Software Project 2

Summer semester 2020

MoDoCoT

“Moodle Dockerized Code Testing

Table Of Contents

Introduction to MoDoCoT

MoDoCoT enables a professor to set up moodle assignments for students, where the students can upload a ZIP file including the URL to their Git repository. MoDoCoT builds the code and runs a predefined JUnit test Launcher suite over the code. The professor provides the JUnit tests as well as a single ZIP file when creating the assignment in Moodle. MoDoCoT shows the JUnit test results to the respective student and the professor similar to other test results in Moodle.

MoDoCoT verifies the type of the uploaded file. Only ZIP files can be uploaded to MoDoCoT. It display the test results of the uploaded Java task files, that means what tests have passed or failed and if there was any compilation error. The MoDoCoT Moodle plugin depends on the MoDoCoT web service.

Goals

The system MoDoCoT enables a professor to set up Moodle exercise hand-ins for students in the HFT Stuttgarts official Moodle system, where MoDoCoT builds the code and runs a Junit test suite over the code. MoDoCoT shows the weighted JUnit test results to the respective student similar to other test results in Moodle.

Architectural Overview

The MoDoCoT system consists of two main parts:

- The MoDoCoT Moodle plugin
- The MoDoCoT web service

The Moodle plugin is of the type assignment submission and connects to the web service using REST and the JSON file format.

Use Cases

- A teacher should be able to upload JUnit test files when creating an assignment
- A student should be able to upload Java files to this assignment. Those files should be tested with the provided JUnit tests.
- The student should see a summary of the test results.
- The teacher should see a summary of all test results of all students, but also be able to view the detailed results.

Reasons for used applications:

Docker

- Easy multi-server maintainability due to just executing the image on the server.
- Keeping track of the container version, thus providing version control of the containers.
- Lightweight since the containers are operating on the process level, making it perfect for software delivery.
- Very low memory usage, since it only requires an operating system, supporting libraries, and system resources to run a specific program.
- Reliable, since it runs the tests on the same image as the production environment.

Jenkins

- Open-source
- Very well documented and has a wide range of useful plugins.
- Allows for running builds for multiple branches dynamically.
- Ability to send email notifications.

Spring Boot

- For all Spring applications, you should start with the Spring Intializr
- The Initialzr offers a fast way to pull in all the dependencies you need for an application
- The Spring Boot Maven Plugin provides Spring Boot support in Apache Maven
- It allows you to package executable jar or war archives run Spring Boot applications, generate build information and start your Spring Boot application prior to running integration tests.
- All Maven projects have a common structure, which makes it easier to understand each project.
- It is declarative. All you have to do was create a .xml file and put your source in the default directory. Maven takes care of the rest.
- It has a lifecycle, which is invoked when you execute `mvn install`. This command tells Maven to execute a sequence of steps until it reaches the lifecycle goal.

Installation Guide

MoDoCoT Moodle Plugin

Moodle Dockerized Code Testing (MoDoCoT) Plugin

A Moodle plugin to assist teachers correcting JUnit exercises. This plugin allows students to submit their Java exercises, let them be tested against a set of JUnit tests (that have been priorly provided by the teacher) and receive immediate feedback on the test results.
For this to work, the plugin communicates with an external webservice providing essentially the following services on the given paths:

- POST **/v1/unittest**: Expects the assignment id and a zip file containing a repo.txt with the repository-link and an optional line for credentials when using a private repository.
  You need to add the credentials like this: username:passwort or username:auth-token.
- POST **/v1/tasks**: Expects the assignment id and a zip file containing a repo.txt with the repository-link and an optional line for credentials when using a private repository.
  You need to add the credentials like this: username:passwort or username:auth-token.
  Returns the results as JSON.
- DELETE **/v1/unittest?assignmentId={id}**: Triggers the deletion of the test files.

MoDoCoT Backend Web Service

See here for an implementation of the webservice:
https://transfer.hft-stuttgart.de/gitlab/HFTSoftwareProject/MoDoCoT-Backend
There is also a ready to go docker container available:
https://hub.docker.com/r/hftstuttgart/modocot-backend

License

GNU General Public License, Version 3, is a free and copyleft license for software. The GNU makes sure it remains free software for all its users, when speaking of free software, it is referred to freedom, not price. General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software, that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, but it is not allowed to relicense the changed source code.If you develop a new program, and you want it to be of the greatest possible use to the public,
develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.
To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively state the exclusion of warranty; and each file should have at least the “copyright” line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of
what it does.> Copyright (C) <year> <name of author>

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as
published by the Free Software Foundation, either
version 3 of the License, or (at your option) any later
version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied
warranty of MERCHANTABILITY or FITNESS FOR A
PARTICULAR PURPOSE. See the GNU General Public
License for more details.

Developers that use the GNU GPL protect your rights with two steps:

1. assert copyright on the software
2. offer you this License giving you legal permission to copy, distribute it.

For more information on this please go to the following link:
https://www.gnu.org/licenses/gpl-3.0.html
You may not propagate or modify a covered work except as expressly provided under this License. Any attempt to propagate or modify it is not acceptable, and will automatically terminate your rights under this License.

Technical Details

- The backend is a RESTful web service that allows you to upload the test files and the task files as HTTP POST parameters.
- The data format for this communication is JSON, i.e when you upload the task ZIP, the Java files will be checked against the previously uploaded tests and the result will be returned as a JSON string.
- The default timeout set in moodle for the requests to our backend is set at 30sec.
- The backend is based on Spring Boot and uses the JavaCompiler API to compile the tests and tasks.
- The backend uses Maven as a build and dependency management tool.
- For continuous deployment Jenkins 2.0 is used.
- For the virtualization technology we chose to use Docker.
- The continuous deployment environment runs on a KVM machine. It uses Jenkins’ Maven plugin, bash scripts and the docker CLI to deploy the new code to the productive environment.
- Jenkins builds the Docker container and pushes it to Docker Hub. On the docker server the newly build container is pulled from Docker Hub.

Here's an example JSON response after uploading the task Java file:

```
{
  "testResults": [
          {
          "testName": "CalculatorTest",
          "testCount": 5,
          "failureCount": 0,
          "successfulTests": [
          "div",
          "mult",
          "sub",
          "add",
          "sum"
          ],
          "testFailures": []
          },
          {
          "testName": "CalculatorSecondTest",
          "testCount": 5,
          "failureCount": 1,
          "successfulTests": [
          "add2",
          "sub2",
          "div2",
          "sum2"
          ] "testFailures": [
          {
          "testHeader": "mult2(CalculatorSecondTest)",
          "message": "expected:<15.0> but was:<10.0>",
 }
 "compilationErrors": [
          {
          "code": "compiler.err.expected",
          "columnNumber": 0,
          "kind": "ERROR",
          "lineNumber": 0,
          "message": "';' expected",
          "position": 46,
          "filePath": "/tmp/TaskNotCompilable.java",
          "startPosition": 46,
          "endPosition": 46
          }
  ] }
```

The above shows the result of two JUnit test files (CalculatorTest and CalculatorSecondTest). The field “testCount” indicates the number of test methods within the test file. The field “failureCount” indicates how many tests have failed and the field “successfulTests” indicates the method names of passed tests. In case a test failed, the necessary information can be found as an entry in the "testFailures" array.

If there was an compilation error the relevant information is part of the "compilationErrors" array as shown below.

Interface description

The web service offers three REST endpoints:
POST /v1/unittest
Used for uploading and creating of assignments. The body needs to contain two fields as form data:
assignmentId: The ID of the created assignment. This is created by moodle.
repositoryTestFile: The ZIP file containing the repository .txt file this assignment.
DELETE /v1/unittest?assignmentId=<111>
Delete the created assignment. The assignment ID of the unit tests which need to be deleted is passed as a query parameter.
POST /v1/task
The upload of the Java files to be tested. The body needs to contain two form fields:
repositoryFile: The ZIP file containing the repository as .txt file.
assignmentId: The id of the assignment. Provided by moodle

Frontend

Moodle originally has been an acronym for Modular Object-Oriented Dynamic Learning Environment. The modularity of Moodle is achieved by having a sophisticated plugin environment, providing vast amounts of existing plugins to choose from – or in case of developing your own plugin – several plugin types to build upon.

This section describes the plugin developed, starting with an overview of the chosen plugin type, then describing the file structure of the plugin and finally its usage.

Part of the following documentation is oriented towards the official Moodle documentation for the Assignment Submission Plugin which can be found here: https://docs.moodle.org/dev/Assign_submission_plugins

Plugin Type – Assignment Submission Plugin

Moodle provides more than 50 standardized plugin types to choose from when writing a plugin and if none of the standardized types fit there is the “local” type for a generic plugin for local customisation.

For our case the Assignment Submission Plugin type was the best fit. It allows you to display custom form fields to the students when they are editing their assignment submission as well as to the teachers when there are editing the assignment settings. It also has full control over the display of the submitted assignment to graders and students.

In short the main features are:

- An assignment submission plugin can add settings to the module settings page.
- An assignment submission plugin can show a summary of the submission to students and graders.
- An assignment submission plugin can add form fields to the student submission page.

File Structure

All the files of the MoDoCoT assignment submission plugin should be located under “mod/assign/submission/modocot” within the root folder of the Moodle installation. This section briefly describes the files and their purpose within the plugin. Some files will be explained in more detail where applicable. For more information about the files described please have a look at the official Moodle documentation and/or the source code of the plugin:

https://transfer.hft-stuttgart.de/gitlab/HFTSoftwareProject/moodle-assignsubmission_modocot

version.php

This file is used to tell Moodle the version information about our plugin, so that it can be installed and upgraded correctly. This information is added to version.php, which is also the case for any other type of Moodle plugin.

For more information please refer to:version.php

settings.php

This settings file allows us to add custom settings to the system wide configuration page for our plugin.

As is there are two settings described in this file for the MoDoCoT plugin:

- default: A checkbox to indicate if the plugin should be enabled by default when creating a new assignment.
- Web service base url: A textfield to define the base url of the web service that is used to communicate the files to and perform the actual tests.

lang/<country_code>/assignsubmission_modocot.php

These are the language files for the plugin. Depending on the language to support, the language files reside in a different subfolder of lang. For example:

- English: lang/en/assignsubmission_modocot.php
- German: lang/de/assignsubmission_ modocot.php
- The filename itself should be the same as the component name of the plugin. The component name of the plugin has the form of <plugintype>_<pluginname>, so assignsubmission_modocot.

Such a language file contains several key,value entries in the form of $string[“key”] = “Value”; where the key is the same throughout the different language files and the value is depending on the given language.

Moodle provides a dedicated String API that allows – given a key – the retrieval of the value depending on the selected language (e.g. get_string(“key”, “default value”)).

db/upgrade.php

This file handles upgrading the plugin to match the latest version. If for example a newer version of the plugin requires additional database tables or columns, this is the place to define them.
See Activity_modules#upgrade.phpfor more information.

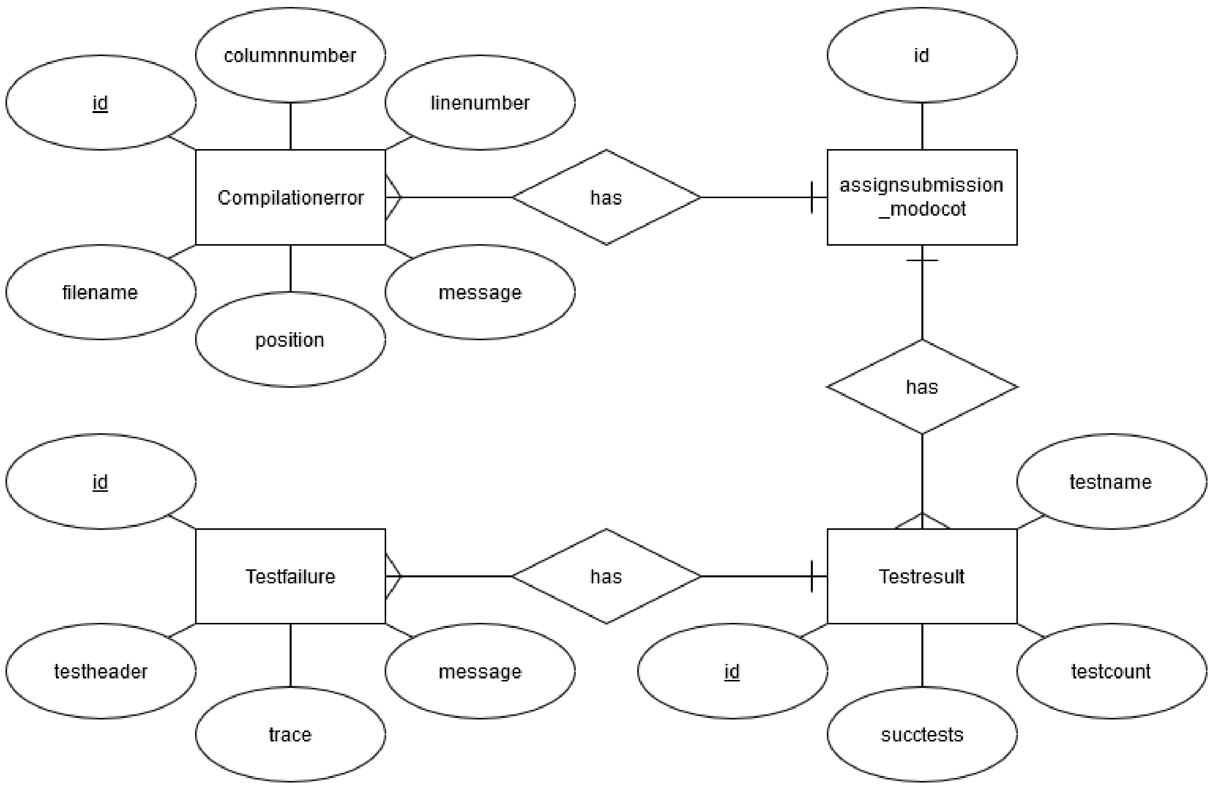
db/install.xml

This is where any database tables required to save this plugins data are defined. Moodle provides a dedicated XML schema to model this kind of information with elements such as TABLENAME, FIELDS and KEY. In addition Moodle comes with a XMLDB editor that supports the creation of the install.xml without having to directly get in touch with XML.

The code below shows for example how the table “modocot_testfailure” is defined, with all its fields as well as primary and foreign key definitions.

```
<TABLE NAME="modocot_testfailure" COMMENT="Info about the failures occured during
test execution.">
 <FIELDS>
   <FIELD NAME="id" TYPE="int" LENGTH="10" NOTNULL="true" SEQUENCE="true"/>
   <FIELD NAME="testresult_id" TYPE="int" LENGTH="10" NOTNULL="true" SEQUENCE="false"/> <FIELD
   NAME="testheader" TYPE="char" LENGTH="255" NOTNULL="true" SEQUENCE="false"/> <FIELD
   NAME="message" TYPE="char" LENGTH="255" NOTNULL="false" SEQUENCE="false"/> <FIELD
   NAME="trace" TYPE="text" NOTNULL="false" SEQUENCE="false"/>
 </FIELDS>
 <KEYS>
   <KEY NAME="primary" TYPE="primary" FIELDS="id"/>
   <KEY NAME="fk_testresult" TYPE="foreign" FIELDS="testresult_id"
REFTABLE="modocot_testresult" REFFIELDS="id"/>
 </KEYS>
</TABLE>
```

The general structure of data is depicted in the following ER-Model.



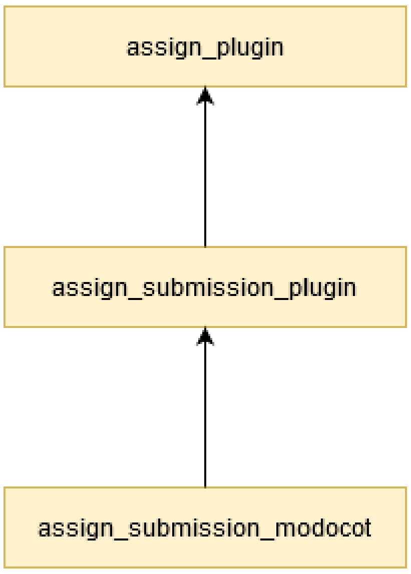
Every time a new MoDoCoT assignment is created in Moodle, a new instance of an “assignsubmission_modocot” entity is created. Again, the name of this table has to be in the form of <plugintype>_<pluginname>. On submission of a task file, the backend web service will be contacted and responds with one or more test results or compilation errors encoded as a JSON-String. The JSON-String will then be parsed and the data is stored in the corresponding database tables for future retrieval.

locallib.php

This is where all the functionality for this plugin is defined. All submission plugins must define a class with the component name of the plugin that extends ***assign_submission_plugin***.

```
class assign_submission_modocot extends assign_submission_plugin {
```

That means we have the following class hierarchy:



The ***assign_submission_plugin*** class is an abstract base class all assignment submission plugins must extend. It contains a small number of additional functions that only apply to submission plugins.
The ***assign_plugin*** class is an abstract class that is the base class for all assignment plugin (feedback or submission plugins). It provides access to the ***assign*** which represents the current assignment instance through “$this->assignment”.
Overall those two classes provide a number of public functions, so called hooks, that can be overridden in order to implement the functionality needed.

In the following a few selected functions will be shortly described to give an impression of the kind of hooks that are present.

- **get_settings()**: The get_settings function is called when building the settings page for the assignment. It allows this plugin to add a list of settings to the form. In case of the MoDoCoT plugin a file manager to allow the teachers to upload their JUnit test ZIP is added. Its overridden from the assign_plugin class.
- **save_settings():** The save_settings function is called when the assignment settings page is submitted, either for a new assignment or when editing an existing one. In the MoDoCoT plugin this function saves the JUnit test ZIP selected by the teacher and transfers the file to the backend web service. Its overridden from the assign_plugin class.
- **get_form_elements_for_user():** This function is called when building the submission form and allows (like the get_settings function for the settings) to add a list of elements to the submission form. In case of the MoDoCoT plugin this function adds file manager to allow the students to upload their repository ZIP file. Its overridden from the assign_plugin class.
- **save():** This function is called to save a user submission. Within the MoDoCoT plugin this function does the following things:

○ Save the uploaded repository ZIP file with tasks in the Moodle database.
○ Call the backend web service to transfer and the test the file
○ Receive and the process the web service response
○ Save the test results in the Moodle database

- **view_summary():** This function is called to display a summary of the submission to both markers (teachers) and students. For the students this summary will be shown within the submission status table and for the teachers within a column of the grading table. In the MoDoCoT plugin this method returns a more compact view (only essential data) for the grading table and a detailed view for the students submission status table.

```
public function view_summary(stdClass $submission, & $showviewlink) {
   global $PAGE;

   if ($PAGE->url->get_param("action") == "grading") {
       return $this->view_grading_summary($submission, $showviewlink);
   } else {
       return $this->view_student_summary($submission);
   }
}
```

- **delete_instance():** This function is called when the assignment has been deleted and is used for clean-up purposes. For the MoDoCoT plugin this means all the test result, compilation error records etc. are deleted. In addition the backend web service is called to trigger the deletion of the assignment’s test files

lib.php

This file is the entry point to many standard Moodle APIs for plugins. An example is that in order for a plugin to allow users to download files contained within a filearea belonging to the plugin, they must implement componentname_pluginfile function in order to perform their own security checks. In case of the MoDoCoT plugin this function is named “assignsubmission_modocot_pluginfile” and checks for example if the user requesting the file download is actually logged in and has the necessary permissions.

Backend

Framework

This is the web service used for the moodle plugin. It is written in Java and uses the Spring Boot framework. Spring is a largely used Java framework and with the Spring Boot extension it provides a fast gettings started experience for Spring development.

Build

As build and dependency management tool, Apache Maven is used. The application can be build using mvn package.

Application configuration

The backend web service is using the application.properties file to configure our application.

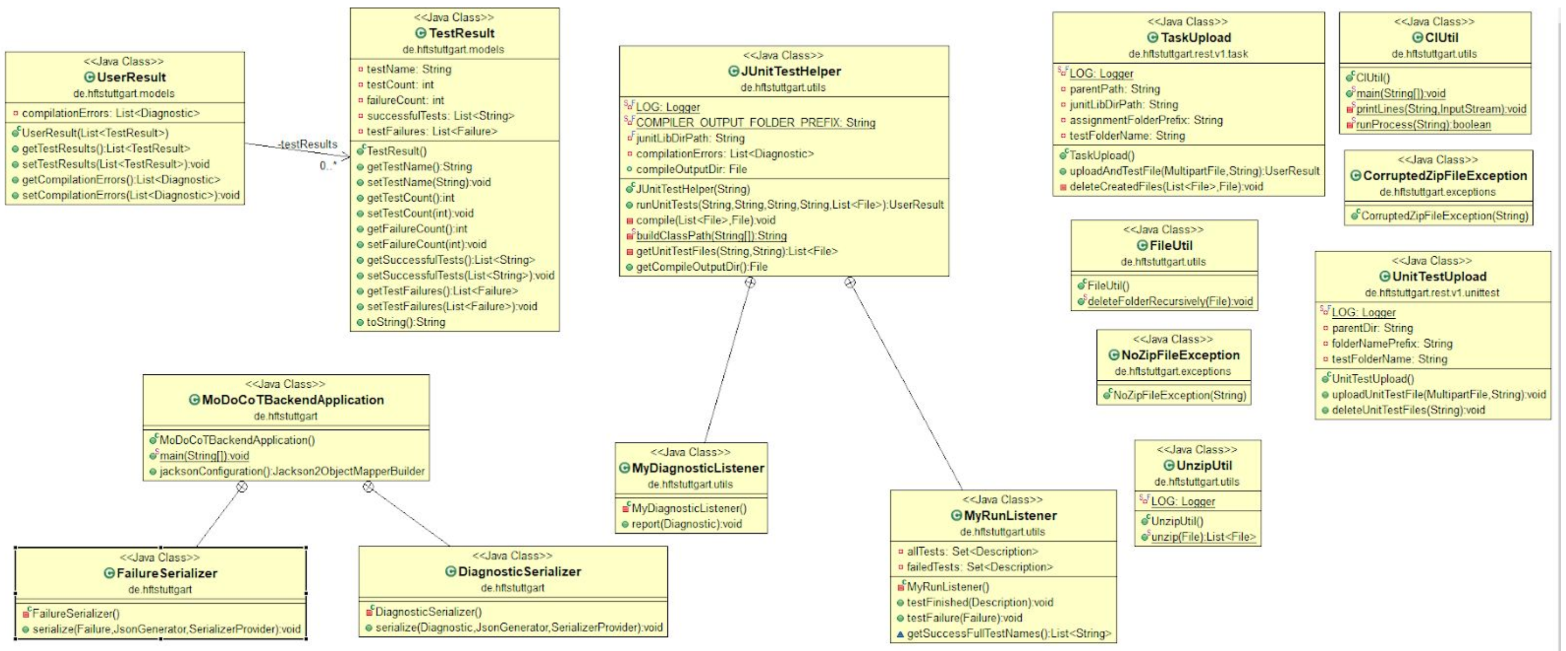
To configure your local configuration create a file called application-local.properties in /src/main/resources/ and override the properties. Afterwards configure the application to use the local profile using the run configuration or adding spring.profiles.active=local to the global application.properties file.

Integration tests

MoDoCoT-Backend has some rudimentary API tests using Spring Boot Testing. This tests assure that there won't be any regressions in the API when changing the backend code.

To be able to run the integration tests the system where the tests are executed needs to be a *nix System because a /tmp/ folder must exist. Also the needed libraries JUnit and Hamcrest need to be downloaded into /opt/mojec/junit/. This is the reason why the tests are disabled by default so it can be build on MS Windows systems. The tests can be enabled by setting -DskipTests=false

Overview of MoDoCoT Class UML



<<Java Class>>
UserResult
de.hftstuttgart.models
compilationErrors: List<Diagnostic>
UserResult(List<TestResult>)
getTestResults():List<TestResult>
setTestResults(List<TestResult>):void
getCompilationErrors():List<Diagnostic>
setCompilationErrors(List<Diagnostic>):void
-testResults
0..*
<<Java Class>>
TestResult
de.hftstuttgart.models
testName: String
testCount: int
failureCount: int
successfulTests: List<String>
testFailures: List<Failure>
TestResult()
getTestName():String
setTestName(String):void
getTestCount():int
setTestCount(int):void
getFailureCount():int
setFailureCount(int):void
getSuccessfulTests():List<String>
setSuccessfulTests(List<String>):void
getTestFailures():List<Failure>
setTestFailures(List<Failure>):void
toString():String
<<Java Class>>
JUnitTestHelper
de.hftstuttgart.utils
LOG: Logger
COMPILER_OUTPUT_FOLDER_PREFIX: String
junitLibDirPath: String
compilationErrors: List<Diagnostic>
compileOutputDir: File
JUnitTestHelper(String)
runUnitTests(String,String,String,String,List<File>):UserResult
compile(List<File>,File):void
buildClassPath(String[]):String
getUnitTestFiles(String,String):List<File>
getCompileOutputDir():File
<<Java Class>>
TaskUpload
de.hftstuttgart.rest.v1.task
LOG: Logger
parentPath: String
junitLibDirPath: String
assignmentFolderPrefix: String
testFolderName: String
TaskUpload()
uploadAndTestFile(MultipartFile,String):UserResult
deleteCreatedFiles(List<File>,File):void
<<Java Class>>
CIUtil
de.hftstuttgart.utils
CIUtil()
main(String[]):void
printLines(String,InputStream):void
runProcess(String):boolean
<<Java Class>>
CorruptedZipFileException
de.hftstuttgart.exceptions
CorruptedZipFileException(String)
<<Java Class>>
FileUtil
de.hftstuttgart.utils
FileUtil()
deleteFolderRecursively(File):void
<<Java Class>>
UnitTestUpload
de.hftstuttgart.rest.v1.unittest
LOG: Logger
parentDir: String
folderNamePrefix: String
testFolderName: String
UnitTestUpload()
uploadUnitTestFile(MultipartFile,String):void
deleteUnitTestFiles(String):void
<<Java Class>>
NoZipFileException
de.hftstuttgart.exceptions
NoZipFileException(String)
<<Java Class>>
MoDoCoTBackendApplication
de.hftstuttgart
MoDoCoTBackendApplication()
main(String[]):void
jacksonConfiguration():Jackson2ObjectMapperBuilder
<<Java Class>>
MyDiagnosticListener
de.hftstuttgart.utils
MyDiagnosticListener()
report(Diagnostic):void
<<Java Class>>
UnzipUtil
de.hftstuttgart.utils
LOG: Logger
UnzipUtil()
unzip(File):List<File>
<<Java Class>>
MyRunListener
de.hftstuttgart.utils
allTests: Set<Description>
failedTests: Set<Description>
MyRunListener()
testFinished(Description):void
testFailure(Failure):void
getSuccessFullTestNames():List<String>
<<Java Class>>
FailureSerializer
de.hftstuttgart
FailureSerializer()
serialize(Failure,JsonGenerator,SerializerProvider):void
<<Java Class>>
DiagnosticSerializer
de.hftstuttgart
DiagnosticSerializer()
serialize(Diagnostic,JsonGenerator,SerializerProvider):void

List of important Classes & Functions

1. UnzipUtil:
    a. unzip(file): Utility function to unzip the uploaded files and saves them to disk. Also checks if the ZIP file is valid.

2. JUnitTestHelper:
    a. runUnitTests: creates a temporary folder for the compilation output, loads compiled classes into the classloader and runs JUnit tests. Returns a list of all successful, failed and not compilable tests.
    b. compile: Sets the compiler option for a specific output path, compiles it, and if the compilation fails, tries to compile again without the not compilable file.
    c. buildClassPath: Builds a custom class path. This is needed because the JUnit.jar dependency needs to be in the classpath when compiling the JUnit tests.
    d. getUnitTestFiles: Gets all the JUnit Test files from the specified path.

3. UnitTestUpload
    a. uploadUnitTestFile: REST resource for the JUnit test upload: Creates one folder per assignment and unzipes the JUnit files into this subfolder.
    b. deleteUnitTestFiles: REST resource for delete uploaded JUnit tests.

4. TaskUpload:
    a. uploadAndTestFile: REST resource for the upload of the Java files. Unzips the files into the subfolder and runs the tests or it clones the repositry from the link which is in the repo.txt file. Afterwards it creates / formats the result for the frontend.

Deployment

Docker installation:

- Installing Docker Toolbox adds these software:
    - For Windows:
        - Docker Client for Windows
        - Docker Machine for Windows
        - Docker Compose for Windows
        - VirtualBox
        - Kitematic for Windows (Alpha)
        - Git for Windows
    - For Linux:
        - sudo apt install docker.io
- Installing Docker Toolbox provides a Docker Terminal for running all Dockerfiles & images
- Then comes creating Dockerfiles as per the requirements
- The Dockerfile runs using Docker Terminal

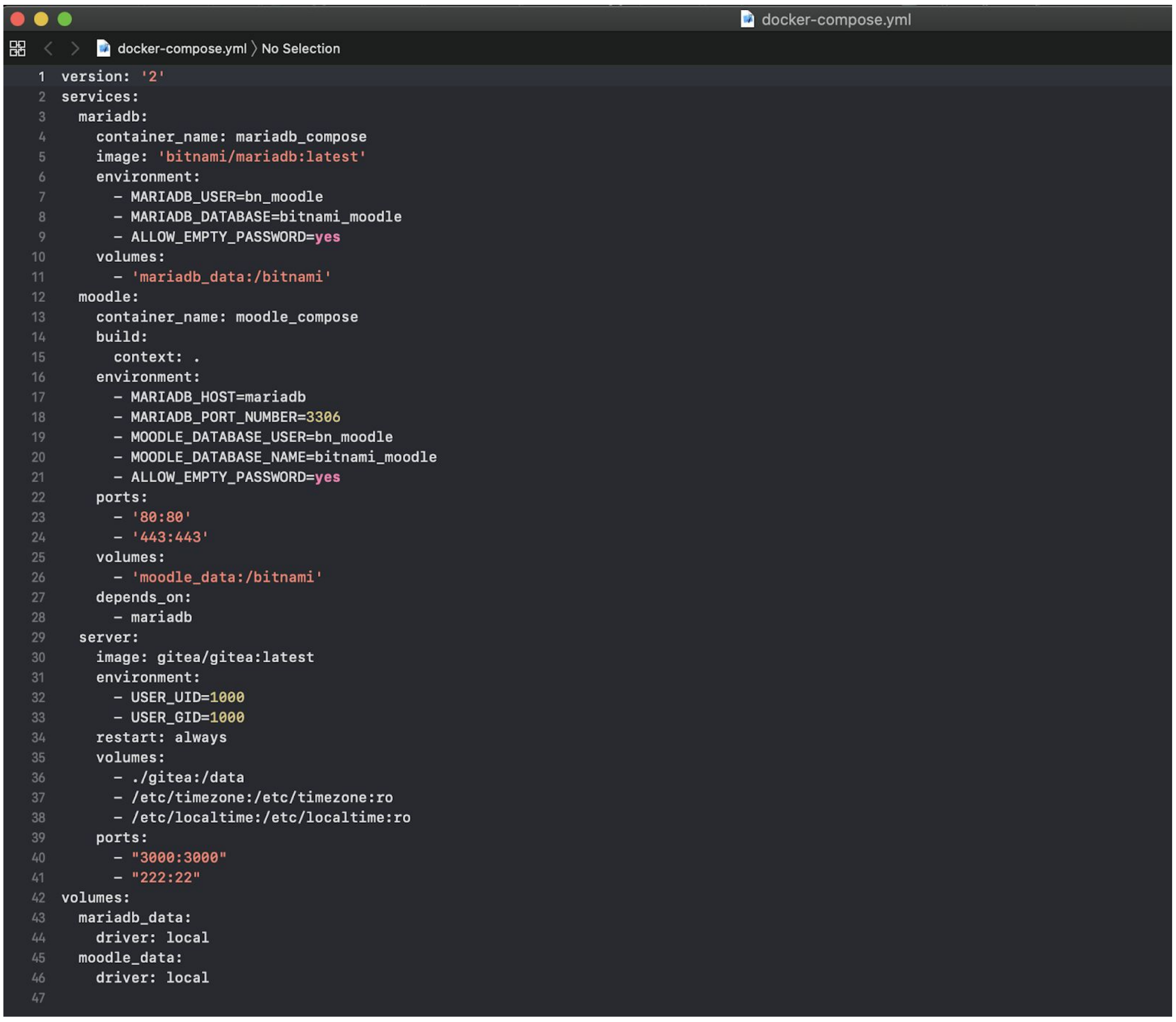
Dockerfile for MoDoCoT:

- This Docker image provides the LAMP stack, installs the latest bitnami-moodle including a SQLite3-Database and the MoDoCoT moodle plugin for JUnit Test Assignments.

Docker-Compose:

- ❖ Docker-compose is used to set up relatively complicated Docker networks and to change or restart them if necessary. In this way you can manage multiple containers. A side effect is that you only need a YML file.
- ❖ Each compose file is named docker-compose.yml and can not be changed, otherwise the container won't start. These are always started with the following command : `sudo docker-compose up –d`

❖ This file is structured like this :



```yaml
version: '2'
services:
  mariadb:
    container_name: mariadb_compose
    image: 'bitnami/mariadb:latest'
    environment:
      - MARIADB_USER=bn_moodle
      - MARIADB_DATABASE=bitnami_moodle
      - ALLOW_EMPTY_PASSWORD=yes
    volumes:
      - 'mariadb_data:/bitnami'
  moodle:
    container_name: moodle_compose
    build:
      context: .
    environment:
      - MARIADB_HOST=mariadb
      - MARIADB_PORT_NUMBER=3306
      - MOODLE_DATABASE_USER=bn_moodle
      - MOODLE_DATABASE_NAME=bitnami_moodle
      - ALLOW_EMPTY_PASSWORD=yes
    ports:
      - '80:80'
      - '443:443'
    volumes:
      - 'moodle_data:/bitnami'
    depends_on:
      - mariadb
  server:
    image: gitea/gitea:latest
    environment:
      - USER_UID=1000
      - USER_GID=1000
    restart: always
    volumes:
      - ./gitea:/data
      - /etc/timezone:/etc/timezone:ro
      - /etc/localtime:/etc/localtime:ro
    ports:
      - "3000:3000"
      - "222:22"
volumes:
  mariadb_data:
    driver: local
  moodle_data:
    driver: local
```

❖ A container with a database is started in the Docker compose (MariaDB), which is called mariadb_compose. So it can be started with docker-compose up -d mariadb_compose. If you want to start several containers, just use docker-compose up -d.

❖ With container_name you give the container a name, this is useful if you have loaded many Docker containers. Because without that, random names would be given and you completely lose the overview.

❖ With a volume, paths on the host system can be linked to paths within the container. In this way, files are retained when the container is updated or when the container / server is restarted.

❖ With an environment, commands are executed within the container.

- ❖ With the specification ports it is possible to address a container via a port
- ❖ Restart tells the Docker daemon that this container should be started every time the server is restarted. Possible commands would be

Continuous Deployment environment:

The software installation required on KVM server:

- JDK:Execute following commands on the terminal for installation
  - `sudo apt-get install default-jdk`
  - `sudo apt-get install default-jre`
- Maven:Execute below command on the terminal to install Maven
  - `sudo apt-get install maven`
- Jenkins: Execute below command on the terminal to get jenkins
  - `wget http://mirrors.jenkins.io/war-stable/latest/jenkins.war`
  - save it in accessible folder, go inside that folder using cd
  - Execute below command on the terminal to install Jenkins
    `java -jar jenkins.war`
- To push the latest image on the Docker machine. To pull the latest changes for the plugin to update the MoDoCoT Front-End plugin.

Docker Toolbox installation:

Click the below link to install the Docker tool box on windows machine
https://docs.docker.com/toolbox/toolbox_install_windows

MoDoCoT - Docker Hub:

https://hub.docker.com/r/hftstuttgart/modocot-backend

Jenkins installation:

Click the below link to get the Jenkins war file for installation
https://jenkins.io/

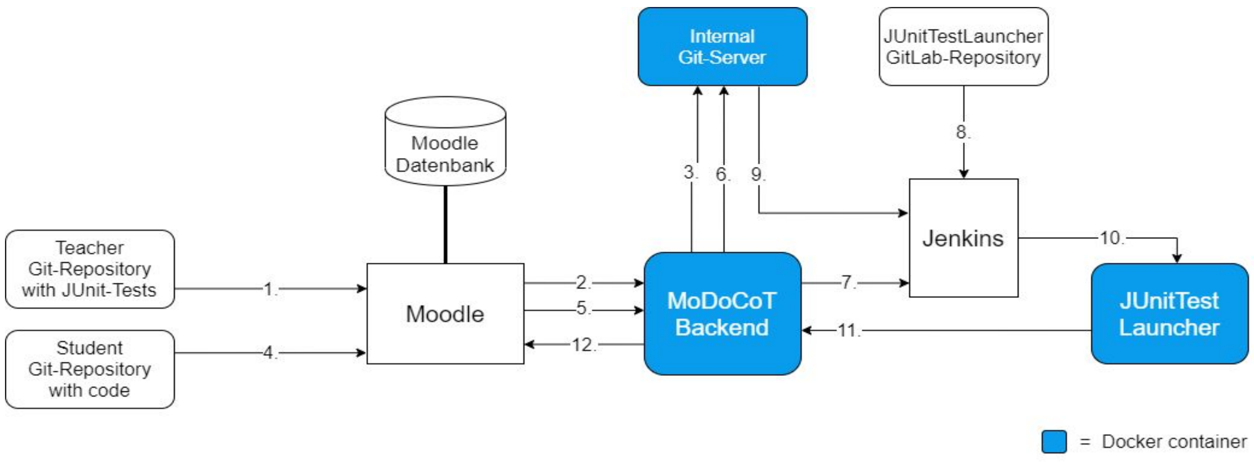
Continuous deployment workflow

- Execute below command on the terminal to install Jenkins
  `java -jar jenkins.war`

- To push the latest image on the Docker machine. To pull the latest changes for the plugin to update the MoDoCoT Front-End plugin.

Docker Toolbox installation:

Click the below link to install the Docker tool box on windows machine
https://docs.docker.com/toolbox/toolbox_install_windows/



1. The Teacher uploads a ZIP-Archive with a Text-File "repo.txt" witch contains a link to a Git-Repository containing the JUnit-Tests to the Moodle-Frontend.
2. Moodle-Frontend passes the ZIP to the MoDoCoT-Backend.
3. The Backend unzips the archive and clones the Repository with the tests on an internal Git-Server.
4. The Student uploads a ZIP-Archive with a Text-File "repo.txt" witch contains a link to a Git-Repository containing the task-code to the Moodle-Frontend.
5. Moodle-Fontend passes the ZIP to the MoDoCoT-Backend.
6. The Backend unzips the archive and clones the Repository with the tasks on an internal Git-Server in an unique Repository together with the tests.
7. Then the Backend starts a Jenkins-Pipeline. This pipeline searches the Jenkinsfile in the Repository, witch setup the 'JUnitTestLauncher'-Docker container, and executes it.
8. The taks and tests need to be cloned into the container. Which are in one Repository, identified with AssignmentID and UUID.
9. The 'JUnitTestLauncher'-Repository needs to be cloned into it as well.
10. Then the container starts and execute the testing.
11. The results are send back to the Backend via a REST-API.
12. Finally the results are shown in the Moodle-Frontend.

Information on SSH Communication

To pull the latest changes of the Moodle plugin we created a shell script that will be use to execute a git pull and get the latest changes for the MoDoCoT Plugin. This is done via Shell Execution command in Jenkins

This will connect to the Docker machine via ssh and executes a script (which exists in the Jenkins machine)

To connect to the Docker Machine via ssh, username and password entry are required at each connection, which made a problem in the automation process. To bypass this problem the authentication using keys is used.

SSH Login without username and password

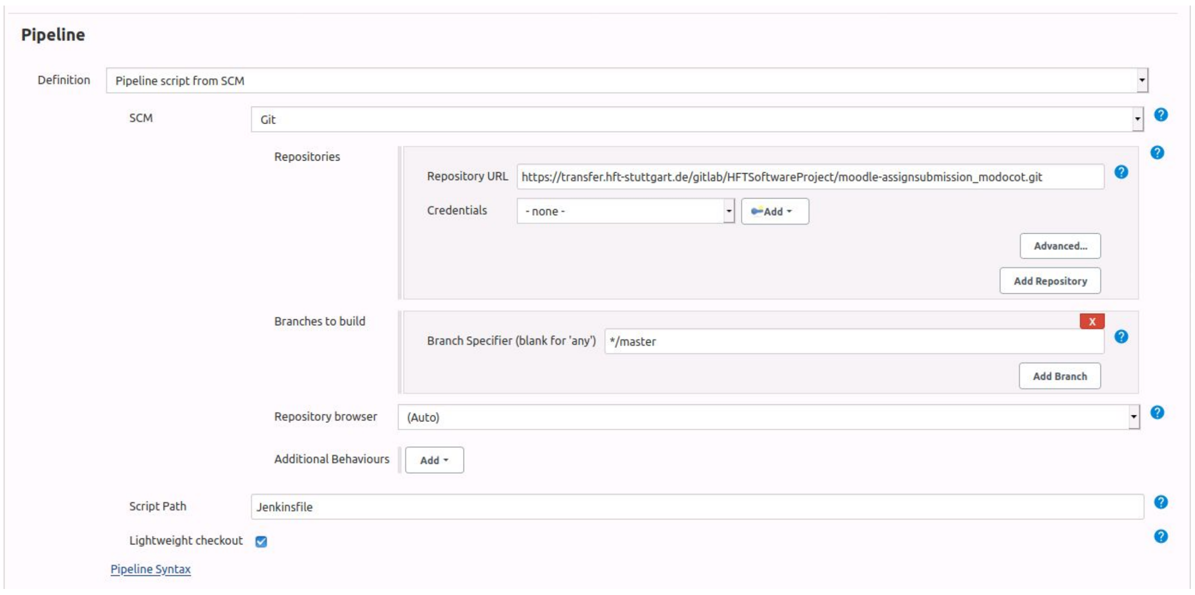
To be able to login from Jenkins to the Docker Machine using ssh the following steps should be done:

1. Run terminal on the Jenkins machine.
2. Execute “ssh-keygen” to generate the public and private key of the Jenkins Machine.
3. Execute “ssh-copy-id –i ~/.ssh/id_rsa.pub 10.40.10.144” (10.40.10.144 Docker Machine IP) which appends the public key to the authorized_keys file in the Docker Machine.
4. Enter the Docker Machine Password.

After following the above steps, all ssh connections from the Jenkins Machine to the Docker Machine will run without any other request for username or password.

Jenkins Job: Settings for MoDoCoT -Frontend:

1. Create new item with Free style project. **MoDoCoT -Frontend**
2. Once the New Item is created go to the Configuration to apply settings as per requirement, you can provide Project Name & Description
3. **Source Code Management:** In source code management we will set the URL of the Git repository from where the code is to be referred


Pipeline

Definition: Pipeline script from SCM

SCM: Git

Repositories
- Repository URL: https://transfer.hft-stuttgart.de/gitlab/HFTSoftwareProject/moodle-assignsubmission_modocot.git
- Credentials: - none -

Branches to build
- Branch Specifier (blank for 'any'): */master

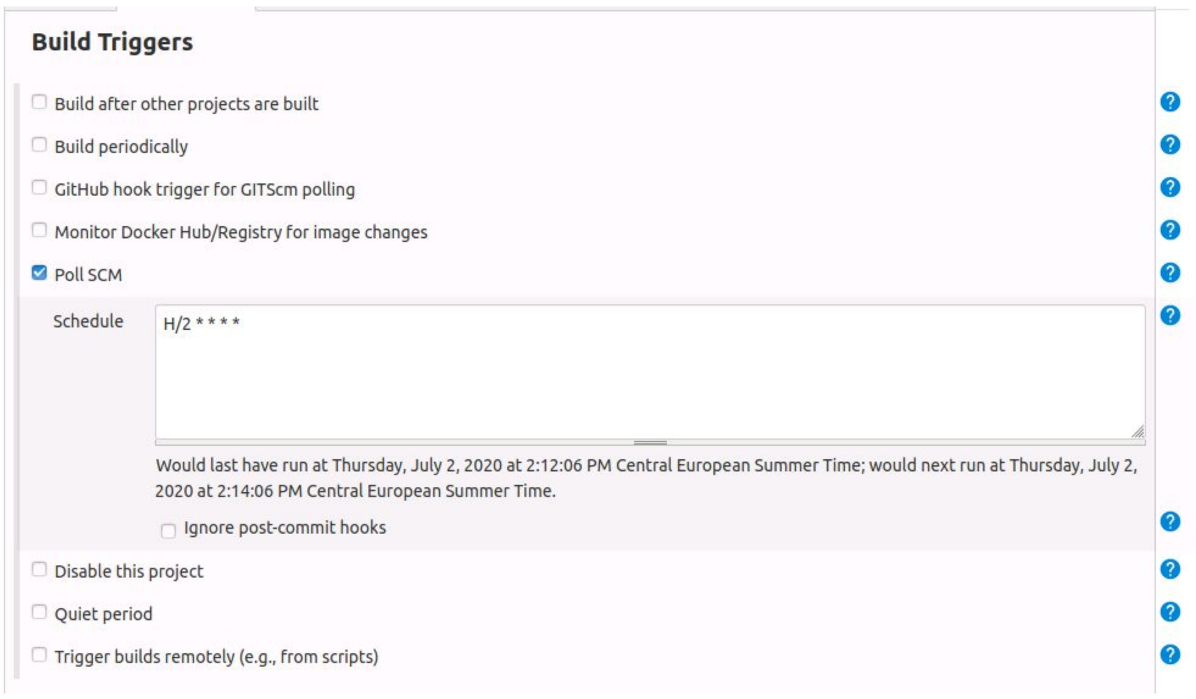
Repository browser: (Auto)

Additional Behaviours

Script Path: Jenkinsfile

Lightweight checkout: ☑

4. **Build Triggers:**Here we specify how often the Jenkins should trigger the Build. In the configuration settings, we have selected Poll SCM and the Schedule as H/2 * * * * that means it will trigger Build every 2 minutes


Build Triggers

- ☐ Build after other projects are built
- ☐ Build periodically
- ☐ GitHub hook trigger for GITScm polling
- ☐ Monitor Docker Hub/Registry for image changes
- ☑ Poll SCM

Schedule: H/2 * * * *

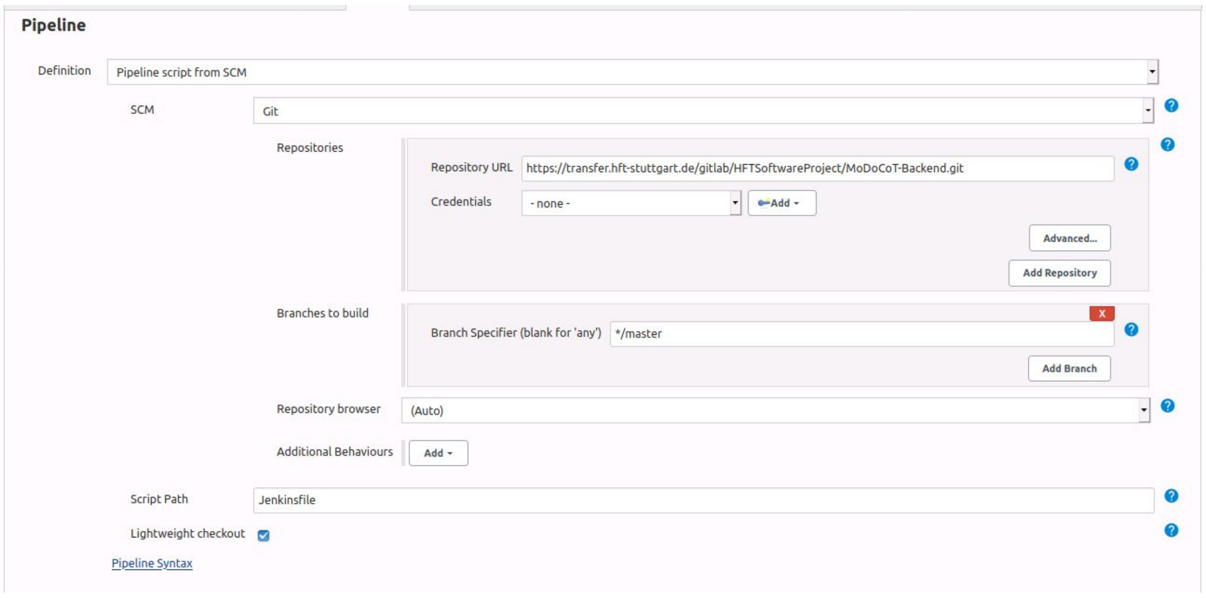
Would last have run at Thursday, July 2, 2020 at 2:12:06 PM Central European Summer Time; would next run at Thursday, July 2, 2020 at 2:14:06 PM Central European Summer Time.

- ☐ Ignore post-commit hooks
- ☐ Disable this project
- ☐ Quiet period
- ☐ Trigger builds remotely (e.g., from scripts)

5. **Build Environment:**Here settings related to what exactly has to be done is entered. In the below screenshot: Execute Shell: Connects to the Docker machine via ssh and executes a script (which exists in the Jenkins machine) to execute a git pull and get the latest changes for the MoDoCoT Plugin.

Jenkins Job: Settings to Automate MoDoCoT -Backend:

1. Create a new item with Free style project. **MoDoCoT -Backend**.
2. Once the New Item is created go to the Configuration to apply settings as per requirement, you can provide Project Name & Description.
3. **Source Code Management:**Here we will set the URL of the Git repository from where the code is to be referred.



Pipeline

Definition: Pipeline script from SCM

SCM: Git

Repositories

Repository URL: https://transfer.hft-stuttgart.de/gitlab/HFTSoftwareProject/MoDoCoT-Backend.git

Credentials: - none -

Branches to build

Branch Specifier (blank for 'any'): */master

Repository browser: (Auto)

Additional Behaviours

Script Path: Jenkinsfile

Lightweight checkout: ✓

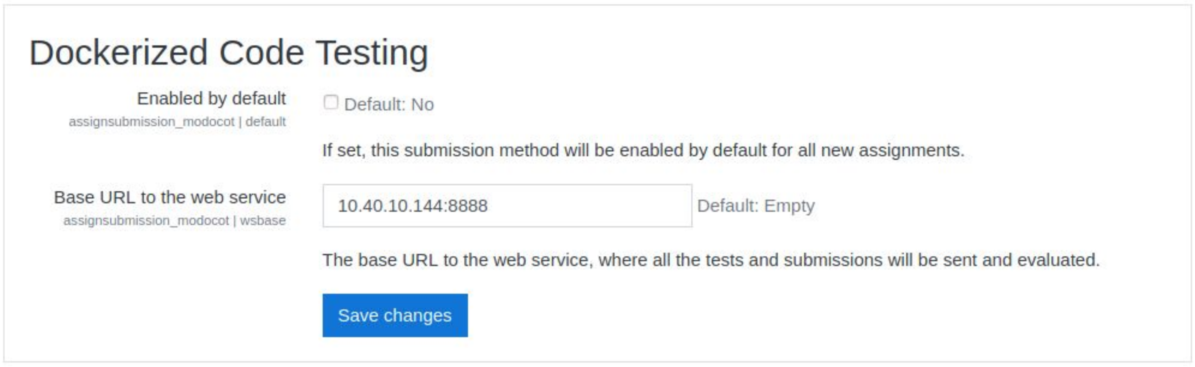
4. **Build Triggers**: Here we specify how often the Jenkins should trigger the Build. In the configuration settings, we have selected Poll SCM and the Schedule as H/2 * * * * that means it will trigger Build every 2 minutes.

5. **Build Environment:**Here settings related to what exactly has to be done is entered In the configuration settings: Clean package using Maven plugin: After the latest changes are pulled from the GitHub, this will remove and again package all. Docker Build & Publish Plugin: To compile, build the new image and push the new image to Docker Hub. Execute Shell: Connects to the Docker machine via ssh and executes a script (which exists in the Jenkins machine) to pull and run the new image.

The plugin used to perform docker operation was ‘Cloudbees Docker Build Push & Plugin’

Demo

The backend webservice has to be configured for the backend. This can be done in moodle: Site administration -> Plugins -> Plugins Overview -> Find "Junit Exercise Corrector". Go to the settings of the plugin and configure the backend URL as shown below.



Dockerized Code Testing

Enabled by default
assignsubmission_modocot | default

☐ Default: No

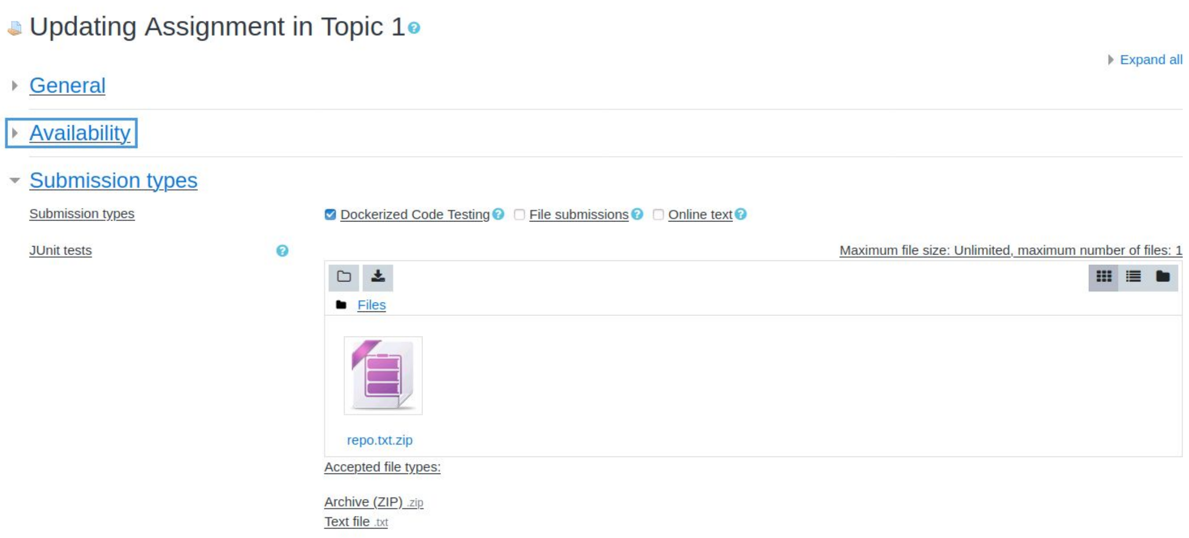
If set, this submission method will be enabled by default for all new assignments.

Base URL to the web service
assignsubmission_modocot | wsbase

10.40.10.144:8888 Default: Empty

The base URL to the web service, where all the tests and submissions will be sent and evaluated.

When adding new assignment, the Dockerized Code Testing is the type of submission to be selected to submit and run JUnit programs. After the selection of the submission types, the zipped repo.txt should be submitted to run the test and get back the weighted results.



Updating Assignment in Topic 1

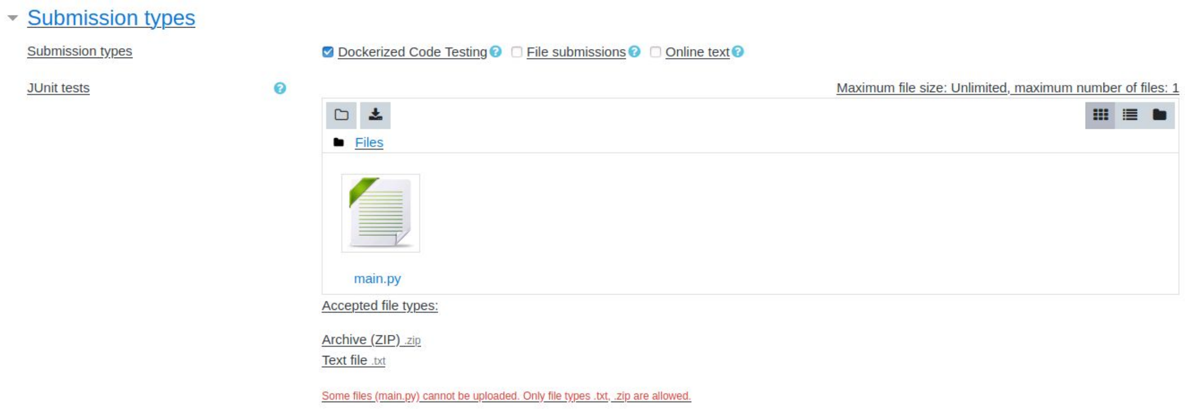
Submission types: ☑ Dockerized Code Testing ☐ File submissions ☐ Online text

JUnit tests — Maximum file size: Unlimited, maximum number of files: 1

Accepted file types:

Archive (ZIP) .zip
Text file .txt

Make sure the submission of the file should be a zipped format, the Exercise Corrector does not accept normal files and throws back Error message to the user.



Submission types

Submission types: ☑ Dockerized Code Testing ☐ File submissions ☐ Online text

JUnit tests

Maximum file size: Unlimited, maximum number of files: 1

Files

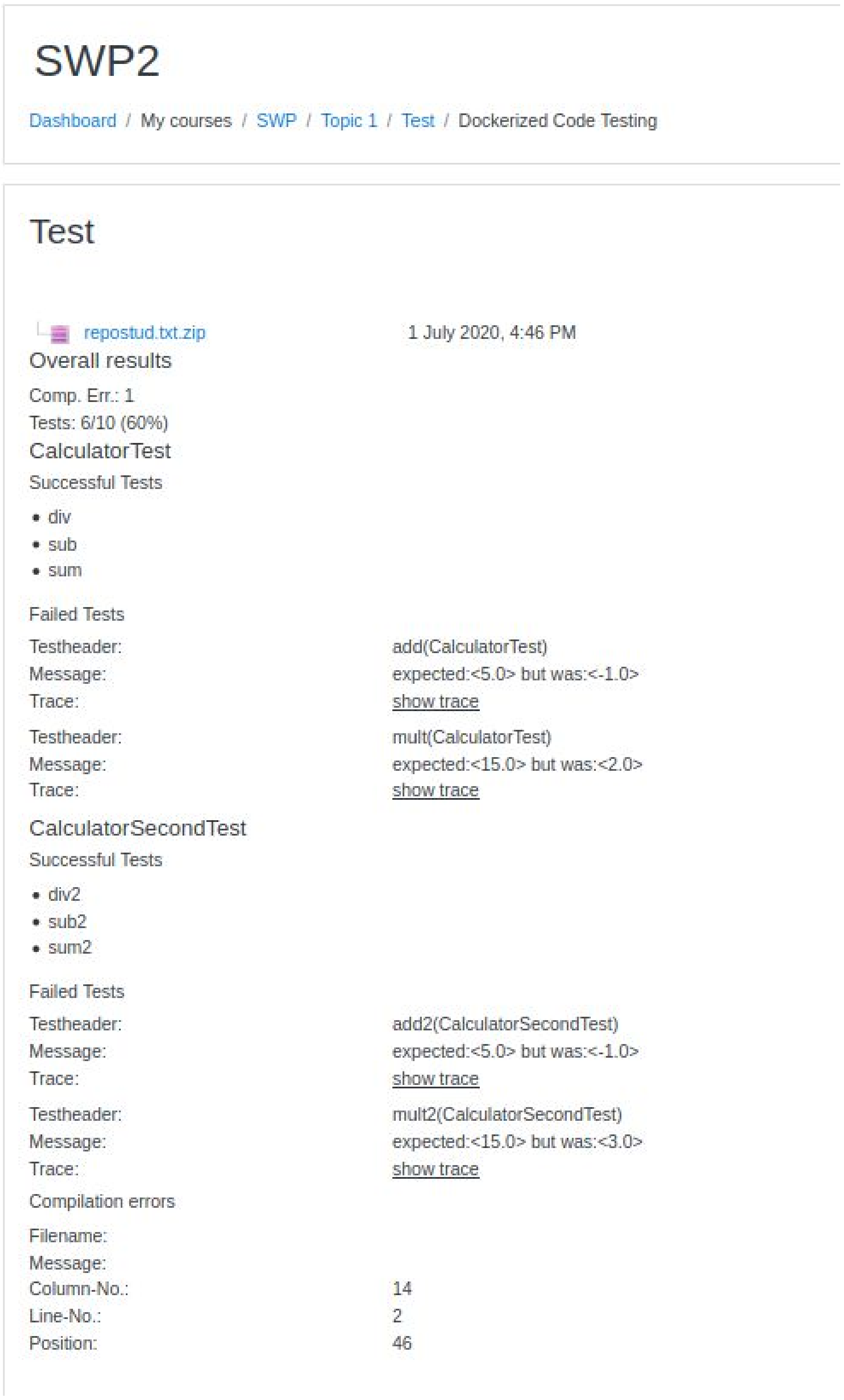
main.py

Accepted file types:

Archive (ZIP) .zip
Text file .txt

Some files (main.py) cannot be uploaded. Only file types .txt, .zip are allowed.

After successfully adding the ZIP file, the Dockerized Code Testing processes, run and compile the ZIP file. This will then calculates the number of successful tests and the compilation error, then submitted for grading. The overall results are presented so the professor grades the students exercise.



SWP2

Dashboard / My courses / SWP / Topic 1 / Test / Dockerized Code Testing

Test

repostud.txt.zip 1 July 2020, 4:46 PM

Overall results

Comp. Err.: 1

Tests: 6/10 (60%)

CalculatorTest

Successful Tests

- div
- sub
- sum

Failed Tests

Testheader:	add(CalculatorTest)
Message:	expected:<5.0> but was:<-1.0>
Trace:	show trace
Testheader:	mult(CalculatorTest)
Message:	expected:<15.0> but was:<2.0>
Trace:	show trace

CalculatorSecondTest

Successful Tests

- div2
- sub2
- sum2

Failed Tests

Testheader:	add2(CalculatorSecondTest)
Message:	expected:<5.0> but was:<-1.0>
Trace:	show trace
Testheader:	mult2(CalculatorSecondTest)
Message:	expected:<15.0> but was:<3.0>
Trace:	show trace

Compilation errors

Filename:	
Message:	
Column-No.:	14
Line-No.:	2
Position:	46