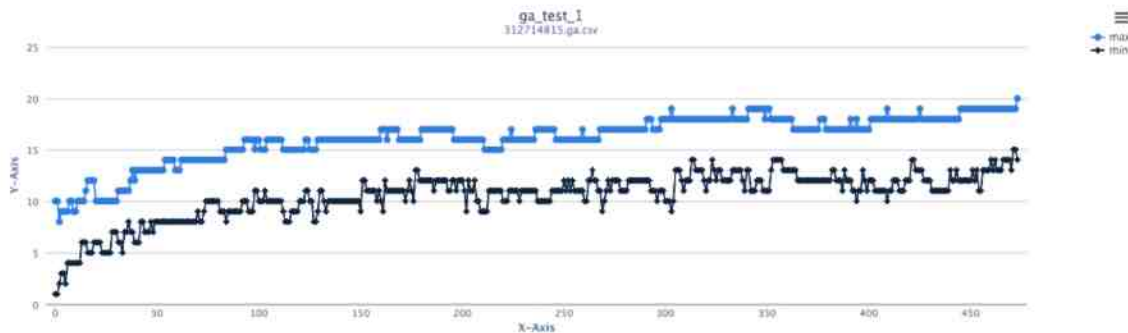


Figure 40.Graph showing two point crossover

Chapter 6: Thesis Results

The primary goal of this thesis was to provide a proof of concept for the feasibility of a cloud based ad hoc laboratory system. The system was to be composed of several parts that would facilitate the creation, modification, and execution of lab assignments. Additionally CloudSuite was to provide teacher/administrator access to specific functions. This chapter discusses how the execution of CloudSuite matches the goals set forth in chapter one.

I. Use Cases

The use cases defined in chapter one served as a guideline for the development of CloudSuite. In this chapter we compare our results with the expectations defined earlier. Any deviations from the initial requirements will be enumerated and solutions will be discussed in the following chapter.

a. Undergraduate Students

The primary goal for undergraduate students was the ability to create and run their own or preconfigured labs. This goal was met completely. Labs are built and executed and the results are delivered to students exactly as the goal defined. The students are only able to view labs that have been shared with them, or that they have created. Once labs have been distributed to students they are able to modify and run the labs as they see fit. One potential issue is that, currently, when a lab is distributed to a student the student becomes the owner of their copy of the lab. It might be more desirable to simply share the lab with the student while the originator retains ownership.

Once a lab has been executed the student has access to the data that has been generated by the lab. This allows the student to analyze the data and draw conclusions from that data. This concludes the requirements for undergraduate use of CloudSuite. In the next section we examine how upper division or graduate students interact with CloudSuite.

b. Upper Division or Graduate Students

Upper division or graduate students have the same ability to access CloudSuite as an undergraduate. These students are able to create, modify, and execute labs in the same way as an undergraduate. An additional use case for graduate, and upper division, students is the ability to create custom modules and functionality for CloudSuite. This is accomplished by supplying the XML schema and generic Python wrapper for use in creating CloudSuite modules. Using these elements as a framework upper division and graduate students are able to create their own modules which may be used in CloudSuite. As mentioned earlier the full text of the XML schema as well as the Python wrappers, are included in appendix E and F, respectively. Adding more documentation would make this easier.

c. Professors

Professors retain all the access of undergraduate and graduate students. Professors have the additional functionality of being able to start and stop the CloudSuite AMI at will as well as distributing labs to students. Starting and stopping the server at preselected times is an additional requirement not discussed in chapter one; however, through the development of CloudSuite it became obvious that this feature would be of great use.

Another requirement put forth in chapter one was the ability for professors to view and edit student generated labs. This functionality is present however it requires the instructor to manually move a lab from the students S3 bucket, to their own lab bucket on Amazon S3. Adding a front end to this functionality is left for future work.

II. Functional Requirements

Functional requirements address areas of CloudSuite that are necessary for the software to be considered functionally complete. Chapter one presented several areas that were necessary for this requirement and chapter four discussed the implementation of these requirements. This chapter will investigate how well the implementation matches the initial requirement starting with creating labs.

a. Creating Labs

Labs are created using a point and click interface exactly as described in the chapter one. Each lab is represented by a unique file that adheres to the criteria defined by the XML schema that defines a lab. This ensures each lab is compatible with CloudSuite. Each lab may contain zero or more modules.

Labs are stored using Amazon S3 which meets the requirement of making the labs available even when the AMI is not running. Through the completion of CloudSuite it became apparent that including data in a lab was not necessary. Data may be operated on by modules contained within a lab but including a data module in the lab itself serves no purpose. Instead data is stored in a collection that is available to the user that generated the data. The user is then able to share a link to that data to whomever they wish.

b. Saving Labs

Labs are saved by clicking the save lab button located in the task bar of CloudSuite. This alerts the user to the success or failure of the save. Additionally each lab is tagged with a unique ID that prevents it from overwriting the labs of other users. Finally the user is able to save as many labs as they wish. Currently there is not mechanism to limit the number of labs a user may create. Limiting the number of labs the user creates maybe an unnecessarily limiting factor. This functionality could be added at a future date if needed.

c. Loading Labs

When a user first logs in to CloudSuite, if a lab has previously been saved, a list of labs is presented for the user to load. Additionally if the user wishes to load other labs clicking a button in the task bar presents a list of labs to which the user has access. Users may access labs they have created or labs that have been distributed to them.

d. Distributing Labs

When distributing a lab the professor is presented with a list of users, and a list of labs. The professor may choose as many users and/or labs as necessary. Once the users are selected the labs are distributed by clicking the ‘Distribute Lab(s)’ button. Each lab is then copied to each selected user. Once labs are distributed they retain the same functionality of a lab that had been created by that user.

One area where the requirement and the implementation differ is that the necessary components of the lab, the modules and/or data, contained within the lab are not distributed. This is simply because the underlying architecture of CloudSuite makes this procedure unnecessary. The modules, as well as the data, have been decoupled from the users and require no additional actions for them to be available for use.

e. Modifying Labs

As the requirements state, each user is only able to modify their own labs, or labs that have been distributed to them. This functionality is currently built in to the system. The ability for administrators or professors to modify labs is not currently implemented. Manipulating the labs of other users requires further development of user authentication. User and group authentication has been left for future work and, as such, does not factor in to the users ability to access labs.

f. Configuring Modules

Configuring modules works precisely as described in chapter one with one exception. Operating on data produced by a module requires the module to be run first then the data may be used in a module.

g. Running a Lab

Labs are processed exactly as described in chapter one. An AMI is activated by a user with the appropriate level of security and a custom daemon process each lab in a ‘first-in-first-out’ queue. As discussed in chapter one, activating the AMI requires the user to possess the correct level of security to start the AMI. Currently the credentials needed to access the AMI are read from a configuration file. However it would be trivial to read the credentials from a user instead of the file. This would facilitate users supplying their own credentials and thus being able to run CloudSuite whenever they wish. This is, of course, predicated upon the user having all the appropriate software as well as an AMI instance. The option to allow users to supply their own credentials has been left for future work.

It is important to note that the way modules, and therefore labs, interact with data is entirely dependent upon how the module was created. Supplying the URI for the data is sufficient for a module to access the data. Additionally it is possible for a module to access data in the S3 bucket directly assuming the proper AWS credentials are supplied.

h. Accessing Student Data

Student data is accessible in two ways. The student may send a URL pointing to their data in an S3 bucket, or the professor may access the students data in the S3 bucket by logging in to the AWS management console and downloading the data directly. A user interface element for this procedure would be more in line with the functional requirements than the current implementation.

Chapter 7:Future Work

Through developing the software that makes up CloudSuite several areas of improvement were identified. These include, but are not limited to, general user management, security, the lab distribution process, and exposure of the data produced by modules. As the goal of this thesis was to provide a proof of concept for CloudSuite, addressing these areas does not fall within the scope of this thesis. However, documenting the changes and identifying areas for improvement will aid in the future development of CloudSuite.

I. User Management

Further development of CloudSuite would necessitate the improvement and expansion of user management. Currently adding a new user is performed by directly editing the underlying XML. Adding a set of API calls and a front end user interface to create and modify CloudSuite users would be an obvious first step. The current user schema is sufficient to support many more functions than are currently offered. Following the outline of the schema would serve as a viable guide for developing users.

Additional users would require the expansion of the group class. Like the user class, a schema exists to define the group structure however group functionality has largely been overlooked. The design of CloudSuite followed standard POSIX/UNIX user management and so much of the needed foundation to implement groups already exists. This limits the amount of rework that is necessary to implement groups. As with users, a group API would need to be developed as would a group user interface element. One of the more challenging aspects of implementing groups would be enforcing the read, write, and execute rights for all of the CloudSuite elements. Though the schemas that have been created all take these permissions into account it would still be necessary to parse these levels and modify the view layer accordingly.

Creating users and groups is currently defined as an administrator or professor level action. It should be possible to add users to a group of sufficient privilege that would allow them to create other users or groups as well as assign group affiliation to users. In this way a ‘normal’ user would become an administrator granting the ability to generate new users or groups.

Users will also be able to edit their own information. This would include changing their password, contact information, and supplying their own Amazon EC2 credentials. The user will only be able to change their group affiliation if they belong to a group of sufficient privilege to change user groups. These elements would require a new user interface element and would follow the design pattern of CloudSuite as well as best practices for user based manipulation of data. This information should be presented when the user clicks on their username in the upper right corner of CloudSuite. This would present the user with an account screen that would include the current ‘Logout’ and ‘Cancel’ buttons.

Additionally it should be possible to generate users from a CSV file, or by uploading a file. The user interface for this functionality would follow the same design pattern

already present in CloudSuite. This would be accomplished by making use of the same APIs used to generate single users but applied to a batch process. Additional API calls will be necessary to facilitate increased security when dealing with user data.

II. Security

Any web based application must be secure, especially any application that stores user data. CloudSuite security is especially important because it is dealing with student data. And, as outlined by FERPA[77] there are numerous student data confidentiality concerns. Additionally using Amazon EC2 instances can lead to monetary charges. Currently CloudSuite does not use secure passwords nor does it encrypt user data. Additional security measures would include limiting the number of failed login attempts and adding a password retrieval mechanism.

Encrypting user data would be achieved by storing the data in an encrypted Amazon S3 bucket. This would be addressed when upgrading the user APIs. No view layer changes would be required. Secure passwords would be implemented at the same time. Securing passwords has a known solution, typically by salting and encrypting the password string, and could be implemented when the user API is upgraded.

An additional feature to be added when modifying the user class would be the addition of a password reset function. This function would require the user to supply the email address associated with their account. CloudSuite would then send an email verification string which would take them to a page where they could enter a new password. This could also be used to unlock the account in the event of multiple failed log in attempts.

Limiting the number of failed login attempts would be accomplished by incrementing a counter for each failed attempt and storing that value in the user object. After each failed attempt the counter would be checked and if a predetermined limit is reached the user would either be locked out until an administrator unlocked them, or for a predetermined amount of time. Either option should be configurable by administrators or professors. These tools will require a user interface element for administrator use.

Finally the API must be secured before it can be made available to the public. This would be accomplished by requiring a valid API key to be sent with each API request. A valid session ID would also be required for calls that return user information.

III. Sessions

Currently CloudSuite does store some user information in a session object; however, it does not make use of this data. When active, persistent sessions would allow CloudSuite to store the username and the last active lab, restoring them when the user returned to CloudSuite. This would require modification of the index.php file to check for session data and load the appropriate view. This feature would be enabled or disabled by a 'Remember Me' checkbox.

IV. Lab Management and Distribution

Deleting CloudSuite labs currently requires a user to load each lab before it's removal. Adding a checkbox and a 'remove lab' button would make this process much simpler. Lab distribution could likewise benefit from the implementation of user groups.

Once user groups are implemented administrators or professors should be able to select individual students, the current functionality, or groups of students for lab distribution. This would work by giving the administrator the choice of listing either individual users, or groups. Toggling between group view, or user view would not deselect the previously chosen recipients of the lab. Once all the users and groups have been selected clicking the 'Distribute Lab' button will copy the lab to all the selected users using the current process.

The option to allow users other than professors or administrators to distribute labs is a configuration option that can be allowed once user groups have been implemented. Any user that has the appropriate level of security, or the appropriate level group, will be able to distribute labs.

V. Editing modules

Currently the only method of editing modules, once they have been added to a lab, is by removing the module and re-adding it to the lab. Ideally it would be possible to edit a module 'in place' by changing the settings present in the lab. There are three ways this could be accomplished: ignoring the previous settings and presenting the user with a 'blank' module configuration screen; by parsing out the XMLRPC string that is stored in the module object and using that data to supply module values; or by restructuring how modules store configuration data to make it more readily accessible.

The third option, restructuring the module, is the most time consuming but offers the best solution. In addition to enabling module editing, by implementing this option it would be possible to list the configuration options on the lab screen when the lab is viewed. This would allow the user to see, instantly, how the lab has been configured.

VI. Collections

Improving the functionality of collections would be a vital part in making CloudSuite extensible. Currently, collections must be created by hand coding the XML necessary to represent them. The API would consist of calls to create, delete, populate and distribute collections. Creating a collection would require a name and a security setting. Deleting a collection would require the user to be of the appropriate level to perform the action. Also deleting a collection would not delete any of the contents of the collection. Populating the collection would require the name of the collection and the name of the module to add to that collection. Distributing a collection would require the correct security level and the name of the user to receive the collection. Finally, collections may be distributed to users of any level; however, accessing the modules contained within the collection would require the correct security level for at least one of the modules contained within the collection. All of these API calls would also require a user interface components.

VII.View Layer

The view layer was largely created in an ‘as needed’ process. Redesigning the view layer in a more deliberate fashion would aid in the usability and extensibility of CloudSuite. Reworking the view layer of CloudSuite to make use of modern JavaScript libraries, like backbone.js and bootstrap.js, would make adding functionality to CloudSuite much simpler. This would also increase the overall portability and platform agnostic nature of CloudSuite.

VIII.Data Import and Use

Data sets must be manually uploaded, or created within CloudSuite, to be used in CloudSuite. A user interface element that allows the upload of data files and automatically configured the appropriate XML file for use in CloudSuite would be a welcome addition to the software. Parsing the XML file and ensuring the integrity of the data, as well as protecting against malicious data, would be the largest challenges when adding the ability to import data.

Currently, a data set must already exist before it is can be flagged for use in a module. In order to allow data to be flagged for use in a module before the originating module has been run, a script that would generate a dummy file would be necessary. This script would be run whenever a module would produce data. Additionally any module that would make use of current system data would require code to scan for extant data and present a list for user selection. This list would be either an HTML multi-select box or an HTML drop down box. Selecting this data would then flag it for use at execution time.

IX.AMI / Data Processing

One of the few areas from the original CloudSuite concept that did not get implemented is the ability for user to supply their own AWS credentials. In the end this may not be a feasible option as the user would also need a complete copy of the CloudSuite backend to make use of their own credentials. Whereas this is possible, by making the AMI available to the user, it does not necessarily fit the philosophy of CloudSuite.

One area that could be improved in CloudSuite is to make use of the parallel processing nature of Amazon EC2. Since each lab can be run independently, and indeed many modules can be run independently as well, CloudSuite is a prime candidate for parallel computing. By making use of the AWS High Performance Computing (HPC) capabilities it would be possible to process each lab simultaneously. However, running a more powerful EC2 instance to handle all the labs could be more expensive.

Processing each lab independently while using the current instant size could be accomplished by spawning a child thread for each lab that needed to be processed instead of processing them in a FIFO queue. Although this would require little change to the code, a great deal of thought and planning would be necessary to ensure maximum efficiency and system stability.

X. Module Uploading / Package Manager

Currently adding a module to CloudSuite requires the creation of several components by hand. Whereas some aspects of module creation, notably the code called by modules, will always need to be created by hand, many aspects could be handled programmatically. A mechanism could be designed by which the module configuration XML and the processing code could be archived and uploaded. This archive could then be parsed by CloudSuite and added automatically to the available modules.

A future possibility for CloudSuite that does not fit within the scope of this thesis is the creation of an APT [66] like package manager. Creating a system where modules may be created, uploaded, and distributed would make CloudSuite much more extensible. This functionality would have to borrow heavily from systems like APT, Easy_Install [67], or Homebrew [68].

XI. API

The current API offers limited access to CloudSuite functionality. Additional work on the API to both expand the API and to make use of the SLIM library would allow greater exposure of CloudSuite functions. Additionally the existing API is not entirely consistent nor does it rigorously follow best practices. Adding these improvements offers enough work for an excellent followup project even though they fall outside the scope of the boundaries of this thesis.

Chapter 8: Conclusion

In this paper we present a proof of concept for CloudSuite through the design, implementation, and analysis of the major necessary components. CloudSuite is a cloud-based software framework to create virtual labs for the demonstration of scientific computer techniques. The software created clearly demonstrates that CloudSuite is a feasible idea and serves as a viable proof of concept.

CloudSuite is considered within the larger context of the applications of cloud-based computing to an educational setting. First, we designed the user interface and necessary functional components, including a modular lab framework accessible by multiple types of users. We then reviewed the existing tools and resources available to assist in the creation of such a framework. We also reviewed existing virtual lab environments. We implemented these ideas into a functional framework that includes a web based user interface, as well as a well-planned, robust backend to carry out the tasks discussed in chapter one. Finally, we analyzed the outcomes, both in terms of CloudSuite itself and how well the original objectives had been met. The few areas where our implementation did not meet the initial requirements have been addressed. We have laid the ground work for addressing them in future versions of CloudSuite.

It is our hope that CloudSuite will continue to grow and evolve over time. It is our belief that this thesis has performed the task it set out to do, namely, demonstrating the validity of the concept of an ad hoc, cloud based, laboratory environment. This thesis presents a vision of the future where advanced ideas in computer science are not limited to those with the knowledge of how to implement them. CloudSuite presents a mechanism by which students in fields other than computer science can make use of computing resources that would otherwise be out of their reach. By allowing students across the disciplines to make use of computer resources we pave the way for new discoveries and new innovations.

References

1. Armbrust, Michael, Armando Fox, et al. "Above the Clouds: A Berkeley View of Cloud Computing." Berkley: 2009. <http://x-integrate.de/x-in-cms.nsf/id/DE_Von_Regenmachern_und_Wolkenbruechen_-_Impact_2009_Nachlese?file/abovetheclouds.pdf>.
2. Frey, Regis, and Wdror-wsu-ap. MVC-Process. 2010. *Wikipedia: The Free Encyclopedia*. Web. 13 Jan 2012. <<http://en.wikipedia.org/wiki/File:MVC-Process.png>>.
3. Mell, Peter and Timothy Grance. "The NIST Definition of Cloud Computing." *NIST Special Publication*. 800-145 <<http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf>>.
4. Blackboard "About Bb" <http://www.blackboard.com/About/Bb/Overview.aspx> n.d. Web. Accessed on 3 February 2012
5. OmniUpdate. "About OmniUpdate" <http://omniupdate.com/company/about> n.d. Web. Accessed on 13 March 2013
6. Cloud9. "About Cloud9" <http://www.cloud9analytics.com/about> n.d. Web. Accessed on 13 March 2013
7. "History of CP-CMS" , *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, n.d. Web. 3 November 2012, http://en.wikipedia.org/wiki/History_of_CP-CMS
8. "Timeline of Virtualization Development", *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, n.d. Web. Accessed on 3 November 2012, http://en.wikipedia.org/wiki/Timeline_of_virtualization_development
9. "Computer Terminal", *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, n.d. Web. 3 November 2012, http://en.wikipedia.org/wiki/Computer_terminal
10. "History of Computer Clusters", *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, n.d. Web. 3 November 2012, http://en.wikipedia.org/wiki/History_of_computer_clusters
11. Milberg, Ken. "IBM and HP virtualization, A comparative study of UNIX virtualization on both platforms." IBM DeveloperWorks. IBM, 29 Sept 2009. Web. 5 NOV 2012. <<http://www.ibm.com/developerworks/aix/library/aixhpvirtualization/index.html>>.
12. Barham, Paul, et al. "Xen and the art of virtualization." ACM SIGOPS Operating Systems Review. Vol. 37. No. 5. ACM, 2003.
13. VMware, "Virtualize Your IT Infrastructure" *VMWare.com*, n.d. Web. Accessed on 13 March 2013
14. Xen® Hypervisor The open source standard for hardware virtualization, "What is the Xen Hypervisor?" <http://xen.org/>, n.d. Web. Accessed on 13 March 2013
15. Virtualbox, "User Manual" <https://www.virtualbox.org/manual/ch01.html> n.d. Web. Accessed on 13 March 2013
16. Warford, Stanley J.. Computer Systems. 2nd Ed. Sudbury, MA: Jones and Bartlett Publishers, 2002. Print.
17. Oracle "The Structure of the Java Virtual Machine" [http://docs.oracle.com/javase/specs/jvms/se7.html/jvms-2.html](http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html) n.d. Web. Accessed on 13 March 2013
18. (Bressoud, and Schneider 90-117) Bressoud, Thomas, and Frew Schneider. "Hypervisor-Based Fault-Tolerance." ACM Transactions on Computer Systems. 14.1 (1996): 90-117. Web. 13 Mar. 2013. <<http://roc.cs.berkeley.edu/294fall01/readings/bressoud.pdf>>.

19. Goldberg, Robert. "Architectural Principles for Virtual Computer Systems." Thesis. Harvard University, 1973. Web. <<http://www.dtic.mil/cgi-bin/GetTRDoc?AD=AD772809&Location=U2&doc=GetTRDoc.pdf>>.
20. "Hypervisor", *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, n.d. Web. 12 November 2012, <http://en.wikipedia.org/wiki/Hypervisor>
21. Xen® Hypervisor The open source standard for hardware virtualization, "How does Xen work?" <http://www.xen.org/files/Marketing/HowDoesXenWork.pdf> n.d. Web. Accessed on 13 March 2013
22. "SIMON (Batch Interactive Test)", *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, n.d. Web 12 November 2012, <[http://en.wikipedia.org/wiki/SIMON_\(Batch_Interactive_test_debug\)](http://en.wikipedia.org/wiki/SIMON_(Batch_Interactive_test_debug))>
23. "CP-40", *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, n.d. Web 12 November 2012, <http://en.wikipedia.org/wiki/IBM_CP_40>
24. IBM, "VM History and Heritage" <http://www.vm.ibm.com/history/> n.d. Web. Accessed on 13 March 2013
25. Smith, J., and R. Nair. Virtual machines: Versatile platforms for systems and processes. 1st. Burlington, MA: Morgan Kaufmann, 2005. Print.
26. Bezos, Jeff "Amazon EC2 Beta" Amazon Web Services Blog, http://aws.typepad.com/aws/2006/08/amazon_ec2_beta.html, 25 August 2006 Web. 14 November 2012
27. Amazon Web Services "What is AWS?" <http://aws.amazon.com/what-is-aws/> n.d. Web. Accessed on 13 March 2013
28. Amazon Web Services "Amazon S3 Pricing" <http://aws.amazon.com/s3/> n.d. Web. Accessed on 13 March 2013
29. Boto boto: A Python interface to Amazon Web Services" <http://docs.pythonboto.org/en/latest/> n.d. Web. Accessed on 13 March 2013
30. Amazon Web Services "Simple Command-Line Access to Amazon EC2 and Amazon S3" <http://aws.amazon.com/developertools/739> n.d. Web. Accessed on 13 March 2013
31. Amazon Web Services "Sample Code & Libraries" <http://aws.amazon.com/code/> n.d. Web. Accessed on 13 March 2013
32. Google Cloud Platform "Google Compute Engine" <http://cloud.google.com/products/compute-engine.html> n.d. Web. Accessed on 13 March 2013
33. Heroku "Cloud Application Platform" <http://www.heroku.com/> n.d. Web. Accessed on 13 March 2013
34. Rackspace "The Open Cloud Company" <http://www.rackspace.com/> n.d. Web. Accessed on 13 March 2013
35. Dreamhost, "About Us" dreamhost.com/about-us/ n.d. Web. Accessed on 13 March 2013
36. Robinson, Tom. "Re: How does Heroku Work" <http://www.quora.com/Scalability/How-does-Heroku-work> n.d. Web. Accessed on 13 March 2013
37. git "About" git-scm.com/about n.d. Web. Accessed on 13 March 2013
38. Heroku Devcenter "Dynos and the Dyno Manifold" <https://devcenter.heroku.com/articles/dynos> n.d. Web. Accessed on 13 March 2013
39. Heroku Devcenter "Slug Compiler" <https://devcenter.heroku.com/articles/slug-compiler> n.d. Web. Accessed on 13 March 2013
40. Coursera "About Coursera" www.coursera.org n.d. Web. Accessed on 13 March 2013

41. Google Developers "Google App Engine General Questions" <https://developers.google.com/appengine/kb/general> n.d. Web. Accessed on 13 March 2013
42. Eucalyptus "Why Eucalyptus" <http://www.eucalyptus.com/why-eucalyptus> n.d. Web. Accessed on 13 March 2013
43. Appscale "Appscale Wiki" <https://github.com/AppScale/appscale/wiki> n.d. Web. Accessed on 13 March 2013
44. PHP "General Information" <http://us3.php.net/manual/en/faq.general.php> n.d. Web. Accessed on 13 March 2013
45. Slim Framework "Slim Documentation" <http://docs.slimframework.com/> n.d. Web. Accessed on 13 March 2013
46. ElementTree. "ElementTree Overview" <http://effbot.org/zone/element-index.htm> n.d. Web. Accessed on 13 March 2013
47. Resig, John. "Selectors in Javascript" <http://ejohn.org/blog/selectors-in-javascript/> n.d. Web. Accessed on 13 March 2013
48. GitHub. "Popular Forked Repositories" <https://github.com/popular/forked> n.d. Web. Accessed on 13 March 2013
49. jQuery. "Sites Using jQuery" http://docs.jquery.com/Sites_Using_jQuery n.d. Web. Accessed on 13 March 2013
50. jQuery, *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, n.d. Web Accessed on 13 March 2013, <<http://en.wikipedia.org/wiki/JQuery>>
51. Coursera, "Support Center" http://help.coursera.org/customer_portal/articles/557884-will-i-receive-a-certificate-or-statement-of-accomplishment-for-each-class-that-i-complete-02 January 2013 Web. 13 March 2013
52. Kahn Academy. "Talks and Interviews" <http://www.khanacademy.org/talks-and-interviews/v/khan-academy-computer-science-launch> n.d. Web. Accessed on 13 March 2013
53. Department of Educational Technology "Home" etec.hawaii.edu n.d. Web. Accessed on 13 March 2013
54. Stuckey-Mickell, Tracey, Stuckey-Danner, Bridget, Taylor, Brandon. "Virtual Labs in the Online Perceptions and Implications for Policy and Practice", TCC 2007 Proceedings Web. Accessed on 13 March 2013 <<http://etec.hawaii.edu/proceedings/2007/stuckey.pdf>>
55. Cristi, Roberto. "EC3400 Digital Signal Processing FPGAs Laboratory" http://faculty.nps.edu/dl/eo3404_dl_lab/ n.d. Web. Accessed on 13 March 2013
56. DiFranco, Mark. "Off Campus Engineering Students Get Virtual Access to NPS Labs." <http://www.nps.edu/About/News/Off-Campus-Engineering-Students-Get-Virtual-Access-to-NPS-Labs.html> n.d. Web. Accessed on 13 March 2013
57. "Grid Computing" , *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, n.d. Web. Accessed on 13 March 2013 http://en.wikipedia.org/wiki/Grid_computing
58. SETI@home "What is SETI@home?" <http://setiathome.ssl.berkeley.edu> n.d. Web. Accessed on 13 March 2013
59. Bonic. "Open-source software for volunteer computing and grid computing." n.d. Web. <http://boinc.berkeley.edu> Accessed on 13 March 2013
60. WLCG "Worldwide LHC Computing Grid" n.d. Web. <http://wlcg.web.cern.ch> Accessed on 13 March 2013
61. EGI "European Grid Infrastructure" n.d. Web. <http://www.egi.eu/about/> Accessed on 13 March 2013
62. Top 500 "Top 500 super computers" June 2012 <http://www.top500.org/lists/2012/06> Accessed on 13 March 2013

63. Debian Administration "Maintaining apache2 sites and modules lists" n.p. Web. <http://www.debian-administration.org/articles/207> Accessed on 13 March 2013
64. PHP "phpinfo" n.d. Web. <http://php.net/manual/en/function.phpinfo.php> Accessed on 13 March 2013
65. Stenberg, Daniel "curl.1 the man page" January 2013 Web. <http://curl.haxx.se/docs/manpage.html> Accessed on 13 March 2013
66. "Advanced Packaging Tool" , *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, n.d. Web Accessed on 13 March 2013, <http://en.wikipedia.org/wiki/Advanced_Packaging_Tool>
67. PEAK "EasyInstall" October 2009 Web. <http://peak.telecommunity.com/DevCenter/EasyInstall> Accessed on 13 March 2013
68. Homebrew "The missing package manager for OS X" n.d. Web. <http://mxcl.github.com/homebrew> Accessed on 13 March 2013
69. Ubuntu "Getting Started with Ubuntu Enterprise Cloud", n.d. Web. <https://help.ubuntu.com/community/UEC> accessed on 24 March 2013
70. Fielding, Thomas "Architectural Styles and the Design of Network-based Software Architectures", 2000, Web. http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm accessed on 24 March 2013
71. "Representational state transfer" , *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, n.d. Web Accessed on 24 March 2013, <http://en.wikipedia.org/wiki/Representational_state_transfer>
72. "Uniform resource identifier" , *Wikipedia: The Free Encyclopedia*. Wikimedia Foundation, n.d. Web Accessed on 24 March 2013, <http://en.wikipedia.org/wiki/Uniform_resource_identifier>
73. builtwith.com "Top in JavaScript Libraries and Functions", n.d. Web. <http://trends.builtwith.com/javascript/top> accessed on 24 March 2013
74. World Wide Web Consortium "Document Object Model" 06 January 2009 Web. <http://www.w3.org/DOM/#what> accessed on 24 March 2013
75. Highcharts JS "Interactive JavaScript charts for your web projects" <http://www.highcharts.com> n.d. Web. Accessed on 25 March 2013
76. LeFlohic, John "Genetic Algorithm", 24 February 1999, Web. <http://www.cs-students.stanford.edu/~jl> Accessed on 25 March 2013
77. Family Educational Rights and Privacy Act (FERPA) "General" n.d. Web. <http://www2.ed.gov/policy/gen/guid/fpco/ferpa/index.html> Accessed on 27 March 2013
78. Nurmi, Daniel, Wolski, Rich, Grzegorzczuk, Chris, Et. Al. "Eucalyptus : A Technical Report on an Elastic Utility Computing Architecture[sic. Linking Your Programs to Useful Systems]", Aug 2008, Web. http://www.cs.ucsb.edu/research/tech_reports/reports/2008-10.pdf Accessed on 25 March 2013

Appendix

Appendix A. PHP Classes

I. cloudsuite.class.php

```
<?php
/**
 * This file is used to call the other CS class files.
 * It should be included whenever CS classes are needed
 *
 * @author Drew A. Clinkenbeard
 */
if (file_exists(dirname(__FILE__) . DIRECTORY_SEPARATOR . 'config.php')) {
    include_once dirname(__FILE__) . DIRECTORY_SEPARATOR . 'config.php';
}

/**
 * CloudSuite: Class include files.
 */
    include_once "collection.class.php";
    include_once "exceptions.class.php";
    include_once "lab.class.php";
    include_once "module.class.php";
    include_once "user.class.php";
    include_once "utils.class.php";
include_once "group.class.php";

?>
```

II. collection.class.php

```
<?php

/**
 * Collections are used for grouping and distributing
 * modules.
 *
 * @author Drew A. Clinkenbeard
 */
class Collection {
    private $schema;
    private $xmlFile;
    private $name;
    private $id;
    private $clearance;
    private $collection;
    private $fileName;
    private $data = array('id' => '',
        'name' => '',
        'ownerID' => '',
        'clearance' => '',);

    function __set($name, $value) {
        if (array_key_exists($name, $this->data)) {
            $this->data[$name] = $value;
            return true;
        } else {
            throw new Exception('No Such Element', '0');
            return FALSE;
        }
    }

    function __get($name) {
        if (array_key_exists($name, $this->data)) {
            return $this->data[$name];
        } else {
            throw new Exception('No Such Element', '0');
            return FALSE;
        }
    }

    function __construct($desc, $ownerID, $name = NULL, $clearance = 10,
        $xmlFile = NULL, $schema = NULL) {

        if ($desc == NULL || $desc == "") {
            throw new Exception("Description can't be null", '1', null);
            return false;
        }
    }
}
```

```
if ($schema == NULL) {
    $schema = $_ENV['cs']['collection_dir'] . "collection.xsd";
}

$this->id = Utils::genID();

$this->name = ($name == NULL) ? Utils::randomName() : $name;

if ($xmlFile == NULL) {
    $xmlFile = Utils::fileName($this->id, $this->name);
}

$this->fileName = $xmlFile;

$this->schema = $schema;
$this->xmlFile = $xmlFile;

$collection = new SimpleXMLElement("<collection></collection>");

$collection->addAttribute('id', $this->id);
$collection->addAttribute('name', $this->name);

$collection->addChild("desc", $desc);
$collection->addChild("ownerID", $ownerID);
$collection->addChild("clearance", $clearance);
$collection->addChild("created", date('Y-m-d'));

$this->collection = $collection;

return $collection;
}

function __destruct() {
    foreach ($this as $key => $value) {
        unset($this->$key);
    }
}

public static function listModules($schema, $xmlFile) {

    $ret = array();
    if (!Utils::load_xml($schema, $xmlFile, $xml)) {
        return false;
    }
    $result = $xml->xpath("//module");
    return $result;
    /*
    foreach ($result as $key => $value) {
```

```
$id = intval((string) $result[$key]["id"]);
$name = (string) $result[$key]["name"];
$ret[$id] = $name;
}

return $ret; */
}

public static function getDesc($schema, $xmlFile) {
    if (!Utils::load_xml($schema, $xmlFile, $xml)) {
        return false;
    }

    return $xml->xpath("/collection/desc");
}

public static function getModuleByID($schema, $xmlFile, $id) {

    if (!Utils::load_xml($schema, $xmlFile, $xml)) {
        return false;
    }

    return $xml->xpath("//module[@id=$id]");
}

public function listTheseModules() {

    $ret = array();

    if (!Utils::load_xml($schema, $xmlFile, $xml)) {
        return false;
    }
    $result = $xml->xpath("//module");

    foreach ($result as $key => $value) {

        $id = intval((string) $result[$key]["id"]);
        $name = (string) $result[$key]["name"];
        $ret[$id] = $name;
    }

    return $ret;
}

public function addModule($moduleObject) {

    if (!Utils::load_xml($xmlSchema, $xmlFile, $xml)) {
        throw new Exceptions("Couldn't access data");
    }
}
```

```
$module = $xml->collection[0]->addChild($module);
$modName = $moduleObject->getName;
$id = Uutil::genID();

$module->addAttribute('name', $modName);
$module->addAttribute('id', $id);
$module->addAttribute('filename', $modName . "." . $id . ".xml");

$sysReqList = $moduleObject->getSystemRequirements;

reset($sysReqList);
while (list($key, $val) = each($sysReqList)) {
    $sysReq = $module->addChild('systemRequirement');
    $sysReq->addChild('product', $moduleObject->getProduct);
    // $sysReq->addChild('version', $moduleObject->get)
}
$fileInfo = $module->addChild('fileInfo');

$fileInfo->addChild('kind', $moduleObject->getKind);
$fileInfo->addChild('path', $moduleObject->getPath);

$params = $fileInfo->addChild('parameter');
$params->addChild('flag');
$params->addChild('type');

$output = $fileInfo->addChild('output');
}

function writeCollection($filename = NULL) {
    //convert to XML and store to the system.

    Uutils::showStuff($this->fileName, 'This file name');

    if ($filename == NULL) {

        $filename = $_ENV['cs']['collection_dir'] . $this->fileName;
    }

    if (file_exists($filename)) {
        //TODO: add lock code.
    }

    $dom = new DOMDocument(1.0);
    $dom->preserveWhiteSpace = false;
    $dom->formatOutput = true;
    $dom->loadXML($this->collection->asXML());

    if (!$dom->save($filename)) {
```



```
        Throw new Exception("Could not save file $filename", '2', NULL);
        return false;
    }
    //return $this->fileName;
    return true;
}

public static function delCollection($fileName, $clearance = 0) {

    if (!($clearance >= $_ENV['cs']['DEL_LEVEL'])) {
        throw new Exception("Not authorized to delete $fileName", '401',
NULL);
    }

    $col = $_ENV['cs']['collection_dir'] . $fileName;

    if(!file_exists($col)) {
        throw new Exception("File $col not found", "404", NULL );
    }

    try {
        unlink($col);
    } catch (Exception $e) {
        throw new Exception("Could not Delete file $fileName", "500", $e);
    }

    return true;
}

/**
 *
 * @return String
 */
function getFileName(){
    return (String) $this->fileName;
}

}

?>
```

III.group.class.php

```
<?php
/**
 * Groups are used for user managment, security, and access rights.
 *
 * @author Drew A. Clinkenbeard
 */
class Group {

    private $__groupXMLFile;
    private $__groupSchema;
    private $myXML;

    function __construct($schema = NULL, $xmlFile = NULL) {

        $this->__groupSchema = ($schema != null) ? $schema : $_ENV['cs']
['schema_dir'] . "group.xsd";
        $this->__groupXMLFile = ($xmlFile != null) ? $xmlFile : $_ENV['cs']
['groupFile'];

        if (Utils::load_xml($this->__groupSchema, $this->__groupXMLFile, $this-
>myXML)) {
            return true;
        } else {
            return false;
        }
    }

    function __destruct() {
        foreach ($this as $key => $value) {
            unset($this->$key);
        }
    }

    function getUserData($id, $value) {
        switch ($value) {
            case "lab":
            case "collection":
            case "lab":
                $xpath = "/groups/group/user[@id=$id]/../$value";
                Utils::showStuff($xpath, "XPATH ");
                return $this->myXML->xpath($xpath);
                break;
            case "groups":
                $returner = array();
                $ret_val = $this->myXML->xpath("/groups/group/user[@id=$id]/../
$id");

                foreach ($ret_val as $key => $value) {
                    echo "key = $key and value = $value";
                    //print_r($value["id"][0]);
                }
            }
        }
    }
}
```

```
$bar = $value["id"][0];
$foo = $this->myXML->xpath("/groups/group[@id=$bar]/@name");
//$returner[(int)$value] = $foo[0];
echo"bar";
$returner[$bar] = $foo[0][0];
print_r($bar);
echo "foo";
print_r($foo[0][0]);

}
return $returner;
////|/groups/group/user[@id=$id]/../@name");
break;
default :
    return false;
}
}

function addMemberToGroup($id, $type, $group) {

switch ($type) {
    case "lab":
    case "collection":
    case "lab":
        $xpath = "/groups/group/user[@id=$id]/../$value";
        Utils::showStuff($xpath, "XPATH ");
        return $this->myXML->xpath($xpath);
        break;
    case "groups":
        return $this->myXML->xpath("//user[@id=$id]/..@id");
        break;
    default :
        return false;
}
}
}

?>
```

IV.lab.class.php

```
<?php
/**
 * Labs are used to process modules.
 * This class contains all the helper functions to create
 * and manage Labs
 *
 * @author Drew A. Clinkenbeard
 */
class Lab {

    private $labSchema;
    private $user;
    private $id;
    private $labname;
    private $fileName;
    private $bucket;
    private $description;
    private $lastRunDate;
    private $lastRunUser;
    private $owner;
    private $permissions = array('owner' => '',
        'group' => '',
        'everyone' => '');
    private $module = array('1' =>
        array('id' => '',
            'moduleName' => '',
            //'seqNumber' => '',
            'method' => '',
            'xmlrpcString' => '',
            'filename' => '',
            'input' =>
                array('type' => '',
                    'filename' => '',
                    'location' => ''
                ),
            'output' =>
                array('type' => '',
                    'filename' => '',
                    'location' => ''
                )
        ),
    );
    private $lab;

    /* * DEPRICATED The data element contains all the data needed for a lab.
     * DEPRICATED Owner, permissions, and an array of module objects stored
with     * DEPRICATED the sequence number as the array key.
```

```
*
* The correct way to access data is by calling it directly. The
* 'magic' getters and setters were too confusing.
*/
private $data = array('owner' => '',
    'permissions' => array('owner' => '7',
        'group' => '5',
        'everyone' => '4'),
    'filename' => '',
    'modules' => array(0 => 'moduleObject'));

function __set($key, $value) {
    if (array_key_exists($key, $this->data)) {
        $this->data[$key] = $value;
        return true;
    } else {
        throw new Exception('No Such Element', '0');
        return FALSE;
    }
}

function __get($key) {
    if (array_key_exists($key, $this->data)) {
        return $this->data[$key];
    } else {
        throw new Exception('No Such Element', '0');
        return FALSE;
    }
}

function __construct($owner, $id = NULL, $labName = NULL, $description =
NULL, $xml=NULL){
    $this->user = $owner;

    Utils::showStuff($labName, 'IN CONSTRUCT LABNAME');

    $this->id = ($id == NULL) ? Utils::genID() : $id;

    if ($labName != NULL) {
        $this->labname = $labName;
    } else {
        $this->labname = Utils::randomName();
    }

    $this->bucket = "cs.user.$this->user.labs";
    $this->bucket = strtolower($this->bucket);

    if ($description != NULL) {
        $this->description = $description;
    }
}
```

```
} else {
    $this->description = "$owner made an unknown lab.";
}

Utils::showStuff($this->labname, 'this lab name...');

$lname = Utils::fileName($this->id, $this->labname);

$this->fileName = Utils::fileName($this->id, $this->labname);

if ($xml != NULL) {
    $this->lab = $xml;
    return true;
}

$lab = new SimpleXMLElement('<lab></lab>');
$lab->addAttribute('id', $this->id);
$lab->addAttribute('labName', $this->labname);

$lab->addChild('owner', $owner);
$lab->addChild('description', $description);

$permissions = $lab->addChild('permissions');
$permissions->addChild('owner', '7');
$permissions->addChild('group', '4');
$permissions->addChild('everyone', '4');

$lab->addChild('lastRunDate', '0');
$lab->addChild('lastRunUser', '0');

/*
    $module = $lab->addChild('module');
    $module->addAttribute('id', '-1');
    $module->addAttribute($name, $value)
*/

$this->lab = $lab;

return $lab;
}

function setFileName($labName) {

    $labName = str_replace(" ", "_", $labName);

    $this->lab['labName'] = $labName;
    $this->fileName = Utils::fileName($this->lab['id'], $labName);
}
```

```
$foo = new SimpleXMLElement("<lab></lab>");
$foo->addChild('bar');
}

function getFileName() {
    return $this->fileName;
}

function setBucket($bucket) {
    $this->bucket = $bucket;
}

function getBucket() {
    return $this->bucket;
}

/**
 *
 * @param string $filename
 * @return Success String
 * @return fail False
 *
 * Always expects a FULL path name so append $_ENV['cs']['labs_dir']
 */
function writeLab($filename = NULL) {
    //convert to XML and store to the system.
    // if (! Utils::validate($this->labSchema, $this->lab->asXML())){
    //     Throw new Exception("invalid Lab structure.", '1', Null);
    //     return false;
    // }
    //Utils::showStuff($this->fileName, 'This file name');

    if ($filename == NULL) {

        $filename = $_ENV['cs']['labs_dir'] . $this->fileName;
    }

    if (file_exists($filename)) {
        //TODO: add lock code.
    }

    $dom = new DOMDocument(1.0);
    $dom->preserveWhiteSpace = false;
    $dom->formatOutput = true;
    $dom->loadXML($this->lab->asXML());

    if (!$dom->save($filename)) {
        Throw new Exception("Could not save file $filename", '2', NULL);
    }
}
```

```
        return false;
    }
    //return $this->fileName;
    return true;
}

function getLab() {
    return $this->lab;
}

static function readLab(String $filename) {
    $domDoc = new DOMDocument();
    $domDoc->load($_ENV['labs_dir'] . $filename);

    //$lab = $domDoc->i

    return $domDoc;
}

/**
 *
 * @param type $xmlFile must append lab directory
 * @param string $xmlSchema
 * @return Lab
 */
public static function loadLab($xmlFile, $xmlSchema = NULL) {

    if ($xmlSchema == null) {
        $xmlSchema = $_ENV['cs']['schema_dir'] . 'lab.xsd';
    }
    if ($xmlFile == null) {
        throw new Exception("XML File must not be null", "1", null);
        return false;
    }

    if (!Utils::load_xml($xmlSchema, $xmlFile, $xml)) {
        throw new Exception("could not load file.", "2", null);
        return false;
    }

    $owner = (String) $xml->owner;
    $id = (String) $xml['id'];
    $labName = (String) $xml['labName'];
    $description = (String) $xml->description;

    return new Lab($owner, $id, $labName, $description, $xml);
}

public static function cloudLoadLab($xmlFile, $xmlSchema = NULL) {
```



```
if ($xmlSchema == null) {
    $xmlSchema = $_ENV['cs']['schema_dir'] . 'lab.xsd';
}

if ($xmlFile == null) {
    throw new Exception("XML File must not be null", "1", null);
    return false;
}

if (!Utils::load_xml($xmlSchema, $xmlFile, $xml, TRUE)) {
    throw new Exception("could not load file.", "2", null);
    return false;
}

$owner = (String) $xml->owner;
$id = (String) $xml['id'];
$labName = (String) $xml['labName'];
$description = (String) $xml->description;

return new Lab($owner, $id, $labName, $description, $xml);
}

public static function cloudWriteLab($filename, $bucket) {

    $s3 = Utils::getS3Instance();
    $fileLocal = "./labs/" . $filename;
    $file = array('fileUpload' => $fileLocal);

    try {

        $s3->create_bucket($bucket, AmazonS3::REGION_US_STANDARD);
    } catch (Exception $e) {
        //don't really care if it fails.
    }

    $response = $s3->create_object($bucket, $filename, $file);
    print_r($response);
    return $response->isOK();
}

public static function cloudListLabs($uname) {
    $s3 = Utils::getS3Instance();

    $bucket = "cs.user." . strtolower($uname) . ".labs";
    return $s3->get_object_list($bucket);
}

/**
```

```

*
* @param SimpleXMLElement $moduleXML
* @param String $xmlFile
* @param String $xmlSchema
* @param String $seqNumber
* @return Boolean
*/
function addModule($moduleXML, $xmlFile = NULL, $xmlSchema = NULL,
$seqNumber = NULL) {

    if ($xmlSchema == NULL && $this->labSchema != NULL) {
        $xmlSchema = $this->labSchema;
    } else {
        $xmlSchema = $_ENV['cs']['schema_dir'] . 'lab.xsd';
    }
    //Utils::showStuff($xmlSchema, 'schema');

    $modules = $this->lab->module;

    $seqNumber = ($seqNumber > sizeof($modules) + 1) ? (sizeof($modules) +
1) : $seqNumber;
    $id = ($moduleXML->module['id'] == NULL) ? Utils::genID() :
$moduleXML['id'];

    if ($seqNumber != NULL && $seqNumber < sizeof($modules)) {
        foreach ($modules as $module) {
            if ($module->seqNumber == $moduleXML->seqNumber) {
                $module->seqNumber = $module->seqNumber + 1;
            } elseif ($module->seqNumber > $moduleXML->seqNumber) {
                $module->seqNumber = $module->seqNumber + 1;
            }
        }
    } else {
        $seqNumber = sizeof($modules) + 1;
    }

    $module = $this->lab->addChild('module');

    $module->addAttribute('id', $id);
    $module->addAttribute('moduleName', (String) $moduleXML['name']);

    $module->addChild('seqNumber', $seqNumber);
    $module->addChild('method', $moduleXML->methodName);
    //$module->addChild('xmlrpcString', $moduleXML->xmlrpcString);
    //$module->addChild('filename', $moduleXML->filename);
    //TEMPORARY HACK FIX

    $paramString = "";
    foreach ($_GET as $key => $value) {
        if (!($key == "addModuleToLab" || $key == "moduleToLoad")) {

```

```
$paramString = $paramString . $key . " ";
if (is_array($value)) {
    foreach ($value as $key2 => $value2) {
        $paramString = $paramString . $value2 . ",";
    }
} else {
    $paramString = $paramString . $value;
}
$paramString = $paramString . " ";
}
}
$module->addChild('xmlrpcString', $paramString);
$module->addChild('filename', Utils::fileName($moduleXML['id'],
$moduleXML['name']));
$module->addChild('description', $moduleXML->description);
/* input and output MUST be pulled from the form.
 * It doesn't make sense to pull it from the module as these values
 * ARE NOT KNOWN to the module. The input and output in the module
 * would more correctly be labeled 'input TYPE' and 'output TYPE'
 *
 * sorry about the confusion.
 *
 * to add to the confusion the input and output are now known to the
module SOMEWHAT.
 * Inputs are now ONLY modules new data is added by creating a new
module.
 * Outputs are, also, modules. A method will be created to download data
 * from modules.
 *
 */

//$fieldset = $module->addChild('fieldset');

/*
$input = $module->addChild('input');
$input->addChild('type', 'input type');
$input->addChild('filename', 'filename');
$input->addChild('location', 'location');

$output = $module->addChild('output');
$output->addChild('type', 'output->type');
$output->addChild('filename', 'output->filename');
$output->addChild('location', 'output->location');
*/
/* $input = $module->addChild('input', $moduleXML->input);
$input->addChild('type', $moduleXML->input->type);
$input->addChild('filename', $moduleXML->input->filename);
$input->addChild('location', $moduleXML->input->location);
```

```
$output = $module->addChild('output',$moduleXML->output);
$output->addChild('type', $moduleXML->output->type);
$output->addChild('filename', $moduleXML->output->filename);
$output->addChild('location', $moduleXML->output->location);
*/

// if (! Utils::validate($xmlSchema, $this->lab->asXML())) {
////    Throw new Exception("Bad module.", '3',NULL);
//    return false;
// }

return true;
}

function getModules() {
    return $this->lab->module;
}

function runLab() {
    $request = xmlrpc_encode_request($method, $params);
    $context = stream_context_create(
        array('http' =>
            array('method' => "POST",
                'header' => "Content-Type: text/xml",
                'content' => $request
            )
        )
    );
};
}

public static function runAllLabs($xmlSchema = NULL, $labDirectory = NULL)
{
    if ($xmlSchema == null) {
        $xmlSchema = $_ENV['cs']['schema_dir'] . 'lab.xsd';
    }
    if ($labDirectory == null) {
        $labDirectory = $_ENV['cs']['labs_dir'];
    }

    $labs = Utils::returnFiles($labDirectory);

    print_r($labs);
}

public function getSimpleXML() {

    return $this->lab;
}
```

```
public static function delLab($fileName, $clearance = 0) {

    if (!$clearance >= $_ENV['cs']['DEL_LEVEL']) {
        throw new Exception("Not authorized to delete $fileName", '401',
NULL);
    }

    $lab = $_ENV['cs']['labs_dir'] . $fileName;

    if (!file_exists($lab)) {
        throw new Exception("File $lab not found", "404", NULL);
    }

    try {
        unlink($lab);
    } catch (Exception $e) {
        throw new Exception("Could not Delete file $fileName", "500", $e);
    }

    return true;
}

static function removeModuleById($labID, $id) {

    $labs = Utils::returnFiles($_ENV['cs']['labs_dir']);

    $xmlSchema = $_ENV['cs']['schema_dir'] . "lab.xsd";

    Utils::showStuff($labID, "lab ID");
    Utils::showStuff($id, "id");
    Utils::showStuff($labs, "labs");

    foreach ($labs as $lab) {
        $parts = explode(".", $lab);

        if ($parts[0] == $labID) {
            $filename = $_ENV['cs']['labs_dir'] . $lab;
            Utils::load_xml($xmlSchema, $filename, $xml);
            break;
        }
    }

    //Utils::showStuff($xml->xpath("//module[@id='$id']"));
    //Utils::showStuff($xml->xpath("//module[@id=$id]"), 'direct');
    // Utils::showStuff($xml, "after unset");

    $dom_sxe = dom_import_simplexml($xml);

    if (!$dom_sxe) {
        throw new Exception("Could not load file", "1", null);
    }
}
```

```
        exit;
    }

    $dom = new DOMDocument('1.0');
    $dom_sxe = $dom->importNode($dom_sxe, true);
    $dom_sxe = $dom->appendChild($dom_sxe);

    $xpath = new DOMXPath($dom);

    foreach ($xpath->query("//module[@id=$id]") as $node) {
        $node->parentNode->removeChild($node);
    }

    $dom->preserveWhiteSpace = false;
    $dom->formatOutput = true;

    if (!$dom->save($filename)) {
        Throw new Exception("Could not save file $filename", '3', NULL);
        return false;
    }
    //return $this->fileName;
    return true;
}

public static function loadLabbyID($labID) {

    $labs = Utils::returnFiles($_ENV['cs']['labs_dir']);

    $xmlSchema = $_ENV['cs']['schema_dir'] . "lab.xsd";

    Utils::showStuff($labID, "lab ID");
    Utils::showStuff($labs, "labs");

    foreach ($labs as $lab) {
        $parts = explode(".", $lab);

        if ($parts[0] == $labID) {
            $filename = $_ENV['cs']['labs_dir'] . $lab;
            Utils::load_xml($xmlSchema, $filename, $xml);
            break;
        }
    }

    Utils::showStuff($xmlSchema, 'schema');
    Utils::showStuff($filename, 'filename');

    Utils::load_xml($xmlSchema, $filename, $xml);

    return new Lab($xml->owner, $labID, $xml['labName'], $xml->description,
$xml);
```

```
}

public static function cloudCHOWN($owner, $filename) {
    $s3 = Utils::getS3Instance();
    $bucket = "cs.user." . strtolower($owner) . ".labs";

    if ($s3->if_bucket_exists($bucket)) {
        $response = $s3->get_object($bucket, $filename);

        if (!$response->isOK()) {
            return false;
        }

        $lab = simplexml_load_string($response->body);
        $lab->owner = $owner;
        $response = $s3->create_object($bucket, $filename, array('body' =>
$lab->asXML()));
        //print_r($response);
        return $response->isOK();
    }

    //load $filename from $owner bucket
    //use body to create simpleXMLElement
    //use simpleXML->lab->owner = $owner
    //use $response = $s3->create_object($bucket, $filename, $file);
    //print_r($response);
}

public static function cloudRM($owner, $filename) {
    $s3 = Utils::getS3Instance();
    $bucket = "cs.user." . strtolower($owner) . ".labs";

    if ($s3->if_bucket_exists($bucket)) {
        $response = $s3->delete_object($bucket, $filename);
        //print_r($response);
        return $response->isOK();
    } else {
        return false;
    }
}

}

?>
```



```
if ($schema != NULL) {
    $this->__moduleSchema = $schema;
} else {
    $this->__moduleSchema = $_ENV['cs']['schema_dir'] . "module.xsd";
}

if ($xmlFile != NULL) {

    $this->__moduleXmlFile = $xmlFile;
    Utils::load_xml($schema, $xmlFile, $xml);
    $this->myXML = $xml;
    $this->fileName = Utils::fileName($xml['id'], $xml['name']);
    return true;
}

$myXML = new SimpleXMLElement('<module></module>');

$id = Utils::genID();
$name = ($name == NULL) ? Utils::randomName() : $name;
$this->fileName = Utils::fileName($xml['id'], $xml['name']);

$myXML->addAttribute('id', $id);
$myXML->addAttribute('name', $name);

$moduleType = $myXML->addChild('moduleType', $type);

$description = $myXML->addChild('description');

$systemReq = $myXML->addChild('systemRequirement');
$systemReq->addChild('product');
$systemReq->addChild('version');

$fieldset = $myXML->addChild('fieldset');
$fieldset->addChild('legend');

$element = $fieldset->addChild('element');
$element->addChild('type');
$element->addChild('name');
$element->addChild('value');
$element->addChild('description');
$element->addChild('input');
$element->addChild('output');
$element->addChild('required');
$element->addChild('default');

$permissions = $myXML->addChild('permissions');
$permissions->addChild('owner');
$permissions->addChild('group');
$permissions->addChild('everyone');
```

```
$permissions->addAttribute('clearance', '9');

$methodName = $myXML->addChild('methodName');
$createdBy = $myXML->addChild('createdBy', $creator);
$dateCreated = $myXML->addChild('dateCreated', date('Y-m-d'));

$modType = $myXML->addChild('moduleType');

$this->myXML = $myXML;

return true;
}

function __destruct() {
    foreach ($this as $key => $value) {
        unset($this->$key);
    }
}

function __get($name) {
    if (array_key_exists($name, $this->__data)) {
        return $this->__data[$name];
    } else {
        throw new Exception('No Such Element', '0');
        return FALSE;
    }
}

function __set($key, $value) {

    if (array_key_exists($key, $this->__data)) {
        $this->__data[$key] = $value;
    } else {
        throw new Exception('no such element', '0', NULL);
        return FALSE;
    }
}

static function getModuleForm($xmlFile, $xmlSchema = NULL, $labFileName =
NULL) {

    if ($xmlSchema == NULL) {
        $xmlSchema = $_ENV['cs']['schema_dir'] . "module.xsd";
    }

    if ($labFileName != NULL) {
        echo "<input type='hidden' value='$labFileName' name='labFileName' /
>";
    }

    if (!Utils::load_xml($xmlSchema, $xmlFile, $xml)) {
```

```

        throw new Exception("Could not load file $xmlFile", '1', NULL);
        return false;
    }
    echo "<h4 class='modTitle'>" . $xml['name'] . "</h4>";
    echo "<form id='addModForm' onSubmit='addFormToLab()' action='>";
    //echo "<form id='addModForm' onSubmit='' action='cloudCommand/
modForm'>";
    echo "<input id='edit-mod-name' type='hidden' value='$xmlFile' />";

    $module['moduleType'] = $xml->xpath("//moduleType");
    $module['description'] = $xml->xpath("/module/description");
    $module['systemRequirement'] = $xml->xpath("//systemRequirement");
    $module['fieldset'] = $xml->xpath("//fieldset");
    $module['permissions'] = $xml->xpath("//permissions");
    $module['methodName'] = $xml->xpath("//methodName");
    $module['createdBy'] = $xml->xpath("//createdBy");
    $module['dateCreated'] = $xml->xpath("//dateCreated");
    $module['id'] = $xml['id'];

    $fs = $xml->xpath("//fieldset");

    foreach ($fs as $key) {
        echo "<fieldset>";
        echo "<legend>" . $key->legend . "</legend>";
        foreach ($key->element as $element) {

            //print_r($element);

            $id = $element['id'];
            echo "<input type='hidden' value='$id'>";
            echo "<input id='$id' type='" . $element->type . "' name=" .
$element->name . " value='$element->value'> $element->description </input>";
            if ($element->input) {
                echo "<div class='moduleInput'>$element->input</div>";
            }
            //echo "</input>";
            echo "<br />";
        }
        echo "</fieldset>";
    }
    $labName = explode('.', $labFileName);
    echo "<div id='cancelModForm-button' class='modButton chiClick
cssshadow'> Cancel </div>";
    if (array_key_exists(1, $labName)) {
        // echo "<button id='addModForm-button' type='submit'
class='modButton chiClick cssshadow'> Add to $labName[1] </button>";
        echo "<div id='addModForm-button' class='modButton chiClick cssshadow'>
Add to $labName[1] </div>";
    } else {
        echo "<div id='' class='modButton cssshadow'> No lab loaded. </div>";
    }

```

```

    }
    echo "</form>";
}

static function loadDataModule($bucket, $filename, $labFileName,
$xmlSchema=null) {

    $data_bucket = 'cloudsuite.data.warehouse';

    if ($xmlSchema == NULL) {
        $xmlSchema = $_ENV['cs']['schema_dir'] . "module.xsd";
    }

    if ($labFileName != NULL) {
        echo "<input type='hidden' value='$labFileName' name='labFileName' /
>";
    }

    $s3 = Utils::getS3Instance();

    $response = $s3->get_object($bucket, $filename);

    if (!$response->isOK()) {
        return false;
    }

    $xmlFile = $response->body;

    if (!Utils::load_xml($xmlSchema, $xmlFile, $xml, true)) {
        throw new Exception("Could not load file $xmlFile", '1', NULL);
        return false;
    }

    $module['moduleType'] = $xml->xpath("//moduleType");
    $module['description'] = $xml->xpath("/module/description");
    $module['systemRequirement'] = $xml->xpath("//systemRequirement");
    //$module['fieldset'] = $xml->xpath("//fieldset");
    $module['permissions'] = $xml->xpath("//permissions");
    $module['methodName'] = $xml->xpath("//methodName");
    $module['createdBy'] = $xml->xpath("//createdBy");
    $module['dateCreated'] = $xml->xpath("//dateCreated");
    $module['id'] = $xml['id'];
    $module['name'] = $xml['name'];
    $module['data'] = $xml->data;

    echo "<h4 class='modTitle'>" . $module['description'][0] . "</h4>";
    //echo "<form id='addModForm' onSubmit='addFormTolab()' action='>";
    echo "<div id='dataDisplay'>";
    //echo "<form id='addModForm' onSubmit='' action='cloudCommand/
modForm'>";

```

```
// echo "<input id='edit-mod-name' type='hidden' value='$xmlFile' />";

try {
    $s3->set_object_acl($data_bucket, $module['data'][0], array(
        array('id' => AmazonS3::USERS_ALL, 'permission' =>
AmazonS3::GRANT_FULL_CONTROL)
    )
);
    $data_url = $s3->get_object_url('cloudsuite.data.warehouse',
$module['data'][0]);
    $data_name = $module['id'] . "." . $module['name'] . ".xml";
} catch (Exception $e) {
    $data_url = $data_name = false;
}

echo "<ul>";
//echo "<li> </li>";
echo "<li>Owner      : " . $module['createdBy'][0] . "</li>";
echo "<li>Created on : " . $module['dateCreated'][0] . "</li>";
echo "<li>data_name : " . $data_name . "</li>";
echo "<li>Module ID : " . $module['id'] . "</li>";
echo "<li>Module Name : " . $module['name'] . "</li>";
if ($data_url) {
    echo "<li><a href=\"\" . $data_url . \"\" target='_blank'>Download Data
File : ".$module['data'][0]."</a></li>";
} else {
    echo "<li>Warning : could not find associated data.</li>";
}
echo "</ul>";
echo "<input id=\"hiddenDataMod\" type=\"hidden\" value=\"\" .
$data_name . \"\"></input> ";

$labName = explode('.', $labFileName);
//echo "<div id='closeData' class='modButton chiClick cssshadow'> Close
</div>";
//echo "<div id='deleteData' class='modButton chiClick cssshadow'> Delete
</div>";
//echo "</form>";
echo "</div>";
}

static function delDataMod($filename, $bucket) {

    $bucket = strtolower($bucket);

    $s3 = Utils::getS3Instance();

    $data_bucket = 'cloudsuite.data.warehouse';

    $xmlSchema = $_ENV['cs']['schema_dir'] . "module.xsd";

    $s3 = Utils::getS3Instance();
```

```
$response = $s3->get_object($bucket, $filename);

if (!$response->isOK()) {
    throw new Exception("Could not load file $filename", '1', NULL);
    return false;
}

$xmlFile = $response->body;

if (!Utils::load_xml($xmlSchema, $xmlFile, $xml, true)) {
    throw new Exception("Could not load file $xmlFile", '1', NULL);
    return false;
}

$data = $xml->data;

echo "deleting $data[0]";

return $s3->delete_object($data_bucket, $data[0])->isOK() && $s3->delete_object($bucket, $filename)->isOK();
}

static function loadModule($xmlFile, $xmlSchema=NULL) {

    if ($xmlSchema == NULL) {
        $xmlSchema = $_ENV['cs']['schema_dir'] . "module.xsd";
    }

    if (!Utils::load_xml($xmlSchema, $xmlFile, $xml)) {
        throw new Exception("Could not load file $xmlFile", '1', NULL);
        return false;
    }

    //__construct($creator, $type, $schema = NULL, $xmlFile = NULL, $name =
    NULL)
    return new Module($xml->createdBy, $xml->moduleType, $xmlSchema,
    $xmlFile, NULL);
}

function removeParam($flag) {

    if (!array_key_exists($this->__data['parameter'][$flag])) {
        throw new Exception('value not found');
        return false;
    } else {

        unset($this->__data[$flag]);
        return true;
    }
}
```

```
}

function addParam($parameterArray) {
    if (!is_array($parameterArray)) {
        throw new Exception('Array expected');
        return false;
    }

    if (!array_key_exists('flag', $parameterArray)) {
        throw new Exception('Flag required');
        return false;
    }

    $this->__data['parameter'][$parameterArray['flag']] =
        array('description' => $parameterArray['description'],
            'value' => $parameterArray['value'],
            'required' => $parameterArray['required'],
            'dataType' => $parameterArray['dataType'],
            'default' => $parameterArray['default'],
            'exclusive' => $parameterArray['exclusive'],
            'input' => $parameterArray['input'],
            'output' => $parameterArray['output']);
}

function listParamters() {
    $params = $this->__data['parameter'];
    //print_r($params);
    return $params;
}

function listParametersByID($id) {

}

/**
 *
 * @return SimpleXMLElement
 */
public function getSimpleXML() {

    return $this->myXML;
}
}

?>
```

VI.¹

```
<?php
/**
 * The code to create, load, and manage users.
 * Most of this class is TBD
 * @author Drew A. Clinkenbeard
 */
class User {
    /**
     private $__id;
     private $__name;
     private $__fname;
     private $__lname;
     private $__clearance;
    */

    private $users = array('1' => 'Drew',
                           '2' => 'Gabbo',
                           '3' => 'Jenny',
                           '4' => 'AJ');

    private $data = array('id' => '',
                          'name' => '',
                          'fname' => '',
                          'lname' => '',
                          'clearance' => '');

    function __set($name, $value) {
        if (array_key_exists($name, $this->data)) {
            $this->data[$name] = $value;
            return true;
        } else {
            throw new Exception('No Such Element', '0');
            return FALSE;
        }
    }

    function __get($name) {
        if (array_key_exists($name, $this->data)) {
            return $this->data[$name];
        } else {
            throw new Exception('No Such Element', '0');
            return FALSE;
        }
    }

    function __construct() {
```

¹ Portions of this class have been edited for security.


```
$this->data['id'] = Utils::genID();
}

function __destruct() {
    foreach ($this as $key => $value) {
        unset($this->$key);
    }
}

public static function login ($uname, $pass, $id=NULL) {

    $users = /*data removed for security reasons*/

    if ($users[$uname] === $pass){
        if (!isset($_SESSION)) {
            session_start();
        }

        $_SESSION['cs']['username'] = $uname;
        $_ENV['cs']['username'] = $uname;
        setcookie("cookie[cs_uname]", $uname);
        return 1;
        // return array ($id => $name);
    }

    return false;
}

public static function checkSession(){
    if (isset($_SESSION['cs']['username'])) {
        return $_SESSION['cs']['username'];
    }

    if (isset($_COOKIE['cs_uname']))){
        return $_COOKIE['cs_uname'];
    }

    else return false;
}

public static function logout () {
    try {

        if (isset($_SESSION['cs']['username']))){
            unset($_SESSION['cs']);
        }
        if (isset($_COOKIE['cs_uname']))){
            unset($_COOKIE['cs_uname']);
        }
    }
}
```

```
        return true;

    } catch (Exception $e) {
        return false;
    }
}

public static function getAWSSettings($xmlFile, $xmlSchema = NULL, $id =
NULL){
    if ($xmlSchema == NULL){
        $xmlSchema = $_ENV['cs']['schema_dir']."user.xsd";
    }

    if ($id == NULL && isset($_SESSION)) {

    }
}
}
?>
```

VII.utils.class.php

```
<?php

/**
 * utils.class.php is a central location for helper functions.
 *
 * @author Drew A. Clinkenbeard
 */
include_once dirname(__DIR__) . DIRECTORY_SEPARATOR . 'aws' .
DIRECTORY_SEPARATOR . 'sdk-1.5.15' . DIRECTORY_SEPARATOR . 'sdk.class.php';

class Utils {

    public static function genID() {
        $idFile = dirname(__FILE__) . DIRECTORY_SEPARATOR . '.idGen';
        $fh = fopen($idFile, 'r') or die("Couldn't get ID file!");
        if (!flock($fh, LOCK_EX)) {
            throw new Exception('Could not get file lock. Try again', '6');
            return -1;
        }

        $id = file_get_contents($idFile);
        $retID = $id + rand(2, 5);
        #$retID = str_pad($retID, 12, "0", STR_PAD_LEFT);

        if (!file_put_contents($idFile, $retID)) {
            flock($fh, LOCK_UN);
            throw new Exception('Could not write file');
            return -1;
        }

        flock($fh, LOCK_UN);
        fclose($fh);
        return $retID;
    }

    public static function fileWrite($xml, $fileDir, $filename) {
        Utils::showStuff($fileDir, 'In utils file directory');
        Utils::showStuff($filename, 'filename');
        Utils::showStuff($xml, 'xml is');
        $fileToWrite = $fileDir . $filename;
        $fh = fopen($fileToWrite, 'r') or die("Couldn't get file!");
        if (!flock($fh, LOCK_EX)) {
            throw new Exception('Could not get file lock. Try again', '6');
            return -1;
        }

        if (!file_put_contents($fileToWrite, $xml)) {
            flock($fh, LOCK_UN);
        }
    }
}
```

```
        throw new Exception('Could not write file');
        return -1;
    }

    flock($fh, LOCK_UN);
    fclose($fh);
    // return $retID;
}

public static function validate($schema, $xmlFile = NULL, $isString=FALSE)
{
    //PROPER ERROR AND RETURN
    //Utils::showStuff($xmlFile, 'UTILS XML FILE');
    //Utils::showStuff($schema, 'UTILS SCHEMA');

    $doc = new DOMDocument();

    try {
        if ($isString) {
            $doc->loadXML($xmlFile);
        } else {
            $doc->load($xmlFile);
        }
    } catch (Exception $e) {
        throw new Exception("Could not load $xmlFile", $e->getCode(), $e-
>getPrevious());
        return false;
    }

    try {
        if ($doc->schemaValidate($schema)) {
            return true;
        } else {
            return false;
        }
    } catch (Exception $e) {
        echo "Neither";
        throw new Exception("Could not validate $xmlFile", $e->getCode(), $e-
>getPrevious());
        return false;
    }
}

/**
 *
 * @param type $xmlSchema
 * @param type $xmlFile
 * @param SimpleXMLElement $xml
 * @return type
 */
```

```
public static function load_xml($xmlSchema, $xmlFile, &$amp;$xml,
$isString=False) {
    if (!Utils::validate($xmlSchema, $xmlFile, $isString)) {
        return false;
    }
    if($isString){
        $xml = simplexml_load_String($xmlFile);
        return true;
    } else {
        $xml = simplexml_load_file($xmlFile);
        return true;
    }
}

/**
 * The following was found at http://www.php.net/manual/en/book.array.php
 * @param type $item
 * @param type $key
 * @param string $array_name
 *
 * produces a javascript array
 */
public static function array_print($item, $key, $array_name) {
    if (is_array($item)) {
        $array_name = $array_name . "[" . $key . "]";
        echo $array_name . " = Array();";
        php_array_to_js_array($item, $array_name);
    } else {
        echo $array_name . "[" . $key . "] = \"" . $item . "\";";
    }
}

public static function php_array_to_js_array($array, $array_name) {
    array_walk($array, 'array_print', $array_name);
}

public static function showStuff($string, $label = NULL) {
    if ($_ENV['cs']['debug'] == TRUE) {
        echo "\n<h1> $label </h1><pre>";
        print_r($string);
        echo "</pre><h1> $label </h1><pre>\n";
    }
}

public static function returnFiles($folder) {
    $ret = array();

    if ($handle = opendir($folder)) {
        while (false !== ($entry = readdir($handle))) {

```

```
        if ($entry != "." && $entry != "..") {
            array_push($ret, $entry);
        }
    }

    closedir($handle);
}

return $ret;
}

public static function randomName() {

    $name = array('Malcom',
        'Kaylee',
        'Jayne',
        'Book',
        'Inara',
        'Simon',
        'River',
        'Wash',
        'Luke',
        'Leia',
        'Anaken',
        'Han',
        'Chewie',
        'Bobbafett'
    );

    return strtolower($name[rand(0, sizeof($name) - 1)]);
}

public static function fileName($id, $filename) {
    $fname = $id . '.' . $filename . '.xml';
    return $fname;
}

/**
 *
 * @param Lab $lab
 * @param XML String $lab
 * @param boolean $source
 * @return string
 *
 * source is a flag to determine if the XML data comes from
 * a Lab object or a string.
 *
 * Formates data for display.
```

```

*
*/
public static function formatLab($lab, $source=false) {
    if ($source) {
        $lab = simplexml_load_string($lab);
    } else {
        $lab = $lab->getSimpleXML();
    }
    $labname = $lab['labName'];
    $filename = Utils::fileName($lab['id'], $labname);

    $return = "<span style=\"text-align:center;\">>\n";
    $return = $return . "\t<h2>" . $labname . "</h2>\n";
    $return = $return . "\t<h4>" . $lab->description . "</h4>\n";
    $return = $return . "</span>\n";
    $return = $return . "<input id=\"labFileNameHidden\" type=\"hidden\" name=\"filename\" value=\"$filename\" />";

    foreach ($lab->module as $module) {
        $return = $return . "<div class=\"lab-content cssshadow lab-slider\">";
        $return = $return . "<ul>";
        $return = $return . "<li>Module Name : " . $module['moduleName'] . "</li>";
        $return = $return . "<li>Description : " . $module->description . "</li>";
        $return = $return . "</ul>";
        $return = $return . "<div onclick=\"delMod(' . $module['id'] . ',' . $lab['id'] . ',' . $module['moduleName'] . ')\", id=\" . $module['id'] . \"_delete\" class=\"status-bar-item labDisplay labButton\">Remove</div>";
        $return = $return . "<div onclick=\"editMod(' . $module['id'] . ',' . $module['moduleName'] . ' . $module->seqNumber . ',' . $lab['id'] . ')\", id=\" . $module['id'] . \"_edit\" class=\"status-bar-item labDisplay labButton\">Edit</div>";
        $return = $return . "</div>\n";
    }
    $return = $return . "<div onclick=\"delLab(' . $filename . ')\", class=\"status-bar-item labDisplay labButton\">Delete $labname</div>";

    return $return;
}

public static function getS3Instance() {

    if (isset($_SESSION['cs']['aws']['credentials'])) {
        $awsCredentials = $_SESSION['cs']['aws']['credentials'];
    } else {
        $awsCredentials = $_ENV['cs']['aws']['credentials'];
    }
    return new AmazonS3($awsCredentials);
}

public static function getEC2Instance() {
    if (isset($_SESSION['cs']['aws']['credentials'])) {

```

```

        $awsCredentials = $_SESSION['cs']['aws']['credentials'];
    } else {
        $awsCredentials = $_ENV['cs']['aws']['credentials'];
    }

    return new AmazonEC2($awsCredentials);
}

public static function getServerStatus($uname) {
    $ec2 = Utils::getEC2Instance();

    if ($uname != "AJ") {
        return;
    }
    echo "<h2>Server Status</h2>";
    //16 == running
    //80 == stopped

    $response = $ec2->describe_instances();
    echo "<div>Server Status for : " . $response->body->reservationSet->item->instancesSet->item->imageId . "</div>";
    echo "<div>Instance Type : " . $response->body->reservationSet->item->instancesSet->item->instanceType . "</div>";
    $startTime = date('Y M d H:i:s', strtotime($response->body->reservationSet->item->instancesSet->item->launchTime));
    echo "<div>Launch Time : $startTime </div>";

    $code = $response->body->reservationSet->item->instancesSet->item->instanceState->code;
    $code = intval($code);
    $name = $response->body->reservationSet->item->instancesSet->item->instanceState->name;
    $dnsname = $response->body->reservationSet->item->instancesSet->item->dnsName;
    echo "<div id='dnsName'>DNS Name : $dnsname</div>";
    echo "<div id='serverCode' name='$code'> Status : $name </div>";
    echo "<div id='statusDiv'></div>";
    if ($code == 80) {
        $loop = false;
        echo "<div id='startServer' class='module adminDisplay chiClick cssshadow'>Start</div>";
    } elseif ($code == 16) {
        $loop = false;
        echo "<div id='stopServer' class='module adminDisplay chiClick cssshadow'>Stop</div>";
    } else {
        echo "<div id='refreshServer' class='module adminDisplay chiClick cssshadow'>Refresh</div>";
    }
}
}
?>

```


Appendix B. Server modules

The following is a listing of all the modules installed and activated on the persistent server hosting `cloudsuite.info`.

- `alias`
- `auth_basic`
- `authn_file`
- `authz_default`
- `authz_groupfile`
- `authz_host`
- `authz_user`
- `autoindex`
- `cgi`
- `deflate`
- `dir`
- `env`
- `mime`
- `negotiation`
- `php5`
- `reqtimeout`
- `rewrite`
- `setenvif`
- `status`
- `wsgi`

Appendix C. Virtual Host File

```
<VirtualHost *:80 >
    DocumentRoot /home/drew/sites/cloudsuite.info/html
    ServerName www.cloudsuite.info
    ServerAlias cloudsuite.info
    CustomLog /home/drew/sites/cloudsuite.info/log/
cloudsuite-access_log combined
    ErrorLog /home/drew/sites/cloudsuite.info/log/cloudsuite-
error_log
</VirtualHost>

<Directory '/home/drew/sites/cloudsuite.info/html">
    Options None
    AllowOverride All
    Order allow,deny
    Allow from all
</Directory>
```

Appendix D. API Reference

This appendix lists the APIs used for accessing CloudSuite function. The following is an example of how the APIs are listed:

2. POST

API Path	Variables	Description
cloudCommand/ birthday/:username/:date	<ul style="list-style-type: none"> • username alphanumeric string • Date yyyy_mm_dd 	Set the birthday belonging to username to date.

Unless otherwise noted the structure of a request is as follows:

cloudsuite.info/cloudCommand/example/drew/1982_02_18

A POST request would set the birthday for the user 'drew' to February 18th, 1982. Currently these APIs can be called 'in the clear'. In the future a valid API key and/or session ID will be required for each transaction as discussed in the chapter seven: Future Work.

1. Get

The following are called with GET requests.

API Path	Variables	Description
cloudCommand/ listMods/:user	<ul style="list-style-type: none"> • user alphanumeric username 	List the modules that have been created by the supplied username.
cloudCommand.queue	<ul style="list-style-type: none"> • N/A 	Get a list of all queued labs.
cloudCommand/ loadLab/:uname/:lab	<ul style="list-style-type: none"> • uname alphanumeric string • lab lab name e.g., 123.lab.xml 	Get the lab that belongs to user and format it for CloudSuite.
cloudCommand/ saveLab/:uname/:filename	<ul style="list-style-type: none"> • uname alphanumeric username • filename lab name e.g., 123.lab.xml 	Saves the lab indicated by filename to the S3 bucket associated with uname
cloudCommand/ distLab/:uname	<ul style="list-style-type: none"> • uname alphanumeric username 	Verify that uname is allowed to share labs and get a list of labs to share.
cloudCommand/ server/:uname	<ul style="list-style-type: none"> • uname alphanumeric username 	Verify that uname is allowed to view lab AMI status and get the status of the AMI.
cloudCommand/ labList/:uname	<ul style="list-style-type: none"> • uname alphanumeric username 	Get a list of labs that belong to uname

API Path	Variables	Description
cloudCommand/ cloudMod/:bucket/:key/:lab name	<ul style="list-style-type: none"> • bucket an S3 bucket • key a key stored in the bucket • labname the name of a lab 	Get the lab results indicated by key from the bucket. :labname is the originating lab.
cloudCommand/ labLister/:uname	<ul style="list-style-type: none"> • uname alphanumeric username 	Get a list of the labs stored in the S3 bucket belonging to uname

The following API calls were implemented before using the SLIM library discussed earlier in this thesis. As mentioned in chapter seven; the following API calls would need to be re-written to offer a more consistent and accurate interface.

The following REST paths require URL parameters in addition to the path. All of the following require a GET request.

API Path	URL parameter	Description
rest.php:colGetModID	<ul style="list-style-type: none"> • xmlScheme Optional, the scheme used to verify the xmlFile. If it is not supplied a default scheme is used. • xmlFile The collection containing the module. • modid the id of the module to load 	Load the module indicated by the ID from the supplied module.
rest.php:colGetDesc	<ul style="list-style-type: none"> • xmlScheme : Optional, the scheme used to verify the xmlFile. If it is not supplied a default scheme is used. • xmlFile : the file representing the desired collection 	Get the description of the collection indicated by xmlFile
rest.php:addModuleToLab	<ul style="list-style-type: none"> • addModuleToLab: the filename of a CloudSuite lab. • moduleToLoad the filename of an XML file representing a module 	Add moduleToLoad to the lab and get the new lab.
rest.php:listLab	<ul style="list-style-type: none"> • uname : alphanumeric username 	list the labs belonging to uname
rest.php:logout	<ul style="list-style-type: none"> • N/A 	logout the currently logged in user. This destroys all session data.

API Path	URL parameter	Description
rest.php/newLab	• newLab : alphanumeric labname	Create a new lab named newLab. Uses Session variable for owner name.
rest.php/saveLab	• saveLab : alphanumeric labname	Writes labname to disk

2. Post

The following commands require a POST request.

API Path	Variables	Description
cloudCommand/shareLab	• N/A	Reads the contents of the \$_POST variable and uses the data there to distribute labs to CloudSuite users.
cloudCommand/queue/labName	• labName : a CloudSuite labname	Adds labName to the execution queue.

3. Delete

The following commands require a DELETE request.

API Path	Variables	Description
cloudCommand/lab/owner/ filename	• owner: alphanumeric username • filename	Delete a lab denoted by filename belonging to owner.
cloudCommand/:filename/: bucket	• filename : a CloudSuite filename • bucket an S3 bucket	delete module denoted by filename in S3 bucket denoted by bucket. Used to remove user generated modules.

4. Put

The following commands require a PUT request.

API Path	Variables	Description
cloudCommand/startServer/username/ instance_id	• uname: alphanumeric username • instance_id : the EC2 instance to start	Starts an EC2 instance indicated by instance_id. Uses uname to verify the user credentials.
cloudCommand/stopServer/username/ instance_id	• uname: alphanumeric username • instance_id : the EC2 instance to start	Stops an EC2 instance indicated by instance_id. Uses uname to verify the user credentials.

Appendix E. XML Schemas

I. collection.xsd

```
<?xml version="1.0" encoding="UTF-8" ?>
<xsd:schema
  version="0.2"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">

  <xsd:annotation>
    <xsd:appinfo>CloudSuite set schema</xsd:appinfo>
    <xsd:documentation xml:lang="en">

      This is the object defenition for the set type object
      used in CloudSuite.info
      Copyright 2011 Drew A. Clinkenbeard.
      All rights reserved.
    </xsd:documentation>
  </xsd:annotation>

  <!-- define simple elements -->
  <xsd:element name="product" type="xsd:string"/>
  <xsd:element name="version" type="xsd:string"/>
  <xsd:element name="kind" type="xsd:string"/>
  <xsd:element name="path" type="xsd:string"/>
  <xsd:element name="flag" type="xsd:string"/>
  <xsd:element name="type" type="xsd:string"/>
  <xsd:element name="desc" type="xsd:string"/>
  <xsd:element name="ownerID" type="xsd:string"/>
  <xsd:element name="clearance" type="xsd:string"/>
  <xsd:element name="created" type="xsd:string"/>
  <xsd:element name="methodName" />
  <xsd:element name="parameterString" type="xsd:string"/>
  <xsd:element name="sequenceNumber" type="xsd:string"/>
  <xsd:element name="input" type="xsd:string"/>
  <xsd:element name="output" type="xsd:string"/>

  <!-- define attributes-->
  <xsd:attribute name="name" type="xsd:string" />
  <xsd:attribute name="id" type="xsd:string" />

  <!-- define complex elements -->

  <xsd:element name="module">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="methodName" />
        <xsd:element ref="parameterString"/>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

```
<xsd:element ref="sequenceNumber" />
<xsd:element ref="desc" />
<xsd:element ref="input" />
<xsd:element ref="output" />
  </xsd:sequence>
  <xsd:attribute ref="name" use="required"/>
  <xsd:attribute ref="id" use="required"/>
</xsd:complexType>
</xsd:element>

<xsd:element name="collection">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="desc" />
      <xsd:element ref="ownerID" />
      <xsd:element ref="clearance" />
      <xsd:element ref="created" />
      <xsd:element ref="module" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute ref="name" use="required"/>
    <xsd:attribute ref="id" use="required"/>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

II. group.xsd

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!--
targetNamespace="http://cloudsuite.info"
xmlns:cs="http://cloudsuite.info"-->

  <xsd:annotation>
    <xsd:appinfo>CloudSuite Group schema</xsd:appinfo>
    <xsd:documentation xml:lang="en">
      This is the object definition for the group type object
      used in CloudSuite.info
      Copyright 2012 Drew A. Clinkenbeard.
      All rights reserved.
    </xsd:documentation>
  </xsd:annotation>

  <!-- define simple elements -->
  <!-- xsd:element name="group" type="xsd:string"/-->
  <!-- xsd:element name="member" type="xsd:string"/-->
  <!-- xsd:element name="id" type="xsd:string"/-->

  <!-- define attributes-->
  <xsd:attribute name="name" type="xsd:string" />
  <xsd:attribute name="id" type="xsd:string" />

  <!-- define complex types -->

  <xsd:element name="user">
    <xsd:complexType>
      <xsd:attribute ref="id" use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="lab">
    <xsd:complexType>
      <xsd:attribute ref="id" use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="collection">
    <xsd:complexType>
      <xsd:attribute ref="id" use="required"/>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="module">
    <xsd:complexType>
      <xsd:attribute ref="id" use="required"/>
    </xsd:complexType>
  </xsd:element>
```



```
<xsd:element name="group">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="user" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="lab" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="collection" minOccurs="0" maxOccurs="unbounded"/>
      <xsd:element ref="module" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute ref="id" use="required"/>
    <xsd:attribute ref="name" use="required"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="groups">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="group" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

III.lab.xsd

```
<?xml version="1.0" encoding="UTF-8"?>

<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:annotation>
    <xsd:appinfo>CloudSuite lab schema</xsd:appinfo>
    <xsd:documentation xml:lang="en">
      This is the object defenition for the lab object type
      used in CloudSuite.info
      Copyright 2012 Drew A. Clinkenbeard.
      All rights reserved.
    </xsd:documentation>
  </xsd:annotation>

  <!-- define simple elements -->
  <xsd:element name="owner" type="xsd:string"/>
  <xsd:element name="group" type="xsd:string"/>
  <xsd:element name="everyone" type="xsd:string"/>
  <xsd:element name="seqNumber" type="xsd:string"/>
  <xsd:element name="method" type="xsd:string"/>
  <xsd:element name="xmlrpcString" type="xsd:string"/>
  <xsd:element name="lastRunDate" type="xsd:string"/>
  <xsd:element name="lastRunUser" type="xsd:string"/>
  <xsd:element name="filename" type="xsd:string"/>
  <xsd:element name="type" type="xsd:string"/>
  <xsd:element name="name" type="xsd:string" />
  <xsd:element name="value" type="xsd:string" />
  <xsd:element name="dataType" type="xsd:string" />
  <xsd:element name="default" type="xsd:string" default="0" />
  <xsd:element name="location" type="xsd:string"/>
  <xsd:element name="description" type="xsd:string"/>
  <xsd:element name="required" type="xsd:boolean" />
  <xsd:element name="legend" type="xsd:string" />

  <!-- define attributes-->
  <xsd:attribute name="id" type="xsd:string" />
  <xsd:attribute name="moduleName" type="xsd:string"/>
  <xsd:attribute name="labName" type="xsd:string"/>
  <xsd:element name="selected" type="xsd:string" />

  <!-- define complex elements -->

  <xsd:element name="input">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element ref="type" />
        <xsd:element ref="filename" />
        <xsd:element ref="location" />
```

```
    </xsd:sequence>
    <!--xsd:attribute ref="id" use="required"/-->
  </xsd:complexType>
</xsd:element>

<xsd:element name="output">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="type" />
      <xsd:element ref="filename" />
      <xsd:element ref="location" />
    </xsd:sequence>
    <!--xsd:attribute ref="id" use="required"/-->
  </xsd:complexType>
</xsd:element>

<xsd:element name="element">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="type" />
      <xsd:element ref="name" />
      <xsd:element ref="value" />
      <xsd:element ref="description" />
      <xsd:element ref="input" maxOccurs="unbounded" minOccurs="0" />
      <xsd:element ref="output" minOccurs="0" maxOccurs="unbounded" />
      <xsd:element ref="required" minOccurs="0"/>
      <xsd:element ref="default" minOccurs="0"/>
      <xsd:element ref="selected" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute ref="id" />
  </xsd:complexType>
</xsd:element>

<!--xsd:element name="fieldset">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="legend" />
      <xsd:element ref="element" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
</xsd:element-->

<xsd:element name="module">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="seqNumber" />
      <xsd:element ref="method" />
      <xsd:element ref="xmlrpcString" />
      <xsd:element ref="filename" />
```

```
minOccurs="0"/><!--xsd:element ref="fieldset" maxOccurs="unbounded"
<xsd:element ref="element" minOccurs="0" maxOccurs="unbounded" />
<xsd:element ref="description" maxOccurs="1" minOccurs="0"/>
</xsd:sequence>
<xsd:attribute ref="id" use="required"/>
<xsd:attribute ref="moduleName" use="required"/>
</xsd:complexType>
<!--xsd:unique name="UniqueSeqNumber">
  <xsd:selector xpath="//module"/>
  <xsd:field xpath="@seqNumber"/>
</xsd:unique-->
</xsd:element>

<xsd:element name="permissions">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="owner" maxOccurs="1"/>
      <xsd:element ref="group" maxOccurs="1"/>
      <xsd:element ref="everyone" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>

<xsd:element name="lab">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="owner" maxOccurs="1"/>
      <xsd:element ref="description" />
      <xsd:element ref="permissions" maxOccurs="1"/>
      <xsd:element ref="lastRunDate" />
      <xsd:element ref="lastRunUser" />
      <xsd:element ref="module" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute ref="id" use="required"/>
    <xsd:attribute ref="labName" use="required"/>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

IV.module.xsd

```
<?xml version="1.0" encoding="utf-8" ?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <!--
targetNamespace="http://cloudsuite.info"
xmlns:cs="http://cloudsuite.info"-->

  <xsd:annotation>
    <xsd:appinfo>CloudSuite module schema</xsd:appinfo>
    <xsd:documentation xml:lang="en">
      This is the object defenition for the module type object
      used in CloudSuite.info
      Copyright 2012 Drew A. Clinkenbeard.
      All rights reserved.
    </xsd:documentation>
  </xsd:annotation>

  <!-- define simple elements -->
  <xsd:element name="product" type="xsd:string"/>
  <xsd:element name="version" type="xsd:string" />
  <xsd:element name="type" />
  <xsd:element name="name" type="xsd:string" />
  <xsd:element name="value" type="xsd:string" />
  <xsd:element name="required" type="xsd:boolean" />
  <xsd:element name="dataType" type="xsd:string" />
  <xsd:element name="default" type="xsd:string" default="0" />
  <xsd:element name="description" type="xsd:string" />
  <xsd:element name="groupname" type="xsd:string" />
  <xsd:element name="everyone" type="xsd:byte" default="4"/>
  <xsd:element name="group" type="xsd:byte" default="4"/>
  <xsd:element name="owner" type="xsd:byte" default="7"/>
  <xsd:element name="input">
    <xsd:complexType>
      <xsd:simpleContent>
        <xsd:extension base="xsd:string"></xsd:extension>
      </xsd:simpleContent>
    </xsd:complexType>
  </xsd:element>
  <xsd:element name="output" type="xsd:string"/>
  <xsd:element name="methodName" type="xsd:string"/>
  <xsd:element name="createdBy" type="xsd:string"/>
  <xsd:element name="dateCreated" type="xsd:string"/>
  <xsd:element name="legend" type="xsd:string" />
  <xsd:element name="selected" type="xsd:string" />
  <xsd:element name="data" type="xsd:string" />
  <xsd:element name="lab" type="xsd:string" />

  <!-- define attributes-->
```

```
<xsd:attribute name="name" type="xsd:string" />
<xsd:attribute name="id" type="xsd:string" />
<xsd:attribute name="clearance" type="xsd:string" />

<xsd:element name="moduleType" >
  <xsd:simpleType>
    <xsd:restriction base="xsd:string">
      <xsd:enumeration value="method" />
      <xsd:enumeration value="data" />
      <xsd:enumeration value="image" />
    </xsd:restriction>
  </xsd:simpleType>
</xsd:element>

<!-- define complex types -->

<xsd:element name="systemRequirement">
  <xsd:complexType>
    <xsd:all>
      <xsd:element ref="product" />
      <xsd:element ref="version" />
    </xsd:all>
  </xsd:complexType>
</xsd:element>

<xsd:element name="element">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="type" />
      <xsd:element ref="name" />
      <xsd:element ref="value" />
      <xsd:element ref="description" />
      <xsd:element ref="input" maxOccurs="unbounded" minOccurs="0" />
      <xsd:element ref="output" minOccurs="0" maxOccurs="unbounded" />
      <xsd:element ref="required" minOccurs="0"/>
      <xsd:element ref="default" minOccurs="0"/>
      <xsd:element ref="selected" minOccurs="0" />
    </xsd:sequence>
    <xsd:attribute ref="id" />
  </xsd:complexType>
</xsd:element>

<xsd:element name="fieldset">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="legend" />
      <xsd:element ref="element" maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:complexType>
```

```
</xsd:element>

<xsd:element name="permissions">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="owner"/>
      <xsd:element ref="everyone"/>
      <xsd:element ref="group"/>
      <xsd:element ref="groupname" maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute ref="clearance" use="required"/>
  </xsd:complexType>
</xsd:element>

<xsd:element name="module">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element ref="moduleType" />
      <xsd:element ref="description" />
      <xsd:element ref="systemRequirement" />
      <xsd:element ref="fieldset" maxOccurs="unbounded" minOccurs="0"/>
      <xsd:element ref="permissions" />
      <xsd:element ref="methodName" />
      <xsd:element ref="createdBy" />
      <xsd:element ref="dateCreated" />
      <xsd:element ref="data" minOccurs="0"/>
      <xsd:element ref="lab" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute ref="id" use="required"/>
    <xsd:attribute ref="name" use="required"/>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Appendix F. Python Wrappers

I. ga.py

```
#!/usr/local/bin/python2.7
from elementtree.ElementTree import Element, SubElement, dump, ElementTree,
tostring
from elementtree.ElementTree import Comment
from optparse import OptionParser as OP
from subprocess import check_output
from boto.s3.key import Key
from cs_error import *
import boto, sys, time, os

logdir = os.environ['HOME'] + '/cs_log/module.log'
logfile = open(logdir, 'a')
error = False
cause = ""

def indent(elem, level=0):
    i = "\n" + level*" "
    if len(elem):
        if not elem.text or not elem.text.strip():
            elem.text = i + " "
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
        for elem in elem:
            indent(elem, level+1)
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
    else:
        if level and (not elem.tail or not elem.tail.strip()):
            elem.tail = i

parser = OP()
...

#These options are necessary for ALL modules.
...

parser.add_option("-u", "--username", dest="username",
                  help="the user who will own the
file")
parser.add_option("-m", "--modname", dest="mod_name",
                  help="The name for the module")
parser.add_option("--not_unique", dest="not_unique", action="store_true",
                  help="the name of the originating
lab.")
parser.add_option("--labname", dest="labname",
                  help="the name of the originating
lab.")
```



```
parser.add_option("--crossover", dest="crossover",
                  help="Type of crossover to use")
(options, args) = parser.parse_args()

if (not options.username == None):
    username = options.username
    uname = username
else:
    error = True
    cause = cause + "No username supplied\n"

if (not options.mod_name == None):
    mod_name = options.mod_name
else:
    cause = cause + "No Module name supplied\n"
    error = True

if (not options.labname == None):
    print "labname is not none"
    labname = options.labname
else:
    cause = cause + "No labname supplied\n"
    error = True
    print "labname IS none"

if (not options.crossover == None):
    crossover = options.crossover
else:
    cause = cause + "No crossover specified\n"
    error = True

if (not error):
    if crossover == 'single':
        executable = "ga/ga_c1.exe"
    else:
        executable = "ga/ga_c2.exe"
    #executable = "ga/ga.exe -c "+crossover
    try:
        output = check_output(executable ,shell=True)
    except:
        error = True
        cause = "\n There was a problem running :\n"+executable

mod_data = "Command called:\n"

for arg in sys.argv:
    mod_data = mod_data + arg + " "

if (error == True) :
```

```
try:
    logfile.write("\n***ERROR***:\n")
    logfile.write("cause : " + cause + "\n")
    logfile.write("mod_data :+" + mod_data + "\n")
    logfile.write("***ERROR***:\n\n")
    raise
except Exception as Inst:
    print "there was an error: " + cause
    raise PreconditionFail(cause)
finally:
    logfile.close()

mod_id = labname.split(".")[0]
if (not options.not_unique):
    mod_id = str(mod_id) + str((long(time.time()*100000)%10000))

filename = mod_id + "."
filename = filename + mod_name.split(".")[0]
data      = filename + ".csv"
filename = filename + ".xml"
directory = os.path.dirname(os.path.realpath(__file__)) + filename

runtime = time.asctime(time.localtime(time.time()))
desc = options.labname.split(".")[1] + "."
desc = desc + mod_name.split(".")[0]
mod_name = mod_name.split(".")[0]
module = Element("module", id=mod_id,name=mod_name)

moduleType = SubElement(module, "moduleType").text = "data"
description = SubElement(module, "description").text = "Data from running "
+ desc

systemReqs = SubElement(module, "systemRequirement")
sr_product = SubElement(systemReqs, "product")
sr_version = SubElement(systemReqs, "version")

fieldset = SubElement(module, "fieldset")
f_legend = SubElement(fieldset, "legend")

f_element = SubElement(fieldset, "element")
e_type = SubElement(f_element, "type")
e_name = SubElement(f_element, "name")
e_value = SubElement(f_element, "value")
e_desc = SubElement(f_element, "description")
e_input = SubElement(f_element, "input")
e_output = SubElement(f_element, "output")
#e_required = SubElement(f_element, "required")
#e_default = SubElement(f_element, "default")
#e_selected = SubElement(f_element, "selected")
```

```
permissions = SubElement(module, "permissions")
permissions.set("clearance", "")
p_owner      = SubElement(permissions, "owner").text = "7"
p_everyone   = SubElement(permissions, "everyone")
p_group      = SubElement(permissions, "group")
p_groupname  = SubElement(permissions, "groupname")

methodName  = SubElement(module, "methodName").text = mod_name
createdBy    = SubElement(module, "createdBy").text = uname
dateCreated  = SubElement(module, "dateCreated").text = runtime
data         = SubElement(module, "data").text = data
lab          = SubElement(module, "lab").text = labname.split(".")[1]

indent(module)
#dump(module)
mod_str = dump(module)
#ElementTree(module).write(filename, encoding="UTF-8", xml_declaration=True)
#Put file in users s3 bucket. BOOM.

c = boto.connect_s3()
bucket = "cs.user."+uname+".modules"
b = c.create_bucket(bucket)
k = Key(b)
k.key = filename
k.set_contents_from_string(tostring(module, encoding="UTF-8"))

b = c.create_bucket("cloudsuite.data.warehouse")
k = Key(b)
k.key = data
k.content_type = 'text/csv'
k.set_contents_from_string(output)

logfile.write("bucket == " + bucket + "\n")
logfile.write("filename == " + filename + "\n")
logfile.write("bucket = cloudsuite.data.warehouse" + "\n")
logfile.write("data == " + data + "\n")
logfile.close()

try:
    logfile = open(logdir, 'a')
    logfile.write("bucket == " + bucket + "\n")
    logfile.write("filename == " + filename + "\n")
    logfile.write("bucket = cloudsuite.data.warehouse" + "\n")
    logfile.write("data == " + data + "\n")
except IOError:
    pass
finally:
    logfile.close()
```

II. grapher.py

```
#!/usr/local/bin/python2.7
from elementtree.ElementTree import Element, SubElement, dump, ElementTree,
tostring
from elementtree.ElementTree import Comment
from optparse import OptionParser as OP
from subprocess import check_output
from boto.s3.key import Key
from lib import graph
from cs_error import *
import boto, sys, time, os

logdir = os.environ['HOME'] + '/cs_log/module.log'
logfile = open(logdir, 'a')

error = False
cause = ""

def indent(elem, level=0):
    i = "\n" + level*" "
    if len(elem):
        if not elem.text or not elem.text.strip():
            elem.text = i + " "
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
        for elem in elem:
            indent(elem, level+1)
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
    else:
        if level and (not elem.tail or not elem.tail.strip()):
            elem.tail = i

parser = OP()
...
#These options are necessary for ALL modules.
...
parser.add_option("-u", "--username", dest="username",
                  help="the user who will own the file")
parser.add_option("-m", "--modname", dest="mod_name",
                  help="The name for the module")
parser.add_option("--not_unique", dest="not_unique", action="store_true",
                  help="the name of the originating lab.")
parser.add_option("--labname", dest="labname",
                  help="the name of the originating lab.")
...
# Modify these options as necessary for the module
...
parser.add_option("--in", dest="infile",
```

```
        help="The CSV file to graph")
parser.add_option("--title", dest="title",
        help="The title of the chart")
(options, args) = parser.parse_args()

if (not options.username == None):
    username = options.username
    uname = username
else:
    error = True
    cause = cause + "No username supplied\n"

if (not options.mod_name == None):
    mod_name = options.mod_name
else:
    cause = cause + "No Module name supplied\n"
    error = True

if (not options.labname == None):
    labname = options.labname
else:
    cause = cause + "No labname supplied\n"
    error = True

if (not options.infile == None):
    infile = options.infile
else:
    cause = cause + "No input file specified\n"
    error = True

if (not options.title == None):
    title = options.title
else:
    cause = cause + "No input file specified\n"
    error = True

if (not error):
    executable = "graph.graphMaker("+infile+", "+title+", "+mod_name+")"
    try:
        output = graph.graphMaker(infile, title, mod_name)
    except:
        error = True
        cause = "\"" + executable + "\" had a problem"

mod_data = "Module command called:\n"

for arg in sys.argv:
    mod_data = mod_data + arg + " "
```

```
if (error == True) :
    try:
        logfile.write("\n***ERROR***:\n")
        logfile.write("cause : " + cause + "\n")
        logfile.write("mod_data :+" + mod_data + "\n")
        logfile.write("***ERROR***:\n\n")
        raise
    except Exception as Inst:
        print "there was an error: " + cause
        raise PreconditionFail(cause)
    finally:
        logfile.close()

mod_id = labname.split(".")[0]
if (not options.not_unique):
    mod_id = str(mod_id) + str((long(time.time()*100000))%10000)

filename = mod_id + "."
filename = filename + mod_name.split(".")[0]
data      = filename + ".html"
filename = filename + ".xml"
directory = os.path.dirname(os.path.realpath(__file__)) + filename
runtime = time.asctime(time.localtime(time.time()))

desc = options.labname.split(".")[1] + "."
desc = desc + mod_name.split(".")[0]

mod_name = mod_name.split(".")[0]
module = Element("module", id=mod_id,name=mod_name)
moduleType = SubElement(module, "moduleType").text = "data"
description = SubElement(module, "description").text = "Data from running "
+ desc

systemReqs = SubElement(module, "systemRequirement")
sr_product = SubElement(systemReqs, "product")
sr_version = SubElement(systemReqs, "version")

fieldset    = SubElement(module, "fieldset")
f_legend    = SubElement(fieldset, "legend")

f_element = SubElement(fieldset, "element")
e_type     = SubElement(f_element, "type")
e_name     = SubElement(f_element, "name")
e_value    = SubElement(f_element, "value")
e_desc     = SubElement(f_element, "description")
e_input    = SubElement(f_element, "input")
e_output   = SubElement(f_element, "output")
#e_required = SubElement(f_element, "required")
#e_default  = SubElement(f_element, "default")
#e_selected = SubElement(f_element, "selected")
```

```
permissions = SubElement(module, "permissions")
permissions.set("clearance", "")
p_owner      = SubElement(permissions, "owner").text = "7"
p_everyone   = SubElement(permissions, "everyone")
p_group      = SubElement(permissions, "group")
p_groupname  = SubElement(permissions, "groupname")

methodName  = SubElement(module, "methodName").text = mod_name
createdBy    = SubElement(module, "createdBy").text = uname
dateCreated  = SubElement(module, "dateCreated").text = runtime
data         = SubElement(module, "data").text = data
lab          = SubElement(module, "lab").text = labname.split(".")[1]

indent(module)
#dump(module)
mod_str = dump(module)
#ElementTree(module).write(filename, encoding="UTF-8", xml_declaration=True)
#Put file in users s3 bucket. BOOM.

c = boto.connect_s3()
bucket = "cs.user."+uname+".modules"
b = c.create_bucket(bucket)
k = Key(b)
k.key = filename
k.set_contents_from_string(tostring(module, encoding="UTF-8"))

b = c.create_bucket("cloudsuite.data.warehouse")
k = Key(b)
k.key = data
k.content_type = 'text/html'

k.set_contents_from_string(output)
logfile.write("bucket == " + bucket + "\n")
logfile.write("filename == " + filename + "\n")
logfile.write("bucket = cloudsuite.data.warehouse" + "\n")
logfile.write("data == " + data + "\n")
logfile.close()

try:
    logfile = open(logdir, 'a')
    logfile.write("bucket == " + bucket + "\n")
    logfile.write("filename == " + filename + "\n")
    logfile.write("bucket = cloudsuite.data.warehouse" + "\n")
    logfile.write("data == " + data + "\n")
except IOError:
    pass
finally:
    logfile.close()
```

III.hashCrack.py

```
#!/usr/local/bin/python2.7
from elementtree.ElementTree import Element, SubElement, dump, ElementTree,
tostring
from elementtree.ElementTree import Comment
#from ElementTree_pretty import prettify
from optparse import OptionParser as OP
from subprocess import check_output
from boto.s3.key import Key
from cs_error import *
import boto, sys, time, os

logdir = os.environ['HOME'] + '/cs_log/module.log'
logfile = open(logdir, 'a')

error = False
cause = ""

def indent(elem, level=0):
    i = "\n" + level*" "
    if len(elem):
        if not elem.text or not elem.text.strip():
            elem.text = i + " "
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
        for elem in elem:
            indent(elem, level+1)
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
    else:
        if level and (not elem.tail or not elem.tail.strip()):
            elem.tail = i

parser = OP()

parser.add_option("-u", "--username", dest="username",
                  help="the user who will own the file")
parser.add_option("-m", "--modname", dest="mod_name",
                  help="The name for the module")
parser.add_option("--not_unique", dest="not_unique", action="store_true",
                  help="the name of the originating lab.")
parser.add_option("--labname", dest="labname",
                  help="the name of the originating lab.")

parser.add_option("--infile", dest="infile",
                  help="The file to be cracked")

parser.add_option("--dictionaries", dest="dictionaries",
                  help="The file to be cracked")
```



```
(options, args) = parser.parse_args()

if (not options.username == None):
    username = options.username
    uname = username
else:
    error = True
    cause = cause + "No username supplied\n"

if (not options.mod_name == None):
    mod_name = options.mod_name
else:
    cause = cause + "No Module name supplied\n"
    error = True

if (not options.labname == None):
    labname = options.labname
else:
    cause = cause + "No labname supplied\n"
    error = True

if (not options.infile == None):
    infile = options.infile
else:
    cause = cause + "No input file specified\n"
    error = True

if (not options.dictionaries == None):
    dictionaries = options.dictionaries
else:
    cause = cause + "No dictionary files specified\n"
    error = True

if (not error):
    executable = "hash_crack/hash_crack.exe"
    try:
        output = check_output(executable ,shell=True)
    except:
        error = True
        cause = "There was a problem running\"" + executable + "\""

mod_data = "Command called:\n"

for arg in sys.argv:
    mod_data = mod_data + arg + " "

if (error == True) :
    try:
```

```
logfile.write("\n***ERROR***:\n")
logfile.write("cause : " + cause + "\n")
logfile.write("mod_data :+" + mod_data + "\n")
logfile.write("***ERROR***:\n\n")
raise
except Exception as Inst:
    print "there was an error: " + cause
    raise PreconditionFail(cause)
finally:
    logfile.close()

mod_id = labname.split(".")[0]
if (not options.not_unique):
    mod_id = str(mod_id) + str((long(time.time()*100000))%10000)

filename = mod_id + "."
filename = filename + mod_name.split(".")[0]
data     = filename + ".txt"
filename = filename + ".xml"
directory = os.path.dirname(os.path.realpath(__file__)) + filename

runtime = time.asctime(time.localtime(time.time()))

desc = options.labname.split(".")[1] + "."
desc = desc + mod_name.split(".")[0]

mod_name = mod_name.split(".")[0]

module = Element("module", id=mod_id,name=mod_name)
moduleType = SubElement(module, "moduleType").text = "data"
description = SubElement(module, "description").text = "Data from running "
+ desc

systemReqs = SubElement(module, "systemRequirement")
sr_product = SubElement(systemReqs, "product")
sr_version = SubElement(systemReqs, "version")

fieldset = SubElement(module, "fieldset")
f_legend = SubElement(fieldset, "legend")

f_element = SubElement(fieldset, "element")
e_type    = SubElement(f_element, "type")
e_name    = SubElement(f_element, "name")
e_value   = SubElement(f_element, "value")
e_desc    = SubElement(f_element, "description")
e_input   = SubElement(f_element, "input")
e_output  = SubElement(f_element, "output")
#e_required = SubElement(f_element, "required")
#e_default  = SubElement(f_element, "default")
#e_selected = SubElement(f_element, "selected")
```

```
permissions = SubElement(module, "permissions")
permissions.set("clearance", "")
p_owner = SubElement(permissions, "owner").text = "7"
p_everyone = SubElement(permissions, "everyone")
p_group = SubElement(permissions, "group")
p_groupname = SubElement(permissions, "groupname")

methodName = SubElement(module, "methodName").text = mod_name
createdBy = SubElement(module, "createdBy").text = uname
dateCreated = SubElement(module, "dateCreated").text = runtime
data = SubElement(module, "data").text = data
lab = SubElement(module, "lab").text = labname.split(".")[1]

indent(module)
#dump(module)
mod_str = dump(module)

c = boto.connect_s3()
bucket = "cs.user."+uname+".modules"
b = c.create_bucket(bucket)
k = Key(b)
k.key = filename
k.set_contents_from_string(tostring(module, encoding="UTF-8"))

b = c.create_bucket("cloudsuite.data.warehouse")
k = Key(b)
k.key = data
k.set_contents_from_string(mod_data)
logfile.write("bucket == " + bucket + "\n")
logfile.write("filename == " + filename + "\n")
logfile.write("bucket = cloudsuite.data.warehouse" + "\n")
logfile.write("data == " + data + "\n")
logfile.close()

try:
    logfile = open(logdir, 'a')
    logfile.write("bucket == " + bucket + "\n")
    logfile.write("filename == " + filename + "\n")
    logfile.write("bucket = cloudsuite.data.warehouse" + "\n")
    logfile.write("data == " + data + "\n")
except IOError:
    pass
finally:
    logfile.close()
```

IV.rsaEncryptor.py

```
#!/usr/local/bin/python2.7
from elementtree.ElementTree import Element, SubElement, dump, ElementTree,
tostring
from elementtree.ElementTree import Comment
#from ElementTree_pretty import prettify
from optparse import OptionParser as OP
from subprocess import check_output
from boto.s3.key import Key
from cs_error import *
import boto, sys, time, os

logdir = os.environ['HOME'] + '/cs_log/module.log'
logfile = open(logdir, 'a')

error = False
cause = ""

def indent(elem, level=0):
    i = "\n" + level*" "
    if len(elem):
        if not elem.text or not elem.text.strip():
            elem.text = i + " "
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
        for elem in elem:
            indent(elem, level+1)
        if not elem.tail or not elem.tail.strip():
            elem.tail = i
    else:
        if level and (not elem.tail or not elem.tail.strip()):
            elem.tail = i

parser = OP()
...
#These options are necessary for ALL modules.
...
parser.add_option("-u", "--username", dest="username",
                  help="the user who will own the file")
parser.add_option("-m", "--modname", dest="mod_name",
                  help="The name for the module")
parser.add_option("--not_unique", dest="not_unique", action="store_true",
                  help="the name of the originating lab.")
parser.add_option("--labname", dest="labname",
                  help="the name of the originating lab.")
...
# Modify these options as necessary for the module
...
parser.add_option("--EnDe", dest="ende",
```

```
        help="encrypt or decrypt")
parser.add_option("--in", dest="infile",
                  help="the file to encrypt/decrypt")
parser.add_option("--out", dest="outfile",
                  help="the name of the output file")
parser.add_option("--inkey", dest="inkey",
                  help="The key to use when encrypting")
parser.add_option("--pubin", dest="pubin",
                  help="the public key used for decryption")

(options, args) = parser.parse_args()

if (not options.username == None):
    username = options.username
    uname = username
else:
    error = True
    cause = cause + "No username supplied\n"

if (not options.mod_name == None):
    mod_name = options.mod_name
else:
    cause = cause + "No Module name supplied\n"
    error = True

if (not options.labname == None):
    labname = options.labname
else:
    cause = cause + "No labname supplied\n"
    error = True
#Custom input checking

if (not options.ende == None):
    ende = options.ende
else:
    cause = cause + "Encrypt or Decrypt not specified\n"
    error = True
...

if (not options.infile == None):
    infile = options.infile
else:
    cause = cause + "No input file specified\n"
    error = True

if (not options.outfile == None):
    outfile = options.outfile
else:
    cause = cause + "No output file specified\n"
```

```
error = True

if (not options.inkey == None):
    inkey = options.inkey
else:
    cause = cause + "No encryption key specified\n"
    error = True

if (not options.pubin == None):
    pubin = options.pubin
else:
    cause = cause + "No public key supplied\n"
    error = True
'''
if (not error):
    executable = "rsa/rsa.exe"
    try:
        output = check_output(executable ,shell=True)
    except:
        error = True
        cause = "\"" + executable + "\" exited with status code " + output

mod_data = "Command called:\n"

for arg in sys.argv:
    mod_data = mod_data + arg + " "

if (error == True) :
    try:
        logfile.write("\n***ERROR***:\n")
        logfile.write("cause : " + cause + "\n")
        logfile.write("mod_data :+" + mod_data + "\n")
        logfile.write("***ERROR***:\n\n")
        raise
    except Exception as Inst:
        print "there was an error: " + cause
        raise PreconditionFail(cause)
    finally:
        logfile.close()

mod_id = labname.split(".")[0]
if (not options.not_unique):
    mod_id = str(mod_id) + str((long(time.time()*100000))%10000)

filename = mod_id + "."
filename = filename + mod_name.split(".")[0]
data = filename + ".txt"
filename = filename + ".xml"
directory = os.path.dirname(os.path.realpath(__file__)) + filename
```

```
runtime = time.asctime(time.localtime(time.time()))

desc = options.labname.split(".")[1] + "."
desc = desc + mod_name.split(".")[0]

mod_name = mod_name.split(".")[0]

module = Element("module", id=mod_id,name=mod_name)

moduleType = SubElement(module, "moduleType").text = "data"
description = SubElement(module, "description").text = "Data from running "
+ desc

systemReqs = SubElement(module, "systemRequirement")
sr_product = SubElement(systemReqs, "product")
sr_version = SubElement(systemReqs, "version")

fieldset = SubElement(module, "fieldset")
f_legend = SubElement(fieldset, "legend")

f_element = SubElement(fieldset, "element")
e_type = SubElement(f_element, "type")
e_name = SubElement(f_element, "name")
e_value = SubElement(f_element, "value")
e_desc = SubElement(f_element, "description")
e_input = SubElement(f_element, "input")
e_output = SubElement(f_element, "output")
#e_required = SubElement(f_element, "required")
#e_default = SubElement(f_element, "default")
#e_selected = SubElement(f_element, "selected")

permissions = SubElement(module, "permissions")
permissions.set("clearance","")
p_owner = SubElement(permissions, "owner").text = "7"
p_everyone = SubElement(permissions, "everyone")
p_group = SubElement(permissions, "group")
p_groupname = SubElement(permissions, "groupname")

methodName = SubElement(module, "methodName").text = mod_name
createdBy = SubElement(module, "createdBy").text = uname
dateCreated = SubElement(module, "dateCreated").text = runtime
data = SubElement(module, "data").text = data
lab = SubElement(module, "lab").text = labname.split(".")[1]

indent(module)
#dump(module)
mod_str = dump(module)
#ElementTree(module).write(filename, encoding="UTF-8", xml_declaration=True)
#Put file in users s3 bucket. BOOM.
```

```
c = boto.connect_s3()
bucket = "cs.user."+uname+".modules"
b = c.create_bucket(bucket)
k = Key(b)
k.key = filename
k.set_contents_from_string(tostring(module, encoding="UTF-8"))

b = c.create_bucket("cloudsuite.data.warehouse")
k = Key(b)
k.key = data
k.set_contents_from_string(mod_data)
logfile.write("bucket == " + bucket + "\n")
logfile.write("filename == " + filename + "\n")
logfile.write("bucket = cloudsuite.data.warehouse"+" \n")
logfile.write("data == " + data + "\n")
logfile.close()

try:
    logfile = open(logdir, 'a')
    logfile.write("bucket == " + bucket + "\n")
    logfile.write("filename == " + filename + "\n")
    logfile.write("bucket = cloudsuite.data.warehouse"+" \n")
    logfile.write("data == " + data + "\n")
except IOError:
    pass
finally:
    logfile.close()
```