

University POLITEHNICA of Bucharest

Faculty of Automatic Control and Computers,  
Computer Science and Engineering Department



## BACHELOR THESIS

# Automated Assignment Grading Using Linux Containers

**Scientific Adviser:**

Șl.dr.ing. Răzvan Deaconescu

**Author:**

Andrei Dorian Duma

Bucharest, 2016



# Abstract

Coding assignments are essential in the training of computer science students. From a teacher's perspective, manually grading hundreds of submissions is a difficult, time-consuming task. Lx-checker is a platform that allows teachers to create and configure assignments. Students submit their code for grading and receive the results as soon as the automated tests are run. This saves time for teachers and gives students the chance to know in real time how well their solution behaves. In this paper, we describe the motivation behind the choices we've made and provide details about the systems's architecture and implementation.

# Contents

<b>Abstract</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Objectives	1
1.1.1 Ease of Use	2
1.1.2 Performance	2
1.2 Use Cases	2
1.2.1 Automated Homework Grader	2
1.2.2 Online Judge	3
<b>2 State of the Art and Related Work</b>	<b>4</b>
2.1 Automated Homework Grader	5
2.2 Online Judge	5
<b>3 Architectural Overview</b>	<b>6</b>
3.1 Architectural Overview	6
3.2 The Lxchecker Binary	7
3.3 Docker Swarm	7
3.4 MongoDB Server	8
<b>4 Implementation</b>	<b>9</b>
4.1 Technologies	9
4.1.1 Docker	9
4.1.2 MongoDB	10
4.1.3 Go	10
4.2 Scheduler Module	11
4.2.1 The API	11
4.2.2 Execution Flow	12
4.3 Storage Interface Module	13
4.3.1 The API	14
4.3.2 Implementation Example	14
4.4 Web Module	15
4.4.1 URL Routing and Handlers	15
4.4.2 Templates	16
4.4.3 Authentication	16
<b>5 Deploying and Using Lxchecker</b>	<b>18</b>
5.1 Deploying Lxchecker	18
5.1.1 Installing Lxchecker	18
5.1.2 Updating Lxchecker	19
5.2 Using Lxchecker	19
5.2.1 Authentication	19

## CONTENTS

iv

---

5.2.2	Working With Subjects . . . . .	20
5.2.3	Managing Assignments . . . . .	20
5.2.4	Submission Upload . . . . .	21
5.2.5	Submission Grading . . . . .	22
<b>6</b>	<b>Results and Future Work</b>	<b>24</b>
6.1	Current State of the Project . . . . .	24
6.2	Future Work . . . . .	25
<b>7</b>	<b>Conclusion</b>	<b>26</b>

# List of Figures

1.1	conceptual view of automated grading . . . . .	2
3.1	architectural overview of Lxchecker . . . . .	6
3.2	Docker hosts pulling images from the Registry . . . . .	8
4.1	execution steps performed by the scheduler . . . . .	12
4.2	database models and schema . . . . .	13
4.3	URL structure of Lxchecker . . . . .	15
4.4	middleware composition . . . . .	17
5.1	logging in . . . . .	19
5.2	creating a new account . . . . .	20
5.3	available subjects list and <i>create subject</i> form . . . . .	20
5.4	subject page for the <i>Operating Systems</i> course . . . . .	21
5.5	<i>Virtual Memory</i> assignment page . . . . .	22
5.6	waiting for the submission to be evaluated . . . . .	22
5.7	submission was tested and score is displayed (full logs not included) . . . . .	23
5.8	grading form for teachers . . . . .	23
5.9	submission after grading by teacher . . . . .	23

# Chapter 1

## Introduction

Practical coding assignments play an important role in the educational development of Computer Science students. Homework and projects allow future engineers to internalize theoretical concepts and develop effective habits and techniques. Even more, technologies such as programming languages or frameworks can only be learned by practice.

In contrast to theoretical lectures and exams, which can be delivered to dozens of students simultaneously, coding assignments require particular attention to each individual. Testing a code submission often involves downloading source files from an email server or a web platform, compiling them locally into an executable, running a suite of tests to produce a numerical score, potentially deducting penalty points due to missed deadlines and eventually filling out a spreadsheet with the final grade. From a technical perspective, different testing environments might produce different results. For the teacher, it is a highly repetitive and time-consuming process. For the student, it means keeping track of different email addresses or submission procedures, deadlines and other policies. Moreover, this approach doesn't allow the student to know in real time how his solution behaves against the tests. One might argue that tests could be made available to all students, but a secret testing mechanism might be desired. Besides that, an update requires all students to redownload the suite. Manually grading coding assignments is difficult and inefficient.

Let's consider a different approach by telling the story of Mihai, first year student in the Ideal Computer Science University (ICSU). He receives his first Computer Programming assignment two weeks into the semester. The homework document asks him to submit his solution in C to <https://lxchecker.icsu.edu>. After creating an account, he visits the Computer Programming page, selects his code archive using a file dialogue and clicks 'submit'. 10 seconds later, he sees the test suite awarded him 90 out of 90 points, he is happy and goes to sleep. One week later, after the deadline had passed, the teacher grades all submissions for coding style and general quality of the code. Potential deadline penalties are automatically subtracted and the final grade is computed. The end. The action flow is simple and intuitive for both students and teachers. All submissions are graded in hermetic and identical environments, guaranteeing reliability, security and fairness. A high-level picture of how such a system would behave is presented in [Figure 1.1](#).

### 1.1 Objectives

The end goal of this project is to provide a ready-to-use software solution for automatically grading programming assignments or challenges. The two main metrics for assessing the success of the project are the platform's ease of use and performance. What we understand by these terms will be explained in the following subsections.

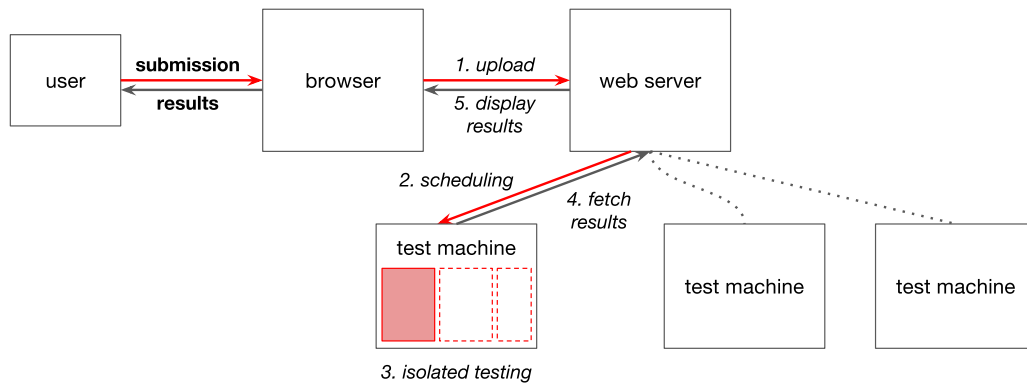


Figure 1.1: conceptual view of automated grading

### 1.1.1 Ease of Use

The platform should be easy to install and maintain by the faculty staff. Going from several fresh Linux hosts to a functional lxchecker cluster should be a matter of installing a few packages and doing minimal configuration, all by following clear steps in the documentation of the project.

Lxchecker should allow global admins to easily create subjects and assign privileged roles to teachers. Teachers should be able to define and configure assignments in a straightforward manner. Uploading submissions and checking results should be quick and intuitive for students.

### 1.1.2 Performance

While the ease of use is a rather subjective quality, performance can be quantified effectively. In the case of lxchecker, we expect the platform to allow at least twice as many simultaneous submissions as the total number of users. The efficiency (actual testing time of submissions to total time ratio) should exceed 90%. Under high load, the waiting time is expected to be inversely proportional to the number of cores available in the cluster.

## 1.2 Use Cases

The project attempts to be flexible enough to be used in a variety of scenarios. The following subsections describe two common use cases.

### 1.2.1 Automated Homework Grader

The scenario we had in mind when developing lxchecker is that of a grading system to be used in universities for week-to-week assignments. Various features were developed for this purpose, such as:

- extensive environment customization (compilers, frameworks, test data, networking support etc.)
- automated grading by tests and manual grading by teachers
- support for soft and hard deadlines



- multi-level permission system (admin, teacher & student roles)

In this regard, lxchecker was inspired by Vmchecker [3], whose features and limitations are described in [Chapter 2](#).

### 1.2.2 Online Judge

Lxchecker can also be used as an online judge for programming contests (e.g. ACM-ICPC<sup>1</sup>). Its flexibility allows for any number of programming languages, time and memory constraints.

---

<sup>1</sup><https://icpc.baylor.edu/>

## Chapter 2

# State of the Art and Related Work

The cloud era has brought numerous advancements in areas such as multitenancy, resource sharing and application sandboxing. Virtualization provides a layer of abstraction that allows developers to build software that runs on any hardware platform. Hypervisors like VMware<sup>1</sup>, VirtualBox<sup>2</sup> or QEMU<sup>3</sup> can run fully separated instances of operating systems on the same host. This makes it possible for multiple applications, potentially untrusted and from different vendors, to be executed simultaneously. Resource constraints (CPU, memory, network, hard-drive etc.) can be enforced by the virtualization platform.

More recently, operating-system-level virtualization allows the existence of multiple isolated user-space instances. Called *containers*, these instances behave like real hosts for their users (applications running inside them). Similar to the standard `chroot`<sup>4</sup> mechanism, containers provide a completely isolated view of the operating environment, including processes trees, mounted file systems, users and networking. In contrast to other virtualization technologies, there is little to no overhead in starting or destroying a container. Isolated applications do not suffer slowdowns, since they are neither subjected to emulation, nor run in a virtual machine. Instead, programs use the operating system's normal system call interface.

Docker<sup>5</sup> is an open-source project that provides an additional layer of abstraction and automation of operating-system-level virtualization on Linux. "Docker containers wrap a piece of software in a complete file system that contains everything needed to run: code, runtime, system tools, system libraries – anything that can be installed on a server. This guarantees that the software will always run the same, regardless of its environment".<sup>6</sup> Containers isolate applications from one another and the underlying infrastructure, providing a lightweight environment that can be quickly instantiated or disposed of.

In the following sections, we are going to discuss available solutions in what regards the two use cases proposed in [Section 1.2](#): automated homework grader and online judge.

---

<sup>1</sup>VMware hypervisor, [vmware.com](http://vmware.com)

<sup>2</sup>VirtualBox hypervisor, [virtualbox.org](http://virtualbox.org)

<sup>3</sup>QEMU, hardware virtualization hypervisor, [qemu.org](http://qemu.org)

<sup>4</sup>Chroot mechanism, [en.wikipedia.org/wiki/Chroot](http://en.wikipedia.org/wiki/Chroot)

<sup>5</sup>Docker homepage, [docker.com](http://docker.com)

<sup>6</sup>What is Docker?, [docker.com/what-docker](http://docker.com/what-docker)

## 2.1 Automated Homework Grader

Despite the advantages of automated assignment grading in academic institutions, few universities have implemented such a system. Initial attempts included the TRY[5] system and the *Scheme-Robo*[4] system implemented at the Helsinki University of Technology for automated assessment of exercises in the Scheme functional language [2]. At the School of Computing of the National University of Singapore, a system called *Online Judge* was successfully employed for several undergraduate courses, proving the effectiveness of programmatic grading [2]. The most advanced of these systems, the *Online Judge*, can be considered primitive given the current advancements in sandboxing technologies – no more than basic `chroot` isolation was used and supported languages only included C, C++ and Java.

A modern approach to automated grading was proposed by Valentin Goşu[3] at the Politehnica University of Bucharest in 2012. As of 2016, the system, called *Vmchecker*, is widely used at the Faculty of Computer Science. It offers support for deadlines, numerical scoring and textual feedback, submission persistence etc. Above all, its popularity arises from its flexibility – *Vmchecker* allows teachers to define a fully customized testing environment by use of virtual machines. Unfortunately, the inherent performance penalty imposed by full virtualization is reflected in slow grading times and, implicitly, prolonged waiting queues for students. Another weak point is its unreliability – the system has proved to be fragile in a number of circumstances.

*Lxchecker* attempts to learn from *Vmchecker*, preserving its flexibility and features while improving its efficiency and dependability. The former is enhanced by switching from virtual machines to Docker containers, while the latter is increased by using Go<sup>1</sup>, a strongly-typed programming language.

## 2.2 Online Judge

In the world of online judges for programming contests, many solutions exist. Most likely, the reason is the reduced complexity of this use case. Usually, a contest-level grading system takes in a singular source file, compiles the code and runs the executable with multiple inputs. Besides compilers and basic sandboxing, not much is needed. Examples of online judges include *Spoj*<sup>2</sup>, *infoarena.ro*<sup>3</sup> and *HackerRank*<sup>4</sup>.

Hiring tool *remoteinterview.io*<sup>5</sup> uses Docker images for packaging its various compiler setups. In addition to the simplified management of dependencies, Docker containers allow code submissions to execute in entirely separate environments, isolating users from one another and protecting the host against malicious attacks.

*Lxchecker* uses Docker to provide an extended online judge functionality by allowing arbitrary uploads and boundless flexibility in specifying test environments (networking, unrestricted file system, complex interaction with the process under test etc.).

---

<sup>1</sup>The Go programming language, [golang.org](http://golang.org)

<sup>2</sup>Sphere online judge, [spoj.com](http://spoj.com)

<sup>3</sup>Infoarena, [infoarena.ro](http://infoarena.ro)

<sup>4</sup>HackerRank, [hackerrank.com](http://hackerrank.com)

<sup>5</sup>remoteinterview.io, [remoteinterview.io](http://remoteinterview.io)

## Chapter 3

# Architectural Overview

In this chapter we will describe the high-level structure of Lxchecker. We will present the components and highlight the interactions between them. The inner workings of individual subsystems will be detailed in [Chapter 4](#).

### 3.1 Architectural Overview

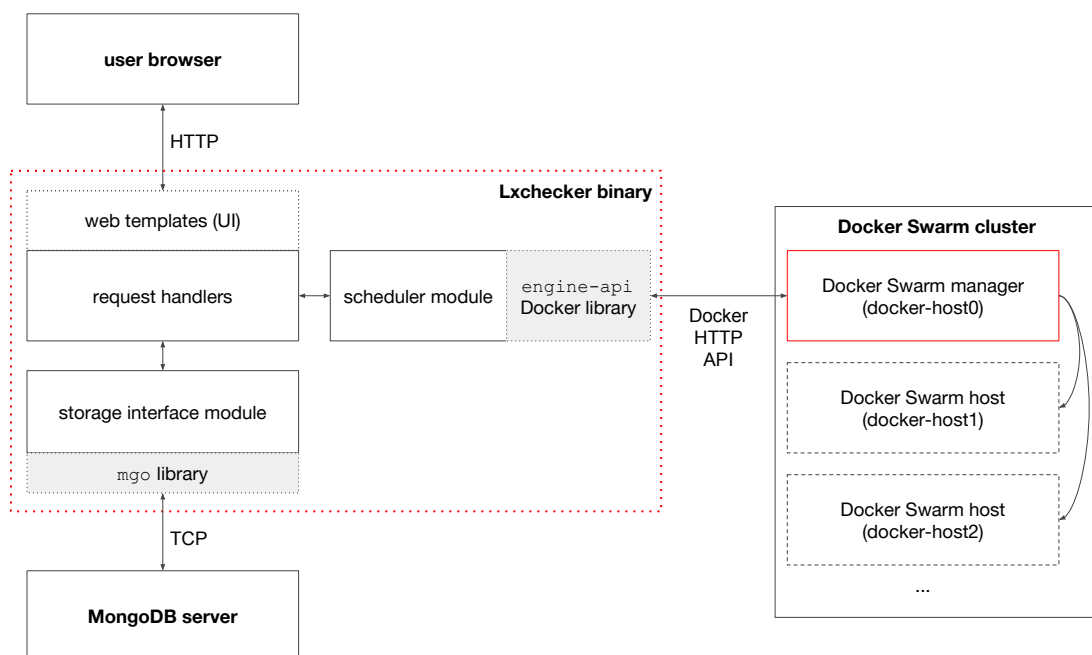


Figure 3.1: architectural overview of Lxchecker

As it is shown in [Figure 3.1](#), the Lxchecker system consists of three components: the Lxchecker binary containing the actual code, a Docker Swarm<sup>1</sup> cluster for testing uploaded submissions and a MongoDB<sup>2</sup> database server. All of them are separate processes, residing on possibly different

<sup>1</sup> Docker Swarm, [docs.docker.com/engine/swarm](https://docs.docker.com/engine/swarm)

<sup>2</sup> MongoDB, [mongodb.com](https://mongodb.com)

hosts and talking to each other over the network. Each of these is described from a high-level perspective in the following sections.

## 3.2 The Lxchecker Binary

All of Lxchecker's code is packaged into a single binary, which is meant to run on a single host and listen for HTTP requests on port 80. This binary is responsible for serving the web interface, handling all the logic and managing the other two components, the database and the Docker cluster. It is comprised of **three modules**: a web module (which implements all user interaction), a scheduler (which runs submissions and fetches results) and a database connector module (providing a higher-level API to the MongoDB collections). Each of these modules is explained in the following paragraphs.

The **web module** handles HTTP requests from the user's browser. On the client side, it serves HTML templates, processes form submits and manages authentication using cookies. Deeper down in the request handlers, the web module deals with data persistence and triggers submission testing. Basically, the web module contains what in the **MVC**<sup>1</sup> terminology would be called **views** and **controllers**. It uses the other modules as dependencies.

Interactions with the Docker Swarm cluster are managed by the **scheduler module**, which provides a simple API for submitting assignments and fetching results. This separates the complicated Docker-specific logic from the web module code. The API consists of a single blocking method, whose most important arguments are **the code to be tested** and **the environment** to use for evaluation. The result contains the **exit code** (to determine whether the testing was successful) and the **evaluation logs**, from which the score can be derived. This module uses the `engine-api`<sup>2</sup> Go library for interacting with the Docker hosts.

The **storage interface module** acts as an **ORM**<sup>3</sup> wrapper around the database. One reason is to simplify database reads and writes in the web module by providing a high-level API. Another motive is to abstract the actual persistence technology (whether it's NoSQL<sup>4</sup> database such as MongoDB or a relational one). Since the upper module is not aware of the implementation details, a future change to a different storage schema can be easily achieved. In **MVC** terms, this module contains database **models**, in-language representations of serializable data.

## 3.3 Docker Swarm

The Docker platform, together with its Swarm setup, is an essential component of the Lxchecker system. When running on a single host, the Docker daemon can be used to schedule a job remotely, specifying at the same time its environment, parameters, resource constraints etc. In conjunction with Swarm, Docker becomes even more powerful. It can transparently turn a cluster of Docker machines into a virtual host controllable by the very same API. This allows theoretically infinite horizontal scaling with no alteration in the code.

In the world of Docker, containers are created from images. An image is a layered file system<sup>5</sup> based on a Linux setup (such as a vanilla Ubuntu distribution). The common and recommended way to define and build an image is by using a `Dockerfile`<sup>6</sup>, which employs a declarative syntax

<sup>1</sup> Model-view-controller, [en.wikipedia.org/wiki/Model-view-controller](https://en.wikipedia.org/wiki/Model-view-controller)

<sup>2</sup> `engine-api`, a Go client library for Docker, [godoc.org/github.com/docker/engine-api](https://godoc.org/github.com/docker/engine-api)

<sup>3</sup> Object-relational mapping, [en.wikipedia.org/wiki/Object-relational\\_mapping](https://en.wikipedia.org/wiki/Object-relational_mapping)

<sup>4</sup> NoSQL Databases Explained, [mongodb.com/nosql-explained](https://mongodb.com/nosql-explained)

<sup>5</sup> UnionFS, [filesystems.org/project-unionfs.html](https://filesystems.org/project-unionfs.html)

<sup>6</sup> Dockerfile reference, [docs.docker.com/engine/reference/builder](https://docs.docker.com/engine/reference/builder)

to specify building steps (such as installing a package or running a command). After building, images are stored in the Docker Registry<sup>1</sup>. When starting a container, the Docker daemon first fetches the specified image from the Registry (if not cached). Figure 3.2 shows two Docker hosts pulling images from the Registry for creating some containers.

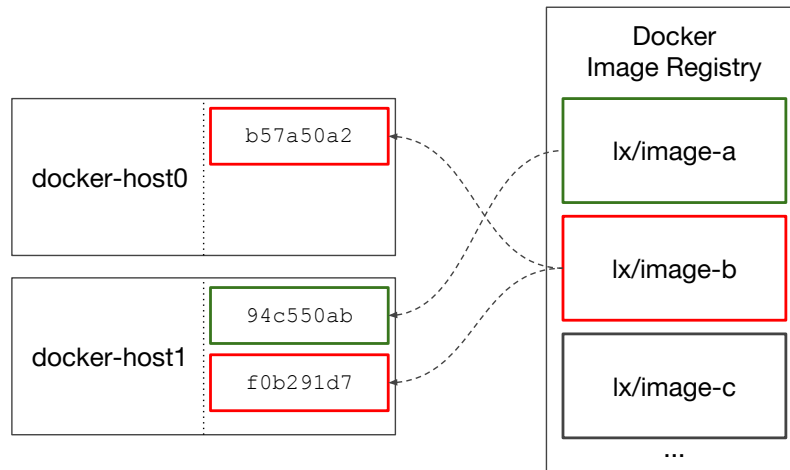


Figure 3.2: Docker hosts pulling images from the Registry

For Lxchecker, we are using `Dockerfiles` to define **per-assignment test environments**. After building, we push the resulting images to the Docker Registry. At evaluation time, when we pass to Docker an image name such as `lxchecker/ml_assignment_3`, the daemon knows what image to pull from the Registry and creates a container with the appropriate testing environment for the given submission.

## 3.4 MongoDB Server

The database server runs in a different process, possibly on a different host. Replication may be used for data redundancy. The Lxchecker binary talks to MongoDB over TCP and stores objects such as subject or assignment configurations, submissions, users and granted permissions.

<sup>1</sup>Docker Registry, [docs.docker.com/registry](https://docs.docker.com/registry)

## Chapter 4

# Implementation

In this chapter, we will show how Lxchecker is implemented. Firstly, [Section 4.1](#) will describe the implementation technologies and explain why we use them. We then proceed to detail the inner workings of the three modules of the application. In [Section 4.2](#), we will see how the interaction with Docker is approached. [Section 4.3](#) presents the API exposed by the storage interface module. After covering these dependencies, we will go over the web module in [Section 4.4](#).

### 4.1 Technologies

The quality of a software implementation greatly depends on the utilized tools. That is why we put a lot of thought into finding the suitable technologies for building Lxchecker. In this section we will first explain the general decision criteria we used. Each subsection will then cover a technology in detail, presenting its advantages, drawbacks and considered alternatives.

When contemplating an option, we analyzed the technology from the following perspectives. While each of them is important, we deemed some aspects more significant in the context of Lxchecker. In descending order of relevance:

- **features:** we looked at what functionality we need and how well the particular technology provides them.
- **development and maintenance costs:** we wanted tools that make development easy, but also allow comfortable future maintenance. We searched for technologies that are easy to understand and use.
- **future support:** since Lxchecker aims to be a long-term solution for automated assignment grading, good backing from industry and community for its software dependencies is important.
- **performance:** both user experience and resource consumption are essential for Lxchecker.

#### 4.1.1 Docker

We already discussed the advantages of containers over virtual machines in [Chapter 2](#). Although using LXC (Linux Containers) directly was possible, our needs were better met by modern container engines. Among them, open-source Docker is by far the most popular. It is widely supported

in all major cloud computing platforms (such as Amazon AWS<sup>1</sup> or Google Cloud Platform<sup>2</sup>), has advanced orchestration and provisioning<sup>3</sup> tools, features native clustering<sup>4</sup>, and comes with a cloud-based image registry service<sup>5</sup>. Docker has the most comprehensive set of features and is battle-tested by the industry, from which it has secured solid support.

An alternative to Docker is the more recent `rkt`<sup>6</sup> from CoreOS, which promises superior security and composability. Given `rkt`'s rather short presence on the market and because security is not crucial to our use case, we preferred the widely supported Docker.

### 4.1.2 MongoDB

For the persistence layer of Lxchecker, we chose to use MongoDB, a NoSQL database with dynamic schemas and intuitive JSON-like<sup>7</sup> document structure. MongoDB is open-source and provides drivers for a variety of programming languages<sup>8</sup>. It has good performance<sup>[1]</sup> and built-in replication (useful for ensuring data redundancy). MongoDB has comprehensive documentation and strong community support.

### 4.1.3 Go

To glue dependencies together and implement the actual functionality, we decided to use Go<sup>9</sup> programming language. Modern yet widely adopted, Go is a compiled, statically typed language with garbage collection and CSP<sup>10</sup>-style concurrency features. It is appropriate for developing servers, balancing the performance and rigor of strongly-typed, compiled languages with the brevity of dynamically-typed languages. Other advantages include an extensive standard library and comprehensive documentation. In the recent years, Go has gained a lot of community traction, which, combined with the active support from companies like Google, suggests a stable future.

The two main alternatives considered were Java and Python. Java, although mature and efficient, promised a longer development time because of its verbosity. Python is at the other end of the brevity spectrum, producing simple and concise code, but its dynamic typing conflicted with the robustness we desired.

For working with Docker, we're using the official client library `engine-api`<sup>11</sup>. It offers programmatic access to all of Docker's functionality as it available using the `docker` command line tool.

For MongoDB, we found the `mgo`<sup>12</sup> library very powerful. One feature that stands out is the ability to serialize and deserialize Go structs transparently, making the library almost as easy to use as an ORM framework.

In the web module, we are making heavy use of the standard library. All frontend templating is done using the `html/template` package and all request handling is managed by the standard `http` package. We decided against using a higher-lever web library to make future maintenance easier. Besides that, the standard library is powerful enough. We did use an external library for routing

<sup>1</sup> Amazon AWS Cloud, [aws.amazon.com](https://aws.amazon.com)

<sup>2</sup> Google Cloud Platform, [cloud.google.com](https://cloud.google.com)

<sup>3</sup> Docker Compose, [docs.docker.com/compose/overview](https://docs.docker.com/compose/overview)

<sup>4</sup> Docker Swarm, [docs.docker.com/swarm/overview](https://docs.docker.com/swarm/overview)

<sup>5</sup> Docker Hub, [docs.docker.com/docker-hub](https://docs.docker.com/docker-hub)

<sup>6</sup> CoreOS's `rkt`, [coreos.com/rkt](https://coreos.com/rkt)

<sup>7</sup> BSON, [bsonspec.org](https://bsonspec.org)

<sup>8</sup> MongoDB Drivers, [docs.mongodb.com/ecosystem/drivers](https://docs.mongodb.com/ecosystem/drivers)

<sup>9</sup> The Go programming language, [golang.org](https://golang.org)

<sup>10</sup> Communicating sequential processes, [en.wikipedia.org/wiki/Communicating\\_sequential\\_processes](https://en.wikipedia.org/wiki/Communicating_sequential_processes)

<sup>11</sup> `engine-api`, Go library for Docker, [godoc.org/github.com/docker/engine-api](https://godoc.org/github.com/docker/engine-api)

<sup>12</sup> `mgo`, Go library for MongoDB, [godoc.org/gopkg.in/mgo.v2](https://godoc.org/gopkg.in/mgo.v2)



requests based on the URL path – the `gorilla/mux`<sup>1</sup> package allows elegant routing schemas in a few lines of code.

## 4.2 Scheduler Module

This component acts as an interface between the web module and the Docker Swarm. It is implemented in the `scheduler` Go package. [Subsection 4.2.1](#) describes the simple API this package exposes, while [Subsection 4.2.2](#) gives a step-by-step explanation of how evaluation scheduling works in LxChecker.

### 4.2.1 The API

The `scheduler` package provides a single `Submit` method that takes as an argument a `SubmitOptions` configuration object and returns a `SubmitResponse` (or an error in case of scheduling failure).

[Listing 4.1](#) shows the `SubmitOptions` struct definition:

- `Image` is the name of the Docker image that should be used as a testing environment (e.g. `lxchecker/ml_assignment_3`)
- `Submission` contains the file uploaded by the user for evaluation – it is copied to the container as instructed by the `SubmissionPath` field (see below)
- `SubmissionPath` tells the scheduler where in the container to put the submission, as expected by the tests in the image
- `Timeout` specifies for how long the scheduler should wait before forcefully killing the container and returning a *time limit exceeded* error

---

```

1 type SubmitOptions struct {
2     Image      string
3     Submission []byte
4     SubmissionPath string
5     Timeout    time.Duration
6 }

```

---

Listing 4.1: the `SubmitOptions` struct

The `SubmitResponse` struct definition is shown in [Listing 4.2](#).

- `Logs` contain the combined `stdout` / `stderr` produced by the tests in the environment.
- `ExitCode` is the status code of the tests execution: 0 for a successful run (no fatal error that interrupted the evaluation), non-zero for error. Note that a successful run of the tests does not require tests to pass – the actual testing results are extracted from the `Logs` based on *metadata* (see below).

---

```

1 type SubmitResponse struct {
2     Logs      []byte
3     ExitCode  int
4 }

```

---

<sup>1</sup>`gorilla/mux`, request router for Go, [gorillatoolkit.org/pkg/mux](http://gorillatoolkit.org/pkg/mux)

Listing 4.2: the `SubmitResponse` struct

LxChecker uses **metadata** to allow environment designers (such as a teacher preparing an assignment) to expose data from the evaluation logs to the web frontend. For all log lines that match the format `@key value`, the frontend adds the corresponding key-value pair to a **metadata** map. A special required key is **score** (used for declaring the submission's evaluation score), but any number of keys can be defined. More on this topic will follow in [Section 4.4](#).

### 4.2.2 Execution Flow

This subsection goes through the steps of running tests on a submission. An overview is presented in [Figure 4.1](#). The dotted box shows the separation between different hosts: the light gray box is part of the Docker cloud, while the dark gray rectangle represents the evaluation results on the host of the Lxchecker binary. Everything inside the dotted box happens on a Docker host selected by the Swarm manager.

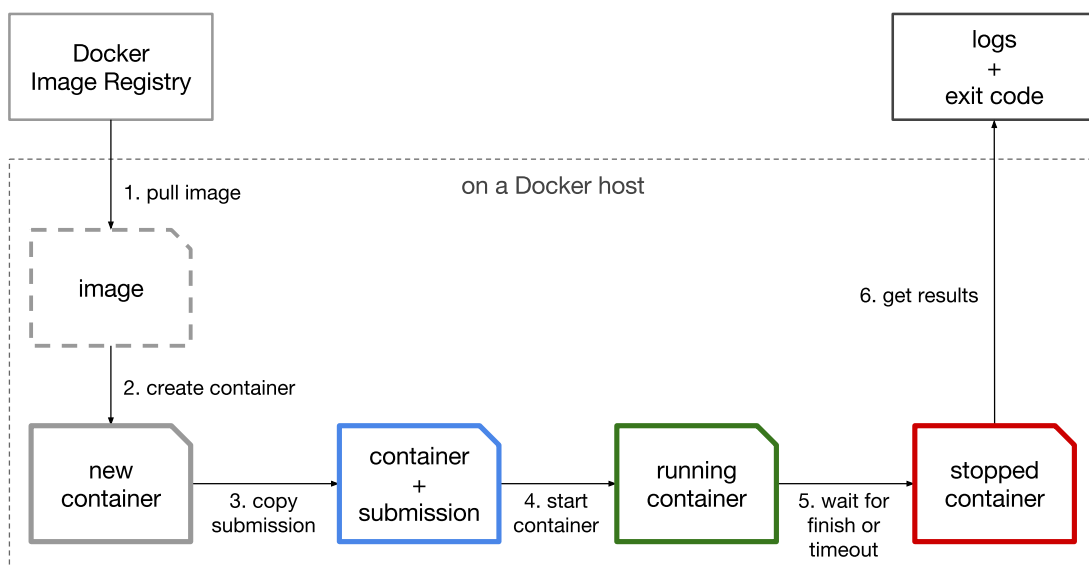


Figure 4.1: execution steps performed by the scheduler

When called, the `Submit` method goes through the following steps:

1. **pull image**: the `Image` field in the `SubmitOptions` struct is used to fetch the appropriate testing environment image from the Docker Hub. This should happen only once on each Docker host, thanks to image caching.
2. **create container**: the pulled image is used to create a new container (stopped for now).
3. **copy submission**: the `Submission` bytes are tarred<sup>1</sup> and then extracted to `SubmissionPath` using Docker's *copy-to-container* mechanism.
4. **start container**: the container is started, invoking the test suite.
5. **wait for finish or timeout**: the scheduler waits for the container to finish its execution. If the specified `Timeout` expires earlier, the container is stopped and an error is returned.

<sup>1</sup>tar archive, [en.wikipedia.org/wiki/Tar\\_\(computing\)](https://en.wikipedia.org/wiki/Tar_(computing))

6. **get results:** the `Logs` and `ExitCode` are extracted from the container and returned in a `SubmitResponse` struct.

### 4.3 Storage Interface Module

This component is a high-level wrapper over the `mgo` library. It provides an ORM-like interface, with database models as first-class objects. Currently, there is support for reading, writing and updating entities.

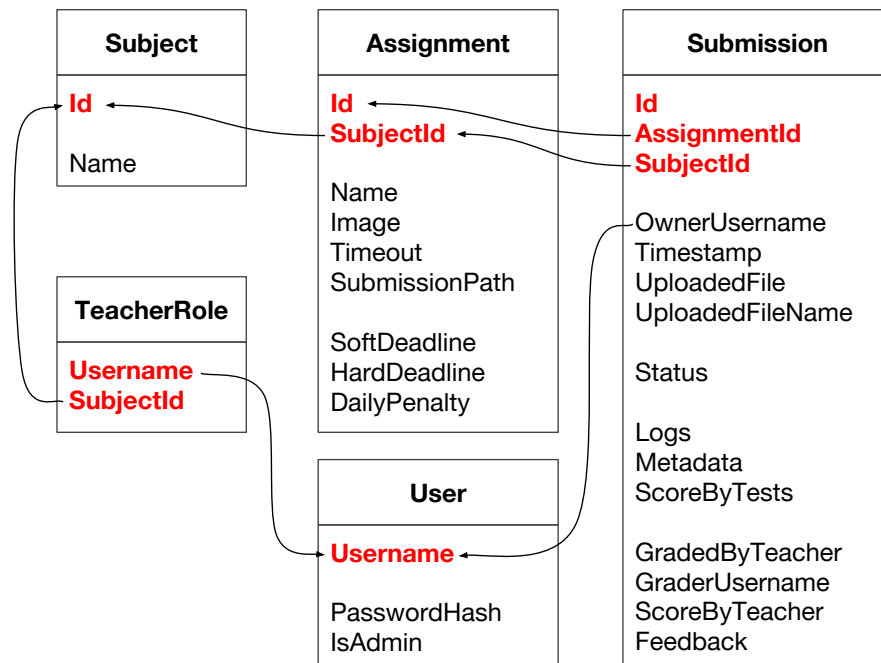


Figure 4.2: database models and schema

Figure 4.2 illustrates database models and relationships between them. The fields in red represent primary keys. One more than one field in red is present, the key is composite. Each entity is stored in a separate MongoDB collection<sup>1</sup>, indexed<sup>2</sup> by the primary key. The following paragraphs briefly present the purpose of each model and cover the most important fields.

**Subject** denotes a course for which automated grading is desired. The `Id` primary key must be a lowercase alphanumeric code (e.g. `ml` for a Machine Learning course), while `Name` should be the full name of the subject (e.g. "Machine Learning").

**Assignment** holds configuration data for testing environments. Assignments are grouped by subject, hence the parent link `SubjectId`. The `Id` should be a short, yet human-readable code for the assignment (e.g. `kmeans`). `Name` should be a lengthier description like "K-means Clustering". Fields `Image`, `Timeout` and `SubmissionPath` are used by the `scheduler` module, detailed in Section 4.2. The remaining fields are necessary for implementing deadline support in the web module (see Section 4.4).

A **Submission** is created when a user uploads a file. The `AssignmentId` and `SubjectId` fields reference the parent assignment and together with `Id` serve as a composite primary key.

<sup>1</sup> MongoDB collection, [docs.mongodb.com/manual/reference/glossary/#term-collection](https://docs.mongodb.com/manual/reference/glossary/#term-collection)

<sup>2</sup> Indexes in MongoDB, [docs.mongodb.com/manual/indexes](https://docs.mongodb.com/manual/indexes)

`OwnerUsername` refers to the user that made the upload, while `Timestamp`, `UploadingFile` and `UploadingFileName` are traits of the upload itself. `Status` can be pending, done or failed. The next group of fields is populated on evaluation. The remaining fields are set when a teacher manually grades the submission, giving feedback.

The `User` refers to an account on Lxchecker. A user can log in using the stored `Username` and a password whose hash is `PasswordHash`. Platform administrators have the field `IsAdmin` set to true.

`TeacherRole` creates a many-to-many relationship between `Users` and `Subjects`. An entry in the `teacher_roles` MongoDB collection gives an user privileged read & write access to an assignment.

### 4.3.1 The API

For each model described above, package `db` provides methods for:

- **querying** objects:
  - `Get*` methods returning exactly one object or an `db.ErrNotFound` error.
  - `GetAll*` methods for returning a slice of object, possibly empty.
- **inserting** objects:
  - `Insert*` methods that return nothing when successful and `db.ErrNotFound` for missing parent objects or `db.ErrAlreadyExists` for duplicate insertions.
- **updating** objects:
  - `Update*` methods that return nothing when successful and `db.ErrNotFound` for non-existing keys.

Support for object deletion is planned for the future. For now, the MongoDB interactive shell can be used by a privileged staff member.

### 4.3.2 Implementation Example

To illustrate how the `mgo` library is used for working with the MongoDB database, [Listing 4.3](#) shows a simple query method from the `db` package.

---

```

1 func GetSubmission(subjectId, assignmentId, id string) (*Submission,
   error) {
2     submission := Submission{}
3     c := mongo.DB("lxchecker").C("submissions")
4     if err := c.Find(bson.M{
5         "subject_id": subjectId,
6         "assignment_id": assignmentId,
7         "id": id,
8     }).One(&submission); err != nil {
9         if err == mgo.ErrNotFound {
10             return nil, ErrNotFound
11         }
12         panic(err)
13     }
14     return &submission, nil

```

15 }

Listing 4.3: GetSubmission query method

## 4.4 Web Module

In this section, we will describe the implementation of Lxchecker's web component. In code, this resides in package `main`. When the server is launched, the execution starts in function `main`: it initializes the `scheduler` and `db` packages, then configures the URL router and finally blocks in the HTTP request handling loop. We have already discussed the two module dependencies in [Section 4.2](#) and [Section 4.3](#), respectively. URL routing and general organization of HTTP handlers will be covered next, followed by HTML templating. For simplicity of exposition, we will initially ignore authentication aspects and assume fully unrestricted access to the platform. Finally, we will discuss permissions and authentication in the last section.

### 4.4.1 URL Routing and Handlers

Lxchecker employs a hierarchical URL structure. Any possible "view" in the browser has a distinct address which can be used to bookmark or share that view. This is a significant user-experience improvement over Vmchecker, which doesn't provide unique URLs for its UI views.

<code>/GET</code>	
<code>/login/GET,POST</code>	
<code>/signup/GET,POST</code>	
<code>/logout/POST</code>	
<code>/add_admin/POST</code>	global handlers
<code>/-/GET</code>	view all subjects
<code>/-/create_subject/POST</code>	
<code>/-/ml/GET</code>	create, view and modify subjects
<code>/-/ml/add_teacher/POST</code>	
<code>/-/ml/create_assignment/POST</code>	
<code>/-/ml/kmeans/GET</code>	create, view and modify assignments
<code>/-/ml/kmeans/create_submission/POST</code>	
<code>/-/ml/kmeans/57cf2db3/GET</code>	
<code>/-/ml/kmeans/57cf2db3/get_upload/GET</code>	create, view and modify submissions
<code>/-/ml/kmeans/57cf2db3/grade_submission/POST</code>	

Figure 4.3: URL structure of Lxchecker

[Figure 4.3](#) groups request handlers by area of interest. URLs marked with the `GET` annotation allow browser navigation and access to data, while `POST` handlers are called by submitting web forms, usually resulting in data mutations.

To understand how request handlers are implemented using Go's `http` library, see [Listing 4.4](#) which shows Lxchecker's simplest handler. `LandingHandler` is the function that manages requests to the landing page (`/`). The first argument, an `http.ResponseWriter`, is where the

HTTP response needs to be written. The second argument is an object containing all request data (such as URL parameters or POST data). In this example, the `http.Redirect` function is used to write a 302-Found HTTP redirection response.

---

```
1 func LandingHandler(w http.ResponseWriter, r *http.Request) {  
2     http.Redirect(w, r, "/-/", http.StatusFound)  
3 }
```

---

Listing 4.4: a simple HTTP handler example

An important aspect in the implementation of request handlers is error management. There two types of error that can occur during the processing of a HTTP request:

- **recoverable errors**, typically generated by bad requests. These should be treated gracefully.
- **internal errors**: these are exceptions that occur either because of a software bug or because some internal dependency has failed (network or disk failure, for example). These are unrecoverable errors and an 500 HTTP response code should be returned to the user.

In Lxchecker's web module, we handle recoverable errors by returning an informative error response to the client. For internal errors we use Go's `panic` feature (similar to exceptions in other languages). The panic propagates along the call stack until explicitly recovered by the top-level `RecoveryHandler`<sup>1</sup> and converted into a 500 internal error response.

#### 4.4.2 Templates

The graphical user interface of Lxchecker is implemented using the standard library `html/template` library and the Twitter Bootstrap<sup>2</sup> UI toolkit.

Templates are structured under the `web/templates/` directory. For avoiding code duplication, a common layout was extracted to `base.html`, inherited by all the other templates. The base layout includes dependencies such as the Bootstrap library and defines a general structures of web pages.

#### 4.4.3 Authentication

Lxchecker is designed to be used in academic institutions and secure grading for students is crucial. The platform offers a basic landing page for unauthenticated guests and allows them to either login or create an account. New accounts are considered "student accounts" and are given limited privileges: accessing subjects and assignments, creating submissions and viewing their own. For unrestricted reading access and grading capabilities within a subject, a user needs to be given a teacher role in that subject. That can be done by an administrator, which has platform-wide privileges. An admin can create subjects and grant administrator and teacher roles.

Privileges are enforced in different ways for reading and writing actions:

- both **reading** and **writing** actions can be allowed or denied at handler-level by middleware – a certain URL might be blocked if the current user doesn't have the right to access it. [Figure 4.4](#) shows the composable middleware architecture of Lxchecker. By routing URLs through any combination of middleware handlers, they can become public, or accessible by simple login, or require higher permissions.

---

<sup>1</sup>Gorilla `handlers` package, [godoc.org/github.com/gorilla/handlers#RecoveryHandler](http://godoc.org/github.com/gorilla/handlers#RecoveryHandler)

<sup>2</sup>Twitter Bootstrap, [getbootstrap.com](http://getbootstrap.com)

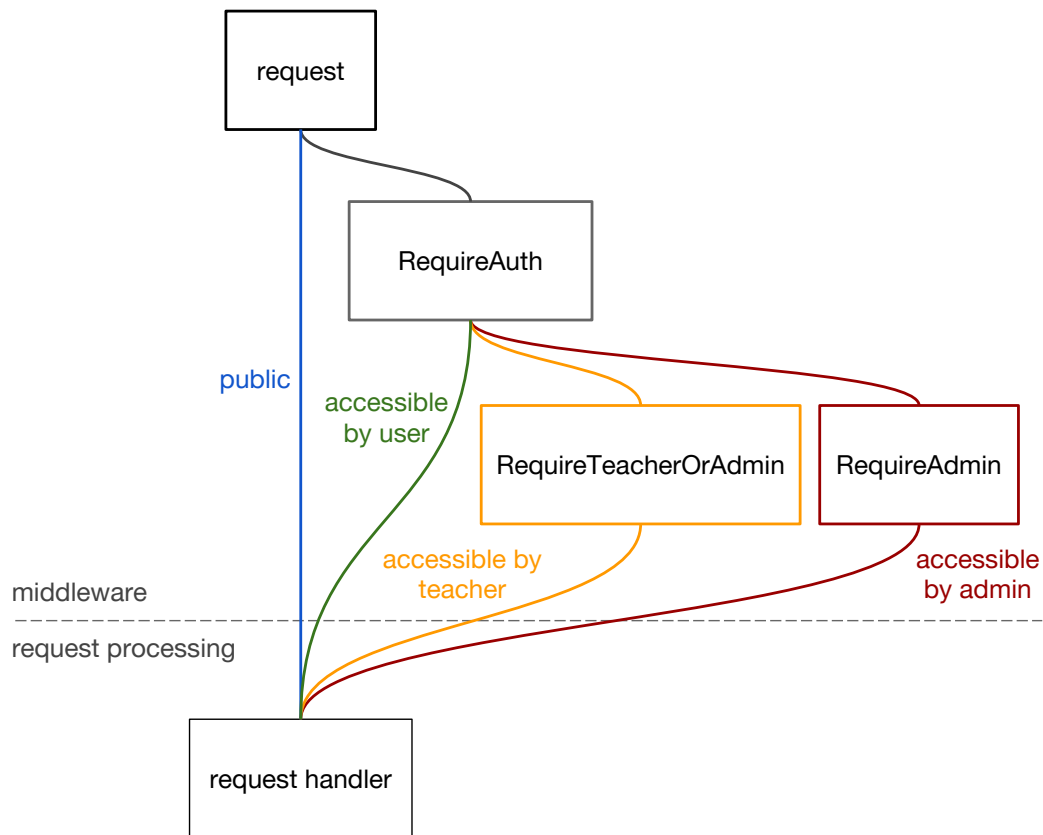


Figure 4.4: middleware composition

- some **reading** limitations are implemented at template-level by only displaying parts of page if required conditions are met. The `RequestData` struct, obtainable in handlers using the `GetRequestData` function from package `util`, can be inspected in [Listing 4.5](#). A `RequestData` object is passed to all executed templates, which can then use the boolean fields to determine available permissions and display information accordingly.

```

1 type RequestData struct {
2     SubjectId    string
3     AssignmentId string
4     SubmissionId string
5
6     User          *db.User
7     UserIsLoggedIn bool
8     UserIsTeacher  bool
9     UserIsAdmin    bool
10 }

```

Listing 4.5: the RequestData struct

Authentication persistence across sessions is implemented using secure cookies<sup>1</sup>.

<sup>1</sup>[gorilla/sessions](https://github.com/gorilla/sessions) library, [gorillatoolkit.org/pkg/sessions](https://gorillatoolkit.org/pkg/sessions)

## Chapter 5

# Deploying and Using Lxchecker

This chapter covers all aspects of using Lxchecker. In [Section 5.1](#), we present the process of installing and maintaining the platform from the perspective of faculty staff members. [Section 5.2](#) shows how teachers prepare assignments and how students submit their solutions for grading. The approach is incremental – we start with steps as easy as logging in and then proceed to more advanced procedures.

### 5.1 Deploying Lxchecker

Lxchecker was designed with easy deployment and maintenance in mind. Given a cluster of Linux hosts, the only prerequisite for installing Lxchecker is Docker.

#### 5.1.1 Installing Lxchecker

For this exposition, let's assume we want to deploy Lxchecker on four machines (either physical or virtual), named `host0` through `host3`. Docker is used as a deployment tool for all of Lxchecker's dependencies, so it needs to be installed on all hosts. The documentation<sup>1</sup> provides easy setup instructions for various Linux distributions.

After installing Docker, we want to prepare the evaluation cluster – let's use `host1`, `host2` and `host3`. We need to install<sup>2</sup> Swarm on all three machines, setup `host1` as a Swarm manager and add `host2` and `host3` to the cluster. From now, `host1` (the Swarm manager) will transparently behave as a giant Docker host and Lxchecker will talk to it for scheduling containers.

The other dependency is MongoDB. We will use `host0` for both the database server and Lxchecker itself. Docker provides an official image<sup>3</sup> for MongoDB, which can be run by a simple command: `docker run --name mongo-server -d mongo`.

Installing Lxchecker follows the same procedure, with two considerations: the container needs to be linked to `mongo-server` so it can reach it, and `host1`'s address must be specified via an environment variable. The resulting command: `docker run --name lxchecker --link mongo-server -e "DOCKER_HOST:host1" -d lxchecker/lxchecker`.

Navigating to `host0` in the browser should display Lxchecker's login page.

---

<sup>1</sup> Installing Docker, [docs.docker.com/engine/installation](https://docs.docker.com/engine/installation)

<sup>2</sup> Build a Swarm cluster for production, [docs.docker.com/swarm/install-manual](https://docs.docker.com/swarm/install-manual)

<sup>3</sup> `mongo` on Docker Hub, [hub.docker.com/\\_/mongo](https://hub.docker.com/_/mongo)



To ensure Lxchecker and its dependencies run continuously, Docker's built-in restart policies<sup>1</sup> can be used. In case of a fatal error in one of the components, a restart policy will rerun the failed container to resume service.

### 5.1.2 Updating Lxchecker

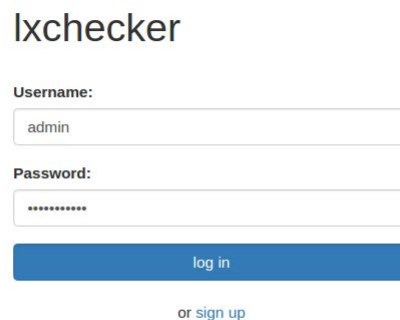
Docker makes it easy to push a software update to production:

- on the development machine, rebuild the `lxchecker/lxchecker` image from the updated code.
- push the new image to Docker Hub: `docker push lxchecker/lxchecker`.
- on `host0`, pull the new image from Docker Hub: `docker pull lxchecker/lxchecker`.
- reissue the `docker run` command.

## 5.2 Using Lxchecker

This section describes the user experience of students, teachers and admins working with Lxchecker. We first cover authentication, followed by subject & assignment creation. Finally, we show how students upload code for testing and how their submissions are graded.

### 5.2.1 Authentication



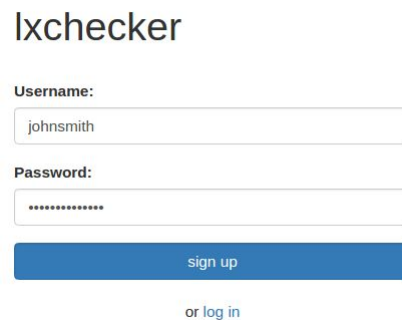
The screenshot shows the Lxchecker login interface. At the top, the word "lxchecker" is displayed in a large, dark font. Below it, there are two input fields: "Username:" with the text "admin" entered, and "Password:" with a masked password "\*\*\*\*\*". A blue "log in" button is positioned below the password field. At the bottom, there is a link that says "or sign up" in a smaller, blue font.

Figure 5.1: logging in

When a user first visits the Lxchecker application, he is redirected to the login form displayed in [Figure 5.1](#). If he provides valid credentials, he is logged in and showed the subject list in [Figure 5.3](#). If he doesn't have an account, he can click the "or **sign up**" link to visit the sign up page (see [Figure 5.2](#)).

The subjects overview page in [Figure 5.3](#) also includes a form for giving admin privileges. The username provided in the textbox must belong to an existing user.

<sup>1</sup> Docker Restart Policies, [docs.docker.com/engine/admin/host\\_integration](https://docs.docker.com/engine/admin/host_integration)

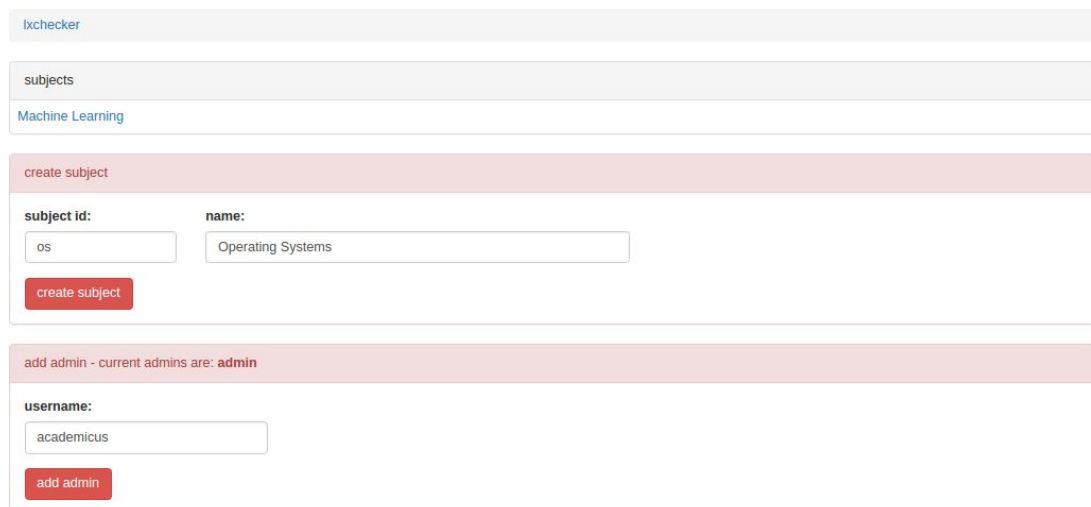


The screenshot shows the 'lxchecker' login and sign-up interface. It features a title 'lxchecker' at the top. Below it, there are two input fields: 'Username:' with the value 'johnsmith' and 'Password:' with a masked password '\*\*\*\*\*'. A blue 'sign up' button is positioned below the password field. At the bottom, there is a link 'or log in'.

Figure 5.2: creating a new account

## 5.2.2 Working With Subjects

In [Figure 5.3](#), users can see the list of all existing subjects. The forms in red (for adding subjects and admins) are only visible to admins. In the example picture, the **create subject** form is used to create an entry for the "Operating Systems" course. The **id** is short and meaningful, as it will be part of all future URLs in this subject.



The screenshot displays the 'lxchecker' interface with a sidebar on the left containing 'lxchecker', 'subjects', and 'Machine Learning'. The main content area shows a list of subjects, with 'Machine Learning' selected. Below this, there are two red-bordered forms. The first form, titled 'create subject', contains two input fields: 'subject id:' with the value 'os' and 'name:' with the value 'Operating Systems'. A red 'create subject' button is at the bottom of this form. The second form, titled 'add admin - current admins are: admin', contains a 'username:' input field with the value 'academicus' and a red 'add admin' button at the bottom.

Figure 5.3: available subjects list and *create subject* form

After submitting the **create subject** form, the admin is redirected to the new "Operating Systems" subject page (see [Figure 5.4](#)). The sections in red are only visible to admins and teachers. Typically, an admin will only create a subject, then delegate permissions to a professor or teaching assistant in that course using the **add teacher** form. Once the teacher gains privileges in this subject, he can create assignments using the **create assignment** form.

## 5.2.3 Managing Assignments

Assignment parameters were explained in previous chapters. The example data filled out in [Figure 5.4](#) is for an assignment on the topic of "Virtual Memory", using the `lxchecker/os-vmem` Docker image as a testing environment.

lxchecker / Operating Systems

assignments

no assignments

create assignment

assignment id: vmem

name: Virtual Memory

docker image: lxchecker/os-vmem

timeout: 120

submission\_path: /submission.zip

soft deadline: 10.12.2016

hard deadline: 17.12.2016

daily penalty: 10

create assignment

add teacher

username: deaconescu

add teacher

Figure 5.4: subject page for the *Operating Systems* course

After submitting the **create assignment** form, the user is redirected to the "Virtual Memory" assignment page, displayed in [Figure 5.5](#). A submission is already included in the figure to serve as an example.

The **assignment info** section contains information about:

- **deadlines**: submissions will be penalized after the *soft deadline* and will be considered "overdue" after the *hard deadline*.
- **daily penalty** tells how many points will be subtracted from the submissions score for each day over the soft deadline.
- the **timeout** imposes a limit on the duration of the evaluation.

Section **my submissions** lists the user's own submissions, with information about evaluation status, points awarded by the system and whether the submission is *active* (selected for final grading by the teacher). The current implementation automatically marks the most recent submission as *active*.

## 5.2.4 Submission Upload

The **upload submission** form at the bottom of the page is used to submit a new file for evaluation. After clicking the **submit** button, the user is redirected to the new submission page.

As shown in [Figure 5.6](#), the submission is initially marked as **pending**. The **on time** label indicates the submission was uploaded before the soft deadline and no penalty is subtracted. Downloading the uploaded file is possible by using the included link. Because the evaluation has not finished, there are no logs yet.

Once the submission is tested, more information becomes available. The **metadata** field is populated and logs are fully displayed in the **execution logs** section. Note, however, that **grading status** is still *not graded* until a teacher gives feedback and a score.

lxchecker / Operating Systems / Virtual Memory

assignment info	
soft deadline	Saturday, 10.12.2016, 23:59
hard deadline	Saturday, 17.12.2016, 23:59
daily penalty	10
timeout	120 seconds

my submissions	
57d1075d09039b5a3235d10e	done score by tests: 70 active

upload submission	
Choose File	os-vmem.zip
submit	

Figure 5.5: *Virtual Memory* assignment page

lxchecker / Operating Systems / Virtual Memory / 57d1052309039b5a3235d10d

submission info	
status	pending on time
execution metadata	not yet available
download submission	link
grading status	not graded
feedback	not yet available
overall grade not available	

execution logs	
not yet available	

Figure 5.6: waiting for the submission to be evaluated

### 5.2.5 Submission Grading

Figure 5.8 shows the grading form. The teacher fills out the **score** and the textual **feedback**. After clicking the **submit** button, the **score by teacher** and **feedback** are populated and the student can see his final grade (score by tests + score by teacher - penalty). Figure 5.9 displays the page of a graded submission.

lxcchecker / Operating Systems / Virtual Memory / 57d1052309039b5a3235d10d

submission info	
status	done on time score by tests: 76
execution metadata	score: 76
download submission	<a href="#">link</a>
grading status	not graded
feedback	not yet available
overall grade not available	

execution logs

```
Archive: submission.zip
inflating: common.h
inflating: common_lin.c
inflating: debug.h
```

Figure 5.7: submission was tested and score is displayed (full logs not included)

grade submission

score:

feedback:

-1: leaking memory

-1: very long functions

grade submission

Figure 5.8: grading form for teachers

submission info	
status	done on time score by tests: 76
execution metadata	score: 76
download submission	<a href="#">link</a>
grading status	score by teacher: 8 graded by: admin
feedback	-1: leaking memory -1: very long functions
score by tests: 76 + score by teacher: 8 = overall grade: 84	

Figure 5.9: submission after grading by teacher

## Chapter 6

# Results and Future Work

In this chapter we describe the current state of the project and attempt to assess whether the initial objectives were met. We also try to identify areas in which our implementation is lacking and suggest a few ideas for improvement.

### 6.1 Current State of the Project

**At this point, Lxchecker is ready for production.** Among the features we have successfully implemented, it's worth to mention:

- full authentication support, implemented using secure cookies
- functional role-based permission system
- submission evaluation on top of Docker
- flexible environment definitions via `Dockerfiles`
- full grading support
- user-friendly interface, with rich URL linking (including redirection after log in)

Compared to Vmchecker, our system offers significantly improved **scalability**. This is due to inherent implications of Vmchecker's design: machines are not shared across subjects. This is not in accordance with how the system is actually used. Students upload submissions for evaluation in spikes, concentrated around deadlines. Deadlines rarely overlap, however, and Vmchecker doesn't make use of this distribution of load over time – most machines sit idle, while one or two sustain all the traffic. Lxchecker solves this issue by using a single pool of resources for all subjects. Thanks to Docker Swarm, scheduling is done as efficiently as possible.

**User experience** received much attention in Lxchecker. Generations of students using Vmchecker taught us about what users value and we incorporated that knowledge when designing the interface of our system.

Lxchecker learned a lot from both Vmchecker's merits and shortcomings. We believe that our system inherited most of the merits and addressed most of the shortcomings. There is one notable feature of Vmchecker that cannot be implemented using Lxchecker's technology stack: **support for Windows environments**. Assignments that require a Windows operating system will continue to be tested using Vmchecker, unless support for virtual machines is added to Lxchecker.

## 6.2 Future Work

The problem of automatically grading programming assignments is a complex one. Lxchecker attempted to incorporate the essential functionality and help us draw conclusions about the feasibility of containers in this problem. Because implementation time was limited, there are features that needed to be left for the future.

Functionality currently missing:

- **deletion support** for database objects: entities such as subjects or assignments can be created and updated, but not deleted. To implement removal, the `db` package and the templates need to be updated.
- a **resubmit** feature: it would be useful to be able to restart the evaluation of a solution without actually reuploading the file.
- **automated container removal**: although keeping old containers for later debugging or inspection might be desired, a mechanism for cleaning up Docker hosts is needed. An idea is to have a cron<sup>1</sup> job that periodically removes old containers.
- **custom selection of active submission**: today the most recent submission of a user on an assignment is automatically selected for grading by the teacher. A student should be able to mark any submission as final. An alternative is to rely on the deletion functionality: the desired submission could be made active by removing all newer submissions.

Broader project ideas that we thought about include:

- **support for Windows environments**: as discussed in [Section 6.1](#), containers are Linux-only. For assignments requiring a Windows system (e.g. "Operating Systems" course), virtual machines could be used. The change would involve extending the `scheduler` module to use a virtual machine manager such as Vagrant<sup>2</sup>. An assignment switch could then select between the two backend technologies.
- **in-browser code inspection**: right now teachers are required to manually download submissions to their local machines in order to check the quality of the code. Making the source tree accessible within the browser would save reviewers a lot of time. One implementation idea is to make use of MongoDB's GridFS<sup>3</sup> for storing unarchived files in the database. The URL schema could be extended to support paths like `/-/os/vmem/57d1075d/-/src/main.cpp`. A syntax highlighter such as `highlight.js`<sup>4</sup> could be used for enhancing readability.

It is clear that Lxchecker offers many opportunities for improvement. An important aspect is user satisfaction – the system can only shine if developers listen to feedback from users. Both students and teachers should contribute with suggestions on how to make Lxchecker better.

---

<sup>1</sup>Cron, [en.wikipedia.org/wiki/Cron](https://en.wikipedia.org/wiki/Cron)

<sup>2</sup>Vagrant, [vagrantup.com](https://vagrantup.com)

<sup>3</sup>GridFS, [docs.mongodb.com/manual/core/gridfs](https://docs.mongodb.com/manual/core/gridfs)

<sup>4</sup>highlight.js, [highlightjs.org](https://highlightjs.org)

## Chapter 7

# Conclusion

In this thesis we proposed **a system for automatically grading programming assignments**. Lxchecker is a web application for teachers and students, that allows code testing in a well-defined environment. It supports any number of subjects and assignments, features a role-based permission model and promises good performance.

In [Chapter 1](#), we described what we want to build and what use cases we have in mind. We continued with presenting recent technical advancements in the field of virtualization and containerization. [Chapter 2](#) also covers existing solutions for automatic grading in universities and online judges. Architectural design of Lxchecker was discussed in [Chapter 3](#) – we showed the high-level overview of the system and then detailed the purpose and interface of each component. In [Chapter 4](#) we explained our choices of tools and libraries. We also investigated the implementation of the three main software modules, insisting on interesting problems we encountered and solutions we found. Because Lxchecker is designed to run in a server environment, [Chapter 5](#) gives instructions on how to install the system on a cluster of Linux machines. A second section describes the usage from the perspective of administrators, teachers and students, with an example scenario included. Finally, in [Chapter 6](#) we present the current state of the project, listing both notable accomplishments and missing features. We also identify areas of improvement, proposing possible ideas and solutions.

We think Lxchecker will prove useful in academic institutions teaching Computer Science. Running the system in production will help us to better understand the needs of our users and allows us to further improve the functionality and behavior. We hope this paper will contribute to the development of superior products in the automated assignment grading market.



# Bibliography

- [1] Compare incomparable: Postgresql vs mysql vs mongodb. <http://erthalion.info/2015/12/29/json-benchmarks>, 2015.
- [2] Brenda Cheang, Andy Kurnia, Andrew Lim, and Wee-Chong Oon. On automated grading of programming assignments in an academic institution, 2003.
- [3] Valentin Goşu. vmchecker - enhancement and scalability, 2012.
- [4] Saikkonen R., Malmi L., and Korhonen A. Fully automatic assessment of programming exercises, 2001.
- [5] K. Reek. A software infrastructure to support introductory computer science courses, 1996.