View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by CSUSB ScholarWorks

California State University, San Bernardino
CSUSB ScholarWorks

Electronic Theses, Projects, and Dissertations
Office of Graduate Studies

12-2016

# CoyoteLab - Linux Containers for Educational Use

Michael D. Korcha
*California State University - San Bernardino*, korcham@coyote.csusb.edu

Follow this and additional works at: https://scholarworks.lib.csusb.edu/etd

Part of the Computer and Systems Architecture Commons, and the Other Computer Engineering Commons

## Recommended Citation

Korcha, Michael D., "CoyoteLab - Linux Containers for Educational Use" (2016). *Electronic Theses, Projects, and Dissertations*. 424.
https://scholarworks.lib.csusb.edu/etd/424

This Project is brought to you for free and open access by the Office of Graduate Studies at CSUSB ScholarWorks. It has been accepted for inclusion in Electronic Theses, Projects, and Dissertations by an authorized administrator of CSUSB ScholarWorks. For more information, please contact scholarworks@csusb.edu.

# COYOTELAB – LINUX CONTAINERS FOR EDUCATIONAL USE

---

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

---

by

Michael Dennis Korcha

December 2016

COYOTELAB – LINUX CONTAINERS FOR

EDUCATIONAL USE

---

A Project

Presented to the

Faculty of

California State University,

San Bernardino

---

by

Michael Dennis Korcha

December 2016

Approved by:

David Turner, Advisor, Computer Science ______ Date ______

Ernesto Gomez ______

Kerstin Voigt ______

© 2016 Michael Dennis Korcha

# ABSTRACT

CoyoteLab is an exploration in the use of Linux container technology as a means to simplify the way students in computing fields access and complete laboratory work in their educational career. This project provides two main benefits: creating a simple way for students to log in and access their coursework without anything more than their web browser, and providing course instructors a way to verify that assigned work is completed successfully. Thanks to advances in container technology and the advent of WebSockets, this becomes a middle layer between a WebSocket opened up on the clients browser and the SSH daemon running in the users container on a remote server.

## ACKNOWLEDGEMENTS

Thanks to my mom, sister, and grandparents for their constant encouragement through my studies. Thanks to Erica, Ammar, and Devin for keeping me sane during my pursuit of this degree. Thanks to the amazing faculty for their support, guidance, and help during this project and through my academic career at CSUSB.

A special thanks to the authors of open source systems, as without you, my project and many others would fall short of the mark.

TABLE OF CONTENTS

*Abstract* . . . . . . . . . . iii

*Acknowledgements* . . . . . . . . . . iv

*List of Figures* . . . . . . . . . . vii

*1. Introduction* . . . . . . . . . . 1

1.1 Overview . . . . . . . . . . 1

1.2 Purpose . . . . . . . . . . 1

1.3 Project Scope . . . . . . . . . . 2

1.4 Defintions, Acronyms, and Abbreviations . . . . . . . . . . 2

*2. Tools and Environment* . . . . . . . . . . 6

2.1 Software . . . . . . . . . . 6

2.1.1 Python . . . . . . . . . . 6

2.1.2 Nginx . . . . . . . . . . 7

2.1.3 PostgreSQL . . . . . . . . . . 7

2.1.4 Redis . . . . . . . . . . 7

2.1.5 LXD . . . . . . . . . . 8

2.2 Libraries . . . . . . . . . . 8

2.2.1 Flask . . . . . . . . . . 8

2.2.2 gevent . . . . . . . . . . 8

2.2.3 SQLAlchemy . . . . . . . . . . 9

2.2.4 WTForms . . . 9
2.2.5 pylxd . . . 9
2.2.6 wssh . . . 9

*3. Software Overview* . . . 10
3.1 Student Interface . . . 10
3.2 Instructor Interface . . . 12
3.3 Workspace . . . 14

*4. System Architecture* . . . 16
4.1 Application . . . 16
4.2 Database . . . 17
4.3 LXD Container System . . . 18
4.4 File System Layout . . . 19

*5. Implementation* . . . 21
5.1 Database Layout . . . 21
5.2 Access Control . . . 27
5.3 Container Control . . . 32
5.4 SSH-WebSocket Bridge . . . 35

*6. Conclusion* . . . 40
6.1 Future Work . . . 40

*References* . . . 42

## LIST OF FIGURES


2.1 Interaction between application and services . . . . . . . . . . . . . . . 6

3.1 Student's view upon login . . . . . . . . . . . . . . . . . . . . . . . . 10
3.2 Student's view of course details . . . . . . . . . . . . . . . . . . . . . 11
3.3 Instructor's view upon login . . . . . . . . . . . . . . . . . . . . . . . 12
3.4 Instructor's student management interface . . . . . . . . . . . . . . . . 13
3.5 Workspace view . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 14

4.1 Basic outline of system interactions . . . . . . . . . . . . . . . . . . . 16
4.2 Application to database interactions . . . . . . . . . . . . . . . . . . . 17
4.3 Application to container system interactions . . . . . . . . . . . . . . . 18
4.4 Host disk partition layout . . . . . . . . . . . . . . . . . . . . . . . . 19
4.5 Visual outline of copy-on-write files . . . . . . . . . . . . . . . . . . . 19

5.1 Database table structure and relationships . . . . . . . . . . . . . . . . 22
5.2 Course object attributes and methods . . . . . . . . . . . . . . . . . . . 23
5.3 Code for password getting and setting . . . . . . . . . . . . . . . . . . 24
5.4 User object attributes and methods . . . . . . . . . . . . . . . . . . . . 25
5.5 Machine object attributes and methods . . . . . . . . . . . . . . . . . . 26
5.6 Code for check if a user has a role . . . . . . . . . . . . . . . . . . . . 27
5.7 Code for before-request permission check . . . . . . . . . . . . . . . . . 28
5.8 Code for adding a role to a user . . . . . . . . . . . . . . . . . . . . . 28
5.9 Code for retrieving a course for an instructor . . . . . . . . . . . . . . . 29

5.10 Code for retrieving a course for a student . . . . . . . . . . . . . . . . 30

5.11 Code for retrieving a student with access to a specific course . . . . . 31

5.12 Code for view where an instuctor can access a student's or their own information with appropriate permission . . . . . . . . . . . . . . . . 31

5.13 Code for establishing a trusted LXD connection . . . . . . . . . . . . 32

5.14 Code for creating a container . . . . . . . . . . . . . . . . . . . . . . 33

5.15 Code for retrieving a container . . . . . . . . . . . . . . . . . . . . . 34

5.16 Code for bridging an SSH connection and a WebSocket (via wssh) [12] 35

5.17 Code for forwarding packets from the WebSocket to the SSH server (via wssh) [12] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 36

5.18 Code for forwarding packets from the SSH server to the WebSocket (via wssh) [12] . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 37

5.19 Code for running and connecting to the container . . . . . . . . . . . 38

# 1. INTRODUCTION

## *1.1 Overview*

CoyoteLab is an exploration of the use of Linux container technology as a means to simplify the way students in technical fields access and complete assignments in their educational career. It is a web application providing educational institutions the ability to assign students to a course, and giving each their own container for that course that is accessible to the student and their instructor. In this way, it provides benefits both to the student who has to complete coursework and to the instructor by providing an easy way to run their assignments in a common environment.

## *1.2 Purpose*

CoyoteLab was inspired from watching several years of undergraduate entry-level computer science courses. In each course, there would be several students who would bring their own machines, running an environment vastly different from that of the labs or what their instructor was expecting. For example, these students would be running the Windows operating system using the Visual C++ compiler, when the lab machines were running Scientific Linux and using the GNU C++ compiler. As these students did not want to install an additional operating system, they would spend several hours trying to create an environment that would ultimately not behave as needed for their lab work.

CoyoteLab solves this problem by being a web-based system accessible from any

device that has an Internet connection and a web browser supporting WebSocket technology. Once logged in, a student has access to a fully-featured Linux container, which works just as any lab machine would, and can be further configured to their liking. All software used on it is the same as that on the lab machines, and remains consistent with the environments of their instructors and classmates. This saves the students time from having to set up an environment that may not work in the same way as is expected for their assigned work.

An added benefit is that a students instructor gets full access to the students container as well. If a student decides they want to use something non-standard, and the instructor cant get their code to work, they are able to connect to the students container and see how the environment differs and verify that the project does indeed work. This saves the instructor time and frustration of not getting a project working, and may prevent a student from getting lower marks on their assignments.

## 1.3 Project Scope

Students are able to view the classes they are enrolled in that use the system, view information about the courses, and access their files and a personal workspace from a web browser. Students can install software to the workspace, restart it, and reset it.

Instructors can create courses in the system, add students to their courses, and access a workspace of their own or one of the workspaces of their students. Instructors have full access to a student's workspace, except for the ability to reset it.

## 1.4 Defintions, Acronyms, and Abbreviations

- bit mask: A technique to modify or retrieve specific bits of a piece of data, typically using bitwise AND and bitwise OR operators.

- bcrypt: A password hashing function based on the Blowfish encryption cipher which can be adapted by increasing computation time.
- Btrfs: A file system created by Oracle Corporation based on the copy-on-write principle, which enables features such as snapshots.
- cgroups: A Linux kernel feature responsible for limiting resources for processes.
- container: Technique to run multiple isolated Linux systems using a single kernel, utilizing the cgroups feature to isolate resources.
- copy-on-write: A method to share a mutable resource without consuming additional resources until the shared resource needs to be modified.
- daemon: Software that runs in the background instead of under direct control of the user.
- ext4: A file system used in the Linux kernel, which is used as the default on many Linux-based operating systems.
- file system: A system that controls how data is stored and represented on a disk drive.
- Flask: A web framework written in the Python programming language.
- foreign key: A database field that identifies a row from another database table.
- gevent: A coroutine-based library for Python for networking.
- hashing: The use of a mathematical cryptography algorithm to map data to a a string of fixed size, designed to be infeasible to reverse.
- HTTP: Hypertext Transfer Protocol - the protocol used to deliver web sites to a user's web browser.

- Linux: An open source operating system kernel which provides core system functionalty such as device drivers and file systems.
- LXC: Linux Containers - a project that develops Linux container technology to have isolated Linux systems on one kernel using various kernel-level features.
- LXD: A daemon providing an API to perform various system operations on a Linux container.
- migrations: A way to control and version a database schema, to improve automation of database operations and improve consistency between deployments.
- Nginx: Software that provides a web server and reverse proxy to serve content over a network.
- paramiko: A Python library providing an implementation of the SSH protocol.
- PostgreSQL: A relational database management system that emphasises standards compliance.
- pylxd: A library that provides access to the LXD REST API without manually constructing requests to the LXD daemon.
- Python: A high-level, general purpose programming language.
- Redis: Software which provides an in-memory data store mapping keys to values of various data types.
- REST API: Representational State Tranfer Application Programming Interface - an interoperable way to manipulate resources using a representation of the data in the form of HTTP requests.
- reverse proxy: A server that retrieves resources for a client from a server.

- session: A way to store user data between pages to denote the state of the application. This is typically accomplished using web cookies stored in the users browser.
- SQL: Structured Query Language - a language designed for accessing and manipulating data in a relational database.
- SQLAlchemy: A Python SQL toolkit that also contains an object-relational mapper to provide an abstraction around standard database operations.
- SSH: Secure Shell - a network protocol used to securely access and operate services over a network.
- UTF-8: A standard for character encoding that provides points for all characters defined in the Unicode standard.
- virtual machine: An emulation of a computer system that provides the functionality of a physical system, usually implemented with software and improved with hardware enhancements.
- web framework: Software or libraries used to assist in the development of web applications.
- WebSocket: A protocol that provides network communication over a TCP connection from a web browser.
- wssh: A Python library providing a bridge between a WebSocket and a server using the SSH protocol.
- WTForms: A Python library that assists in the creation and validation of web forms.
- ZIP: An archive data format that compresses files without loss of data.

# 2. TOOLS AND ENVIRONMENT

## *2.1 Software*

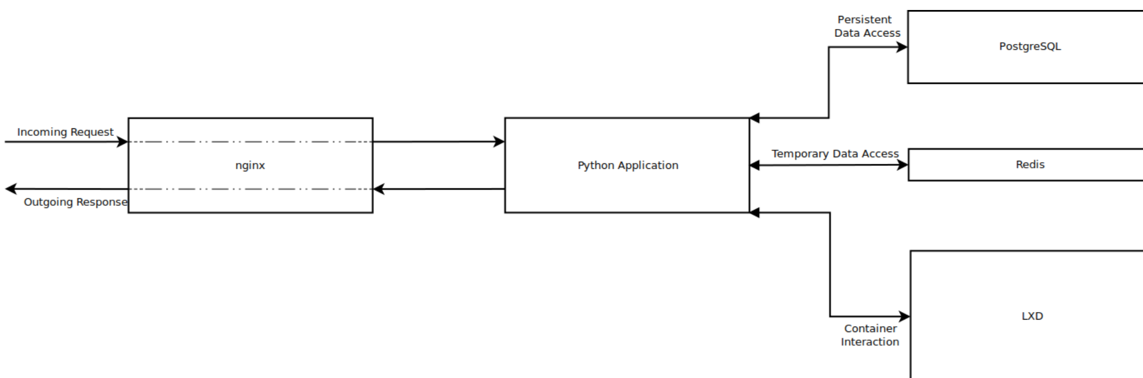


*Fig. 2.1:* Interaction between application and services

Several open source software projects are used in the development implementation of CoyoteLab. Some of these components can be replaced with others due to implementation features. Tools such as text editors and database management software are not included, as they are not within the scope of the project.

### *2.1.1 Python*

Python is the programming language used in this project. It was chosen due to familiarity, ease of use with database systems, and the libraries available that made the project much simpler to implement.

The Python application runs a gevent server with several threads to handle incoming traffic and return responses to the requests. These responses are generated using sofware written with the Flask web framework, along with a handful of other libraries, to provide functionality the application requires.

### *2.1.2 Nginx*

In front of the Python application is the Nginx reverse proxy server, which is used to serve dynamic content over HTTP [5]. Nginx could be replaced with another similar system, such as the Apache web server, with additional configuration. In some configurations, a proxy server is not required, as the gevent server used in the Python application fully supports the HTTP protocol.

### *2.1.3 PostgreSQL*

PostgreSQL is the database system in use, which was chosen because of it's full UTF-8 compatibility, built-in safety checks and improved query optimizer compared to other database systems [1]. The Python application uses the database to store user information, course information, and container references.

PostgreSQL could be replaced in this system with another relational database system, thanks to the SQLAlchemy library used in the Python application and database migrations created with the project. Because of this, a simple change to the connection string in the application configuration allows use of other database systems [7].

### *2.1.4 Redis*

Redis is a key-value store system that is used to store session information and temporary data in a fast in-memory store. This session data is accessed in the application,

with information from the cookie on the user's web browser, to store information relating to the current session.

### 2.1.5 LXD

LXD is a daemon which provides LXC functionality either locally on the host machine or over a network from another machine [3]. This functionality includes starting, stopping, restarting, and destroying Linux containers. For this project, LXD's API is configured to accept local connections to interact with containers.

## 2.2 Libraries

Several libraries are used to ease the development of this project.

### 2.2.1 Flask

The Flask web framework is used to define web application routes that respond to the given request parameters. By design, it doesn't include form processing or database handlers, as it's meant to be as light as possible while providing core web request handling functionality [2]. These features are provided by other libraries used in this project.

### 2.2.2 gevent

The gevent library provides a Python networking library to provide a high performance implementation of networking threads [10]. It was chosen for it's built-in support of WebSockets and ease of implentation with Flask applications.

### *2.2.3 SQLAlchemy*

SQLAlchemy is a library that interfaces with SQL databases to query data from it and create data models from different tables in the database. By it's design, a change in the configuration will allow one to change database systems almost seamlessly [7]. When combined with the Alembic database migration system from the same author, one can bring up a new database with minimal effort.

### *2.2.4 WTForms*

WTForms provides an API to create, validate, and process web forms with ease. Each form is declared in a class, with each field having a validator attached that is run when the form is validated [8]. Almost all input processing in this project is handled using WTForms.

### *2.2.5 pylxd*

The pylxd library is a wrapper around LXD's REST API which allows one to create, start, stop, destroy, and otherwise interact and manage the containers used in the application. It allows use synchronously or asynchronously, meaning it can run operations without blocking the execution of other functions, if desired.

### *2.2.6 wssh*

The wssh library simplifies the bridging of an SSH connection to a WebSocket for the user to access in their browser. It does this by connecting a WebSocket to the paramiko library, providing an SSH implementation in Python, and passing data sent over the WebSocket into this library to be sent to the container.

# 3. SOFTWARE OVERVIEW

This application provides a class with Linux containers in their web browser to complete course work. Each user gets a container for each course they are involved with, either as a student or as an instructor. The course instructor is able to access the containers of each of the students in the courses they are the instructor of.

The software provides three primary interfaces: the student interfaces, the instructor interfaces, and the workspace interface used by both students and instructors.

## *3.1 Student Interface*

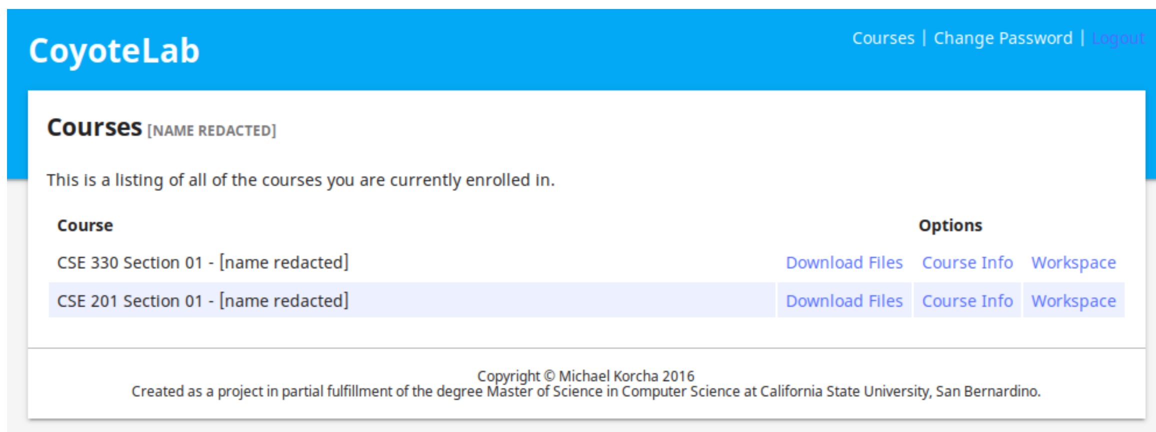


*Fig. 3.1:* Student's view upon login

On login, the student sees a screen which lists all of the courses they are a part of. This listing does not include courses that they have been dropped from. For each of these courses, they are presented a link to see more course information, a link to download the files from the workspace for that course, and a link to access their workspace for that course.

The download link initiates a download of a ZIP archive containing all of the files from the user's directory. The course information link will take the user to a screen displaying more information about the course. The workspace link will take the user to their workspace specifically for that course.

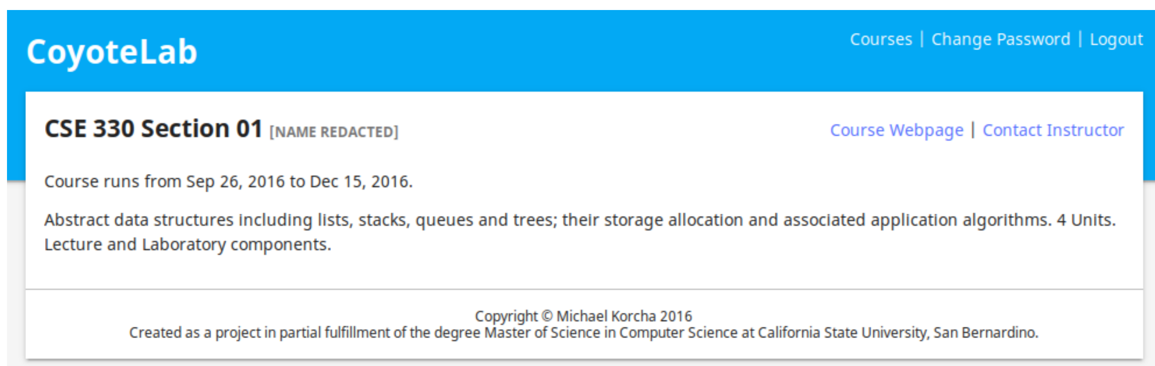


*Fig. 3.2:* Student's view of course details

The course information screen shows information provided by the instructor about the course. This information includes a link to contact the instructor via email, a link to visit the course webpage, a description of the course, and the duration of the course.

When clicked, the link to contact the instructor will open the user's email software to compose an email to the instructor. The course webpage link will open the course webpage provided by the instructor in a new window. If the instructor does not

provide a course webpage, the link will not appear.

### 3.2 Instructor Interface

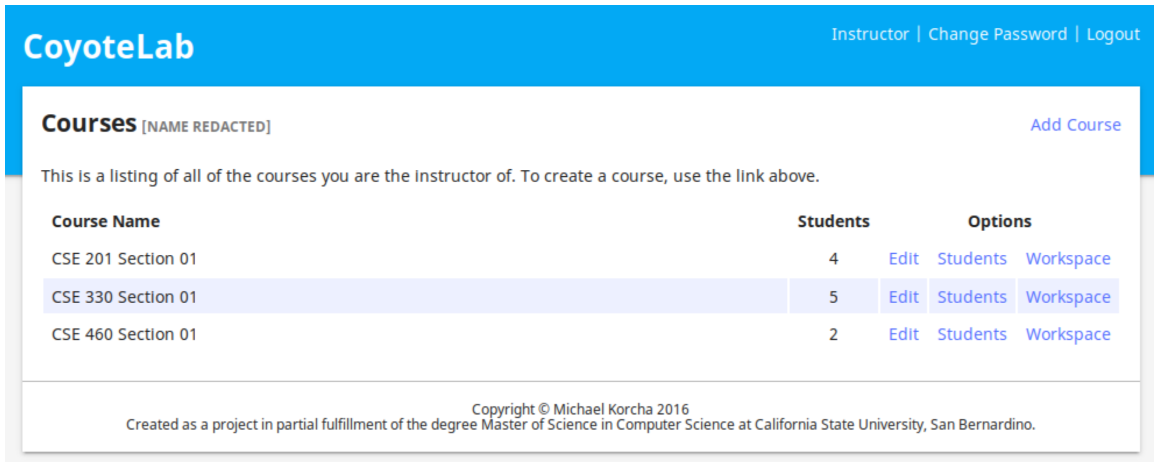


*Fig. 3.3:* Instructor's view upon login

On login, the instructor sees a screen with all of the courses they are the instructor of. For each course, the instructor is able to edit the course information, view the students in the course, and go to a workspace for their course. This workspace is separate from student workspaces to provide an environment for the instructors to teach with, if desired.

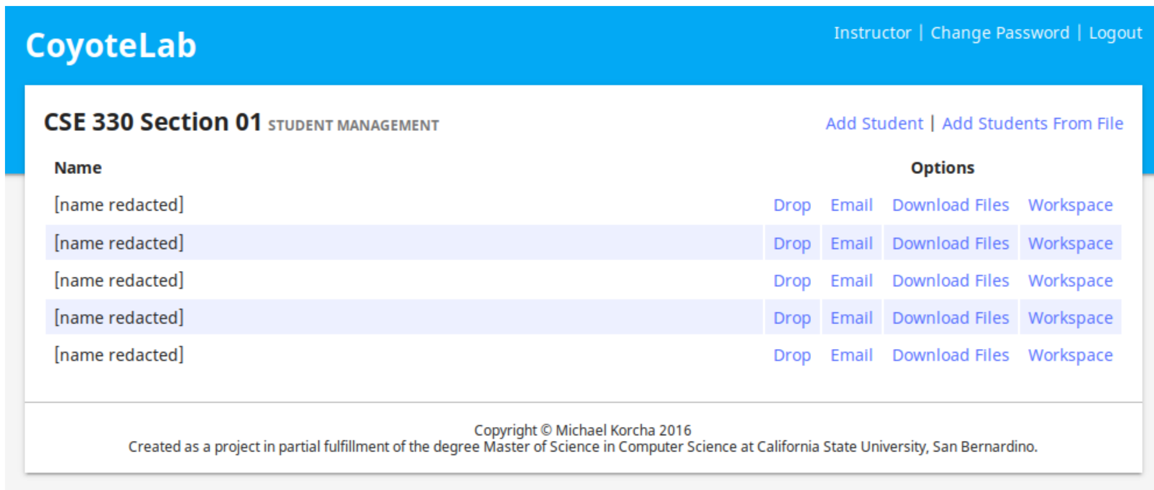


*Fig. 3.4:* Instructor's student management interface

In the students view, the instructor can drop or email any student in their course. They also have access to download their files from the container and access the student's workspace.

When a student is dropped from a course, they are no longer able to access their container or information about the course. However, the instructor is able to re-add them, as the link changes when a student is dropped to do so.

The download files link behaves exactly as it does for students, but for a selected student instead of the currently logged in user. It initiates a download of a ZIP archive containing the selected student's files from their container. The workspace link will take the instructor to the student's workspace.

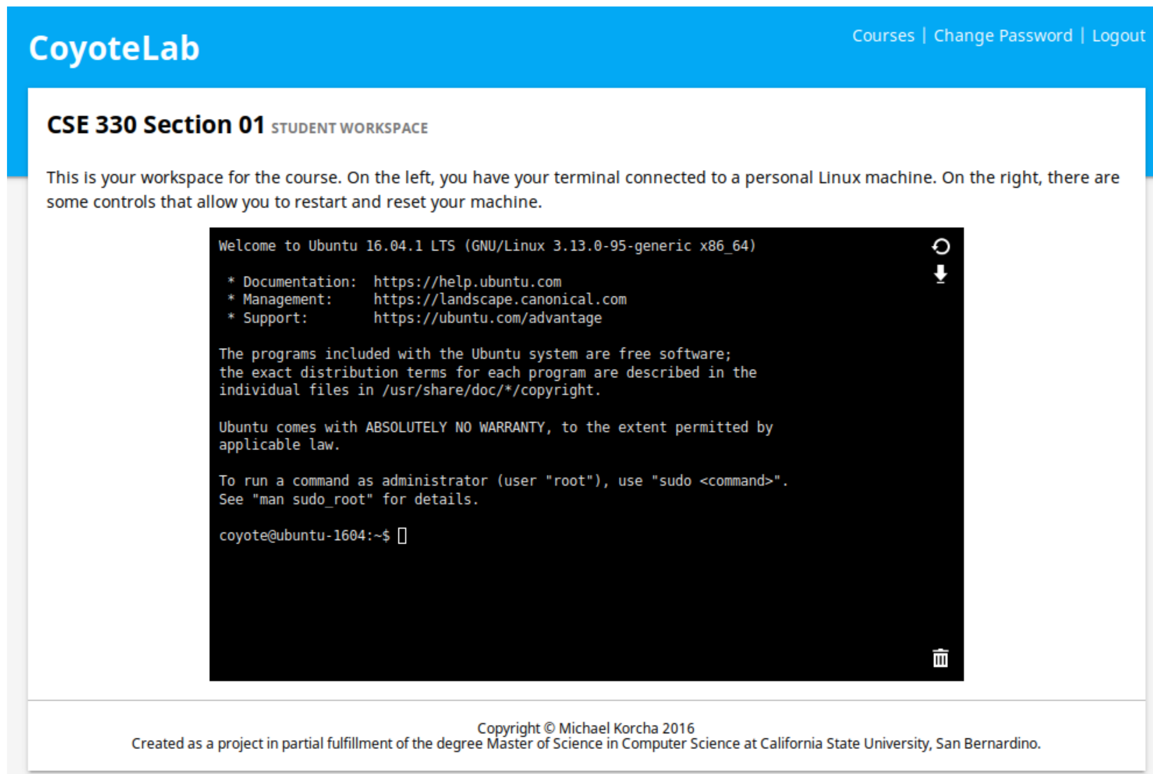


*Fig. 3.5:* Workspace view

When the workspace is loaded, the WebSocket connects to the web server to attempt to connect to the appropriate container. If a container doesn't exist, it will be created. Otherwise, the existing one is started. The container is then connected to the web-based terminal emulator, where it can be interacted with by the end user.

When the page is closed, the socket times out, or the connection is closed, the container is shut down to free resources on the server.

The workspace features three buttons to control the container. The top button refreshes the page, which will reboot the container. The second button is a shortcut

to download files from the container. The final button on the bottom will reset the container back to the state of the course's base image. An alert is shown when it's clicked to confirm that the user intends to perform this action, as it's irreversible. Instructors cannot reset a student's workspace, only their own.

# 4. SYSTEM ARCHITECTURE

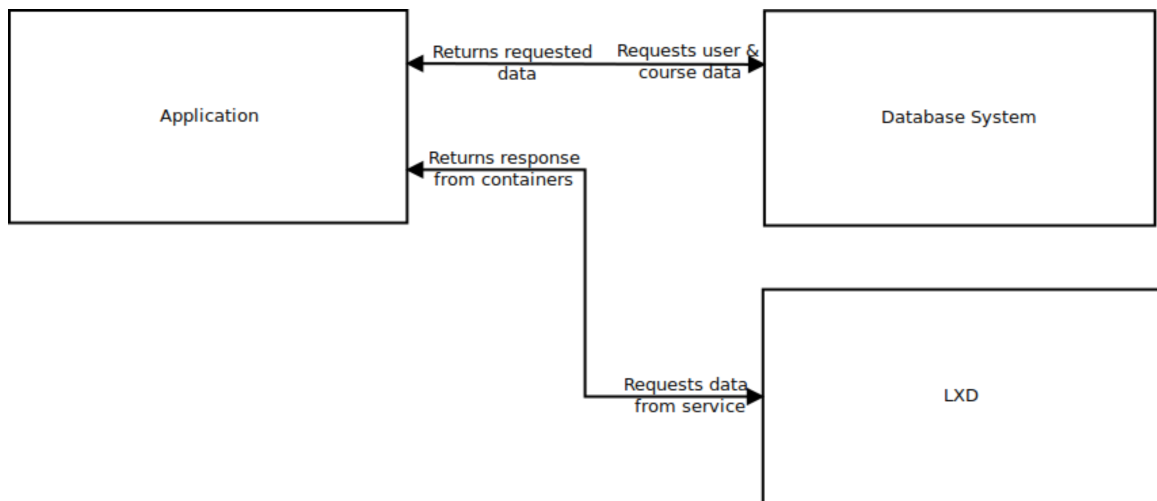


*Fig. 4.1:* Basic outline of system interactions

The system is built up using a custom file system layout and three different key services: the application, the database, and the LXD container system.

## *4.1 Application*

The application is built with the Python programming language and uses the Python interpreter to run [6]. The application is built with the Flask web framework, using gevent for network thread handling. Some additional libraries are used to add form validation, database interaction, and container control.

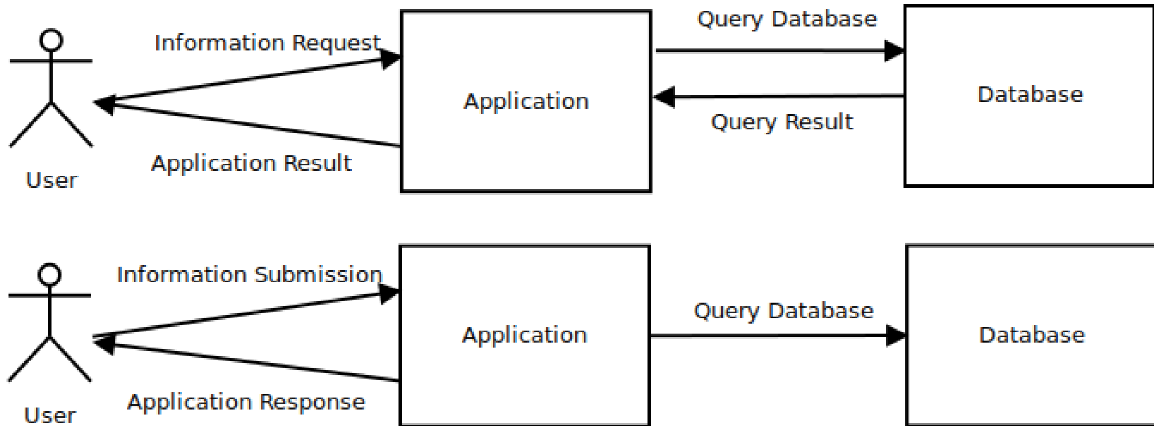


*Fig. 4.2:* Application to database interactions

The database is a core requirement of this project, as it is the central store for user and course information, and holds a table to keep track of container information. It is directly accessible to the application. Users are able to submit data to the application, which stores it in the database, and retrieve data from the application, fetching it from the database for the user.

The database interactivity is provided by the SQLAlchemy library, which allows for most relational database systems to be used interchangeably (with the help of migrations) [7]. For this project, the PostgreSQL database system was chosen.

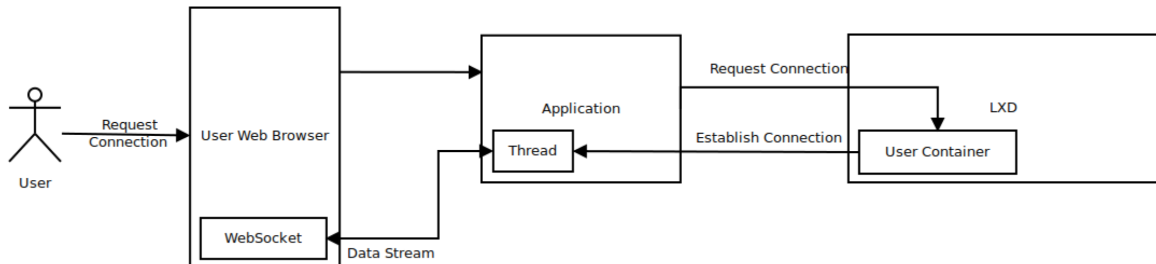


*Fig. 4.3:* Application to container system interactions

The LXD daemon is a system for managing Linux containers over a REST API, providing operations similar to those for full virtual machines [3]. The difference is that containers share a running Linux kernel with the hosting machine, rather than running their own kernel, and don't have any hardware emulation. As a result, containers start, stop, and are created much faster.

Containers are run out of directories that mock a root file system, and are isolated using cgroups and user namespaces (a kernel mechanism that isolates security-related identifiers such as file systems and users). From the container host, a container appears as it's own process with it's own subprocesses running in a specific directory. From within the container, it appears as it's own machine, with the directory it's installed in appearing as the root directory.

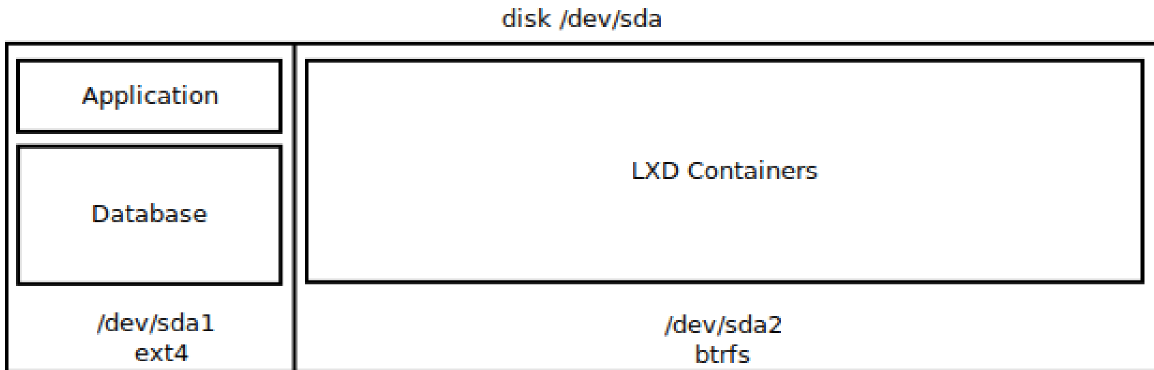


*Fig. 4.4:* Host disk partition layout

The host machine has two different file systems set up on it. One of them is for the main operations of the host machine, such as running the application and database, which runs on the default ext4 Linux file system. The other is dedicated exclusively for LXD container storage, running on Btrfs.

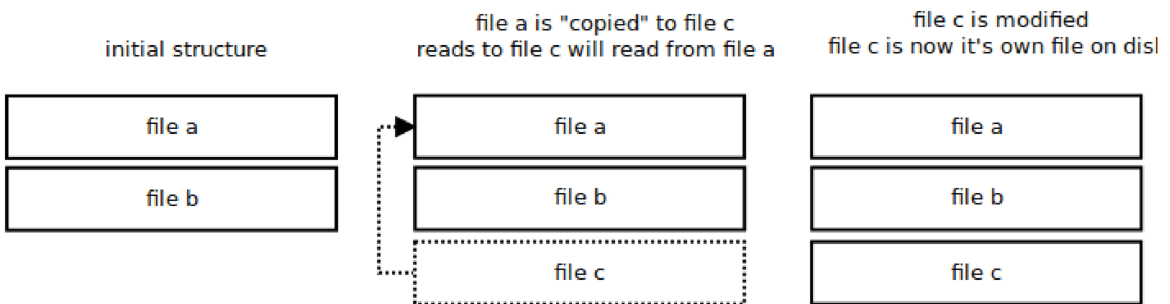


*Fig. 4.5:* Visual outline of copy-on-write files

Btrfs was chosen for container storage because it is a copy-on-write file system.

When a file is copied in a copy-on-write system, the newly created file redirects to the original file instead making a true copy. This only becomes it's own file when changes are made and saved to disk [9]. This saves disk space, as there is no need to keep several copies of identical files when unedited copies are made. This saves several gigabytes of disk space, as a container will typically be a few gigabytes of duplicate files otherwise.

# 5. IMPLEMENTATION

## *5.1 Database Layout*

The database schema consists of four tables, each representing a specific data object of the application.

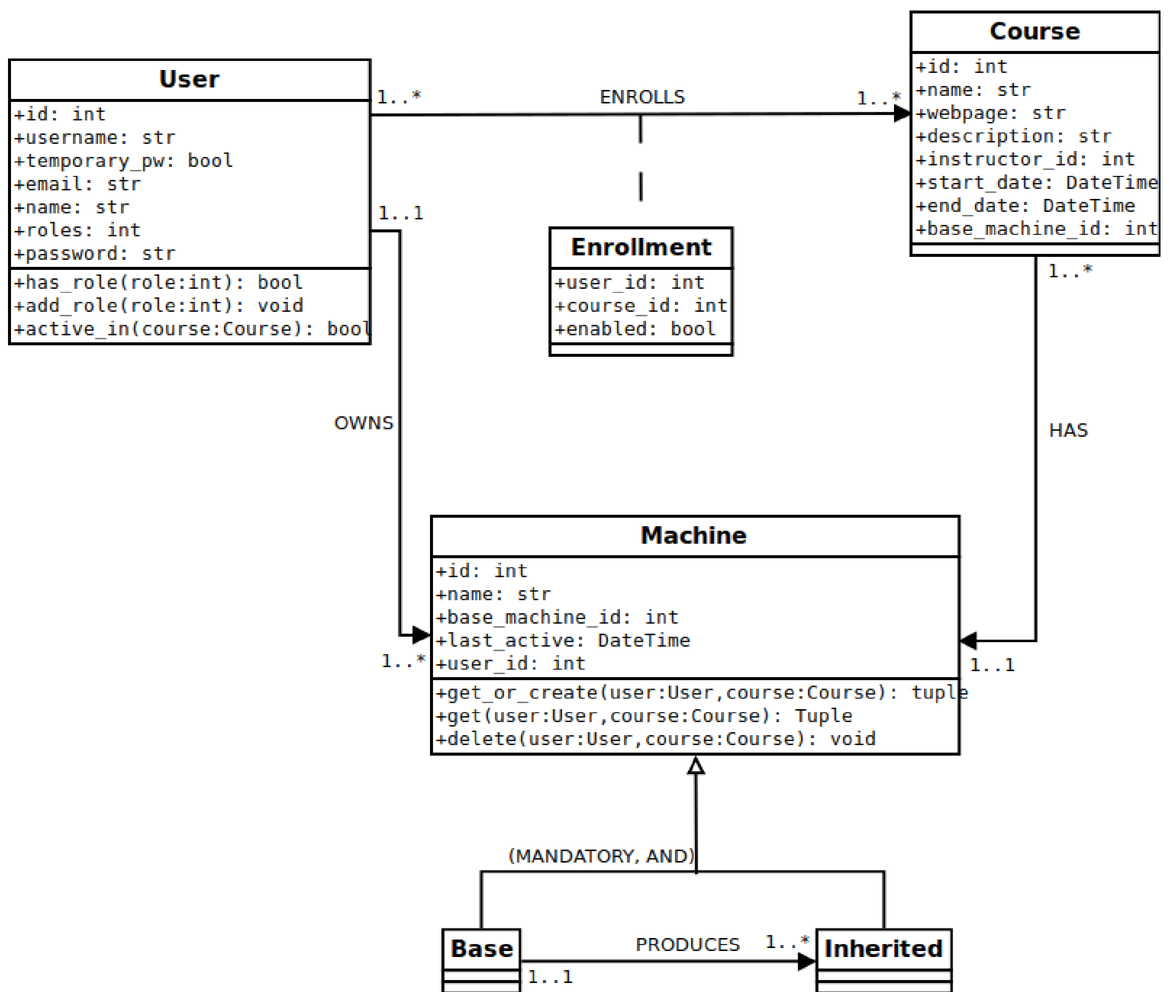


*Fig. 5.1:* Database table structure and relationships

| Field | Description |
|---|---|
| id | The auto-generated ID of the course |
| name | The name of the course |
| webpage | The URL to the course webpage |
| description | The course description |
| instructor_id | The ID of the user who is the instructor of the course |
| start_date | The date the course starts |
| end_date | The date the course ends |
| base_machine_id | The ID of the container which student machines will be cloned from |

*Fig. 5.2:* Course object attributes and methods

A Course is an object that describes course that is run by an instructor, which multiple students can be enrolled in. It is owned by a User (the course instructor), and can have multiple Users as students, via the Enrollment table. It also has a Machine object, which defines which container the initial container for the course will be based on. There are also fields for additional data: the webpage, course description, start date, and end date.

```
@hybrid_property
def password(self):
    '''
    Returns the password hash field
    '''
    return self._password

@password.setter
    def password(self, value):
    '''
    Sets the password to the hash of the provided value
    '''
    from ..util.auth import pwhash
    self._password = pwhash(value)
```

*Fig. 5.3:* Code for password getting and setting

| **Field** | **Description** |
|---|---|
| id | The auto-generated ID of the user |
| username | The user's login name |
| temporary_pw | Flag determining if the current password is temporary |
| email | The user's email address |
| name | The user's name |
| roles | The user's assigned roles |
| password | The user's hashed password |
| has_role(int) | Returns if the user has the specified role |
| add_role(int) | Adds the specified role to the user |
| active_in(Course) | Returns if the user is currently enrolled in the given course |

*Fig. 5.4:* User object attributes and methods

The User is a model of a user and it's attributes. There are two interesting fields of importance: the password field and the roles field. When added, the password is hashed using the bcrypt hashing algorithm, so that user passwords are not actually stored, and instead compared to a value representing it. The roles field is a bit field used instead of a separate role system in another table, due to the limited number of roles and simplicity of the system. The currently available roles are the roles of student and instructor.

| Field | Description |
| --- | --- |
| id | The auto-generated ID of the container |
| name | The name of the container in LXD |
| base_machine_id | The ID of the container which the referencing container originates from |
| last_active | The date that the container was last used |
| user_id | The user who is the owner of the container |
| get_or_create(User, Course) | Gets or creates a container for the specified user/course combination and returns it |
| get(User, Course) | Returns the container for the given user/course combination |
| delete(User, Course) | Deletes the container for the given user/course combination |

*Fig. 5.5:* Machine object attributes and methods

A Machine is a description of a container in LXD. The name field corresponds with the name of the container it's meant to reference which is handled by the LXD system. It has a relation to another Machine object, which is the base machine object which represents the container that it originated from. Each Machine is owned by a User.

The Enrollment object allows for a many-to-many relationship between User and Course objects. This relationship allows a User to be a member of multiple courses, and multiple courses to have multiple users enrolled. The enabled field determines if the user is actually enrolled - if it's set, the user is currently enrolled, otherwise the user has been dropped. This is set instead of deleting enrollment information, so if a student were to be re-added, their data would not be lost. This relationship is designed in this way so that a student can be part of multiple courses, with multiple instructors, and require only a single account for their academic career.

## 5.2 Access Control

Each user has one or more roles defined in the role field. The role field is a simple integer, with specific bits of the integer set up to either grant or deny a specific role. There are two roles defined: instructor and student. Checking a role is done by using a bit mask. This mask is put across the user's role using a bitwise AND operation, and if the result is not 0 then the user has the given role.

To verify appropriate access to resources protected to only certain roles, this is done before any other processing is done. In this application, it's done along certain application endpoints. A helper method called has_role was created on the User object to simplify the code.

```
def has_role(self, role):
    '''
    Checks if the user has a role
    '''
    return bool(self.roles & role)
```

*Fig. 5.6:* Code for check if a user has a role

As an example, for instructor interfaces, the role is checked for the authenticated user before any interface is allowed to load. If they don't have the appropriate roles, they are redirected away.

```
@blueprint.before_request
def filter():
    '''
    Verify that the user has permission to be here
    '''
    if not authenticated() or not session_user().has_role(ROLE_INSTRUCTOR):
        return redirect(url_for('auth.login'))
```

*Fig. 5.7:* Code for before-request permission check

Adding a role is similar to checking a role, just using a different operation. Instead of using the mask as a way to check using the bitwise AND operation, the bitwise OR operation is used to add the appropriate bit to the integer (see above figure). Another helper method was added to the user object to facilitate this.

```
def add_role(self, role):
    '''
    Adds a role to a user
    '''
    if self.roles:
        self.roles |= role
    else:
        self.roles = role
```

*Fig. 5.8:* Code for adding a role to a user

Other access controls need to be in place for course-related interfaces, to ensure that only members of a course (either instructor or student) have access to the course material and workspace. A function was written for each endpoint that has course-

related material, which retrieves the course with the given user parameters, and returns a page not found error to those not involved with the course.

```
def get_course(course_id):
    '''
    Returns a course from the given course_id if the course is taught by the
    currently authenticated user. Otherwise, the application will display a 404
    error
    '''
    course = Course.query.filter_by(id=course_id, instructor=session_user())\
                 .first()

    if course is None:
        abort(404)

    return course
```

*Fig. 5.9:* Code for retrieving a course for an instructor

```
def get_course(course_id):
    '''
    Returns a course from the given course ID if the student is enrolled in it,
    otherwise will abort
    '''
    course = Course.query.get(course_id)
    user = session_user()

    if course is None or not course in user.enrolled or not \
            user.active_in(course):
        abort(404)

    return course
```

*Fig. 5.10:* Code for retrieving a course for a student

In the workspaces, additional logic is required for instructors accessing their student's workspaces. The endpoints accept an optional student parameter, corresponding to the student whose workspace is being accessed. In this case, we first perform the check for course accessibility for the instructor. Afterwards, when retrieving the workspace, we check if the user is the instructor, and if not, pull in the student's information to load the workspace. By checking if the user is the instructor of the course first, we can be sure that the user has access to the workspaces already without adding a redundant check

```
def get_student(course, student_id):
    '''
    Returns the student if the instructor can access this student's work
    '''
    student = User.query.get(student_id)

    if student is None or course.instructor != session_user() or \
            not student.active_in(course):
        abort(404)

    return student
```

*Fig. 5.11:* Code for retrieving a student with access to a specific course

```
@blueprint.route('/<course_id>')
@blueprint.route('/<course_id>/<student_id>')
def workspace(course_id, student_id=None):
    '''
    The main workspace view
    '''
    course = get_course(course_id)
    student = get_student(course, student_id) if student_id else None

    return render_template('workspace.jinja', course=course, user=student)
```

*Fig. 5.12:* Code for view where an instuctor can access a student's or their own information with appropriate permission

## 5.3 Container Control

Container control is provided by the pylxd library, which uses the LXD REST API. In order to manage containers with the library, the LXD service first requires that anything using it authenticate itself to be trusted by the system.

```
def lxd_client():
    '''
    Returns an LXD client object to interact with containers. Returns None if
    a trusted connection cannot be established
    '''
    client = LXD(endpoint=current_app.config['LXD_ADDRESS'],
                 cert=(os.path.join(os.getcwd(), 'cert.crt'),
                       os.path.join(os.getcwd(), 'cert.key')),
                 verify=False)
    client.authenticate(current_app.config['LXD_TRUST_PASSWORD'])

    return client if client.trusted else None
```

*Fig. 5.13:* Code for establishing a trusted LXD connection

Once authenticated, containers can be accessed and modified using the provided client functions. Utility methods were created on the Machine object in order to create machines that don't yet exist, or get a reference to the container if it does.

```
@staticmethod
def get_or_create(user, course):
    '''
    Creates a new container for the given user/course combination and
    returns a tuple of the form (lxd_container, model)
    '''
    if not user.active_in(course) and course.instructor != user:
        return None

    from ..util.lxd import lxd_client
    lxd = lxd_client()

    # ...

    container = lxd.containers.create({
        'name': name,
        'source': {
            'type': 'copy',
            'source': course.base_machine.name
        },
        'config': {
            'limits.cpu': current_app.config['LXD_LIMIT_CPU'],
            'limits.memory': current_app.config['LXD_LIMIT_MEMORY']
        }}, wait=True)

    # ...

    return container, machine
```

*Fig. 5.14:* Code for creating a container

```
@staticmethod
def get(user, course):
    '''
    Gets a container/model pair for the given user/course combination. or
    None if it doesn't exist
    '''
    if not user.active_in(course) and course.instructor != user:
        return None

    from ..util.lxd import lxd_client
    lxd = lxd_client()

    name = current_app.config['USER_CONTAINER_NAME']\
             .format(course_id=course.id, user_id=user.id)

    try:
        container = lxd.containers.get(name)
        return container, Machine.query.filter_by(name=name).first()
    except LXDAPIException:
        return None
```

*Fig. 5.15:* Code for retrieving a container

Once a container is retrieved, pylxd makes it trivial to manage container state. This is done using the start and stop methods of the LXD container object, which are used in the SSH-WebSocket bridge.

```
def _bridge(self, channel):
    """ Full-duplex bridge between a websocket and a SSH channel """
    channel.setblocking(False)
    channel.settimeout(0.0)
    self._tasks = [
        gevent.spawn(self._forward_inbound, channel),
        gevent.spawn(self._forward_outbound, channel)
    ]
    gevent.joinall(self._tasks)
```

*Fig. 5.16:* Code for bridging an SSH connection and a WebSocket (via wssh) [12]

The SSH-WebSocket bridging functionality is provided by a library called wssh using it's bridge function. This function takes the WebSocket on connection, passes the data to another library called paramiko, which handles passing SSH messages between the application and the SSH daemon running in the container. This bridge creates two threads - one for forwarding inbound traffic and one for forwarding outbound traffic.

```
def _forward_inbound(self, channel):
    """ Forward inbound traffic (websockets -> ssh) """
    try:
        while True:
            data = self._websocket.receive()
            if not data:
                return
            data = json.loads(str(data))
            if 'resize' in data:
                channel.resize_pty(
                    data['resize'].get('width', 80),
                    data['resize'].get('height', 24))
            if 'data' in data:
                channel.send(data['data'])
    finally:
        self.close()
```

*Fig. 5.17:* Code for forwarding packets from the WebSocket to the SSH server (via wssh) [12]

When inbound traffic to the container comes in, the thread passes the data coming from the WebSocket to the SSH daemon. The data received comes in a JSON format.

```
def _forward_outbound(self, channel):
    """ Forward outbound traffic (ssh -> websockets) """
    try:
        while True:
            wait_read(channel.fileno())
            data = channel.recv(1024)
            if not len(data):
                return
            self._websocket.send(json.dumps({'data': data}))
    finally:
        self.close()
```

*Fig. 5.18:* Code for forwarding packets from the SSH server to the WebSocket (via wssh) [12]

When the SSH daemon sends to the client, the outbound thread formats the data in a JSON format and sends it back to the WebSocket. In the browser, this response is decoded and the result handled by the web terminal.

```
# start the machine if need be and wait for the network to come online,
# then get the address to connect to
container.start(wait=True)

while len(container.state().network['eth0']['addresses']) < 2:
    time.sleep(1)

for addr in container.state().network['eth0']['addresses']:
    if addr['family'] == 'inet':
        address = addr['address']

# ...

bridge = WSSHBridge(request.environ['wsgi.websocket'])

try:
    bridge.open(hostname=address, username='coyote', password='coyote')
except:
    request.environ['wsgi.websocket'].close()
    container.stop()
    return str()

bridge.shell()

request.environ['wsgi.websocket'].close()

# once it's closed, we want to stop the machine to get back the resources
container.stop()
```

*Fig. 5.19:* Code for running and connecting to the container

The workspace socket brings this all together, by first creating or grabbing the container that will be used, starting it, and waiting for the network to start. Once it's ready, the bridge is made to the SSH daemon in the container. As soon as the socket connection is terminated, the container is shut down to free system resources.

# 6. CONCLUSION

This project aims to provide an easy way for instructors and students to work on class assignments in their computing courses. It provides an easy way for a student to log in and get to a container, that can then be accessed by their instructor. This makes it easier for both parties as a common set of tools can be used, which keeps discrepancies in results to a minimum, while also saving time for both parties by removing the need to set up and install a machine of their own.

## *6.1 Future Work*

Several improvements could be made to make this project work in a more general case to allow for more courses to be able to use it.

- A VNC client could be installed to a container, and a VNC JavaScript implementation could be used for graphical programs.
- A system could be put in place for an instructor to make a more tailored base container for their particular course needs. Additionally, more base container options could be added, such as CentOS or OpenSUSE, instead of just Ubuntu, as other Linux operating systems provide different packages and security features.
- Mechanisms could be implemented for users to have multiple containers, to do parallel programming work. This could extend to use the libvirt software, which can manage both LXD containers and virtual machines, to use VM-assigned hardware for GPU work.

- A grouping system could be created to allow multiple users to share a common container for team project work.

The potential for this projects continuation is great, and can provide many improvements to the way computing is taught.

# REFERENCES

[1] Anand Chitipothu. Ten reasons why you should prefer postgresql to mysql. Presented at Root Conf 2015. [Online]. Viewed 2016 November. Available: http://www.slideshare.net/anandology/ten-reasons-to-prefer-postgresql-to-mysql.

[2] Flask Documentation. (undated). [Online]. Viewed 2016 November. Available: http://flask.pocoo.org/docs/0.11/.

[3] LXD Official Documentation. (undated). [Online]. Viewed 2016 November. Available: https://help.ubuntu.com/lts/serverguide/lxd.html.

[4] MDN WebSockets Documentation. (undated). [Online]. Viewed 2016 October. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets$_{A}PI$.

[5] Nginx Documentation. (undated). [Online]. Viewed 2016 October. Available: https://nginx.org/en/docs/.

[6] Python Documentation. (undated). [Online]. Viewed 2016 November. Available: https://docs.python.org/2/.

[7] SQLAlchemy Documentation. (undated). [Online]. Viewed 2016 October. Available: http://docs.sqlalchemy.org/en/latest/index.html.

[8] WTForms Documentation. (undated). [Online]. Viewed 2016 October. Available: http://dev.mysql.com/doc/refman/5.5/en/.

[9] Neeta Garimella. Understanding and exploiting snapshot technology for data protection, part 1: Snapshot technology overview. [Online]. Viewed 2016 October. Available: https://www.ibm.com/developerworks/tivoli/library/t-snaptsm1/index.html.

[10] gevent Documentation. (undated). [Online]. Viewed 2016 October. Available: http://www.gevent.org/contents.html.

[11] PostgreSQL 9.6 Maunal. (undated). [Online]. Viewed 2016 November. Available: https://www.postgresql.org/docs/9.6/static/index.html.

[12] wssh GitHub Repository. (undated). [Online]. Viewed 2016 October. Available: https://github.com/aluzzardi/wssh/.