On-demand virtual laboratory environments for ~~an~~ Internetworking e-learning

ANDREAS KOKKALIS

Master's Thesis at ICT
Supervisor: Anders Västberg
Examiner: Gerald Q. Maguire Jr.

Abstract

Learning Management Systems (LMSs) are widely used in higher education to improve the learning, teaching, and administrative tasks for both students and instructors. Such systems enrich the educational experience by integrating a wide range of services, such as on-demand course material and training, thus empowering students to achieve their learning outcomes at their own pace.

Courses in various sub-fields of Computer Science that seek to provide rich electronic learning (e-learning) experience depend on exercise material being offered in the forms of quizzes, programming exercises, laboratories, simulations, etc. Providing hands on experience in courses such as Internetworking could be facilitated by providing laboratory exercises based on virtual machine environments where the student studies the performance of different internet protocols under different conditions (such as different throughput bounds, error rates, and patterns of changes in these conditions). Unfortunately, the integration of such exercises and their tailored virtual environments is not yet very popular in LMSs.

This thesis project investigates the generation of on-demand virtual exercise environments using cloud infrastructures and integration with an LMS to provide a rich e-learning in an Internetworking course.

Is this perhaps one of the main points of the thesis - that one can dynamically instantiate virtual exercise environments without having to have the CAPEX of running ones own infrastructure for this. This is especially important as the usage of such systems is very bursty (due both to the academic calendar and other factors).

Sammanfattning

Add swedish section

Acknowledgements

I would like to acknowledge ...

Contents

List of Figures

List of Algorithms

List of Tables

Listings

List of Acronyms and Abbreviations

Keep alphabetical order!

AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
CA	Certificate Authority
CS	Computer Science
CPU	Central Processing Unit
DOM	Document Object Model
DSL	Domain Specific Language
e-learning	electronic learning
EC2	Elastic Compute Cloud
ERB	Embedded RuBy
GLUE!	Group Learning Uniform Environment
GUI	Graphical User Interface
HTML	Hyper Text Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
IT	information technology
JSON	JavaScript Object Notation
KTH	Kungliga Tekniska Högskolan
LIS	Learning Information Services
LMS	Learning Management System

LTI Learning Tools Interoperability

LXC Linux Containers

MIT Massachusetts Institute of Technology

MIME Multipurpose Internet Mail Extensions

MOOC Massive Open Online Course

OCI Open Container Initiative

RDBMS Relational Database Management System

SCROM Sharable Content Object Reference Model

SHA Secure Hash Algorithm

SSH Secure Shell

SQL Structured Query Language

TC Tool Consumer

TCP Transmission Control Protocol

TLS Transport Layer Security

TP Tool Provider

TTL Time To Live

UI User Interface

URL Uniform Resource Locator

XML Extensible Markup Language

Chapter 1

Introduction

The use of electronic learning (e-learning) technologies has been well established in modern education to assist both students and instructors in their learning, teaching, and administrative tasks. One of the e-learning technologies most widely adopted by the academic community is Learning Management Systems (LMSs). A LMS is a software application that handles all aspects of the learning process [1], enabling instructors to design rich e-learning courses and students to experience self-paced learning using a variety of features, such as on-demand course material, video lectures, automatic delivery and evaluation of assignments, collaboration tools, etc.

Many courses, especially in various sub-fields of Computer Science depend on training events in the form of programming assignments, laboratory exercises, simulations, etc. These activities are crucial for students to gain hands-on experience with complex concepts and systems [2]. Although LMSs support on-line training events, such as interactive quizzes with automatic evaluation and analysis of results, providing training events that depend on using complex virtual environments and software are not yet very popular (and hence not widely supported or used).

One of the main advantages of using an LMS is that it supports the integration of external applications to provide personalized, domain specific e-learning, such as messaging and video streaming services, on-line office suites, collaboration tools, or even training environments with exercises tailored to the needs of a specific course.

1.1 Background

Hands-on experience is very important to achieve understanding of complex systems and concepts. For example, when studying computer networks, laboratory exercises are a common student activity. An Internetworking course often involves students studying the performance of different Internet protocols under different conditions (such as varying throughput bounds, error rates, and patterns of changes in network conditions).

These experiments depend on specific software, network topologies, and local or virtual hardware. Traditional approaches for realizing such environments depend upon the student's own hardware or on-site computer labs with pre-configured software [3]. More modern approaches involve remote access to virtual machines running on central servers or cloud infrastructures [4].

Currently LMSs do not have built-in support for such laboratory environments. However, one of the main advantages of designing an on-line course on top of an LMS that supports the integration of extenal applications is to provide tailored functionality for the course's and student's specific needs. Today, many LMSs, such as Instructure Inc.'s Canvas [5] LMS, implement the IMS Global Learning Consortium Tools Interoperability® (LTI®) specification. Learning Tools Interoperability (LTI) allows the exchange of information between the LMS and third party components, thus exposing internal functionality of the LMS to external applications in a controlled manner.

Supporting virtual laboratory environments in a LMS in order to meet the needs of an Internetworking course, requires the design of a software framework that implements the LTI interoperability specification in order to exchange relevant information between the laboratory environment and the LMS.

1.2 Problem definition

Hands on experience is very important aspect of the learning process in several fields of Computer Science, including computer networks. Understanding the domain specific concepts and problems of an Internetworking course, depends greatly on exercise material and laboratory practice. Today, such exercises, are not usually designed to extract suitable analytics for the instructor (as an instructor ideally wishes to evaluate each student's level of understanding of each of the different concepts covered in an exercise). Assessing the student's understanding is currently achieved by using additional training material, such as quizzes or assignments in forms of reports which are manually evaluated by instructors or by other students in the form of peer reviews. These alternative methods both introduce a delay in feedback to the student (hence reducing the student's rate of learning) and are not scalable (for example, preventing their use in Massive Open Online Courses (MOOCs)).

Supporting an on-line version of an Internetworking course through a LMS that enables students to achieve the course's learning outcomes at their own pace, depends greatly on designing interactive practice environments. Such environments should be easily modified by the instructor to fit the needs of different exercises. Although today LMSs support a variety of training events, such as quizzes and assignments through integration of external services, on-line virtual laboratory environments that fulfill the requirements of an Internetworking course are not yet well supported and hence not widely used.

However, similar practice environments are common in on-line courses that teach programming languages. Such environments are part of systems that provide tools for designing coding assignments, and support several assessment methods, including automatic evaluation and grading of code [6] and programming quizzes. These systems often provide standalone web applications or LTI integrations in LMSs that expose functionality for developing code, submitting assignments, and presenting feedback to users [7, 8].

This project aims to design a software framework that supports interactive training material for an Internetworking course, integrates with a LMS to provide a rich e-learning experience, and offers dynamic installation of laboratory environments that scale according to the needs of the virtual classroom.

1.3 Goals

The design of such a laboratory environment for an Internetworking course has to meet several user requirements from the perspective of both students and instructors, and integrate with a LMS to offer a rich e-learning experience. The expected outcome of this project is a software framework that supports instantiation of on-demand laboratory environments using cloud based technologies to enrich the learning experience of students, allowing them to proceed at their own pace. Additionally, the framework should enable a teacher to customize the environment according to different exercises' requirements, and provide the instructor with constructive feedback about each student's progress and understanding.

The process of designing this framework can be realized by achieving the following goals:

- Devise a method to easily build virtual laboratory environments,
- The framework should enable the instructor to easily create and manage different versions of laboratory environments, as such environments can be reused for different assignments.
- The framework should be integrated with the LMS to enable students to access the training environments via the LMS,
- The method of integration of such exercise environments should be usable by others - thus an important part of this thesis project is documenting the selected method to facilitate the integration of a diverse set of external environments (for example, an ns-3 simulator configured for a particular simulation),
- The framework should scale in such way that it enables students to do assignments at any given time, thus offering on-demand availability of the underlying services, and

- A student should be able to access a training environment within reasonable upper bounded time from the moment she requests from the LMS to start an assignment.

1.4 Research Methodology

Design science research addresses important unsolved problems in unique or innovative ways or solved problems in more effective or efficient ways. It focuses on the design and construction of information technology (IT) artifacts that have utility in real-world, application environments. The artifacts, as the outcome of the research process, aim to improve domain-specific systems and processes [9, 10]. The utility, quality, and adequacy of a design artifact, is thoroughly evaluated under varying experimental setups to verify that it successfully fulfills the stated requirements.

Design, in several research fields, including IT, is an iterative process of planning, generating alternatives, and selecting a satisfactory outcome. Design science research, although it is not performed using strictly defined processes, can be summarized by three closely related cycles of activities (these cycles are the relevance cycle, the rigor cycle, and the design cycle) [11], that act as guidelines for designing, constructing, and evaluating an artifact. The relevance cycle establishes the application context that not only provides the requirements for the research as inputs, but also defines acceptance criteria for the evaluation of the research results. The rigor cycle provides past knowledge to the research project to ensure its innovation. It is contingent on researchers to thoroughly research and reference this knowledge base in order to guarantee that the designs produced are research contributions and not routine designs based upon the application of well-known processes. The central Design Cycle iterates between the core activities of building and evaluating the design artifacts and processes of the research [9], until the acceptance criteria, as defined in the Relevance Cycle, are met.

This project is carried out using the design science research approach. The resulting software and documentation attempt to solve the problem of designing and realizing a framework for rich on-line laboratory environments for an elearning course on Internetworking, that is to be accessible via a specific learning management system (Canvas LMS). The two different domains that define the context of this problem are the Internetworking course domain, and the LMS along with the method(s) of integration of external applications into Canvas (in this case via LTI).

1.5 Deliminations

This project addresses the problem of designing and integrating virtual laboratory environments to support e-learning in an LMS for an Internetworking course. The laboratory framework, the expected outcome of this project, has to fulfill several requirements: usability for different types of users (instructor, administrator, and student,), integration into the Canvas LMS via the LTI specification, and satisfy the laboratory and pedagogical challenges of this particular course. Although there are different specifications for integrating external applications and services into a LMS [12], this project addresses only the LTI specifications, as this method is supported by Canvas (along with many other LMSs, for example LTI can be used together with edX as either a consumer or provider [13]). The laboratory framework, is designed to suit the needs of a typical classroom (in this case approximately 30 students), thus its scalability is limited.

Testing the scalability of the designed system regarding the number of users is outside of the scope of this thesis project. However, a system might be scaled up by using larger virtual instances (vertical scaling) or by creating multiple instances (horizontal scaling). Additionally, scaling up and down of services in clouds has been investigated by others [14].

1.6 Structure of the thesis

Chapter 2 explains what an LMS is, introduces the LTI specification for integrating external learning applications into such systems, and presents an example of an external learning tool which is integrated with Canvas LMS. Furthermore, it presents the related technologies that was used to implement the software outcome of this project, along with projects that addressed problems related to the e-learning process in other fields of Computer Science (CS). Chapter 3 explains the methods used to evaluate the proposed artifact. Chapter 4 presents the software artifact that was designed to facilitate student understanding of Internetworking via e-learning, and finally, Chapter 5 presents the results and the future work required to prepare the software artifact for use in production with Canvas LMS.

Chapter 2

Background

This chapter explains what ~~is~~ a LMS and how learning applications are integrated in such systems to support rich e-learning. Moreover, it introduces research artifacts that offer on-line training environments for various courses in the CS domain. Lastly, it introduces those technologies that were used to design the framework that supports training events for an Internetworking course.

2.1 LMS

LMSs are software applications that automate the training, teaching, and administrative tasks of the learning process [1]. They have been widely adopted by higher education institutions to automate their organizational functions and provide a rich e-learning experience for both instructors and students.

Such systems are designed to provide self-guided services; rapid delivery and composition of learning material; tracking and reporting of progress through training programs, classroom, or on-line events; personalized content; and centralization and automation of administration [15]. From a learner's perspective the most common use cases of a LMS are planning ones own learning experience and collaboration with colleagues; while from an instructor's perspective the most common use cases are the design and delivery of educational content along with tracking and analysis of students' learning evolution [16].

The main functionality of a LMS concerns content organization and delivery, communication and collaboration, and assessment* of student's learning process. Some of the most commonly used features of an LMS for e-learning are video streaming of lectures, on-line notes and presentations, quizzes and practice environments, automatic evaluation of assignments (usually exercises with

*According to Wynne Harlen and Mary James [17], formative assessment is performed by teachers during the learning process, to modify and improve the teaching and learning activities. It is based on observation of students' individual efforts and development; thus, having a qualitative and diagnostic nature. Summative assessment, performed by both instructors and students, is based on public criteria that aim to measure student's achieving of the course learning outcomes.

predefined input and output), wikis, and discussion forums [18]. These services are either offered directly by the LMS or by integrating external applications that are designed according to specific interoperability standards. Section 2.2 describes this interoperability and integration in detail.

Although LMSs provide built-in learning applications for designing e-learning courses, their functionality is often very limited and might not suit the needs of every course. Moreover, not all LMSs support the same learning tools, nor provide the same functionality for e-learning. On the contrary, external learning tools can be integrated with multiple different LMSs, and allow re-using existing material thus minimizing the effort for designing an e-learning course. Usually such tools are web services* that are discoverable by an LMS via the service's Uniform Resource Locator (URL) and authorization parameters (such as secret keys). The communication between the LMS and the tool is performed by exchanging messages whose format and content is defined by the interoperability specification. Section 2.3 shows several web frameworks that can be used to design external learning tools as web services.

There are several LMSs in the market (Blackboard, Moodle, Kanu, . . .) that are used by multiple institutions. In the scope of this project the chosen learning management platform is Canvas [5]. This LMS was chosen because the system is open source, supports a well defined interoperability specification, and was selected in 2016 by KTH as their LMS.

2.2 LTI

Interoperability is the ability to communicate, execute programs, or transfer data among functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units [19]. An e-learning platform usually consists of several services such as course and user administration modules, and learning applications that exchange information in a formal and standardized way.

The IMS Global Learning Consortium Tools Interoperability (LTI) specification establishes a way of integrating rich learning applications (often remotely hosted and provided through third-party services) with platforms, such as LMSs, portals, learning object repositories, or other educational environments [20]. The main goal of LTI is to standardize the process of building links for sharing information and exposing functionality between external learning tools and the LMS [21]. There are two major pieces of software involved in LTI. The first is called a Tool Consumer (TC) and it refers to the software (such as an LMS) that consumes the output of

*In service oriented architectures, a web service is a piece of software that makes itself available over the Internet and allows third-party software to communicate with them by exchanging strictly defined messages formatted in Extensible Markup Language (XML), JavaScript Object Notation (JSON), etc.

external tools, and the second, is a Tool Provider (TP) which provides an external tool for use by the TC.

An example of a basic learning tool, is a service that accepts a request to perform a course assignment such as multiple choice question via a web form, evaluates the user's input, and returns a pass/fail grade. In this scenario, the service is the TP and Canvas LMS is the TC. A user of Canvas with administrative access (e.g., teacher), configures the integration of the external tool, a course assignment for which the tool will be launched, and finally, chooses whether the interface of the tool will be embedded in Canvas, or run in a new browser window. Figure 2.1 shows a basic flow for launching a TP from the TC. The user requests from the LMS that they want to do an assignment. This specific assignment has been configured to launch a specific LTI capable external tool together with arguments that are passed to the TP. The TP authenticates and accepts the LTI Launch request by the TC and starts a session for that particular user that allows this user to interact with the assignment.

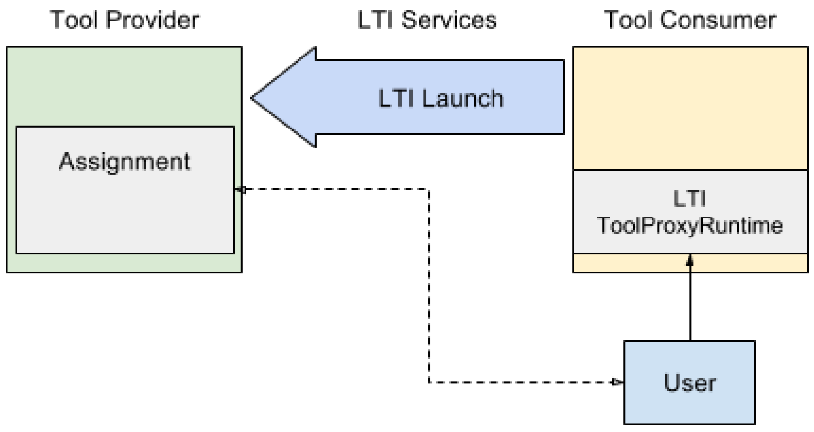


Figure 2.1: User launching an external tool

A TP often requires access to course related information, such as people, groups, memberships, courses, and outcomes. This information along with standardized ways of retrieving it are defined by the IMS Global Learning Consortium Learning Information Services (LIS) specification [22]. These services can be provided either by the TC or by a third party system. Canvas LMS implements the LTI version 1.1 which includes a subset of the LIS specification, called the LTI Basic Outcomes Service. In the example mentioned above, the information that Canvas provides to the TP when performing an LTI Launch are: how to access the LIS services, the resource identifier (assignment) for which a grade will be reported, and user information such as the unique identifier of the student. Figure 2.2 shows how a TP can communicate with LIS services to get user data and report the grade of the assignment back to the TC.

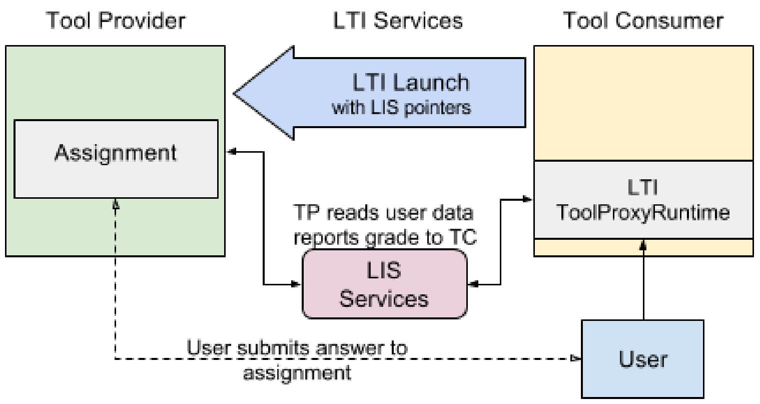


Figure 2.2: A TP using LIS services

2.3 Sinatra DSL

A simple web server is a piece of software designed to process Hypertext Transfer Protocol (HTTP) requests. Many web frameworks have been developed in several programming languages that allow to quickly develop web servers and applications. Sinatra [23, 24] amongst them, is a Domain Specific Language (DSL) for writing web applications in Ruby. A Sinatra web application is organized around routes which are HTTP methods paired with a URL-matching pattern. Listing 2.1 presents a minimal sinatra application. The route `"/"` is paired with a `get` HTTP method. Every time this route is invoked, it provides a `"Hello World!"` text response.

Listing 2.1: Sinatra basic route

```ruby
# hello_world.rb
require 'sinatra'

get '/' do
  'Hello world!'
end
```

A file named `hello_world.rb` contains the code shown in Listing 2.1, which is called a route block. A route block starts with a keyword such as `get, post, put, ...` and corresponds to an HTTP method, and finishes with the keyword `end`. Executing the web application is as simple as running the command `ruby hello_world.rb`. This will start a sinatra web server on the default host (`localhost`) that listens for Transmission Control Protocol (TCP) connections on the default port (`4567`). By visiting the URL `http://localhost:4567/` with a browser, the route `"/"` is invoked and the response returned to the user.

A route can also utilize HTTP GET query parameters as shown in Listing 2.2. In this case, if a `course_id` is provided as a parameter of query string, then its value is loaded into the local variable `courseID`. The same concept could be applied if the route was an HTTP POST method and course_id was one of the post's parameters.

Listing 2.2: Sinatra route with HTTP GET parameters

```
get '/assignments' do
  # matches "GET /assignments?course_id=IKXXX"
  courseID = params['course_id']
  # uses course_id variable; query is optional to the / route
end
```

Sinatra also supports the use of wildcards to match all parameters of the query string. Such parameters are called splat, are symbolized with a "*" router pattern, and are accessible via the `params['splat']` array. In the Listing 2.3, the route `'/department/*/course/*'` represents the course catalog of a university. The splat parameters match the department (`informatics`) and course (`ID001`) identifiers respectively.

Listing 2.3: Wildcard route pattern

```
get '/department/*/course/*' do
  # matches /department/informatics/course/ID001
  params['splat'] # => ["informatics", "ID001"]
end
```

Templates are a text injection mechanism, that allows static text to be enriched using dynamic content (e.g., an Hyper Text Markup Language (HTML) template might contain some static text and variables, where the variables are replaced during runtime). In Sinatra a template by default is stored under the directory `./views`, and can be used in many different ways, including rendering HTML pages, constructing a **!** (**!**) object as a response to an HTTP request, etc. Listing 2.4 shows the route `get '/assignments'` which stores the value of the `course_id` parameter into an instance variable `@courseID` which makes the value of this variable available for use in the template shown in Listing 2.5.

Listing 2.4: Sinatra route with template

```
get '/assignments' do
  @courseID = params['course_id']
  erb :index
end
```

Calling the `assignments` route by visiting the url `http://localhost:4567/assignments?course_id=IK1552` will parse the query parameter, invoke the `index.erb` template[*] stored under the directory `./views` and substitute the text `<%= @courseID%>` ~~with~~ the value of the variable `@courseID`. The response that will be rendered by the browser will be an HTML page that contains the text "List of assignments for IK1552" in its body.

[*]Embedded RuBy (ERB) is part of the Ruby standard library, and serves as the mechanism for variable substitution within template files.

Listing 2.5: index.erb

```
<!DOCTYPE html>
<html>
  <head>
    <title>Assignments</title>
  </head>
  <body>
    <p>List of assignments for <%= @courseID%></p>
  </body>
</html>
```

A Sinatra route can be used to serve static files. By default, static files are served from the `./public` directory that is located under in the same directory as the application. A Sinatra application, though it is minimalistic, it is not limited to default options, thus one can configure different port numbers, root directories, custom template engines and locations, etc. Other web servers similar to Sinatra are: Flask in Python, and Netty in Java.

A collection of URL routes such as `/department/*/course/*` and `/assignments` describe a server-side web Application Programming Interface (API), that is based on HTTP request-response message exchange. In the context of web application development, such routes are ~~named~~ API endpoints ~~that~~ describe the method for accessing application resources. An endpoint is consumed by a client-side application or a web service, and are either publicly accessible, or protected by some sort of authorization scheme.

2.4 LTI tool provider

This section presents a TP written in Ruby Sinatra that implements the Basic Outcomes Service of the LTI specification, and is integrated into the Canvas LMS which will act as a TC. The TP has three routes (listed in Table 2.1).

Table 2.1: Routes of the TP

`launch`	route for launching the external tool
`assignment`	route for starting an assignment
`report`	route for reporting the result of the assignment to Canvas LMS

The `launch` route implements the LTI Launch functionality of the LTI specification, accepts requests for launching the external tool, and initiates a unique session per request. The `assignment` route checks for a valid session, and then returns an HTTP response with an HTML form. The form is the assignment and in this example contains a simple arithmetic question that the student has to reply to by submitting her answer in the form's input. Finally, the `report` route validates the student's input, and reports a pass/fail grade to the TC.

This example assumes that a Canvas instructor has created an assignment and configured it to launch the TP. The following code snippets present the code implementation of the TP (inspired by lti_example from ~~this~~ github repository [25] of Instructure Inc.), the functionality of each route, and the XML messages that are used to communicate with the TC.

Listing 2.6 shows the code dependencies to implement the TP. First it requires the `sinatra` gem* and the `oauth` gem (used to implement the service provider, according to the LTI specification for authorization between a a TP and a TC). The `$oauth_keyt` and `$oauth_secret` variables define the key and secret that is used by the TP to identify the TC. These variables are configured in a Canvas LMS when specifying the external tool. Finally the `disable :protection` statement allows for the HTML content produced by the Sinatra application to be embedded into an HTML frame of the TC, and the `enable :sessions` statement allows for session information to be used between subsequent HTTP requests to Sinatra routes.

Listing 2.6: Code dependencies and some global variables of the TP

```
# dependencies
require 'sinatra'
require 'oauth'
require 'oauth/request_proxy/rack_request'

# key and secret for authenticating requests from the TC
$oauth_key = "test"
$oauth_secret = "secret"

# disable x-frame to allow embedding the TP in the TC
disable :protection

# ennable sessions for uniquely identifying students
enable :sessions
```

The `launch` route shown in Listing 2.7 is responsible for authorizing a request from the TC to launch the assignment. First it verifies the `request` against the `secret` variable. If the authorization fails, then a text message is returned to inform the Canvas user that the integration of the tool was not successful. After the authorization succeeds, the HTTP request parameters `lis_outcome_service_url` and `lis_result_sourcedid` (these correspond to the LTI LIS services) are read. The first corresponds to the TC URL that is used to report a grade for an assignment, while the latter is a unique identifier that is used to map an assignment grade for a particular student. If these parameters were not provided when Canvas invoked this route, then the request will fail. By default Canvas sets these parameters when a tool provider is correctly configured as graded assignment. After the successful verification of the afore mentioned

*Ruby gems are versioned packages of ruby source code. In practice they are libraries that are hosted in public servers that make them available for download via ruby package management systems.

parameters, their values are stored in corresponding session objects and the route redirects to the `get /assignment` route.

Listing 2.7: Launch route

```
post "/launch" do
  # verify the request of the TC
  begin
    signature = OAuth::Signature.build(request, :
   consumer_secret => $oauth_secret)
    signature.verify() or raise OAuth::Unauthorized
  rescue OAuth::Signature::UnknownSignatureMethod,
         OAuth::Unauthorized
    return %{Unauthorized attempt. Make sure you used the
   consumer secret "#{$oauth_secret}"}
  end

  # Verify that this is a valid request to perform an
   assignment
  unless params['lis_outcome_service_url'] && params['
   lis_result_sourcedid']
    return %{It looks like this LTI tool was not launched as
   an assignment, or you are trying to do the assigment as a
   teacher rather than as a a student.}
  end

  # store the relevant parameters from the launch into the
   user's session, for
  # access during subsequent HTTP requests.
  %w(lis_outcome_service_url lis_result_sourcedid).each { |v|
   session[v] = params[v] }

  # Go to the assignment
  redirect to("/assignment")
end
```

The `/assignment` route, presented in Listing 2.8, starts by validating the session variable `lis_result_sourceid`. If this parameter was not set, then the tool was not launched via the TC, hence an error text message is returned. This error message will be visible ~~to~~ the user's browser (either as a frame within Canvas LMS or as a new tab on the user's browser). If the session is valid, then the route replies with an HTML form that is rendered by the user's browser. This form includes a simple arithmetic addition question and an input field for the student to reply. The form action sends the form to the `report` route ~~and~~ using HTTP `post` method. When the student presses the submit button within the browser, the `report` route is invoked. Note that in this listing the form ha been included directly in the route block, but it could have been placed in a ruby template such as the ~~one of~~ Listing 2.5.

Listing 2.8: Assignment route

```
get "/assignment" do
  # Verify the validity of the session
  unless session['lis_result_sourcedid']
    return %{You need to take this assignment through Canvas.}
  end

  # Render a form with the assignment question.
  <<-HTML
  <html>
    <head><title>Demo LTI Assignment</title></head>
    <body>
      <form action="/report" method="post">
        <p>What is the sum of 100 + 200 ?</p>
        <input name='sum' type='text' width='5' id='sum'
   required />
        <input type='submit' value='Submit' />
      </form>
    </body>
  </html>
  HTML
end
```

The `report` route is displayed in Listing 2.9 and is invoked when the student submits the form. If the form parameter `sum` is not provided, then the user is redirected (again) to the assignment via the corresponding route. Upon successful validation of the form input, an XML response message is defined and sent to Canvas via the appropriate LIS services to report the student's grade for this assignment. The format of the XML message is based upon the `imsx_POXEnvelopeRequest` class defined in the XML schema of the IMS General Web Services documentation [26] and described in the LTI 1.0 implementation guide [27].

The body of the message contains the field `sourceID` that is assigned the value of the session variable `#session['lis_result_sourcedid']` , and the resultScore field that corresponds to the assignment's grade and gets the value 1 in the `textString` subfield if the provided sum was 300 or 0 otherwise. The corresponding assignment had been configured earlier in Canvas to accept a maximum of 1 point for the grade for this assignment.

The message is signed according to OAuth 1.0 protocol[*] using the same consumer key and secret that were provided during the LTI launch request (`launch` route). The message is posted synchronously to the Canvas LIS service defined by `session['lis_outcome_service_url']` using a Multipurpose Internet Mail Extensions (MIME)[†] encoding, and the response is stored in the `response`

[*]OAuth provides a method for clients to access server resources on behalf of a resource owner (such as a different client or an end-user). It also provides a process for end-users to authorize third-party access to their server resources without sharing their credentials (typically, a username and password pair), using user-agent redirections.[28]

[†]The MIME-type is a two-part identifier for file formats and format of contents [illegible]smitted on

variable. Because the post was done synchronously the code will wait until the response to this post is received. Thus the body of the response can be used to compute the message to be displayed to the user via their browser.

Listing 2.9: Report the assignment grade to Canvas

```
post "/report" do
  sum = params['sum']
  if !sum || sum.empty?
    redirect to("/assignment")
  end

  # now post the score to canvas. Make sure to sign the POST
   correctly with
  # OAuth 1.0, including the digest of the XML body. Also make
    sure to set the
  # content-type to application/xml.
  xml = %{
<?xml version = "1.0" encoding = "UTF-8"?>
<imsx_POXEnvelopeRequest xmlns = "http://www.imsglobal.org/lis
   /oms1p0/pox">
  <imsx_POXHeader>
    <imsx_POXRequestHeaderInfo>
      <imsx_version>V1.0</imsx_version>
      <imsx_messageIdentifier>12341234</imsx_messageIdentifier
   >
    </imsx_POXRequestHeaderInfo>
  </imsx_POXHeader>
  <imsx_POXBody>
    <replaceResultRequest>
      <resultRecord>
        <sourcedGUID>
          <sourcedId>#{session['lis_result_sourcedid']}</
   sourcedId>
        </sourcedGUID>
        <result>
          <resultScore>
            <language>en</language>
            <textString>#{sum == 300 ? 1 : 0}</textString>
          </resultScore>
        </result>
      </resultRecord>
    </replaceResultRequest>
  </imsx_POXBody>
</imsx_POXEnvelopeRequest>
  }
  consumer = OAuth::Consumer.new($oauth_key, $oauth_secret)
  token = OAuth::AccessToken.new(consumer)
```

the Internet.

```
  response = token.post(session['lis_outcome_service_url'],
    xml, 'Content-Type' => 'application/xml')

  headers 'Content-Type' => 'text'
  %{
Your score has #{response.body.match(/\bsuccess\b/) ? "been
    posted" : "failed in posting"} to Canvas. The response was:
#{response.body}
  }
end
```

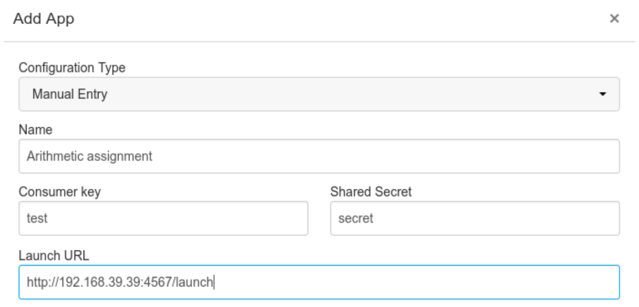
Lastly the contents of `reponse` are evaluated and checked whether posting the grade was successful or not, and a text message is sent to the user to be rendered by her browser informing her about the status of posting the grade to Canvas. The response of a successful post is highlighted in Listing 2.10 in the `imsx_codeMajor` xml field.

Listing 2.10: XML response from Canvas

```
<?xml version="1.0" encoding="UTF-8"?>
<imsx_POXEnvelopeResponse xmlns="http://www.imsglobal.org/
    services/ltiv1p1/xsd/imsoms_v1p0">
  <imsx_POXHeader>
    <imsx_POXResponseHeaderInfo>
      <imsx_version>V1.0</imsx_version>
      <imsx_messageIdentifier/>
      <imsx_statusInfo>
        <imsx_codeMajor>success</imsx_codeMajor>
        <imsx_severity>status</imsx_severity>
        <imsx_description/>
        <imsx_messageRefIdentifier>12341234</
    imsx_messageRefIdentifier>
        <imsx_operationRefIdentifier>replaceResult</
    imsx_operationRefIdentifier>
      </imsx_statusInfo>
    </imsx_POXResponseHeaderInfo>
  </imsx_POXHeader>
  <imsx_POXBody><replaceResultResponse/></imsx_POXBody>
</imsx_POXEnvelopeResponse>
```

2.4.1 Integration of an external application into Canvas LMS

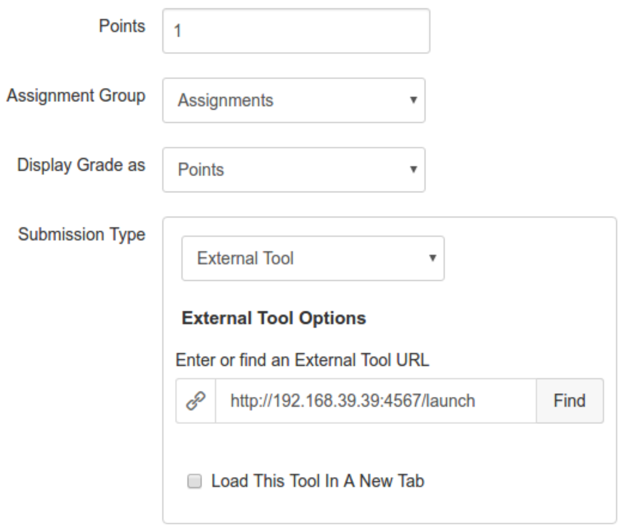
The text above presented how to develop a simple LTI provider that supports graded assignments. The Graphical User Interface (GUI) of Canvas LMS allows the integration of external applications via different options such as manual configuration forms, launch URLs, and pasting XML entries. This section will present how to configure the external tool of the previous section using a manual configuration form via the `Settings->Apps->External Apps->Add App` menu for a course. Here we assume that an instructor wishes to add an external app for a particular course. The input form shown in Figure 2.3 is loaded. The instructor inputs a name for the application, the LTI Launch URL, and the consumer key and secret.


Add App

Configuration Type	Manual Entry
Name	Arithmetic assignment
Consumer key	test
Shared Secret	secret
Launch URL	http://192.168.39.39:4567/launch

Figure 2.3: Adding an external application to Canvas

After adding this external tool, the instructor creates a new assignment, configures it to launch the application within Canvas, or using an external window as shown in Figure 2.4, and specifies a grading scheme. Once the assignment is configured and published in Canvas, a student can do this assignment via the course page. Section 2.5 explains how to integrate external applications using URLs and XML configuration.


Points	1
Assignment Group	Assignments
Display Grade as	Points
Submission Type	External Tool

External Tool Options

Enter or find an External Tool URL: http://192.168.39.39:4567/launch

☐ Load This Tool In A New Tab

Figure 2.4: Configuring an assignment to use an external tool

2.4.2 Securing the connection between a TP and a TC

The communication between Canvas LMS and external application tools is by default expected to be performed using the Hypertext Transfer Protocol

Secure (HTTPS)[*] protocol. In the example presented in previous section, the communication between the TP and the TC was over HTTP, hence Canvas generated a corresponding error while launching the TP. The Sinatra web-server can be easily configured to listen for HTTPS connections on some port. Listing 2.11 shows such a configuration of the Sinatra web server (named Webrick). HTTPS requires a TLS certificate which for the purposes of this example was issued and signed using the OpenSSL [29] cryptography and TLS toolkit, rather than a trusted third party Certificate Authority (CA).

[*]HTTPS is a protocol for communication over HTTP within a connection encrypted by Transport Layer Security (TLS). TLS uses a public and a private encryption key to generate a session key which is used to encrypt the data flow between client and server. An HTTP message is encrypted prior to transmission and decrypted upon arrival.

Listing 2.11: TLS configuration of a Sinatra application

```
require 'sinatra/base'
require 'webrick'
require 'webrick/https'
require 'openssl'

CERT_PATH = '/opt/CA/'

webrick_options = {
  :Port                 => 8443,
  :Logger               => WEBrick::Log::new($stderr, WEBrick::
   Log::DEBUG),
  :DocumentRoot         => "/ruby/htdocs",
  :SSLEnable            => true,
  :SSLVerifyClient      => OpenSSL::SSL::VERIFY_NONE,
  :SSLCertificate       => OpenSSL::X509::Certificate.new(File.
   open(File.join(CERT_PATH, "cert.pem")).read),
  :SSLPrivateKey        => OpenSSL::PKey::RSA.new(File.open(File
   .join(CERT_PATH, "key.pem")).read),
  :SSLCertName          => [ [ "CN", '127.0.0.1' ] ]
}

class MyServer  < Sinatra::Base
    post '/' do
      "Hellow, world!"
    end
end

Rack::Handler::WEBrick.run MyServer, webrick_options
```

Listing 2.12 shows how to generate a TLS certificate using the OpenSSL command line tool. The command is `openssl req` and it takes several arguments such as `-new` (request new certificate), `-x509` (format of the public key), `-extensions v3_ca` (the extensions to add for a self signed certificate, shown in the corresponding block of Listing 2.13, `-keyout key.pem` (the output file for storing the key), `-out cert.pem` (the output file for storing the self-signed certificate), `-days 365` (the number of days until the certificate expires), and finally the sample configuration file `openssl.conf` for reading the default values.

Listing 2.12: Generating a self signed TLS certificate and encryption key

```
openssl req -new -x509 -extensions v3_ca -keyout key.pem -out
   cert.pem -days 365 -config ./openssl.conf
```

The OpenSSL configuration shown in Listing 2.13, is a sample file containing default values for generating a TLS certificate and a public key file, and is available for download in Markus Redivo's page "Creating and Using SSL Certificates" [30]. More details regarding the use of the `req` command of the OpenSSL toolkit can be found in the corresponding man page [31], and information about the configuration file can be found in Phil Dibowitz's blog page "Openssl.conf walkthru" [32].

Listing 2.13: Sample OpenSSL configuration for issuing SSL/TLS certificates

```
---Begin---
# OpenSSL configuration file.

# Establish working directory.
dir = .

[ ca ]
default_ca     = CA_default

[ CA_default ]
serial         = $dir/serial
database       = $dir/index.txt
new_certs_dir  = $dir/newcerts
certificate    = $dir/cacert.pem
private_key    = $dir/private/cakey.pem
default_days   = 365
default_md     = md5
preserve       = no
email_in_dn    = no
nameopt        = default_ca
certopt        = default_ca
policy         = policy_match

[ policy_match ]
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ req ]
default_bits        = 1024     # Size of keys
default_keyfile     = key.pem  # name of generated keys
default_md          = md5      # message digest algorithm
string_mask         = nombstr  # permitted characters
distinguished_name  = req_distinguished_name
req_extensions      = v3_req

[ req_distinguished_name ]
# Variable name           Prompt string
```

```
#----------------------   ----------------------------------
0.organizationName      = Organization Name (company)
organizationalUnitName  = Organizational Unit Name (department, division)
emailAddress            = Email Address
emailAddress_max        = 40
localityName            = Locality Name (city, district)
stateOrProvinceName     = State or Province Name (full name)
countryName             = Country Name (2 letter code)
countryName_min         = 2
countryName_max         = 2
commonName              = Common Name (hostname, IP, or your name)
commonName_max          = 64

# Default values for the above, for consistency and less typing.
# Variable name                Value
#----------------------       ------------------------------
0.organizationName_default  = The Sample Company
localityName_default        = Metropolis
stateOrProvinceName_default = New York
countryName_default         = US

[ v3_ca ]
basicConstraints       = CA:TRUE
subjectKeyIdentifier   = hash
authorityKeyIdentifier = keyid:always,issuer:always

[ v3_req ]
basicConstraints     = CA:FALSE
subjectKeyIdentifier = hash

----End----
```

2.5 LTI applications

Edu App Center [33] is an open database for learning tools maintained by Instructure [34] and among its several services, it offers a collection of open learning applications that implement the LTI specification. These applications can be integrated with different LMSs. The user can apply filters to locate an appropriate tool and can browse tutorials about integrating a tool with the LMS of their choice. Often these tools are hosted by third party services (e.g GitHub, Youtube, Turnitin). The goal of Edu App Center is to enable instructors to easily configure these external applications to their courses, thus providing and fostering a market place for LTI applications.

Section 2.4.1 presented how an instructor can integrate a Ruby Sinatra external application into Canvas LMS using a web form. This approach is limited to the functionality of Canvas LMS. An alternative method for integrating external applications via XML configuration can be used across different LMSs. Edu App Center offers such configurations for every LTI tool listed in the marketplace. Additionally, it provides the XML Config Builder service, that allows instructors to generate XML for integrating custom built external LTI applications into different LMSs. Listing 2.14 shows an example of such XML entry (generated by the Edu App Center's XML Config Builder) that was used to integrate the Ruby Sinatra application (presented in the previous section) into Canvas.

First, the XML version and the charset encoding are defined. Then the `cartridge_basiclti_link xmlns` specifies that this is an LTI link that can be used for integrating an external application. This block contains the whole XML configuration. It starts by defining the IMS Global XML schema that is used to describe this entity. Then the LTI Launch URL is specified (`blti:launch_url`), and it is followed by metadata, regarding the title (`blti:title`) and description (`blti:description`) of the external application. Finally, it defines a block for ~~lti~~ extensions (`blti:extensions platform`) that specifies the LMS platform to act as a TC for this TP. This block of XML code can contain information that is specific to each LMS that is supported by the TP.

Listing 2.14: I am a comment

```
<?xml version="1.0" encoding="UTF-8"?>
<cartridge_basiclti_link xmlns="http://www.imsglobal.org/xsd/
   imslticc_v1p0"
  <!-- Definition of the XML Schema -->
  xmlns:blti = "http://www.imsglobal.org/xsd/imsbasiclti_v1p0"
  xmlns:lticm ="http://www.imsglobal.org/xsd/imslticm_v1p0"
  xmlns:lticp ="http://www.imsglobal.org/xsd/imslticp_v1p0"
  xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation = "http://www.imsglobal.org/xsd/
   imslticc_v1p0 http://www.imsglobal.org/xsd/lti/ltiv1p0/
   imslticc_v1p0.xsd
  http://www.imsglobal.org/xsd/imsbasiclti_v1p0 http://www.
   imsglobal.org/xsd/lti/ltiv1p0/imsbasiclti_v1p0.xsd
  http://www.imsglobal.org/xsd/imslticm_v1p0 http://www.
   imsglobal.org/xsd/lti/ltiv1p0/imslticm_v1p0.xsd
  http://www.imsglobal.org/xsd/imslticp_v1p0 http://www.
   imsglobal.org/xsd/lti/ltiv1p0/imslticp_v1p0.xsd">

  <!-- The LTI Launch url -->
  <blti:launch_url>http://192.168.39.39:4567/launch</
   blti:launch_url>
  <!-- Title of the External Application -->
  <blti:title>Arithmetic Assignment</blti:title>
  <!-- Description for the external application -->
  <blti:description>Sample arithmetic assignment tool</
   blti:description>
  <-- Configuration specific to the TC -->
  <blti:extensions platform="canvas.instructure.com">
    <lticm:property name="privacy_level">public</
   lticm:property>
  </blti:extensions>
</cartridge_basiclti_link>
```

2.6 Previous efforts to provide on-line exercise material

Traditional practice events in Computer Science involve laboratory environments and exercises based on physical or virtual hardware and domain specific software. One of the problems is creating and managing these environments. Previously such material was packaged in virtual machines or run in an isolated environment (such as a sandbox or linux container as will be described in Section 2.7).

With the rapid growth of e-learning courses, the need for on-line exercise material has grown. Efforts in fields of cybersecurity include ”A Comparison of Virtual Lab Solutions for Online Cyber Security Education” [35]~~,~~ and “Top 10 Hands-on Cybersecurity Exercises” [36]. In addition to the environment, there is a need for domain specific source material. Some useful references and sources for exercise material regarding networking include “Hands-On Experience to a Massive Open Online Course on openHPI” [4], ”Some Experiences in Using Virtual Machines for Teaching Computer Networks” [3], and ”V-Lab: A Mobile Virtual Lab for Network Security Studies” [37].

2.7 Linux Containers

A container is a light weight operating system running inside the host system, executing instructions native to the Central Processing Unit (CPU), eliminating the need for instruction level emulation or just in time compilation [38]. Linux Containers (LXC) [39] is an operating-system-level virtualization method for running multiple isolated Linux systems (containers) on a host using a single Linux kernel. Its purpose is to virtualize a single application rather than a whole operating system inside a virtual machine. LXC uses cgroups[*] to isolate resources (such as CPU, memory, network, etc.) and namespaces[†] to isolate the application from the operating system [41].

Docker [42] is a Linux container engine that provides the ability to manage containers as self contained images. Docker utilizes LXC for the container implementation, has image management capabilities, and implements a Union File System (UnionFS). It features resource isolation via cgroups and namespaces, network and file system isolation through LXC functionality, and allows managing the lifecycle of a container [38]. Although docker initially utilized LXC as the only execution driver for resource isolation, lately it introduced libcontainer [43], which includes its own implementation for resource isolation, but also has bindings to leverage other technologies (such as LXC, libvirt-lxc [44] and systemd-nspawn [45]), thus libcontainer realizes a cross-system abstraction layer for packaging, delivering, and running applications in isolated environments. The implementation and functionality of libcontainer is defined by the Open Container Initiative (OCI) [46] specification which defines the image formats, the image management interface, and the container runtime life-cycle.

Docker leverages a client-server architecture. The server is called a docker daemon, and it is responsible for the container’s runtime environment. It also has capabilities for building, running, and distributing docker containers. The Docker client is a user interface for communicating with the docker daemon. The client has several implementations, including a command line tool [47] and the Docker Remote API [48]. The Docker ecosystem includes different technologies and tools

[*]Control groups (cgroups) is a Linux kernel feature that is responsible for managing resources such as CPU, memory, disk I/O, network, etc.

[†]A namespace wraps a global system resource (process IDs, mount points, network devices, network stacks, ports, etc.) in an abstraction that makes it accessible to the processes. Within a namespace each process has its own isolated instance of the global resource. Changes to the global resource are visible to other processes that are members of the namespace, but are invisible to other processes [40].

for managing images, container and application runtime, infrastructure deployment and orchestration, etc. The Docker Hub is an image registry that stores container images in a similar way as traditional package management stores software artifacts. An image is part of a repository and has an author and a version, thus making the image and its configuration easy to distribute and discover.

Listing 2.15 illustrates how a container image can be downloaded from the Docker Hub using the command line interface of the docker daemon. The command `docker pull ubuntu:14.04` requests a download of the image of Ubuntu from the repository that is tagged with version 14.04. To realize this pull, the Docker daemon connects to the Hub and then requests this particular image of that repository, and starts downloading the image together with its configuration and dependencies. Finally, after the downloading is complete, the Docker daemon creates a hash string of the image using the Secure Hash Algorithm (SHA) algorithm. Subsequently this hash is used uniquely identify the image in the local registry of this docker daemon.

Listing 2.15: Docker pull command

```
$: docker pull ubuntu:14.04
14.04: Pulling from library/ubuntu

ba76e97bb96c: Pull complete
4d6181e6b423: Pull complete
4854897be9ac: Pull complete
4458f3097eef: Pull complete
9989a8de1a9e: Pull complete
Digest: sha256:062bba17f92e749bd3092e7569aa0\
    6c6773ade7df603958026f2f5397431754c
Status: Downloaded newer image for ubuntu:14.04
```

Using the command line client, docker can list all downloaded images along with a set of metadata for these images. Listing 2.16 shows the output of the command `docker images`, which contains the name of the repository, the repository tag, a unique identifier of the image, and additional information (such as when the image was created and stored in the Docker Hub), and its size.

Listing 2.16: Docker images command

```
$: docker images
REPOSITORY    TAG       IMAGE ID        CREATED        SIZE
ubuntu        14.04     4d44acee901c    3 days ago     187.9 MB
```

The container runtime, defines the different states of a container: created, started, paused, stopped, and deleted. In order to run an application inside an isolated environment, first a container has to be created from an existing image and then started. Listing 2.17 shows the command `docker run` which specifies the execution of a container from a particular image and causes it to execute a particular application (in this case `/bin/bash`).

Listing 2.17: Docker run command

```
$: docker run -t -i ubuntu:14.04 /bin/bash
```

In more detail, the command causes the runtime to create a container from the image `ubuntu:14.04`, and configures it according to the specified arguments. The command argument `-t` requires allocates a pseudoterminal (pty) [49], and the argument `-i` attaches the standard input to this pseudoterminal. Finally, the container starts and executes the command `/bin/bash`.

Listing 2.18 illustrates to `docker ps` command which lists the containers that are in the running state. The output of the command includes information such as the unique identifier of the container, the container image, the command that is running, and other information such as when the container was created it, when it started running, what port bindings the container has with the host operating system, and a unique name.

Listing 2.18: Docker ps command

```
$: docker ps
CONTAINER ID  IMAGE         COMMAND       CREATED        STATUS        PORTS  NAMES
91af84830636  ubuntu:14.04  "/bin/bash"   3 seconds ago  Up 2 seconds
    lonely_lichterman
```

The commands presented previously are just a subset of those available via the command line interface of the docker client. The complete set of commands can be found by running docker without any arguments or with the argument ”help”. Figure 2.5, from the documentation about the Docker Remote API, shows a state diagram of a container, along with the various commands and events that are responsible for containers transitioning between different states.

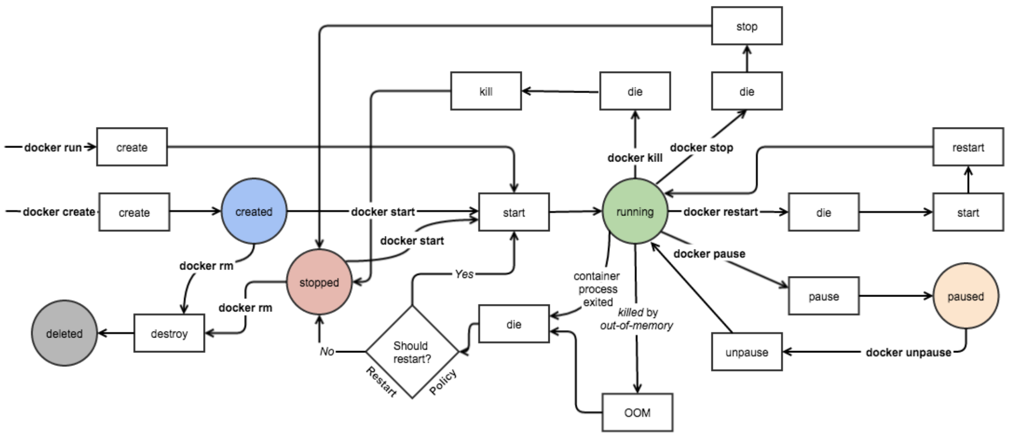


Figure 2.5: States of the container lifecycle

Listing 2.17 showed how to run the bash shell process inside a linux container.

The code snippets of Listings 2.19 and 2.20 illustrate how one can install a package in the operating system of the container and then create a new image of the resulting container.

Listing 2.19: Installing a package in the container Operating System

```
root@91af84830636:/# apt-get install traceroute
```

Using the apt package manager of Ubuntu, the root user installs the traceroute package. ~~Then~~ this running container is used to create a new image, that will contain the current state of this container.

Listing 2.20: Create a new docker image out of a running container

```
$: docker commit -m "traceroute-package" -a "KTH" 91af84830636
    my-ubuntus:traceroute
```

The command `docker commit` accepts a `-m` parameter containing a commit message, a `-a` parameter specifying the author of this commit, the id of the container that will be used to create a new image (in this case 91af84830636), the name of the repository (my-ubuntu), and the reference tag for this repository (:traceroute). Executing the command `docker images` as shown in Listing 2.21, will verify that the image was created.

Listing 2.21: List the docker images, shows the newly created image

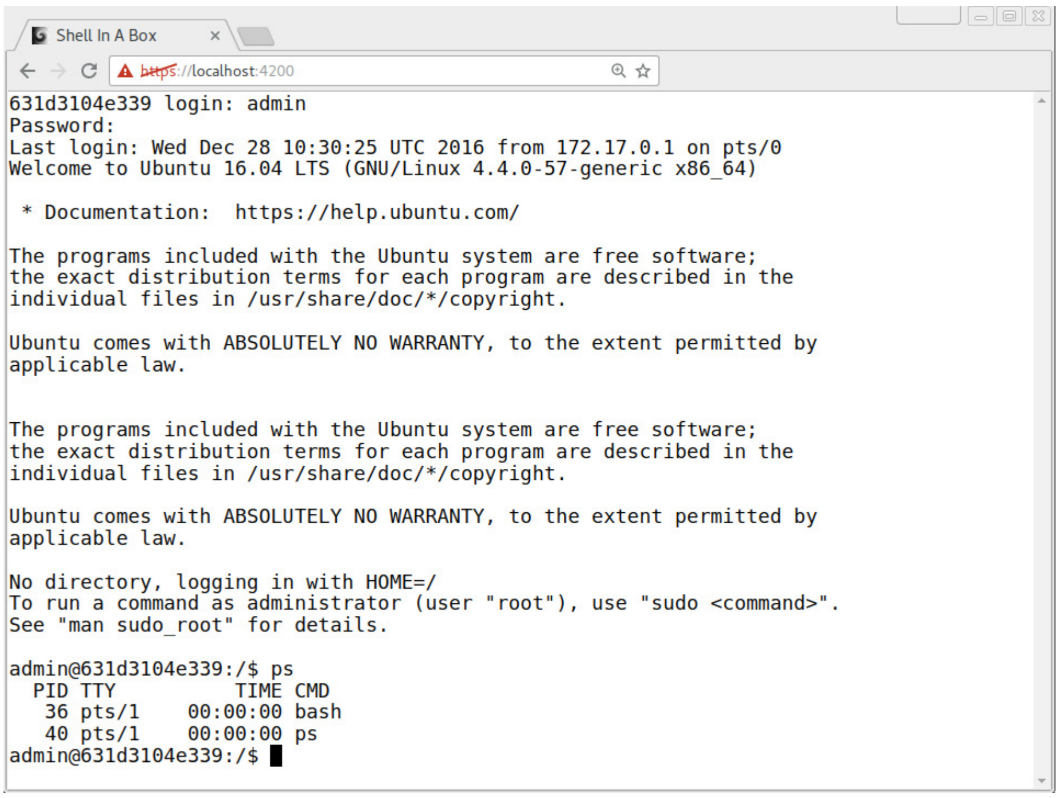
```
$: docker images
REPOSITORY   TAG          IMAGE ID      CREATED        SIZE
ubuntu       14.04        4d44acee901c  3 days ago     187.9 MB
my-ubuntus   traceroute   1261c79eb3da  4 seconds ago  166.9 MB
```

Linux containers can be used to create pre-configured machines for laboratory assignments of an Internetworking course. By creating container images tailored to the needs of each assignment, a student can focus on the exercise, while avoiding details that are not relevant to the learning process. A software solution that supports creating images and running containers on demand, can be very useful for e-learning, as it takes a student just a few seconds to access a unique laboratory environment via her web browser.

2.8 Web based shell emulators

When it comes to e-learning assisted by LMSs, students are used to performing most of their learning tasks via their web browser. Using pre-configured laboratory environments based on docker images entails the same risks as traditional labs, as the student has to install docker and manually execute a series of commands before she will be able to focus on the learning process. An alternative solution would be to support such environments in a remote server, and then simply provide the student access to the remote environment via a web browser. The software that provides access to a linux shell via a web browser is often called a web based terminal emulator. The technology that provides communication between the server (the terminal emulator) and the client (the web browser) is called Web-based Secure Shell (SSH). The server side of the implementation involves a web application that accepts requests for keyboard events and forwards these keyboard events to a secure shell client communicating with the connected SSH server. The terminal output is either passed to the client where it is converted into HTML via JavaScript or it is translated into HTML by the server before it is transmitted to the client [50].

There are several implementations of web based shell emulators, such as GateOne [51] and Shell In A Box [52]. The latter, implements a web server that can export arbitrary command line tools to a web based terminal emulator. This emulator is accessible to any JavaScript and CSS enabled web browser. The server listens on a specified port and publishes services that are displayed by a VT100 [53] emulator implemented as an Asynchronous JavaScript and XML (AJAX) [54] web application. Figure 2.6 shows the web based emulator running in a web browser that enables the user to access the remote system via an SSH session. ~~The~~ Shell In A Box web server is a process running on a docker container based on the `ubuntu:16.04` docker image, and is listening for secure TLS connections on port `4200`.



```
631d3104e339 login: admin
Password:
Last login: Wed Dec 28 10:30:25 UTC 2016 from 172.17.0.1 on pts/0
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0-57-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.


The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

No directory, logging in with HOME=/
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

admin@631d3104e339:/$ ps
  PID TTY          TIME CMD
   36 pts/1    00:00:00 bash
   40 pts/1    00:00:00 ps
admin@631d3104e339:/$ █
```

Figure 2.6: Shell In A Box emulator

The default configuration settings of the server require a TLS certificate for the server to start. If no certificate is provided, a self signed certificate is generated. In addition to the certificate, Shell In A Box requires users that want to access the linux server via an SSH session to authenticate themselves using a username and a password. Such credentials are also passed as parameters to the server startup process.

The docker image was configured to run the Shell In A Box web server according to the instructions of the GitHub repository *docker-shellinabox*[55] of the Github user *sspreitzer*. This repository, mentions two different methods of acquiring the docker image. The first downloads the image from the Docker image registry using the remote image repository `sspreitzer/shellinabox`. The downloading of the image is initiated by the `docker pull` command as explained in the previous section. The second method, specifies configuration rules to use when building the image in a local image repository with the `docker build` command.

Figure 2.6 shows that the emulator is accessible via the URL `https://localhost:4200`, where `localhost` is the host system that is running the Docker daemon, and `4200` is a TCP port of the host system that is reserved by Docker and is used to forward network packets to the container that is running the Shell In A Box web server process, and is listening for connections on the container's TCP network port `4200`. When Docker is installed on a Linux host, a network interface named `docker0` is created. The `docker0` is an Ethernet bridge[*] that enables packet transmission between physical and virtual network interfaces [57], and enables the host machine to receive and send packets to containers connected to this bridge interface. Additionally, the docker server has functionality that allows a network port of the host system to be ~~binded~~ to a network port of the container. For example, the `docker run` command accepts a parameter `-p IP:host_port:container_port` which specifies which host port should bind to a container port. The command below shows how to run a container (running a Shell In A Box web server process) from the image repository `sspreitzer/shellinabox` with version `latest`, and map the TCP port 4200 of the host system to the TCP port 4200 of the container.

```
docker run -p 4200:4200 -e SIAB_PASSWORD=123 -e SIAB_USER=
    admin -e SIAB_SUDO=true sspreitzer/shellinabox:latest
);
```

The parameter `-e` specifies environment variables that are saved in the in the linux operating system of the container during its creation. Those environment variables are parsed by the Shell In A Box web server initialization script (~~such~~ variables are explained in detail ~~by~~ GitHub repositories referenced aboce) to configure the web server, the authentication credentials, and `sudo` access for the Linux user.

[*]A bridge is a way to connect two Ethernet segments together in a protocol independent way. Packets are forwarded based on Ethernet address, rather than IP address (like a router). Since forwarding is done at Layer 2, all protocols can go transparently through a bridge [56].

2.9 Related work

The support for interoperability specifications by several LMSs has allowed rapid experimentation and implementation of external application frameworks that offer a variety of on-line training events for various Computer Science courses. This section presents some of these frameworks and describes how they are relevant to this project.

2.9.1 EDURange

Designing on-line training environments for the field of cyber security requires overcoming some technical constraints, such as high availability and scalability, and pedagogical limitations, such as teaching analysis skills to understand complex systems and concepts via practicing [2]. EDURange addresses these issues by designing an open source framework that provides interactive security exercises in an elastic cloud environment [58].

EDURange is a software framework, designed to work on Amazon Elastic Compute Cloud (EC2) [59]. It allows teachers to easily build and scale dynamic virtual environments to host cybersecurity training [60]. This framework provides ease of use for instructors, by offering the flexibility to specify exercises at a high level and allowing the instructor to configure different aspects of the training scenarios in order to provide a tailored learning experience that focuses on analysis skills.

2.9.2 GLUE!

Group Learning Uniform Environment (GLUE!) is a middle-ware integration architecture that aims to standardize the integration of existing external learning tools into several LMSs [61]. It facilitates the instantiation and enactment of collaborative learning situations within LMSs, by using the distinctive administrative features of these systems to manage users and groups. LTI and the Sharable Content Object Reference Model (SCROM) are two specifications for the integration of external learning tools into an LMS. However, each LMS usually supports only a single interoperability specification; thus, developing a universal external tool requires a substantial development effort to support the different interoperability standards. In contrast, GLUE! proposes a software architecture that takes advantage of the common integration features of LMSs to integrate multiple existing learning tools into multiple LMSs.

2.9.3 INGInious

Programming exercises are the most common form of practice for students learning CS. Traditionally, the evaluation of these exercises, requires grading of reports, reading source code, and testing source code, thus making it time consuming, especially for large classes (i.e., large numbers of students). INGInious

[7, 62, 63, 64] is a software framework that empowers instructors to easily construct coding tasks and it supports automatic evaluation and grading of the code, thus providing both students and teachers with constructive feedback.

The framework consists of two main components: the frontend and the backend. The frontend provides a web interface where students perform programming tasks and an administration module that allows instructors to design these tasks. The backend is responsible for running and grading the code inside remote isolated Linux containers. Each container is specifically built for a particular programming language, according to configuration provided by the instructor or the administrator of the system, thus supporting the evaluation of tasks written in any programming language that runs in a Linux environment.

One of the main features of INGInious is that the frontend component can be used either as a stand-alone web application or as an external learning tool that is integrated into an LMS using the LTI specification. Additionally, the backend component scales horizontally very easily, since it utilizes a docker container for every task request, therefore it is suitable for MOOC platforms.

A programming task in INGInious is designed using a configuration file (`task.yaml`) that identifies the problem to be solved by the student, and the evaluation process, a template file (template.py) that presents the task to the student, and defines the input field for the code, and finally, a file (`run`) that executes the student code, and validates the output. The following code samples show the minimum configuration required by the instructor, to design a simple "Hello World" task in Python. Listing 2.22 is the `task` file. It starts with key-value pairs that are used to describe the `name` and `context` of the task.

Listing 2.22: Definition of a task in task.yaml

```
name: "Hello World!"
context: "In this task, you will have to write a python script
    that displays 'Hello World!'."
problems:
  question1:
    name: "Let's print it"
    header: "def func():"
    type: "code"
    language: "python"
limits:
  time: 10
  memory: 50
  output: 1000
environment: "default"
```

Then it defines the `problems` that have to be solved to complete this task. Each problem has a unique name within the task (`question1`) and a series of metadata such as the programming language to be used for solving the problem, and the text input to print in the input form. Finally it contains other metadata that defines the resources of the virtual environment that will be used to evaluate the code.

Listing 2.23: Code input of question1 in template.py

```
def func():
  @ @question1@@

  func()
```

Listing 2.23 defined the input into field in which the student will input their code. Finally, the `run` file defined in Listing 2.24, is a shell script, that parses the input code using the INGInious commands `parsetemplate`, then evaluates the expected output against the results of the input function using the command `run_student`. Finally it prepares the result of the task using the `feedback` command.

Listing 2.24: Evaluation of student code by the run file

```
#! /bin/bash

# Parse the template and put the result in studentcode.py
parsetemplate --output studentcode.py template.py

# Verify the output of the code...
output=$(run_student python studentcode.py)
if [ "$output" = "Hello World!" ]; then
  # The student succeeded
  feedback --result success --feedback "Success!"
else
  # The student failed
  feedback --result failed --feedback "Your output is $output"
fi
```

Detailed information for specifying a task in INGInious platform can be found in the official teacher documentation [65]. As part of the research in this thesis project, the LTI component of INGInious was configured with Canvas LMS, to perform sample programming tasks like the "Hello World!" code that was explained earlier.

2.10 Summary

Canvas LMS is an open source system that aims to assist in every aspect of the learning process. It offers functionality for e-learning activities such as rich media, interactive quizzes, methods for automatically evaluating assignments, and finally allows developers to design and integrate their own learning tools via the LTI specification. The LTI specification standardizes the method of integrating external learning applications in LMSs via XML configurations, and allows the LMS to exchange structured messages with a TP to share information such as user sessions, and learning outcomes.

LTI is only one of the several specifications for integrating learning applications into LMSs. GLUE! is a middleware implementation that supports the integration of external learning tools into different LMS that implement different specifications.

Designing assignments for an Internetworking course relies heavily on laboratory environments. Creating and managing such environments can easily be performed by using Linux Containers. Docker offers a high level API that allows to create container images with provisioned software, tailored to the requirements of different assignments. The Docker runtime can nearly instantly create and execute software realizing a particular laboratory environment. Using web based shell emulators, students can access the environment and focus on the learning process, rather than configuring the environment themselves.

Similar approaches that address the problem of virtual laboratory environments, and automatic assignment evaluation have been proposed by researchers in other fields of Computer Science. These approaches were evaluated, and provided useful guidelines for designing the software artifact of this project. The Edurange project focuses on devising a set of exercises that train students in the Cybersecurity domain. Moreover, if offers a method for deploying the framework in cloud infrastructures, to increase availability of the system for students and instructors, and also reduce the cost of hosting the framework for educational institutions. The Inginious framework focuses on providing an environment for evaluating coding assignments in all programming languages whose runtime is supported by the linux kernel. The system offers high availability for evaluating code using unit tests, and the actual evaluation is performed within a docker container.

Chapter 3

Methodology

This thesis project is carried out using the Design Science research method. This type of research focuses on the design and construction of IT artifacts that have utility in the real world, in this case as an application environment, and aim to improve domain-specific systems and processes. In the context of this research, the real world problem is the lack of interactive virtual laboratory environments in the form of e-learning tools.

3.1 Research Process

Vijay Vaishnavi and Bill Kuechler in *Design Science Research in Information Systems* [66] describe the process for performing Design Science Research in the following five steps: Awareness of the Problem, Suggestion, Development, Evaluation, and Conclusion. In the scope of this project, the first two steps are reflected by the Introduction and Background Chapters. The literature study that was performed, provided understanding of the problem, of how other researchers have addressed similar problems, and how existing technologies can be combined to devise a solution for the problem of this thesis work. The Development step is reflected by Chapter Implementation, which describes the designed software artifact. The Evaluation step, reflected by the corresponding Chapter, evaluates the functionality of the artifact against a set of criteria (listed in ~~the~~ section). Finally, the Conclusion step, summarizes the results, and proposes a series of actions to be taken as part of the future work of this project.

3.2 Evaluation Process

The literature study that was carried out within the scope of this project revealed two important software solutions (EduRange and INGinious) that address similar problems in different domains of CS. Further analysis ~~on the~~ functionality and implementation ~~of those projects~~ inspired the work of this project and ~~concluded~~ a ~~few~~ high level requirements ~~that are listed below:~~

- The laboratory environments can be designed using Docker containers, in a similar way that INGInious ~~is using~~ them to perform the evaluation of student assignments. ~~Generating a~~ laboratory environment can be realized by creating a container image which includes all software require~~ments of~~ an Internetworking assignment.
- High availability of these environments can be achieved by creating and running a docker container for each student session. This approach was ~~both supported~~ by INGInious for the evaluation of each coding assignment~~,~~ and Edurange which relies on preconfigured virtual machines that are used to facilitate Cybersecurity training~~.~~
- The instructor should be able to dynamically update the underlying software and the assignments, similarly to the way INGInious creates a new assignment.
- ~~Design a system that is not~~ specific to a cloud infrastructure provider. ~~The~~ Docker runtime is supported by most linux operating system distributions which can run on dedicated or virtual hardware.

Furthermore, a series of goals were decided to be used as guidelines for the system design. These guidelines focus on the interaction ~~for~~ the main two user roles of the system, the instructor and the student~~, and are described below~~:

- The instructor should have complete control over the software used for a particular assignment (for example install the software and create a container image that will be used for a particular assignment).
- The instructor should always know which container images exist in the system, and should have sufficient privileges to delete and create these images.
- The system should provide a way for the instructor to access a laboratory environment, in similar to the way a student is expected to access it.
- The system should provide sufficient configuration for the instructor to create an assignment in Canvas LMS and connect it with a particular container image.
- The student should be able to launch a laboratory environment from Canvas LMS, by simply pressing the assignment button. The resulting container should be available to the user instantly.

The evaluation of the software artifact was performed in two steps. First, the system was evaluated against the requirements mentioned earlier to validate whether the solution is align~~ing~~ with the goals of this project, and then, additional evaluation methodologies such as unit testing were used to test that the implemented code ~~was~~ perform~~ing~~ as intended.

Chapter 4

Implementation

The artifact that was designed within the scope of this thesis work consists of two different modules: a TP and a Tool Client. The TP enables students to access a laboratory environment via LTI integrations with Canvas LMS, while the Tool Client provides an administrative tool which exposes functionality enabling the instructor to preconfigure the laboratory environments and configure the integration with the LMS acting as a TC. Figure 4.1 presents a high level overview of these user interactions with the system.

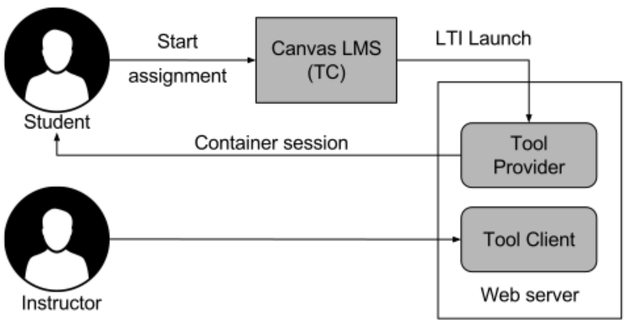


Figure 4.1: High Level Overview of the System Architecture

Tool Provider (TP) in figure.

The TP and the Tool Client are not separate systems, but different components of the same web server ~~that is allowing~~ them to share common functionality such as the container runtime, and management of user sessions.

4.1 Software architecture

Section 2.3 introduced an example of a web server which had the role of a TP that accepted and authenticated requests from Canvas LMS to launch assignments. Similarly to that approach, an HTTP web server was used to support the functionality of the LTI Tool Client and the LTI Tool Provider. The Docker daemon provided the required functionality to manage the container runtime and container image manipulation. This functionality was exposed to the web server via a Docker Remote API client library. API endpoints accessible via HTTP request methods were developed as part of the web server functionality that consume~~d~~ the Docker client library to support the various use cases of the Tool Client and the TP. The web server ~~was~~ communicat~~ing~~ with two different data stores: (1) the session and (2) the persistent storage for storing and retrieving user session information and storing and retrieving assignment configurations respectively. Figure 4.2 presents these components.

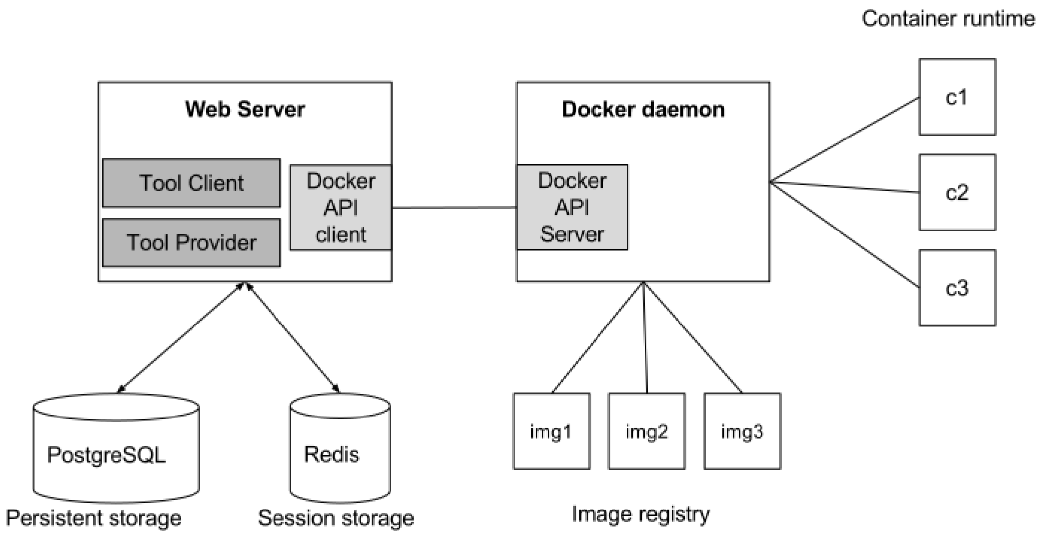


Figure 4.2: Architecture of the system components

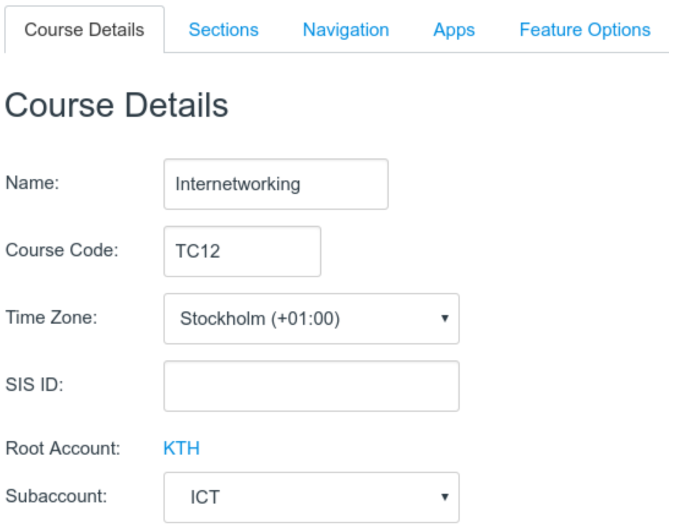
4.1.1 Canvas LMS

Canvas LMS was used during the development phase of this project to understand and test the functionality of the LTI integration with the TP. Canvas is based on the Ruby on Rails framework [67] and has several software dependencies. To facilitate the installation of Canvas, a virtual machine was configured to run the Ubuntu 14.04 operating system. The software dependencies of Canvas were installed in the operating system as explained in the "Quick Start" wiki page of the official Canvas LMS GitHub repository[68]*.

After the installation was complete and the system was running successfully, Canvas was configured to have an administrator account. This account was used to register two additional user roles (the *instructor* and the *student*), the institution (*KTH*), the department (*ICT*) and a course (*Internetworking*). The instructor user

*A simplified method for installing Canvas LMS in a virtual machine using Vagrant [69] and VirtualBox [70] was used in this project. This method is documented in a public GitHub repository [71].

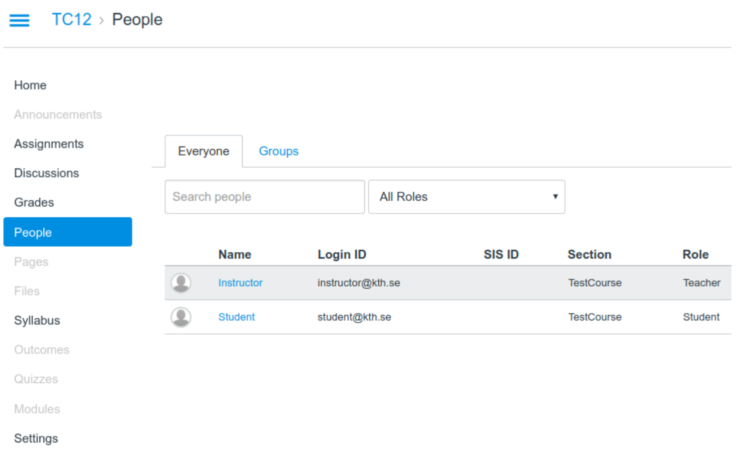
was configured to have the Canvas role *teacher* for this course, while the student user was configured to participate in this course. Figure 4.3 shows the configuration performed via the Canvas User Interface (UI).



Course Details

Name:	Internetworking
Course Code:	TC12
Time Zone:	Stockholm (+01:00)
SIS ID:	
Root Account:	KTH
Subaccount:	ICT

(a) Structure of a course in Canvas



Name	Login ID	SIS ID	Section	Role
Instructor	instructor@kth.se		TestCourse	Teacher
Student	student@kth.se		TestCourse	Student

(b) People participating in a Canvas Course

Figure 4.3: Sample configuration of a course and its participants in Canvas LMS

The configuration of course assignments was explained in Figures 2.3 and 2.4 of Section 2.4.1, thus this material is not included in this section.

4.1.2 Web server

The TP and Tool Client components are sets of API endpoints that are served by the same web server. Each endpoint is responsible for carrying out a specific task, such as authentication, exposing system resources to users and launching LTI integrations. These endpoints were implemented using the Go programming language (Go or Golang) [72, 73]. The web server itself, is implemented in Go and is part of the standard `net/http` [74] package. Listing 4.1 shows an example of an HTTP web server that is configured to listen for TLS connections on port 443 and has a single endpoint that ~~is~~ repl~~ying~~ to HTTP GET requests for ~~to~~ the root (`"/"`) URL path.

Listing 4.1: Golang simple HTTPS web server

```
import (
  "net/http"
  "github.com/julienschmidt/httprouter"
)

func handler(w http.ResponseWriter, req *http.Request, _
    httprouter.Params) {
  w.Header().Set("Content-Type", "text/plain")
  w.Write([]byte("This is an example server.\n"))
}

func main() {
  router := httprouter.New()
  router.GET("/", handler)
  http.ListenAndServeTLS(":443", "cert.pem", "key.pem", router
    )
}
```

The line `import "net/http"` includes the package which implements the HTTP web server. The line `import "github.com/julienschmidt/httprouter"` includes a Go package developed by Julien Schmidt [75], which maps URL paths such as the root path `"/"` to HTTP Request methods (such as GET, POST, DELETE, PUT, etc.), and Go functions such as `handler(w http.ResponseWriter, req *http.Request, _ httprouter.Params)` that processe~~s~~ the corresponding HTTP request.

Go language is a strictly typed language, ~~like~~ C and C++. The declaration of variables, function parameters, and function return types is performed by first writing the corresponding parameter, variable or function name ~~succeeded~~ by its type. In the example above the function `handler` has three parameters `w`, `req`, and _. The type `http.ResponseWriter` is an interface that exposes functionality such as setting an HTTP response header and writing an HTTP response. The prefix `http.` ~~dictates~~ that `ResponseWriter` is a type that is part of the package `http`. Functions, types and variables that are declared in a package and start with an uppercase letter, are exported by the compiler, ~~and~~ are available for use in other packages, by first invoking the package name followed by a dot, and then ~~inferring~~ to th~~at~~ type, function, or variable. The type `http.Request` is an interface that exposes functionality for reading request parameters, form data, etc.~~, and~~

~~t~~he type `httprouter.Params` is a key-value data structure that maps ~~http~~ request parameters names to their values. Since such data are not relevant in the function, instead of naming the parameter, the blank identifier (represented by the underscore symbol) is used.

The server starts executing following the call to `ListenAndServeTLS(":443", "cert.pem", "key.pem", router)` function. This first parameter is the port number that the server will be listening on for incoming TLS connections, `cert.pem` and `key.pem` are the TLS certificate and key respectively that were generated similarly to the instructions of Section 2.4.2, and router is the URL router created above. The router parameter is declared and initialized using the symbols `:=`. This syntax tells the compiler to infer the type of the variable router from the return type (Router) of function `New()` that is declared in package `httprouter`. The call `router.GET`, takes two parameters, the URL path `"/"` and the function `handler`. The function GET, registers that every GET HTTP Method to the root path should be handled by function handler.

The implementation ~~of~~ this project, ~~contains~~ a series of functions such as the handler function mentioned earlier, that implement functionality, such as creating a docker image, launching an assignment, etc. Each function is mapped to a specific URL path and an HTTP Method. Table 4.1 shows the URL Paths, the HTTP Method, and explains the functionality realized by each endpoint.

Table 4.1: Endpoints of the HTTP Web Server

URL Path & HTTP Method	Endpoint functionality
`/admin/login` POST	Implements the login functionality for the admin user of the LTI Tool Client
`/admin/logout` GET	Implements the logout functionality for the admin user of the LTI Tool Client
`/admin/containers/run/:id` POST	Handles the container run functionality for the admin user of the LTI Tool CLient. Parameter `:id` is the identifier of the image to be used for creating and starting a container.
`/admin/containers/kill/:id` DELETE	Handles the container kill and remove functionality for the admin user of the LTI Tool CLient. Parameter `:id` is the identifier of the running container.
`/admin/containers/commit/:id` POST	Handles the image creation functionality for the admin. It uses a specific running container as a seed for the new image. Parameter `:id` is the identifier of the running container.
`/admin/images` GET	Lists all container images available to the admin user of the LTI Tool CLient.
`/admin/images/history/:id` GET	Returns ~~the~~ information ~~of~~ a particular image to the admin user of the LTI Tool CLient. Parameter `:id` is the identifier of the image.
`/admin/images/delete/:id` DELETE	Deletes a particular image. Parameter `:id` is the identifier of the image.
`/lti/launch/:id` POST	The LTI Tool Provider. Handles the LTI Launch request. Parameter `:id` is the identifier of the image that should be used to create and start a container for this particular request.
`/ui/*filepath` GET	Handles requests for all static files that are located in a custom directory. The syntax of the URL route is related to the specification of `httprouter` package.

4.1.3 Docker Remote API Consumer

The web server communicates with the Docker daemon via the Docker Remote API [48] to create and delete images, create, start, and delete docker containers. The web server ~~is using~~ the Go implementation of the Remote API Client library [76] to ~~perform~~ requests to the Docker server. The version of the Docker Server ~~that was~~ used in this implementation is `1.12.4`, and the version of the server API was `1.24`. The version of the API is very important when initializing the client library from the Go code, as a matching version ensures the client will communicate using the same version of the API calls that the server is responding to. This library has functionality similar to the Docker command line client that was introduced in Section 2.7. Listing 4.2 shows how a request is performed by the client `Cli` to start a container using the `ContainerStart` function of ~~the~~ `Cli`. It is assumed that the container was previously created~~,~~ using the `ContainerCreate` function.

Listing 4.2: Start container request

```
Cli.ContainerStart(context.Background(), containerID, types.
    ContainerStartOptions{})
```

The first parameter expects a variable of type Context[*], the second parameter is the unique identifier of the container to start. The last parameter is a Go struct of type `types.ContainerStartOptions` and its members are initialized with the *zero values* of their corresponding type using the curly brackets {}[†].The `ContainerStartOptions` is part of the Docker Checkpoint & Restore [79] functionality that is not relevant to this project, ~~thus not explained~~.

The functionality of the TP relies on facilitating a connection to a laboratory environment via the web shell emulator Shell in a Box. In order to support this functionality, a docker image with a pre-configured installation of Shell in a Box was chosen to serve as the initial container image of the system. The Tool Client allows the administrator to choose this initial image as a seed for creating new laboratory environments. The docker daemon stores images in its local image registry from various remote image repositories. The system was designed to ~~have~~ access ~~to~~ only a particular subset of images of the local image registry. Section 2.7 explained that a container image is identified by a repository, ~~has an~~ author, and a version. In Docker Remote API, the repository and the version of an image are identified by

[*]The package context defines the Context type, which carries deadlines, cancellation signals, and other request-scoped values across API boundaries and between processes [77]. The background function returns a non-nil, empty Context. This context is never canceled, has no values, and has no deadline. The context is typically used by the main function, initialization, and tests, and as the top-level Context for incoming requests.

[†]The Go language specification [78] describes the initialization of variables as follows: When storage is allocated for a variable, either through a declaration or a call of new, or when a new value is created, either through a composite literal or a call of make, and no explicit initialization is provided, the variable or value is given a default value. Each element of such a variable or value is set to the zero value for its type: false for booleans, 0 for integers, 0.0 for floats, "" for strings, and nil for pointers, functions, interfaces, slices, channels, and maps.

a parameter named `RepoTags`, i.e, an image of Ubuntu with version 14.04 has the RepoTag `ubuntu:14.04` where the semicolon is the delimiter between the repository and the version. The system is allowed to operate on images that belong to ~~only~~ a particular repository, to satisfy the requirement for containers that can be accessed via a web shell emulator. In this implementation the repository was named `dc` and ~~could not~~ be changed by any user of the system, while the image version ~~was used to~~ identify the different images~~,~~ and ~~it was~~ a parameter that the administrator ~~could~~ set when a new image ~~was~~ created.

The source code of this implementation includes a Go package named `dc` (named after docker containers). This package is responsible for manipulating the images of the homonym repository, initializes the API client, and contains functions that consume the Docker API Client library. These functions are invoked by several HTTP route handlers in the Tool Client and the Tool Provider to deliver the desired functionality to the end users. The list below introduces the names of these functions along with brief descriptions of their intended functionality~~, that~~ is explained in more detail in the next sections of this chapter.

- `ListImages` requests the Docker API to return all container images, and afterwards, iterates over the results to filter only images of the `dc` repository. This function is invoked by the endpoint `/admin/images`.

- `ImageHistory` requests the Docker API to return detailed information ~~of~~ a particular image such as the author, the `RepoTags`, when was it created, and a text message that identifies the creation of the image. This function is invoked by the endpoint `/admin/images/history/:id`.

- `ImageRemove` requests the Docker API to remove a particular container image from the local repository. This function is invoked by the endpoint `/admin/images/delete/:id`.

- `RunContainer` first requests the Docker API to create a container from a specific image, and then start the container. ~~It~~ returns configuration information for the user to access the container via the web SSH emulator. This function is invoked by the endpoint `/admin/containers/run/:id` and the endpoint `/lti/launch/:id`.

- `RemoveContainer` first requests the Docker API to stop a running container, and then to remove it from the container runtime (used after a container session expires for a user). The laboratory environment is purged and is no longer available for the system or the users. This function is invoked by the endpoint `/admin/containers/kill/:id`.

- `CommitContainer` requests the Docker API to create a new image using a running container as seed. ~~An usage example is~~ when an instructor is running a container instance to configure software for a new laboratory environment. Once she is done with the configuration, she performs a request to "commit the container" as an image, in the local repository. This function is invoked by the endpoint `/admin/containers/commit/:id`

4.1.4 Session Storage

The system ~~is using~~ an in-memory key-value storage to store and retrieve information for user and container sessions. When a container is running for a particular user, whether that user is an administrator of the Tool Client~~,~~ or a student ~~that~~ is accessing a laboratory environment via the LMS, information ~~for~~ the system to uniquely identify the user~~, and~~ the running container is stored in this storage. This mechanism prevents users from running multiple laboratory environments at the same time, thus preventing resource exhaustion.

The session storage is ~~powered~~ by the open source in-memory data structure store Redis [80]. The server ~~is~~ communicat~~ing~~ with Redis using the client library for Go [81]. The information ~~that is~~ stored for a student session has the format `key-value`, where the key is a unique identifier for the user, ~~and~~ the value is ~~is~~ a JSON object containing information ~~for~~ the running container. Every data entry has a Time To Live (TTL) value that defines when the key expires. For the system, an expired key means that the session has expired, ~~and~~ a container should neither exist in a running state~~,~~ nor the user ~~should~~ be able to access it. The code sample ~~below~~ shows the value of a Redis key~~,~~ used to identify a running container for an admin user of the Tool Client:

Listing 4.3: Redis session value for a container run configuration

```
{
  id:b79803d58414fd7786,
  port:4200,
  username:admin,
  password:password
  url:https://localhost:4200
}
```

The `id` is the identifier of the container. A Shell in a Box web server that is running in a container ~~is~~ listen~~ing~~ for connections on a specific port number. The attribute `port` is the port number that the host system is using to forward data packets to the port of the running container. The attributes `username` and `password` are additional parameters that the user should use to authenticate herself to access the emulated unix shell, and finally, `url` is the URL containing the hostname and the port to access the shell emulator. For an admin user, such entry has a key with format such as `run:adm:7ff10abb653dead4186089acbd2b7891`, where `run:adm:` is the prefix, and `7ff10abb653dead4186089acbd2b7891` is a hash of the administrators numeric account identifier. For a student the corresponding key has the format `run:usr:7272818191010`, where the prefix is `run:usr:` and `7272818191010` is the user identifier that is returned by Canvas via the LTI Launch integration.

Additional key-value data entries are stored in Redis such as HTTP cookie information for users of the Tool Client. Such keys have the format `adm:7ff10abb653dead4186089acbd2b7891`, ~~has~~ a TTL of one day, and ~~is~~ created when the administrator successfully authenticates herself to the Tool Client.

4.1.5 Persistent Storage

The system ~~is using~~ a Relational Database Management System (RDBMS) to store persistent information such as login credentials for the administrative user. The database server is PostgreSQL [82] with version number 9.6. A combination of two Go packages are used to establish connections, store and retrieve data from PostgreSQL database. The first is `database/sql` [83]~~, and~~ provides a generic interface interface ~~around~~ SQL databases. This package is intended to be used in conjunction with a database driver that implements the SQL interface functions. In this implementation the database driver is provided by the Go package `pq` [84].

Although the data stored in the persistent storage are not enough to justify the use of a RDBMS, ~~it~~ was chosen to support future engineering choices that will extend the functionality of the system, such as storing information for assignments~~,~~ and analytics regarding the usage of the system ~~that~~ will be available to the instructor via the Tool Client interface. This future work is documented in ~~the corresponding section of this report~~.

The relational schema consists of a single table called `admins` that stores information such as the unique numeric identifier (`id`) of an admin user, the `username` and `password` that the administrator is using to sign into the Tool Client, a status that can be `active` or `deleted`, and optional information such as the `name` of the user and~~,~~ timestamps that ~~dictate~~ when the admin account was created~~,~~ and when ~~was the~~ last ~~time the user~~ signed into the Tool Client. ~~The code listing below~~ presents the Structured Query Language (SQL) database schema definition ~~with~~ PostgreSQL specific syntax~~;~~

```
CREATE TYPE enum_admin_status AS ENUM('active', 'deleted');
CREATE TABLE admins(
  id SERIAL PRIMARY KEY,
  username varchar(60) NOT NULL UNIQUE,
  password varchar(100) NOT NULL,
  name varchar(100),
  status enum_admin_status NOT NULL DEFAULT 'active',
  created_at TIMESTAMP WITHOUT TIME ZONE DEFAULT
   CURRENT_TIMESTAMP,
  last_login TIMESTAMP WITHOUT TIME ZONE
);
```

The method for accessing and storing data using the `lib/pq` package in Go is ~~not important~~ for this project, ~~and~~ ~~is~~ left out~~, as~~ the official documentation of the package covers such methods in detail.

Binding network ports of the host system to container ports

The web server of the Tool Client and the TP is required to run multiple containers for several users at the same time. ~~E~~Each container is running the Shell In A Box web server process ~~as explained in Section 2.8~~. In order for the shell emulator to be accessible from a user's browser, network packets from the host system should be forwarded to the corresponding docker container via a network bridge interface (docker0). This is achieved by binding ~~a~~ TCP ports of the host system to the TCP port 4200 of each container running the shell emulator web server process. To avoid port collision on the host server, the system ~~is~~ reserv~~ing~~ and utiliz~~ing~~ a specific port range between `4200-4399`. This means that the system has the ability to support 200 running containers at the same time~~, and~~ the web server can serve ~~the~~ maximum ~~number~~ of 200 requests to run containers via the `/admin/containers/run:id` and `/lti/launch:id` endpoints.

Several mechanism have been used to avoid port collision~~,~~ and guarantee that the system has sufficient port resources to create new containers. During the startup process of the web server, a key-value data structure is initialized that stores the TCP port numbers as keys, ~~and~~ values of boolean type ~~that dictate~~ whether the port is in use by a container or not. ~~Below is t~~he definition of the data structure in Go cod~~e:~~

```
type portResources struct {
  portsAvailable map[int]bool
}
```

`PortResources` is ~~the~~ struct that contains the map data structure `portsAvailable`.

When the function of the `dc` package `RunContainer executes` following a request ~~for running~~ a container ~~to~~ any of the `/admin/containers/run:id` or `/lti/launch:id` endpoints, the system will check if there is an available port in the map, and if so it will set ~~its~~ value to `true` to indicate that the port is in use. Similarly, when the function `RemoveContainer` is invoked, the system will locate the port in the map~~,~~ and set its value to `false`, thus making the port reusable. This functionality covers the use cases when users manually request to run and kill containers.

Section 4.1.4 introduced the user sessions and their corresponding running configuration keys in Redis~~, that~~ are defined to expire after some specific TTL. When a container session expires, the container is still running, but the key is removed in Redis. This indicates that the container should be terminated, and the port should become available for reuse by the system. A module named `PeriodicChecker` been developed, that periodically checks whether the ports used by Docker containers are consistent with the `PortResources` map entries, and the session keys in Redis. If for some reason a container is running and is using a port within the specified range, but the corresponding map entry does not have the value `true`, the mechanism will fix this inconsistency. Similarly, the system will check for incosistencies in the Redis storage. If a key is missing for a container that is running, it assumes that the container has expired, and the container should be killed, and the port resources should be returned to the system for use. Algorithm 1 shows pseudocode that describes the the functionality of the `PeriodicChecker` module.

Algorithm 1: Module PeriodicChecker

```
$usedPorts$ := getPortsOfDockerContainers()
foreach $port \in PortResources$ do
    if $port \in usedPorts$ then
        $PortResources[port]$ := true
    else
        $PortResources[port]$ := false
foreach $port, containerID \in usedPorts$ do
    if $port \notin redisPorts$ then
        RemoveContainer($containerID$, $port$)
```

The first line ~~is~~ performing a series of calls to the Docker Remote API, to determine ~~the~~ which containers are running in the system, and what host ports used for these containers. The function returns the `usedPorts` map, with ~~keys~~ the ports, ~~and~~ values the container identifiers. The following loop iterates over the `PortResources` map, and resets its entries. A port entry of `PortResources` that exists in `usedPorts`, gets the value `true`, while an entry that does not exist in `usedPorts` get the value `false`. Finally, the last loop, iterates over the entries of the `usedPorts` map, checks whether an entry for such port exists in redis storage, and if it does not, it invokes the `RemoveContainer` function of the `dc` package ~~that~~ requests the Docker API to remove the container from the Docker runtime, and then releases the port from `PortResources` map.

4.2 LTI Tool Client

The LTI Tool Client acts as an administration panel for the system. It allows a user with admin privileges* to create and delete docker images that act as pre-configured laboratory environments~~, and~~ are used ~~to~~ configure an LTI integration for a course assignment in Canvas. This section explains the intended functionality of the Tool Client, presents the web pages of the Tool Client UI, and describes the key concepts the implementation.

Authentication

The API endpoints consumed by the Tool Client UI have access restrictions to prevent unauthorized requests, i.e., requests not from an administrator. The process for authenticating an administrator and creating a user session is

*A user with admin privileges is defined to be a user that has an entry in the table `admins` of the PostgreSQL database~~, and~~ the `status` column ~~has~~ the value `active`.

performed by submitting a web form with username and password parameters. The form is presented in Figure 4.4.

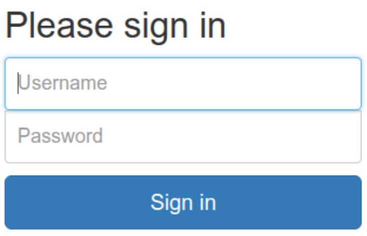


Figure 4.4: Sign in form - LTI Tool Client Interface

When the *Sign in* button is clicked by the user, the Javascript function shown in Listing 4.4 executes~~, and~~ perform~~s~~ the following steps. First it ~~gets~~ access ~~to~~ the form parameters, then it sets the HTTP request URL to `/admin/login`, the header Content-Type to `application/json`, then it sets the HTTP request body to contain the following JSON object:

```
{
  "username": "admin",
  "password": "password"
}
```

Listing 4.4 shows the jQuery function `jQuery()`, which locates the HTML form using the HTML class attribute with value `form-signin`. The following function call `.submit` specifies which function will execute, when the submit button (Sign in) of the HTML form is clicked. The body of that function calls the jQuery function `$.ajax()` [85] which performs an AJAX request to the `/addmin/login` endpoint. The `$.ajax()` function has the parameters `url`, `type`, `dataType`, `data`, `contentType`, `success`, and `error`.

Listing 4.4: Javascript function consuming the /admin/login/ endpoint

```
jQuery('.form-signin').submit(function() {
  $.ajax({
    url: "/admin/login",
    type: 'post',
    dataType: 'json',
    data: JSON.stringify({
      username: $("#userName").val(),
      password: $("#Password").val(),
    }),
    contentType: "application/json",
    success: function(data) {
      window.location.replace("/ui/images.html");
    },
    error: function(response) {
      // error handling
    }
  });
});
```

The parameter **url**, specifies the url path that is used to perform the HTTP request, and correspond to the server endpoint that handles the request. The parameter type, is the HTTP method to use for this request. The parameter `type` specifies the format of the data that is passed in the HTTP request body, and the parameter `data`, contains the JSON object shown earlier that is generated by locating the HTML input elements with identifiers `#userName` and `#Password` using the jQuery function `#("")`, and extracting their values by invoking the jQuery function `.val()`. These data are converted to a JSON object using the function `stringify()` of the Javascript object `JSON`. The parameter `contentType` sets the HTTP header to `application/json`, and is the HTTP request header that the server is expecting. The parameter `success` specifies the javascript function to execute if the server responds with an HTTP StatusOK status code (200), while parameter `error` specifies the function ~~that should~~ execute if the HTTP response contains a status code different than 200*.

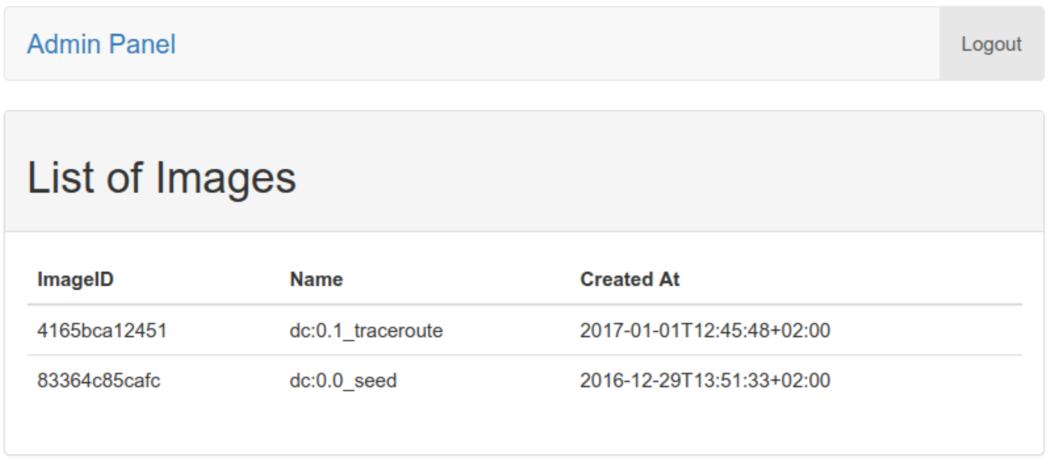
If the server replies that an error occurred, a corresponding error message is presented to the user, while if the request was successful, the user is redirected to the home page of the Tool Client interface.

The server validates the form data and compares the given parameters with the corresponding values of the user entry in the persistent storage. If the credentials match, a user session is created and stored in the session storage, and then an HTTP cookie is created~~, with~~ ~~unique~~ information ~~about~~ this administrator. Afterwards, every subsequent request to other endpoints of the Tool Client, verifies that a cookie exists for that particular user~~,~~ and that its value matches an existing entry in the session storage. If no such value exists either in the cookie value~~,~~ or in Redis, the user is redirected to th~~is~~ sign in form.

*The HTTP response status codes are explained in detail by RFC 7231 [86].

Home Page - List of Images

The home page of the application is titled List of Images" (see Figure 4.5). This page contains a table with three columns: the image identifier (ImageID), the name of the image (Name), and the date the image was created (Created At). The user can click on each row of the table to go to the next page named "Image History" which provides more detailed information about this specific image.



List of Images

ImageID	Name	Created At
4165bca12451	dc:0.1_traceroute	2017-01-01T12:45:48+02:00
83364c85cafc	dc:0.0_seed	2016-12-29T13:51:33+02:00

Figure 4.5: Tool Client page “List of Images”

When the browser renders the HTML elements of this web page, it performs an HTTP GET request (as shown in Listing 4.5) for URL path `/admin/images` using the same `$ajax()` function that was presented earlier, with different parameters, ~~such as~~ `GET` HTTP method as `type`, `/admin/images` as `url`. The server upon successful authentication of the request~~, requests~~ calls the `dc` function `ListImages` to request the Docker Remote API to return the list of images of the `dc` image repository, and afterwards, prepares a JSON array as a response, containing image information as a response.

```
{
  "data": [{
    "Id": "00db67e76050",
    "RepoTags": "dc:0.1_traceroute"
    "CreatedAt": "2016-12-29T13:51:33+02:00"
  }, {
    "Id": "83364c85cafc",
    "RepoTags": "dc:0.0_seed"
    "CreatedAt": "2017-01-01T12:45:48+02:00"
  }]
}
```

If the server responds with HTTP status code 200, the jQuery function `$.each()` iterates over the JSON array contained in the response to parse the data and append a row in the HTML table (using the jQuery function `append()`) for each array element.

Listing 4.5: Javascript function consuming the /admin/images/ endpoint

```
$(document).ready(function() {
  $.ajax({
    url: "/admin/images",
    success: function(response) {
      $.each(response.data, function(k, v) {
        $("#image-table").append(
          '<tr onclick=\'toImageHistory("' + v.Id + '")\' role
    ="button">'+
            '<td>' + v.Id + '</td>'+
            '<td>' + v.RepoTags + '</td>'+
            '<td>' + v.CreatedAt + '</td>'+
          '</tr>')
      });
    },
    error: function(response) {
      handleError(response)
    }
  });
});
```

The jQuery function `$(document).ready()` provides a way to run Javascript code~~,~~ when the page's Document Object Model (DOM) becomes safe to be manipulated~~,~~ and before the user can view or interact with the page content. When the "List Images" page is loaded, this function executes a call to the `$ajax()`. On `success`, it parses the `response` object provided by `$ajax`, and iterates over the `response.data`. Function `function(k,v)`, specifies the action to be performed for each key (k)~~,~~ and value (v)~~,~~ of the JSON array. The call to `$("#image-table").append` locates the HTML table with css identifier `#image-table`, and appends a table row `<tr>`. The attribute `onclick` specifies the action to be performed when the user clicks on a table row. This action is a call to a Javascript function named `toImageHistory()`, which requests the server to return the HTML page "Image History" that contains details about a particular docker image.

Home Page - Image History

When the administrator clicks on the table row of page "List of Images", an HTTP GET request to the `/admin/images/:id` endpoint is performed, similar~~ly~~ to the call presented in Listing 4.5. The server authenticates the request and afterwards it requests the Docker API to return information about a particular image. The server returns a JSON object with the requested information as shown below:

break page if needed, when report is final

```
{
  "Id"          : "4165bca12451"
  "RepoTags"    : "dc:0.1_traceroute"
  "Comment"     : "Installed traceroute package"
  "Created At" : "2017-01-01T12:45:48+02:00"
}
```

This JSON object is parsed by the `success` function, and its content is injected in the HTML page using the `$(#id).append()` function ~~as~~ shown earlier. The resulting page is shown in Figure 4.6.

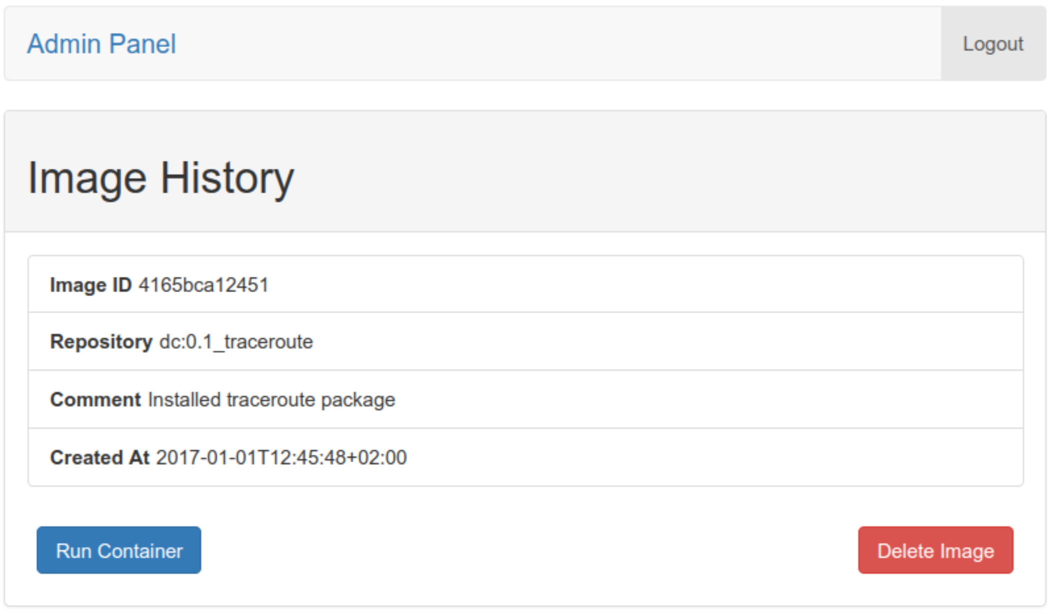


Image History

Image ID 4165bca12451

Repository dc:0.1_traceroute

Comment Installed traceroute package

Created At 2017-01-01T12:45:48+02:00

Figure 4.6: Tool Client page "Image History"

~~Apart from~~ the information returned by the server, the user is presented with two button options, "Run Container" and "Delete Image". Run Container requests the page of the Tool Client that consumes the `/admin/containers/run/:id` endpoint, and gives the admin ~~to~~ access to a running container, while Delete Image requests the Tool Client page that consumes the `/admin/images/delete/:id` endpoint and presents the admin with an option to delete the container image from the system.

Run Container page

The "Run Container" page works similarly to the previously explained pages. When the user clicks on the corresponding button of the page "Image History", before the HTML of the page is rendered by the browser and presented to the user, a request to the `/admin/containers/run/:id` endpoint is performed. The parameter `:id` is the identifier of the image, that is used to create and start a container. The server performs the following steps following the POST request to the endpoint:

1. Verifies that the correct HTTP cookie was sent with the request. If the request is not authorized, an HTTP response with HTTP status code `401 Unauthorized` is returned.

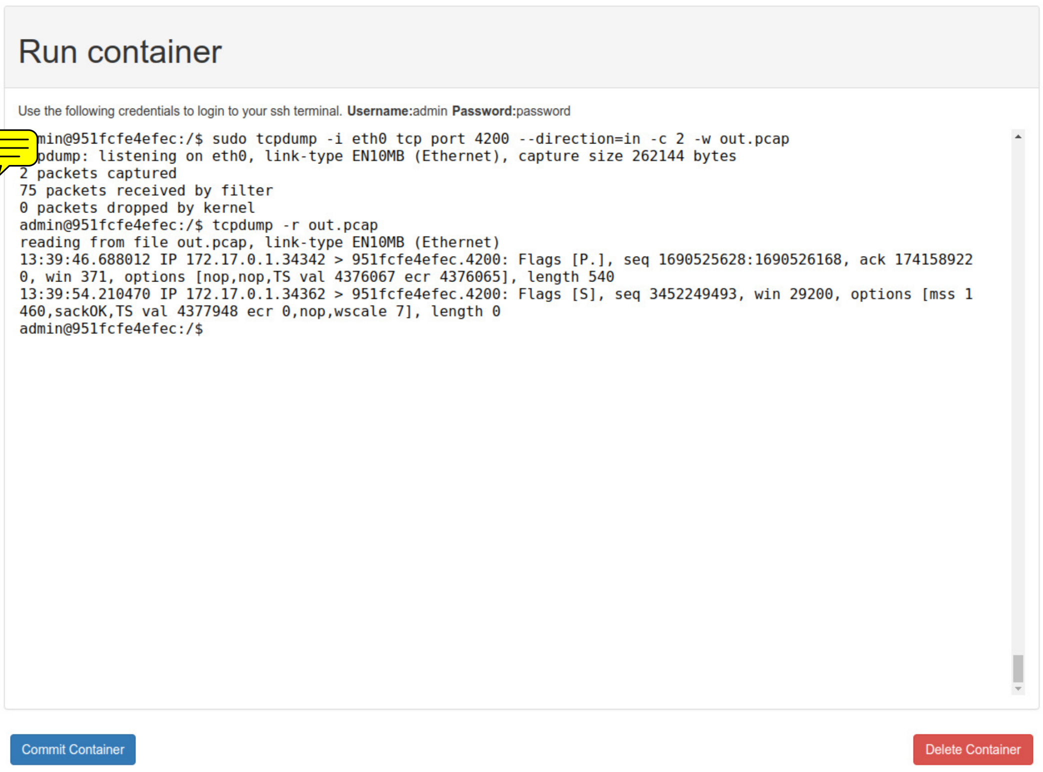
2. Validates the request parameter `:id`. If the image identifier is not valid a response with HTTP status code `400 Bad Request` is returned.

3. Extracts the session key from Redis, and then looks for a container run configuration. This check provides information for the endpoint handler, to know whether an existing container session exists and should be returned as response, or a new request to the Docker API for running a container ~~should be performed~~. This mechanism prevents subsequents requests from running new containers~~,~~ if ~~previous~~ sessions exists, thus preventing resource exhaustion for ports.

4. If a container session already exists, the TTL value of the Redis key is renewed, and the JSON value of the key ~~as~~ shown in Listing 4.3 is returned as part of the HTTP response.
   If no container session was present in Redis storage, a request to the Docker Remote API ~~is performed~~ to run a new container. The server will first look for an unused port resource. If no ports are available, an HTTP response with an error message is returned. If an unused port is found, it is reserved, and a container is created using the configuration parameters port, username, and password that are required for the Shell In A Box web server. The request for running the container is performed similarly to the docker command line example `docker run` presented in Section 2.8. The major difference is that the web server ~~is~~ perform~~ing~~ two requests to the Docker Remote API via the Go client, by calling the functions `ContainerCreate` and `ContainerStart` of the client library.

   Finally, if the Docker API ~~successfully~~ responds and starts the container, a new JSON configuration entry is stored in Redis~~,~~ and is returned as response to the calling jQuery function.

Figure 4.7 presents the contents of the web page "Run Container". The page contains an an HTML iframe, that embeds the Shell In A Box shell emulator, with an active SSH session. ~~It~~ shows the user which credentials to use to login into the Linux shell (Username and Password), and below the iframe ~~it has~~ two buttons ~~called~~ "Commit Container" and "Delete Container" ~~that when they~~ are clicked~~,~~ request the corresponding web pages of the Tool Client (the first is responsible for creating an image from a running container~~,~~ and the second is responsible deleting the running container).



Run container

Use the following credentials to login to your ssh terminal. **Username:**admin **Password:**password

```
min@951fcfe4efec:/$ sudo tcpdump -i eth0 tcp port 4200 --direction=in -c 2 -w out.pcap
pdump: listening on eth0, link-type EN10MB (Ethernet), capture size 262144 bytes
2 packets captured
75 packets received by filter
0 packets dropped by kernel
admin@951fcfe4efec:/$ tcpdump -r out.pcap
reading from file out.pcap, link-type EN10MB (Ethernet)
13:39:46.688012 IP 172.17.0.1.34342 > 951fcfe4efec.4200: Flags [P.], seq 1690525628:1690526168, ack 174158922
0, win 371, options [nop,nop,TS val 4376067 ecr 4376065], length 540
13:39:54.210470 IP 172.17.0.1.34362 > 951fcfe4efec.4200: Flags [S], seq 3452249493, win 29200, options [mss 1
460,sackOK,TS val 4377948 ecr 0,nop,wscale 7], length 0
admin@951fcfe4efec:/$
```

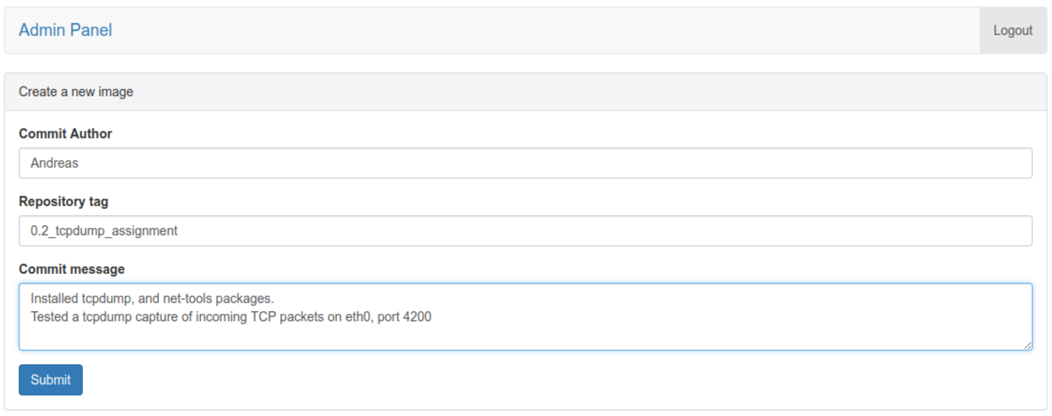
Figure 4.7: Tool Client page "Run Container"

The contents of the terminal presented in ~~the figure~~ show the ~~administrator~~ user running the `tcpdump` program [87] to perform a capture of incoming TCP packets to the network interface `eth0` and the port `4200`. The program ~~is~~ captur~~ing~~ packets sent from the Shell In A Box emulator while the user is typing commands in the terminal~~, and that~~ are forwarded to the web server that is running inside the container~~,~~ and is listening for connections on port `4200`. There are two commands ~~that are~~ shown in the figure. The first is `tcpdump -i eth0 tcp port 4200 --direction=in -c 2 -w out.pcap`~~,~~ ~~and has several parameters~~. The parameter `-i` specifies the network interface ~~`eth0`~~ to listen on. The parameter `tcp` specifies to listen only for TCP packets. The parameter `port` specifies the ~~port~~ destination port on which the TCP packets arrive. The parameter `--direction=in` specifies to listen only for incoming TCP packets. The parameter `-c` specifies ~~to~~ stop listening after capturing the first 2 packets~~, and~~ finally, the parameter `-w` specifies the output file in which tcpdump should write the results. The second command `tcpdump -r out.pcap` takes the parameter `-r` which specifies the input file from which it should read the captured output, and ~~print it in~~ the standard output.

This process can be used by an instructor to specify an assignment that involves the use of `tcpdump` program. The instructor can configure the required software for the assignment, and ~~perform~~ the assignment inside the emulator to verify that the laboratory environment is configured correctly and is ready to be used by students. Once the configuration is complete, the admin~~istrator~~ user of the Tool Client can click on the “Commit Container” button to visit the corresponding page and create a container image, to be used for configuring an LTI integration in Canvas.

4.2.1 Commit Container page

The “Commit Container” page allows the admin~~istrator~~ ~~od~~ the Tool Client to create new images using a running container as configuration. The page (shown in Figure 4.8) contains an HTML form with three input fields to be used as metadata for storing the image in the Docker local image repository. The first is Commit Author ~~and is~~ the name of the author ~~that~~ issues the commit command. The second is Repository Tag ~~that~~ will be used to identify the image~~, and the~~ last input field is ~~called~~ Commit message ~~and~~ allows the admin~~istrator~~ to provide additional information for the image. ~~Such~~ input data were presented earlier ~~in~~ “Image History” page. The form has a submit button, that when ~~is~~ clicked~~, is~~ perform~~ing~~ an HTTP POST request to the `/admin/containers/commit/:id` endpoint. The Javascript function that parses the form data and performs the request is similar to the ~~one presented in Listing 4.4~~ of the “Login page”.



Admin Panel | Logout

Create a new image

Commit Author: Andreas

Repository tag: 0.2_tcpdump_assignment

Commit message:
Installed tcpdump, and net-tools packages.
Tested a tcpdump capture of incoming TCP packets on eth0, port 4200

Submit

Figure 4.8: Tool Client page “Commit Container”

The handler function of the endpoint authenticates the user request, then performs validation of the form input fields, and then requests the Docker Remote API to create a new container image. This is performed using the function `ContainerCommit` of the Go client library. If the image is created ~~successfully~~, the server ~~is~~ issu~~ing~~ a request to delete the container~~,~~ and its corresponding configuration from Redis storage ~~and the port mappings~~.

4.2.2 Delete Container page

The “Delete Container” page presents the user a message ~~for~~ delet~~ing~~ a running container, and a button named “Delete Container”. When the button is clicked, a Javascript function is triggered to perform an HTTP DELETE request the `/admin/containers/kill/:id` endpoint to delete the container. The handler function of the endpoint authenticates the user request, verifies that the container is actually running by checking for container run configurations in Redis. Finally, the handler requests the Docker Remote API to remove the container using the function `ContainerRemove` of the Go client library, and afterwards, all related session keys are removed from Redis storage.

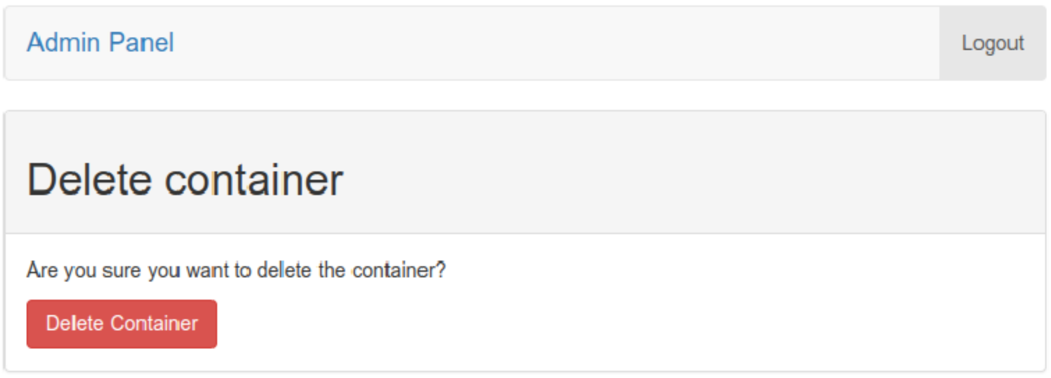
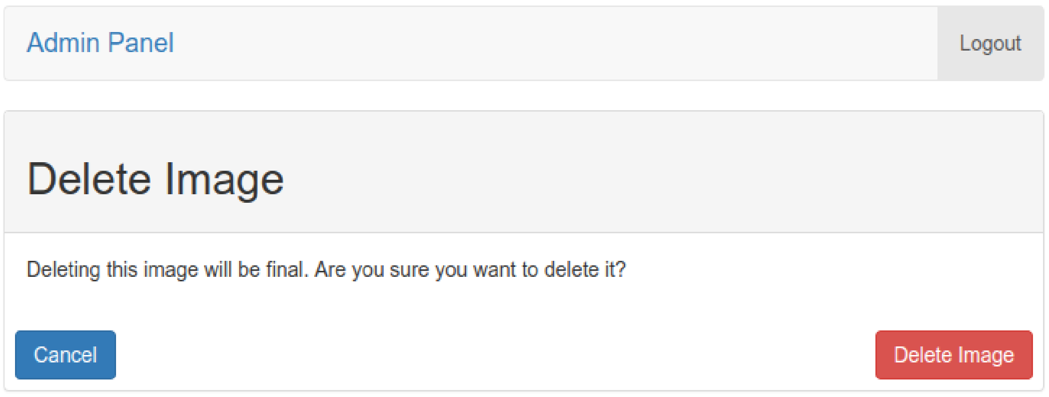

Admin Panel
Logout
Delete container
Are you sure you want to delete the container?
Delete Container

Figure 4.9: Tool Client page “Delete Container”

4.2.3 Delete Image page

The “Delete Image” page is loaded after clicking the corresponding button in the “Image History” page. It works similarly to the “Delete Container page”~~, and~~ performs an HTTP DELETE request to the `/admin/images/delete/:id` endpoint. The handler of the endpoint authenticates the user request, performs validation ~~for~~ the image identifier, and requests the Docker Remote API to delete the image by calling the function `ImageRemove` of the Go client library.



Delete Image

Deleting this image will be final. Are you sure you want to delete it?

Figure 4.10: Tool Client page “Delete Image”

4.3 LTI Tool Provider

The TP is realized by a single endpoint called `/lti/launch/:id`. The parameter `:id` is used to identify the container image that should be used to run a container. The handler function of the endpoint has a mechanism for authenticating requests from Canvas LMS in a similar way ~~with~~ the Sinatra web application presented in Section 2.4. This mechanism is implemented using a Go library for LTI integrations[88]. The route definition is ~~visible in the listing~~ below:

```
router.POST("/lti/launch/:id", route.OAuth(route.LTILaunch))
```

The HTTP method is POST, while the handler functions that serve the request for the URL `/lti/launch/:id` are `OAuth`, and `LTILaunch`. The first is the authentication mechanism and its implementation is shown in Listing 4.6, ~~and~~ the latter is the function that serves requests for running laboratory environments.

The `OAuth function` is of type `httprouter.Handle` (this type is defined as `type Handle func(http.ResponseWriter, *http.Request, Params)`), has the parameter `handler` that is a function of the same type, and has the same return type. Several programming languages including Go support passing functions as arguments to other functions~~,~~ or specifying them as return values. Such languages are often categorized as programming languages with support for “First Class Functions”. In the listing below, the OAuth function, defines the return function right after the reserved word `return`. This function is responsible for authenticating requests for the LTI route.

The authentication is performed using the The OAuth 1.0 Protocol. A request to this route is expected to have an ~~oauth~~ signature that matches a predefined key and a secret. Such signature is sent by Canvas~~, and it~~ is based on the key and secret values defined during the integration of an external application such as the TP. When Canvas performs the LTI Launch request, it sends an ~~oauth~~ signature. The server verifies that signature as shown in the code ~~below:~~

Listing 4.6: Authentication of the LTI Launch requests in Go

```
func OAuth(handler httprouter.Handle) httprouter.Handle {
  return func(res http.ResponseWriter, req *http.Request,
   params httprouter.Params) {

    // OAuth authentication of the TP requires ~~to match~~ the
    // request URL ~~with~~ the expected path. Since image IDs
    // change all the time, the path is constructed using
    // the imageID as extracted from the HTTP Header.
    path := fmt.Sprintf("https://%s%s",req.Host,req.URL.Path)

    p := lti.NewProvider("oauth_secret", path)
    p.ConsumerKey = "oauth_key"

    ok, err := p.IsValid(req)
    if !ok {
      res.Write([]byte("Invalid request"))
      return
    }
    if err != nil {
      res.Write([]byte("An error occured"))
      return
    }
    handler(res, req, params)
  }
}
```

The parameter `req` contains the ~~singature~~ value and method that are sent by the LMS. The OAuth signature provider is created following a call to the function `NewProvider()`, which takes two arguments `oauth_secret` (the secret that protects the route) and path (the URL path of the route). The key of the TC is configured by the assignment `p.ConsumerKey = "oauth_key"`. The call to the function `IsValid(req)` creates a server-side signature and compares it against the signature sent by the TC. This function has two return values, a boolean ok and and error. If the result contains an error, or `ok` does not have the value `true`, error messages are returned in the HTTP response. If the signature matches, the `handler` function is invoked to create a new laboratory environment.

The `handler` function `LTILaunch` operates similarly to the function that handles requests to the `/admin/containers/launch/:id` endpoint (explained in Section 4.2), but instead of returning a JSON object as a response, LTILaunch handler returns an HTML page containing the credentials for logging into the shell~~, and~~ an iframe with the shell emulator embedded. The resulting page is shown in Figure 4.13, while a simplified version of the handler's code is shown in Listing 4.7.

Listing 4.7: LTILaunch route handler function

```
func LTILaunch(res http.ResponseWriter, req *http.Request,
   params httprouter.Params) {
```

```
  t, _ := template.ParseFiles("templ/html/assignment.html")

  // Validate imageID
  if !vImageID.MatchString(imageID) {
    t.Execute(res, Resp{Error: "Invalid URL. Contact the
   administrator"})
  }

  // Parse LTI Post params
  err := req.ParseForm()
  // Error handling is omitted in listing

  // extract Canvas userID and store is as session key
  userID := req.PostFormValue("user_id")
  sessionExists, err = dc.ExistsUserRunConfig(userID)
  // Error handling is omitted in listing

  if sessionExists {
    cfg, err = dc.GetUserRunConfig(userID)
    // Update the TTL
    err = dc.SetUserRunConfig(userID, cfg)
  } else {
    // SESSION didn'texist, Generate username and password
    username := "guest"
    password := newPassword()

    // Run container request
    cfg, err = dc.RunContainer(imageID, username, password)
  }

  // Set session
  err = dc.SetUserRunConfig(userID, cfg)

  // Return HTML template with data
  t.Execute(res, getResp(cfg))
}

type Resp struct {
  ContainerID string
  Port        string
  Username    string
  Password    string
  URL         string
  Error       string
}
```

The first action of the LTILauch function is to create a text template following a call to function `ParseFiles()` of Go package `html/template`. The function takes an HTML file as argument~~,~~ and produces a variable of type `Template`. ~~F~~unction `t.Execute(res, RespError: "text message")` is used to inject string values into the template `t`, and write the output to the HTTTP response `res`. The Go `struct Resp` contains values of type `string` that are used in the template to inject the URL of the iframe containing shell emulator, the username, password, and container identifier. For example, a variable containing an error is passed in the HTML template using the `.Error` syntax. The `Execute` function will replace the contents of `.Error` with the value of the `Error` variable.

```
<span>Error:</span> {{ .Error }}
```

After the template variable is initialized, the handler validates the image identifier parameter `:id` via a call to `vImageID.MatchString(imageID)`. The variable `vImageID` is a compiled regular expression defined as `var vImageID = regexp.MustCompile(([A-Fa-f0-9]12,64)$)`. Function `MatchString` verifies whether the parameter imageID of type string, matches ~~against~~ the regular expression (an alphanumeric sequence of 12-64 characters) and returns a boolean value as a result.

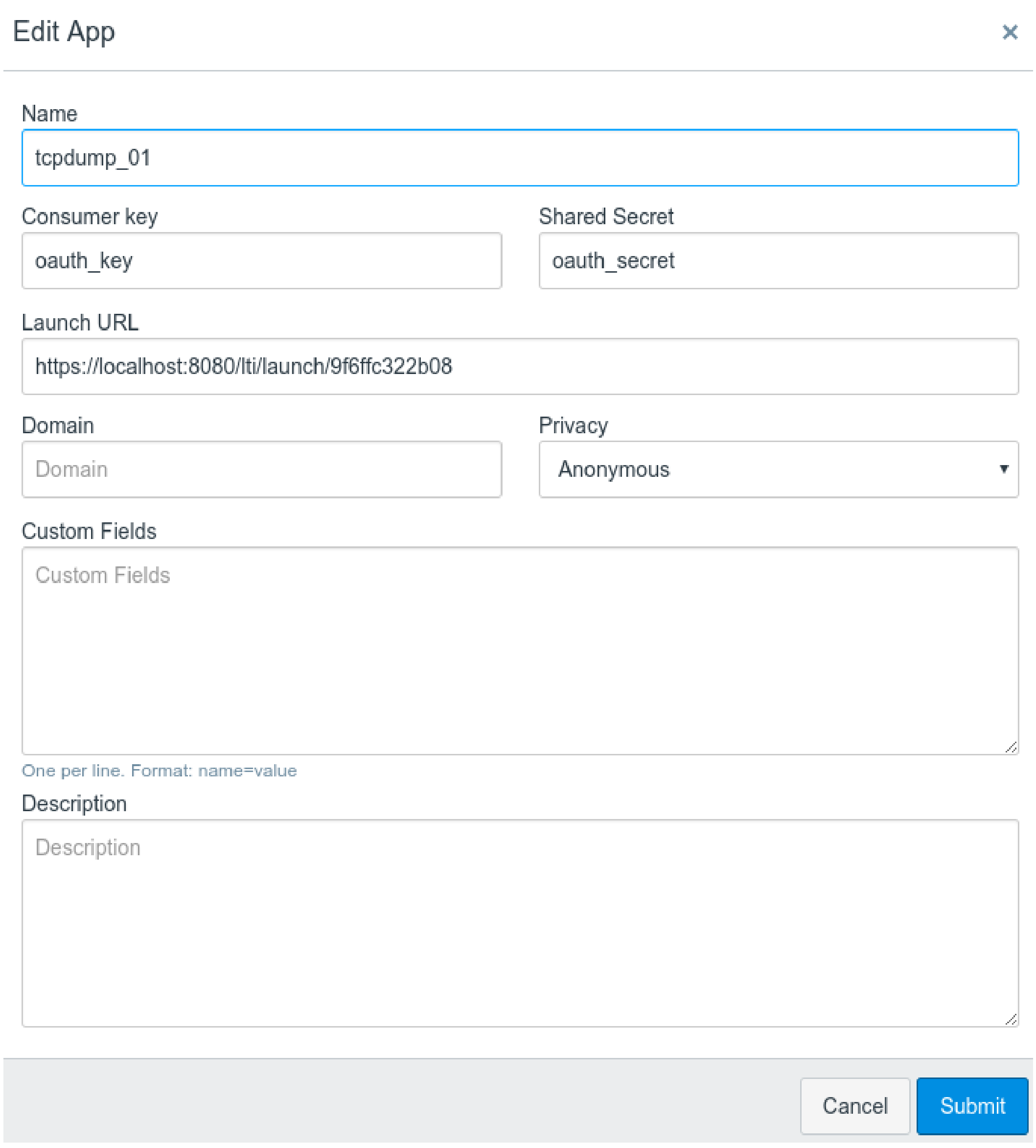
Afterwards, the handler reads the `user_id` form parameter sent by Canvas, and checks whether a container run configuration exists in Redis for that particular user. If such configuration exists, it is loaded in the `cfg` variable, and the TTL value of the Redis entry is renewed. If such configuration was not present, a `username`, and a random `password` are created and passed as parameters to the function `RunContainer` of package `dc` to create and start a new container for this user session. The new container run configuration is stored in Redis following a call to `SetUserRunConfig`.

Finally, a call to `t.Execute(res, getResp(cfg))` is performed, to write the configuration values in the HTML template, and return ~~it~~ to Canvas LMS. The function `getResp(cfg)` initializes a `Resp` struct with the values returned from `RunContainer` function.

The following sections contain an example of configuring an `/lti/launch/:id` route as an external application in Canvas LMS, and a student accessing a laboratory environment through an assignment that was configured to launch the external application.

Configuration of an Assignment

Figure 4.11 shows how a specific image was configured in Canvas, as an external application. The name of the application is `tcpdump_01`, the consumer key and the share secret have values `oauth_key` and `oauth_secret` respectively ~~which were~~ configured in the OAuth handler function ~~of~~ Listing 4.6. The Launch URL is `https://localhost:8080/lti/launch/9f6ffc322b08` where the identifier of the container image, is the one created in the Tool Client from the "Commit Container" page of Figure 4.8.

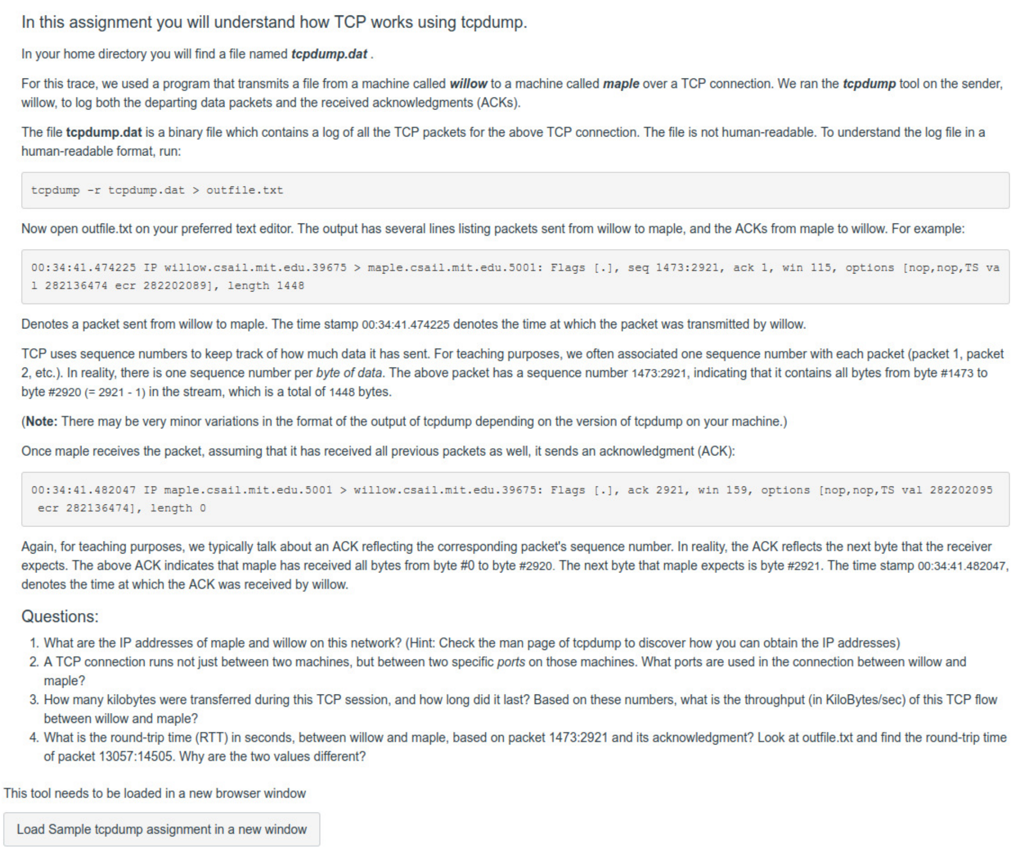


Edit App

Name	tcpdump_01
Consumer key	oauth_key
Shared Secret	oauth_secret
Launch URL	https://localhost:8080/lti/launch/9f6ffc322b08
Domain	Domain
Privacy	Anonymous
Custom Fields	Custom Fields
	One per line. Format: name=value
Description	Description

Figure 4.11: Configuration of the TP in Canvas

An assignment configuration was created in Canvas to run the external tool shown above. The tool was instructed to run in a new browser window, rather than embed the response of the TP in the same page. For the configuration of the assignment, the laboratory assignment “Understanding TCP and tcpdump”[89] from the course “Computer System Engineering” of the Massachusetts Institute of Technology (MIT)~~, department~~ “Electrical Engineering & Computer Science~~”~~ was used. ~~The~~ description of ~~the~~ assignment is shown in Figure 4.12.



In this assignment you will understand how TCP works using tcpdump.

In your home directory you will find a file named ***tcpdump.dat*** .

For this trace, we used a program that transmits a file from a machine called ***willow*** to a machine called ***maple*** over a TCP connection. We ran the ***tcpdump*** tool on the sender, willow, to log both the departing data packets and the received acknowledgments (ACKs).

The file **tcpdump.dat** is a binary file which contains a log of all the TCP packets for the above TCP connection. The file is not human-readable. To understand the log file in a human-readable format, run:

```
tcpdump -r tcpdump.dat > outfile.txt
```

Now open outfile.txt on your preferred text editor. The output has several lines listing packets sent from willow to maple, and the ACKs from maple to willow. For example:

```
00:34:41.474225 IP willow.csail.mit.edu.39675 > maple.csail.mit.edu.5001: Flags [.], seq 1473:2921, ack 1, win 115, options [nop,nop,TS va
l 282136474 ecr 282202089], length 1448
```

Denotes a packet sent from willow to maple. The time stamp 00:34:41.474225 denotes the time at which the packet was transmitted by willow.

TCP uses sequence numbers to keep track of how much data it has sent. For teaching purposes, we often associated one sequence number with each packet (packet 1, packet 2, etc.). In reality, there is one sequence number per *byte of data*. The above packet has a sequence number 1473:2921, indicating that it contains all bytes from byte #1473 to byte #2920 (= 2921 - 1) in the stream, which is a total of 1448 bytes.

(**Note:** There may be very minor variations in the format of the output of tcpdump depending on the version of tcpdump on your machine.)

Once maple receives the packet, assuming that it has received all previous packets as well, it sends an acknowledgment (ACK):

```
00:34:41.482047 IP maple.csail.mit.edu.5001 > willow.csail.mit.edu.39675: Flags [.], ack 2921, win 159, options [nop,nop,TS val 282202095
 ecr 282136474], length 0
```

Again, for teaching purposes, we typically talk about an ACK reflecting the corresponding packet's sequence number. In reality, the ACK reflects the next byte that the receiver expects. The above ACK indicates that maple has received all bytes from byte #0 to byte #2920. The next byte that maple expects is byte #2921. The time stamp 00:34:41.482047, denotes the time at which the ACK was received by willow.

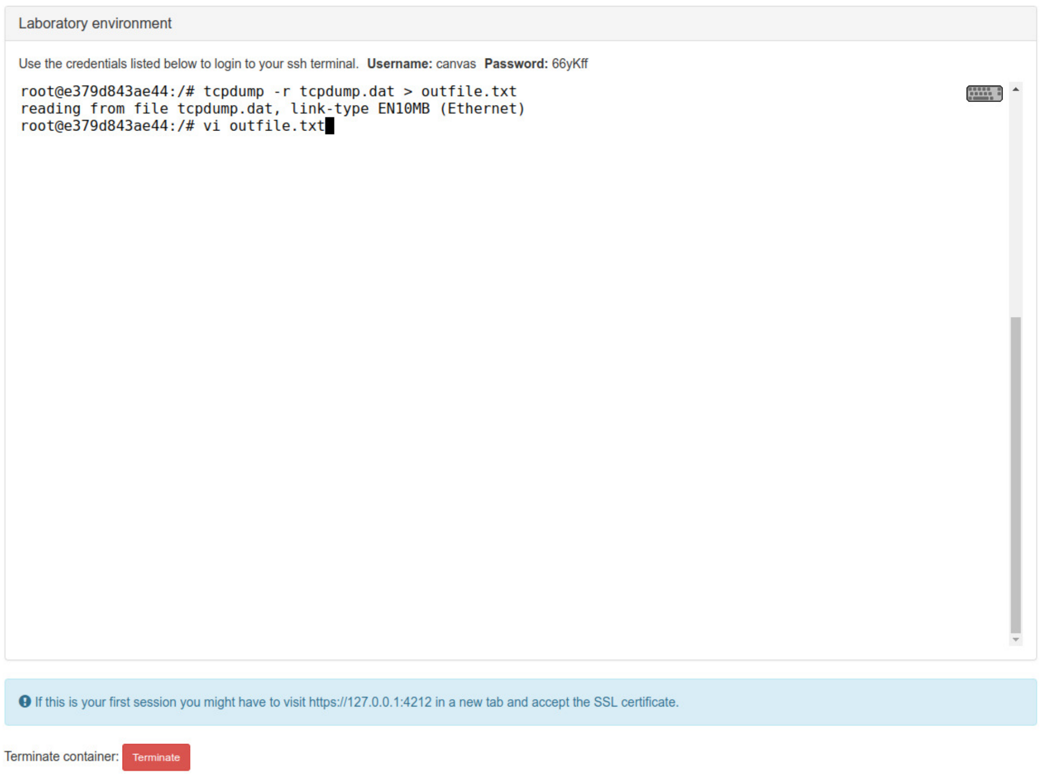
Questions:

1. What are the IP addresses of maple and willow on this network? (Hint: Check the man page of tcpdump to discover how you can obtain the IP addresses)
2. A TCP connection runs not just between two machines, but between two specific *ports* on those machines. What ports are used in the connection between willow and maple?
3. How many kilobytes were transferred during this TCP session, and how long did it last? Based on these numbers, what is the throughput (in KiloBytes/sec) of this TCP flow between willow and maple?
4. What is the round-trip time (RTT) in seconds, between willow and maple, based on packet 1473:2921 and its acknowledgment? Look at outfile.txt and find the round-trip time of packet 13057:14505. Why are the two values different?

This tool needs to be loaded in a new browser window

Figure 4.12: Assignment Description in Canvas LMS

The description instructs the user to use the `tcpdump` command line program that is configured in the laboratory environment to study TCP packets that were sent from a server called `willow` to a server called `maple`. It provides some information regarding the output of `tcpdump`, and asks the student a series of questions to complete the assignment. Methods for replying to ~~such~~ questions ~~were~~ not ~~involved~~ in this example, as they are not relevant to the use of the laboratory environment. At the bottom of the assignment, an HTML button with content “Load Sample tcpdump assignment in a new window” is visible. When this button is clicked, Canvas performs the HTTP POST request to the `/lti/launch/:id` endpoint, and requests the browser to render the HTML response in a new window (shown in Figure 4.13).



Laboratory environment

Use the credentials listed below to login to your ssh terminal. **Username:** canvas **Password:** 66yKff

```
root@e379d843ae44:/# tcpdump -r tcpdump.dat > outfile.txt
reading from file tcpdump.dat, link-type EN10MB (Ethernet)
root@e379d843ae44:/# vi outfile.txt
```

If this is your first session you might have to visit https://127.0.0.1:4212 in a new tab and accept the SSL certificate.

Terminate container: Terminate

Figure 4.13: Laboratory environment via Canvas LMS

~~The figure~~ shows a student accessing the laboratory environment via the HTML page returned as response from the `LTILaunch` route handler. The ~~user~~ has already authenticated herself in the shell, and is following the instructions of the assignment to execute the command `tcpdump -r tpdump.dat > outfile.txt` to write the output of the TCP packet trace in a human readable ~~way~~ into file `outfile.txt`.

The page contains a section that instucts the user to open the shell emulator in a new browser window, to leverage full screen capabilities of the shell. Finally, a button called "Terminate" provides the option for the user to terminate the container session.

4.4 Evaluation

Chapter 5

Conclusions and Future Work

5.1 Conclusions

5.2 Limitations

5.3 Future work

Scalability

The architectural design of the implementation presented in ~~c~~hapter 4 has limitation in terms of scalability. The docker daemon and the Tool Provider are running in the same virtual ~~of~~ physical ~~environment~~, ~~resulting to~~ bound~~ing the~~ ~~cpu~~ and memory resources that the containers consume.

In addition this setup has limitations on the number of ports the docker daemon can use to bridge network connections between the containers and the host system.

A lot of work has been done in deploying scalable clusters of container runtime environments. Kubernetes being one of them ...

Web SSH

Shell In A Box was ~~the~~ chosen web terminal emulator, but there are alternatives implementations that have not being investigated ~~within~~ the scope of this project. I addition, the base container image used by the backend system was not evaluated. The configuration of shellinabox package has more capabilities than supported in the docker image. An alternative would be to manually create a base image...

Problem with commit containers. SIAB_USER is not used anymore. All users have ~~guest~~ username, but different password.
In order to use custom usernames, new method should be implemented using the entrypoint.sh and Dockerfile, to delete the last user, group, etc. before the commit command.
In addition, the create process should be changed and use more environment variables for shell in a box

Investigate an implementation that instead of returning an emulator shell, ~~it~~ provides configuration settings, such as an ssh key for download, so that the user can ssh directly in the container from her terminal. This will probably be more useful, as users will not rely on the browser communicating with the emulator, but ~~on~~ ssh agent ~~directly~~.

User Tests

This system is intended to be use~~r~~ by instructors and students. Evaluation of the front-end performance and usability of the system should be performed, using UX prototypes ...

Additional Evaluation

The implementation of the port mapper makes various assumptions.
* Investigate the use of goroutine channels instead of mutex.locks
* Measure and evaluate how the periodic checks affect the performance of the system. * PeriodocChecker relies on Redis. Eliminate this. Only use docker and in memory.

Frontend technologies

The system relies on containers running all the time. This is not guaranteed. From the frontend heartbeats should be sent to the API asynchronously, to determine faults, and inform the user accordingly.

Assignment evaluation

- Moreover, the environment must provide feedback for both students and teachers.

- Internetworking assignments~~,~~ and extraction of relevant analytics have ~~also~~ limitations. Make sure that they are reflected in this section.

Desired Features

- The system should evaluate if an image is functional, and if it can be used to integrate with Canvas via LTI. It should give feedback to the instructor if something went wrong~~,~~ and what the errors were.

References

[1] William R. Watson and Sunnie Lee Watson. An argument for clarity: what are learning management systems, what are they not, and what should they become? *TechTrends*, 51(2):28–34, 2007.

[2] Stefan Boesen, Richard Weiss, James Sullivan, Michael E. Locasto, Jens Mache, and Erik Nilsen. EDURange: Meeting the Pedagogical Challenges of Student Participation in Cybertraining Environments. In *7th Workshop on Cyber Security Experimentation and Test (CSET 14)*, San Diego, CA, August 2014. USENIX Association.

[3] Ricardo Nabhen and Carlos" Maziero. *Education for the 21st Century — Impact of ICT and Digital Resources: IFIP 19th World Computer Congress, TC-3, Education, August 21–24, 2006, Santiago, Chile*, chapter Some Experiences in Using Virtual Machines for Teaching Computer Networks, pages 93–104. Springer US, Boston, MA, 2006.

[4] Christian Willems, Johannes Jasper, and Christoph Meinel. Introducing hands-on experience to a massive open online course on openhpi. In *Teaching, Assessment and Learning for Engineering (TALE), 2013 IEEE International Conference on*, pages 307–313. IEEE, 2013.

[5] Instructure, Inc. Canvas learning management system. `https://www.canvaslms.com/`. [Online; accessed 2016-02-21].

[6] Daniela Fonte, Daniela da Cruz, Alda Lopes GanÃ§arski, and Pedro Rangel Henriques. A flexible dynamic system for automatic grading of programming exercises. In *OASIcs-OpenAccess Series in Informatics*, volume 29. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2013.

[7] Guillaume Derval, Anthony Gego, Pierre Reinbold, Benjamin Frantzen, and Peter Van Roy. Automatic grading of programming exercises in a MOOC using the INGInious platform.

[8] Ricardo Queirós and José Paulo Leal. Programming exercises evaluation systems - An interoperability survey. In *CSEDU (1)*, pages 83–90, 2012.

[9] Alan Hevner and Samir Chatterjee. Design Science Research in Information Systems. In *Design Research in Information Systems*, volume 22, pages 9–22. Springer US, Boston, MA, 2010.

[10] Vijay K. Vaishnavi and William Kuechler, Jr. *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. Auerbach Publications, Boston, MA, USA, 1st edition, 2007.

[11] Alan R. Hevner. A three cycle view of design science research. *Scandinavian journal of information systems*, 19(2):4, 2007.

[12] C. Alario and S. Wilson. Comparison of the main alternatives to the integration of external tools in different platforms. In *ICERI2010 Proceedings*, 3rd International Conference of Education, Research and Innovation, pages 3466–3476. IATED, 15-17 November, 2010 2010.

[13] Open edX as an LTI Tool Provider. `https://open.edx.org/blog/open-edx-lti-tool-provider`. [Online; accessed 2016-02-28].

[14] Md. Iqbal Hossain and Md. Iqbal Hossain. Dynamic scaling of a web-based application in a cloud architecture. Master's thesis, KTH, Radio Systems Laboratory ~~(RS Lab)~~, 2014.

[15] Ryann K Ellis. Field guide to learning management systems, 2009.

[16] José Paulo Leal and Ricardo Queirós. A comparative study on lms interoperability. *Higher Education Institutions and Learning Management Systems: Adoption and Standardization*, page 142, 2011.

[17] Wynne Harlen and Mary James. Assessment and Learning: differences and relationships between formative and summative assessment. *Assessment in Education: Principles, Policy & Practice*, 4(3):365–379, November 1997.

[18] Janne Malfroy Kevin Ashford-Rowe. E-Learning Benchmark Report: Learning Management System (LMS) usage. `http://www.uws.edu.au/__data/assets/pdf_file/0007/452077/Griffith_UWS_Elearning_Benchmark_Report.pdf`, 2009.

[19] ISO. Information technology vocabulary. ISO 2121317 - 2382:2015, International Organization for Standardization, 2015.

[20] IMS GLOBAL Learning Consortium. Learning Tools Interoperability ®(LTI®). `http://www.imsglobal.org/activity/learning-tools-interoperability`. [Online; accessed 2016-02-23].

[21] Ricardo Queirós, José Paulo Leal, and José Paiva. Integrating rich learning applications in LMS. In *State-of-the-Art and Future Directions of Smart Learning*.

[22] IMS Learning Information Services. `https://www.imsglobal.org/lis/`. [Online; accessed 2016-02-28].

[23] Ruby Sinatra - official documentation page. `http://www.sinatrarb.com/documentation.html`. [Online; accessed 2016-07-17].

[24] Alan Harris and Konstantin Haase. *Sinatra: Up and Running*. O'Reilly Media, Inc., 1st edition, 2011.

[25] LTI Outcome Service Example using Canvas LMS. `https://github.com/instructure/lti_example`. [Online; accessed 2016-04-23].

[26] IMS Global General Web Services. `https://www.imsglobal.org/gws/index.html`. [Online; accessed 2016-07-27].

[27] IMS Global Learning Tools Interoperability$^{\text{TM}}$ Implementation Guide. `https://www.imsglobal.org/specs/ltiv1p1/implementation-guide`. [Online; accessed 2016-07-27].

[28] Ed. E. Hammer-Lahav. The o[illegible]h 1.0 protocol, April 2010.

[29] OpenSSL cryptography and SSL/TLS toolkit. `https://www.openssl.org/`. [Online; accessed 2016-08-07].

[30] Marcus Redivo. Creating and using SSL certificates. `http://www.eclectica.ca/howto/ssl-cert-howto.php`. [Online; accessed 2016-08-07].

[31] OpenSSL - official documentation of command `req`. `https://www.openssl.org/docs/manmaster/apps/req.html`. [Online; accessed 2016-08-07].

[32] Phil Dibowitz. Openssl.conf walkthru. `https://www.phildev.net/ssl/opensslconf.html`. [Online; accessed 2016-08-07].

[33] An open lti app collection. https://www.eduappcenter.com/. [Online; accessed 2016-07-11].

[34] Instructure. `https://www.instructure.com/`. [Online; accessed 2016-07-17].

[35] Joon Son, Chinedum Irrechukwu, and Patrick Fitzgibbons. A Comparison of Virtual Lab Solutions for Online Cyber Security Education. *Communications of the IIMA*, 12(4), 2012.

[36] Richard Weiss, Jens Mache, and Erik Nilsen. Top 10 hands-on cybersecurity exercises. *J. Comput. Sci. Coll.*, 29(1):140–147, October 2013.

[37] Yugesh Suresh Bhosale and Jenila Livingston M. Article: V-lab: A mobile virtual lab for network security studies. *International Journal of Computer Applications*, 93(20):35–38, May 2014. ~~Full text available.~~

[38] Rajdeep Dua, A Reddy Raja, and Dharmesh Kakadia. Virtualization vs Containerization to Support PaaS. [illegible]es 610–614. IEEE, March 2014.

[39] Linux containers - lxc. https://linuxcontainers.org/lxc/introduction/. [Online; accessed 2016-02-28].

[40] Linux programmer's manual, overview of linux namespaces. http://man7.org/linux/man-pages/man7/namespaces.7.html. [Online; accessed 2016-02-28].

[41] Rami Rosen. Linux containers and the future cloud. *Linux J*, 240, 2014.

[42] Docker. https://www.docker.com/. [Online; accessed 2016-02-28].

[43] Libcontainer implementation. https://github.com/opencontainers/runc/tree/master/libcontainer. [Online; accessed 2016-11-20].

[44] Lxc container driver. https://libvirt.org/drvlxc.html. [Online; accessed 2016-11-20].

[45] systemd-nspawn. https://www.freedesktop.org/software/systemd/man/systemd-nspawn.html. [Online; accessed 2016-11-20].

[46] Open container initiative. https://www.opencontainers.org/about. [Online; accessed 2016-11-20].

[47] Docker command line reference. https://docs.docker.com/engine/reference/commandline. [Online; accessed 2016-11-20].

[48] Docker remote api. https://docs.docker.com/engine/reference/api/docker_remote_api. [Online; accessed 2016-11-20].

[49] Unix pseudoterminal interface. http://man7.org/linux/man-pages/man7/pty.7.html. [Online; accessed 2016-11-20].

[50] Web based ssh. https://en.wikipedia.org/wiki/Web-based_SSH. [Online; accessed 2016-11-19].

[51] Gateone. https://github.com/liftoff/GateOne. [Online; accessed 2016-11-20].

[52] shellinabox. https://github.com/shellinabox/shellinabox. [Online; accessed 2016-11-20].

[53] *VT100 Series Technical Manual*, 1979.

[54] Anthony T. Holdener, III. *Ajax: The Definitive Guide*. O'Reilly, first edition, 2008.

[55] shellinabox for docker. `https://github.com/sspreitzer/docker-shellinabox`. [Online; accessed 2016-11-20].

[56] The Linux Foundation. The linux foundation wiki - bridge.

[57] Docker documentation - customize the docker0 bridge. `https://docs.docker.com/engine/userguide/networking/default_network/custom-docker0/`. [Online; accessed 2016-12-18].

[58] Edurange: A cybersecurity competition platform to enhance undergraduate security analysis skills. `http://blogs.evergreen.edu/edurange/`. [Online; accessed 2016-02-28].

[59] Amazon elastic compute cloud (amazon ec2). `https://aws.amazon.com/ec2/`. [Online; accessed 2016-02-28].

[60] EDURange Github project. 2014. [Online; accessed 2016-02-28].

[61] Carlos Alario-Hoyos, Miguel L. Bote-Lorenzo, Eduardo Gómez-Sánchez, Juan I. Asensio-Pérez, Guillermo Vega-Gorgojo, and Adolfo Ruiz-Calleja. Glue!: An architecture for the integration of external tools in virtual learning environments. *Computers & Education*, 60(1):122–137, 2013.

[62] Inginious by universitÃ© catholique de louvain. `http://inginious.org/`. [Online; accessed 2016-02-28].

[63] Github repository of inginious. `https://github.com/UCL-INGI/INGInious`. [Online; accessed 2016-02-28].

[64] Technical documentation of inginious. `http://inginious.readthedocs.org`. [Online; accessed 2016-02-28].

[65] Teacher documentation of inginious. `http://inginious.readthedocs.io/en/latest/teacher_documentation.html`. [Online; accessed 2016-02-28].

[66] Vijay K. Vaishnavi and William Kuechler, Jr. Design science research in information systems. January.

[67] Ruby on Rails - official web page. `http://rubyonrails.org/`. [Online; accessed 2016-07-17].

[68] Instructure, Inc. Canvas l[illegible] istallation quick start wiki page. `https://github.com/instructure/canvas-lms/wiki/Quick-Start`. [Online; accessed 2016-11-07].

[69] HashiCorp. Vagrant home page. `https://www.vagrantup.com/`. [Online; accessed 2016-08-07].

[70] Oracle. Virtualbox home page. https://www.virtualbox.org/. [Online; accessed 2016-08-07].

[71] Andreas Kokkalis. Canvas lms installation using vagrant. https://github.com/andreas-kokkalis/canvas_lms_vagrant. [Online; accessed 2016-08-07].

[72] The go programming language. https://golang.org/. [Online; accessed 2016-08-07].

[73] Alan A.A. Donovan and Brian W. Kernighan. *The Go Programming Language*. Addison-Wesley Professional, 1st edition, 2015.

[74] Golang package net/http. https://golang.org/pkg/net/http/. [Online; accessed 2016-08-07].

[75] Httprouter - a trie based high performance http request router. https://github.com/julienschmidt/httprouter. [Online; accessed 2016-12-07].

[76] Go implementation of the docker remote api library. https://godoc.org/github.com/docker/docker/client. [Online; accessed 2016-08-07].

[77] Go context package. https://golang.org/pkg/context/. [Online; accessed 2016-12-07].

[78] The go programming language specification. https://golang.org/ref/spec#The_zero_value. [Online; version 2016-05-31].

[79] Docker checkpoint and restore. https://github.com/docker/docker/blob/master/experimental/checkpoint-restore.md. [Online; accessed 2016-12-18].

[80] Redis. https://redis.io/. [Online; accessed 2016-12-18].

[81] Type-safe redis client for golang. https://github.com/go-redis/redis. [Online; accessed 2016-12-18].

[82] Postgresql - open source object relational database system. https://www.postgresql.org/. [Online; accessed 2016-12-18].

[83] Go package database/sql. https://golang.org/pkg/database/sql/. [Online; accessed 2016-12-18].

[84] Pure go postgres driver for database/sql. https://github.com/lib/pq. [Online; accessed 2016-12-18].

[85] ajax function of jquery for performing asynchronous http (ajax) requests. https://api.jquery.com/jquery.ajax/. [Online; accessed 2016-12-18].

[86] J.Reschke R. Fielding. Hypertext transfer protocol (http/1.1): Semantics and content, June 2014.

[87] tcpdump - command line packet analyzer. `http://www.tcpdump.org/`. [Online; accessed 2016-12-18].

[88] Jordi Collell. Golang lti - go tools for working with the lti specification. `https://github.com/jordic/lti`. [Online; accessed 2016-12-18].

[89] ~~Computer System Engineering~~ M.[illegible] DEP[illegible]TMENT OF EECS. Understanding tcp using tcpdump. [illegible]`ttp://web.mit.edu/6.033/www/assignments/handson-tcp.html`. [Online; accessed 2017-01-02].

Andreas_Kokkalis_report_20161120-commented-a.pdf contains important feedback for the Citations

Appendix A

Appendix Name X

content X