Coding in the cloud

New ways to teach programming classes

Matthias Möller

Overview

- **Motivation:** OOSP-C++ course and lessons learnt last years
- **INGInious:** Overall concepts and adjustments for OOSP-C++ course
- **Conclusions:** Got interested? Then give it a try….

Motivation

Course: Object-Oriented Scientific Programming with C++ (OOSP-C++)

- Started as PhD seminar with 5-8 participants in 2015
- Has become a BSc/MSc/PhD course with about 80-100 participants
- Participants from all faculties within DCSE with very varying background
- **Hands-on course** (2h lectures + 4h lab sessions per week):
  - Need for sufficiently many and well-trained TAs (difficult to find!)
  - Reduce technical problems and use TAs for content-related support

Learning Objectives

1. Students will learn to **design, implement, and systematically validate well-structured and maintainable efficient computer programs in C++** for **solving scientific problems from their field of applications**.

This requires a good **knowledge of modern C++** features:

- OOP techniques: polymorphism, inheritance, encapsulation, abstraction
- Template meta-programming and compile-time optimisation techniques
- New (and really powerful) concepts introduced in C++11, 14, 17

together with **discipline-related knowledge** not taught in this course.

Learning Objectives

2. Students will learn to **use professional software development tools and workflows (version control systems, IDEs, build systems, debuggers, …)** for **developing software projects in teams**.

This requires

- TU-wide availability of pre-defined software stack
- Willingness of students to use ICT infrastructure from TU
- Understanding of the need for it (not just among students)

Data Champions

TU Delft Cluster

Data Stewards

TU Delft ICT-services

DCSE D:Dream

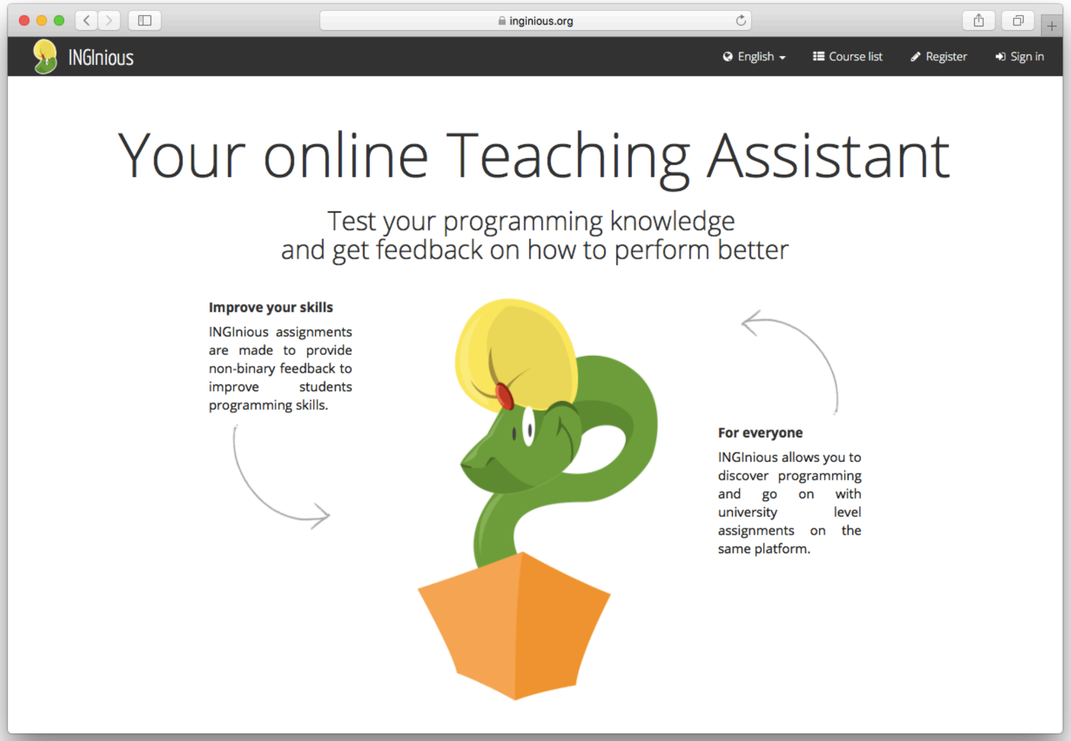
Lessons learned last years

1. Client-side solutions on students' computers are doomed to fail:
   - 1-2 weeks spent on giving installation support during lab sessions
   - $|S_{OSs} \times S_{Compilers}| >> 1$, so one cannot test all possibilities before
   - C++ *is* platform dependent, so what is the reference for grading
   - New C++14 or 17 features not available in outdated compilers

2. Students are reluctant to use TU computers (even if they worked)

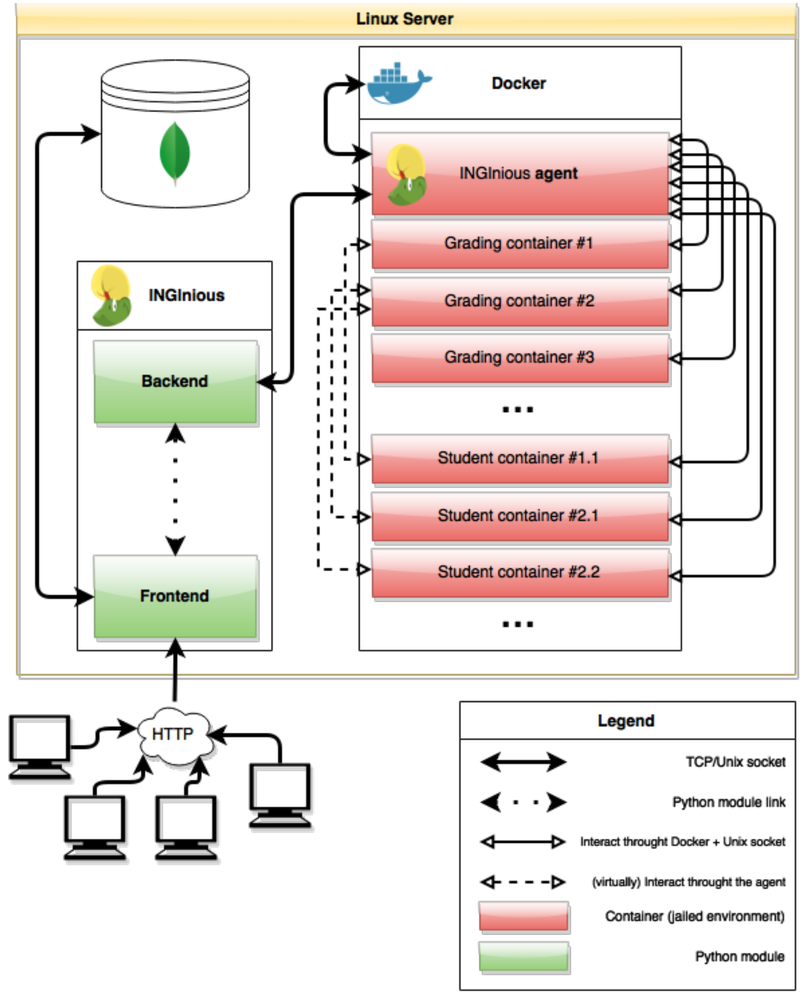
Conclusion: **Web-based server-side solution (with admin rights) needed; step-by-step integration of software development aspects in next years**

INGInious



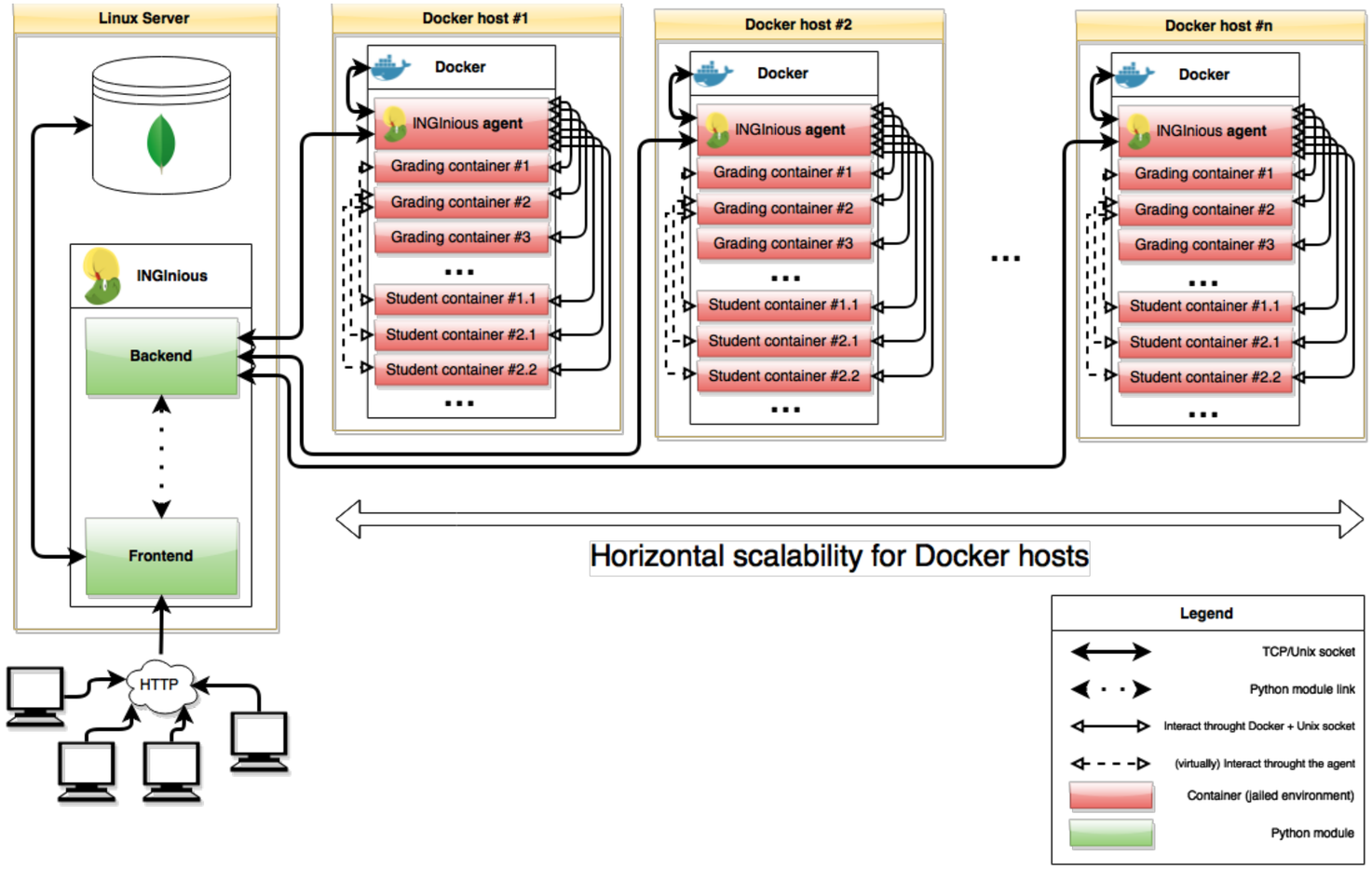
- Open-source tool developed by the CSE department at UCLouvain/BE
- Used at UCL and for edX courses

INGInious workflow

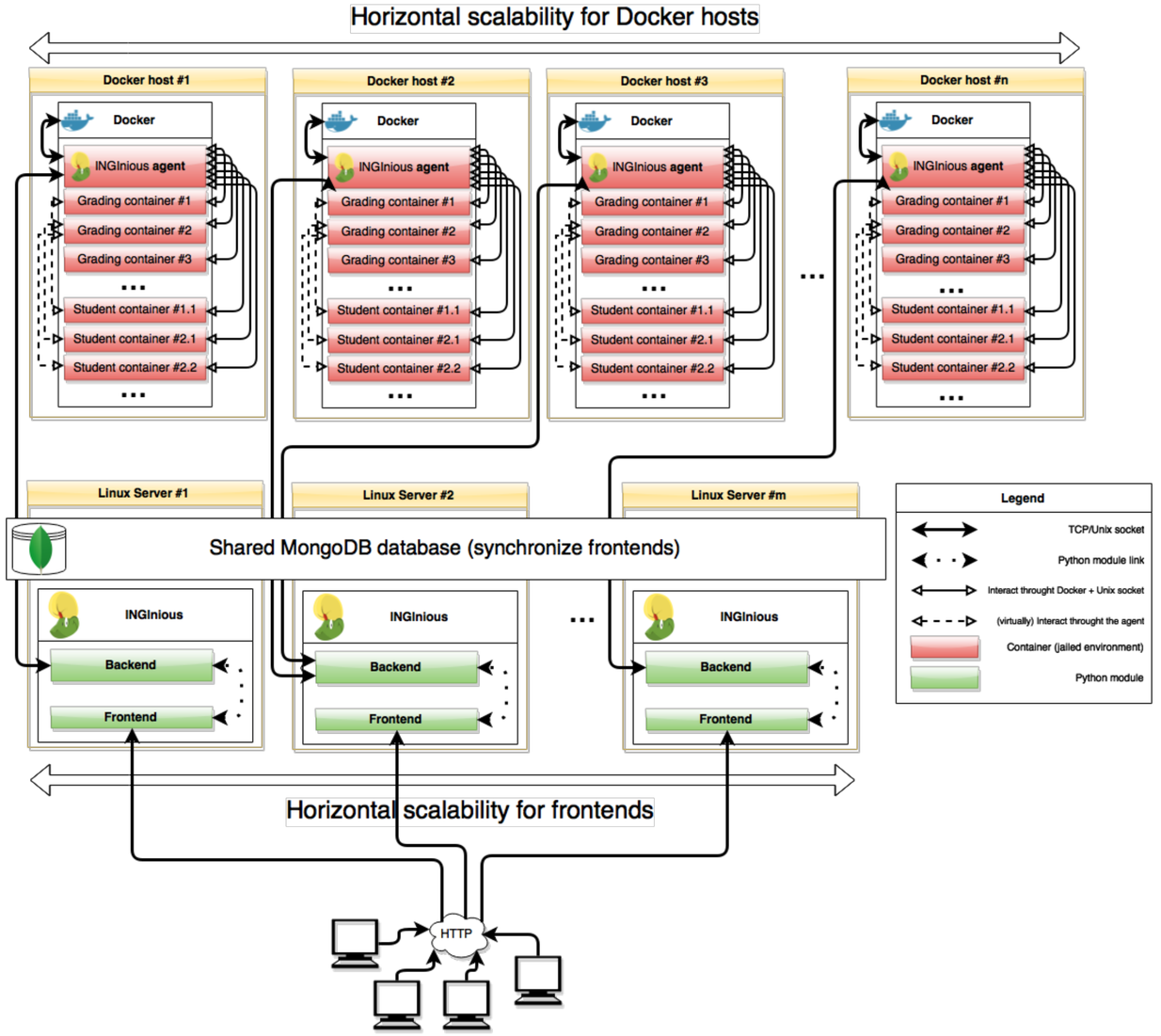


1. Student logs in to course web-site
2. Works on assignment in web formular
3. Submits the solution (history is stored)
4. *System assesses the solution in Docker container and returns feedback to student*
5. Student revises/accepts submission
6. *Final check by TAs/instructor for grading*

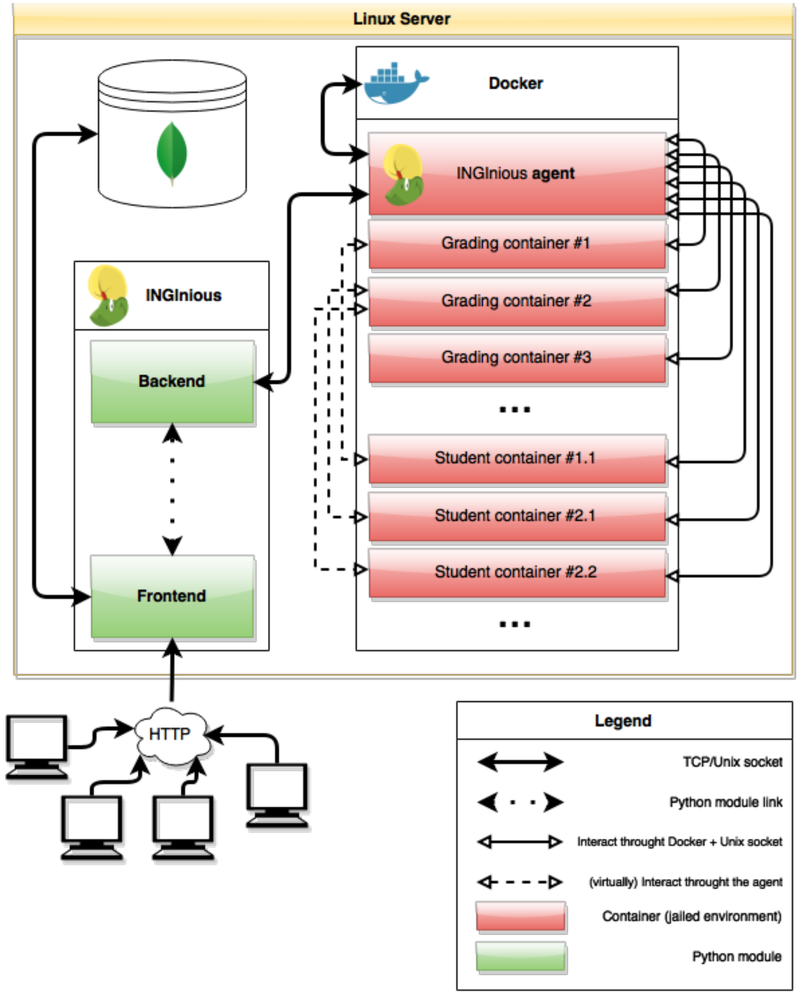
Scalability (near-term)



Scalability (long-term)



INGInious.EWI.tudelft.nl



- Runs on virtual server (CentOS 7) hosted, administrated and backed up by TUD-ICT
- First practical test in Q2 2018/19 for my OOSP-C++ course (130 registrations)

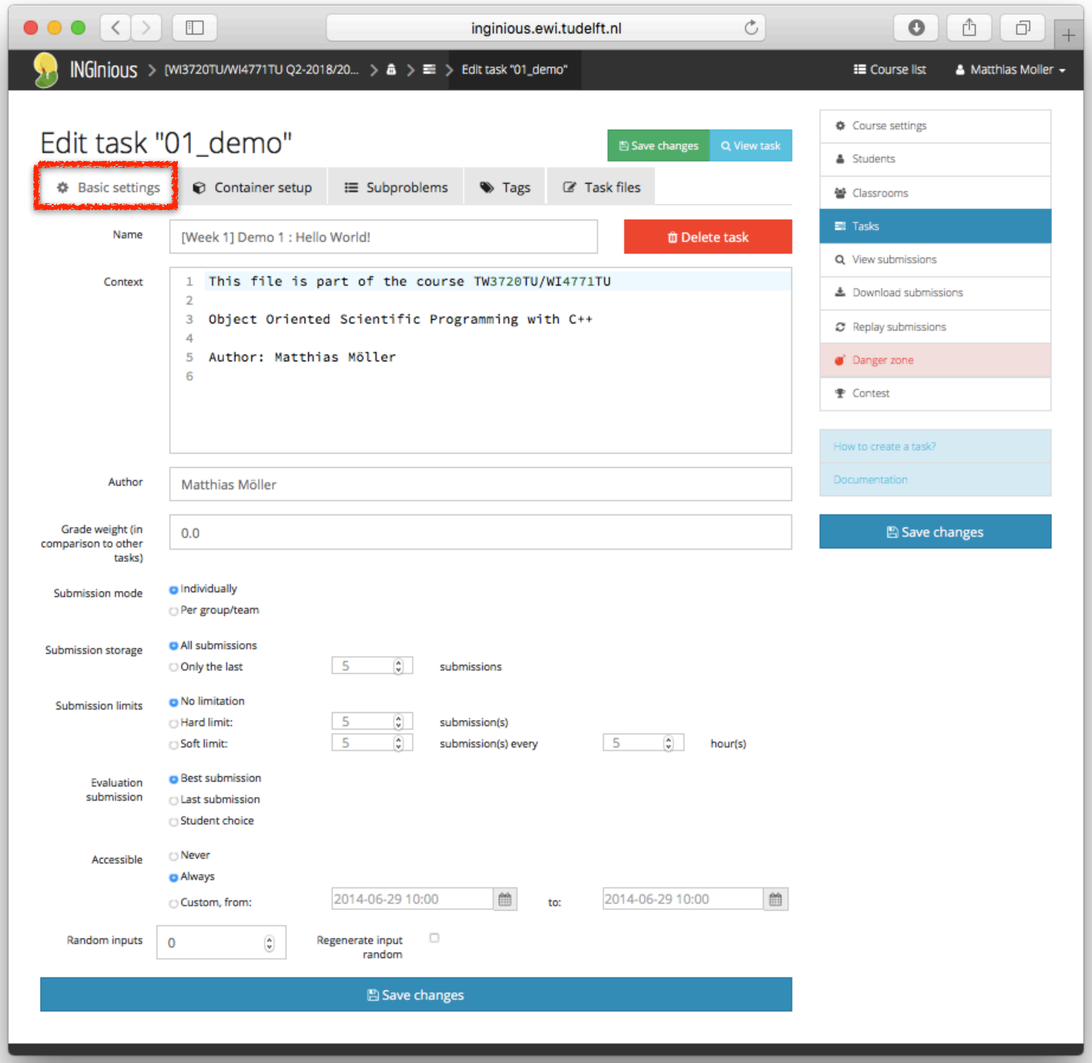
File-based course structure

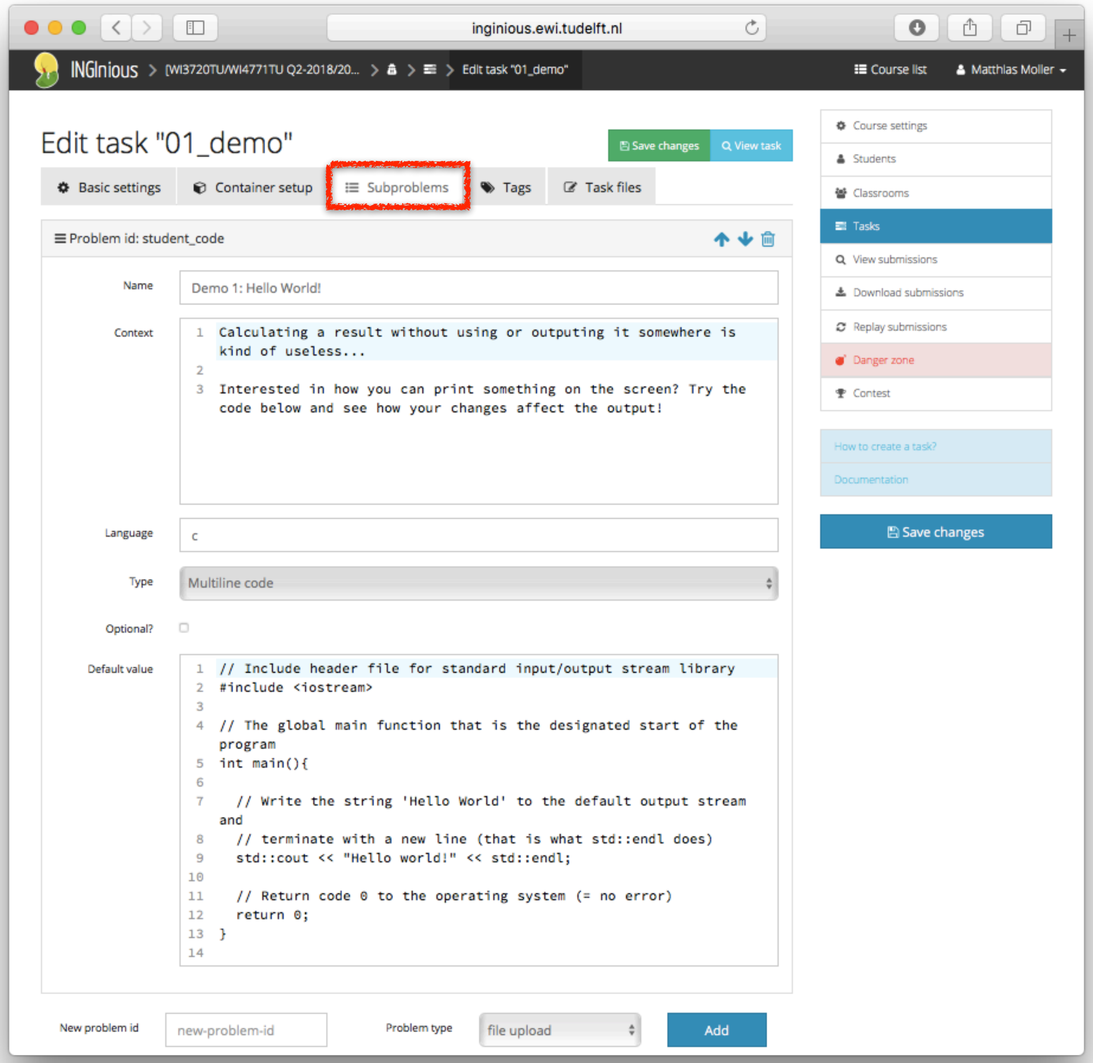
- /var/www/INGInious — Webserver top-level directory
  - /Course1 — Course1 top-level directory
    - /Task1 — Task1 top-level directory
      - task.yaml — Description of task and assignment implemented in markdown language
      - run — Instructions for running/checking student submission and giving feedback
      - …
    - /Task2 — Task2 top-level directory
      - task.yaml
      - run
    - …
  - /Course2 — Course2 top-level directory
    - /Task1
    - …

Good for backups and batch editing (if you know what you are doing)

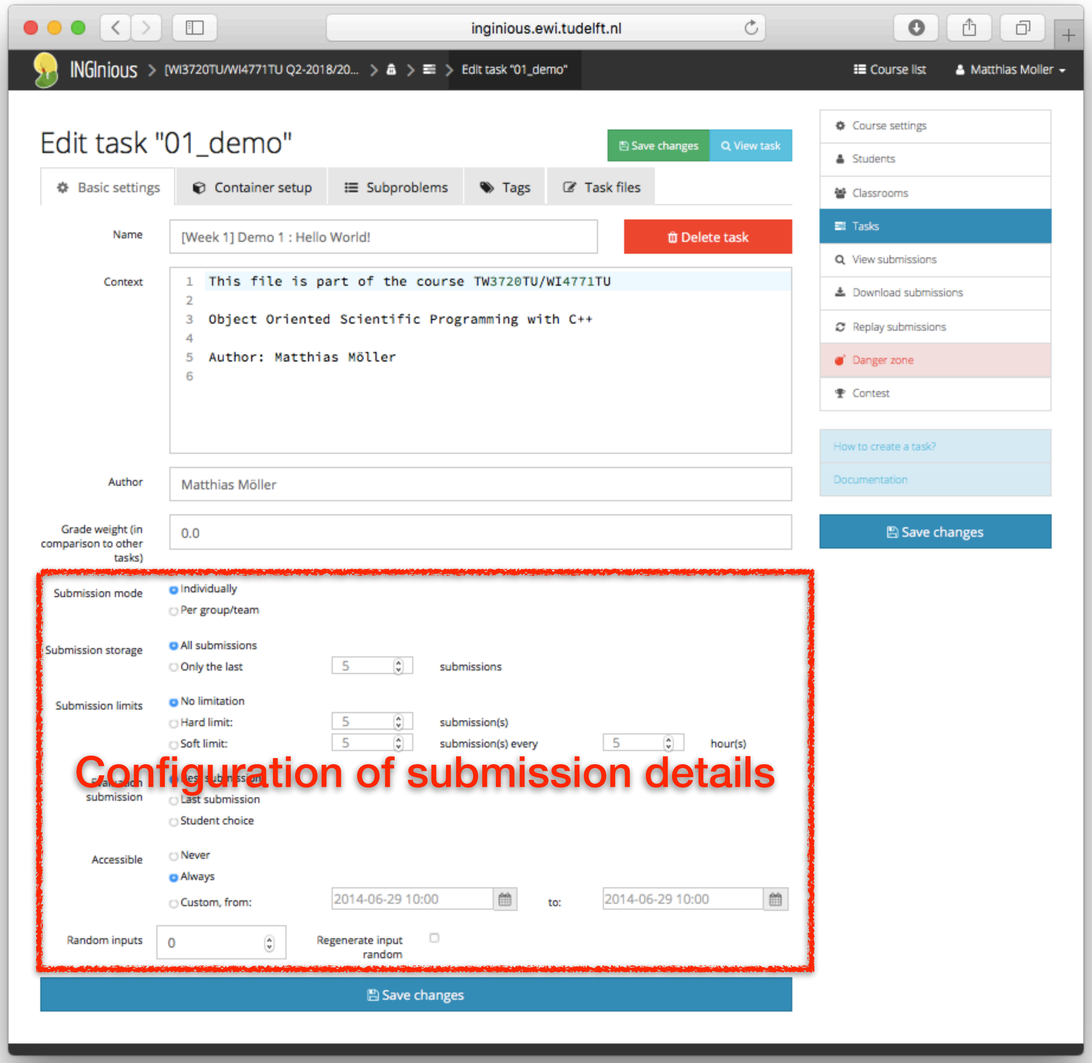
Direct SSH login to host needed

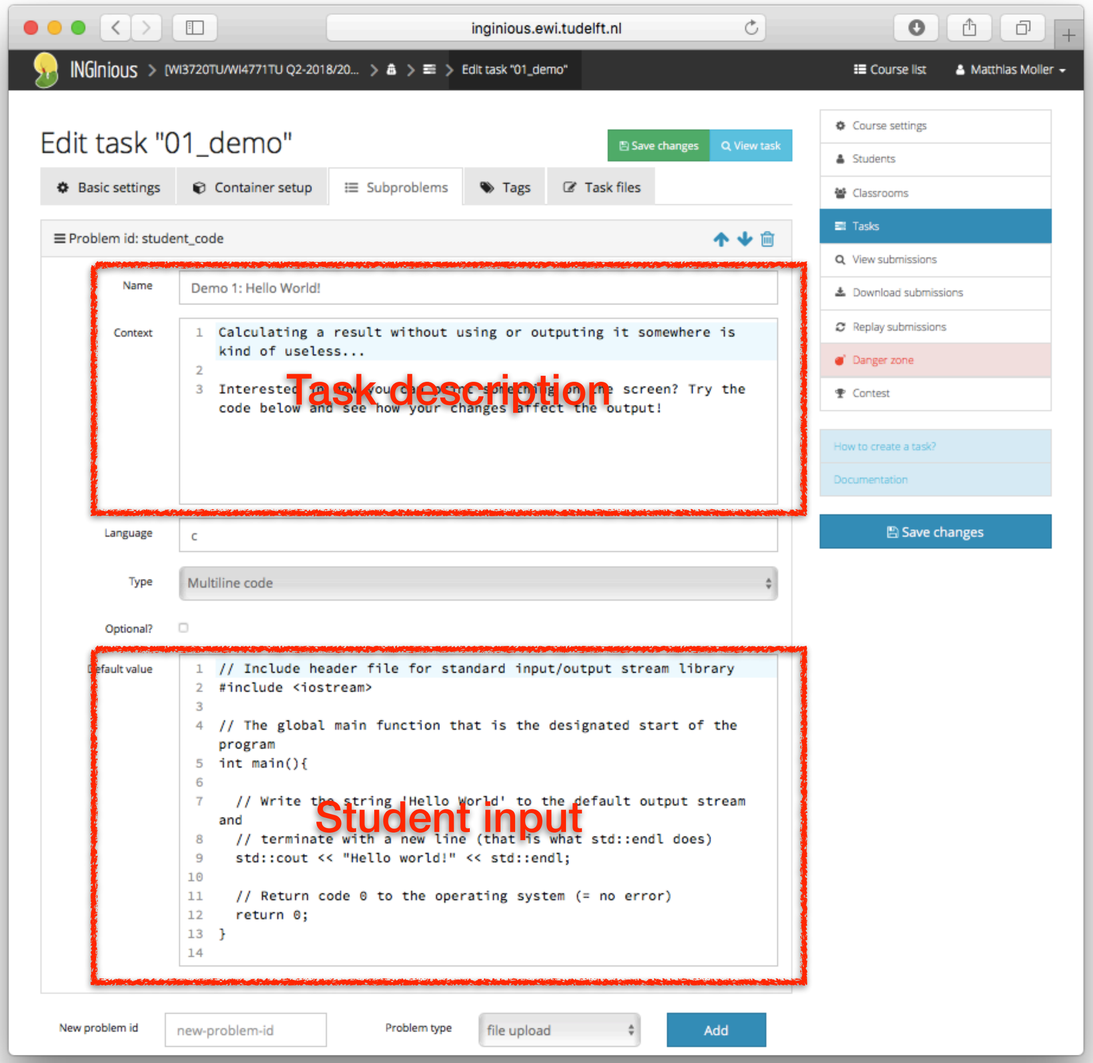
Web-based task editor



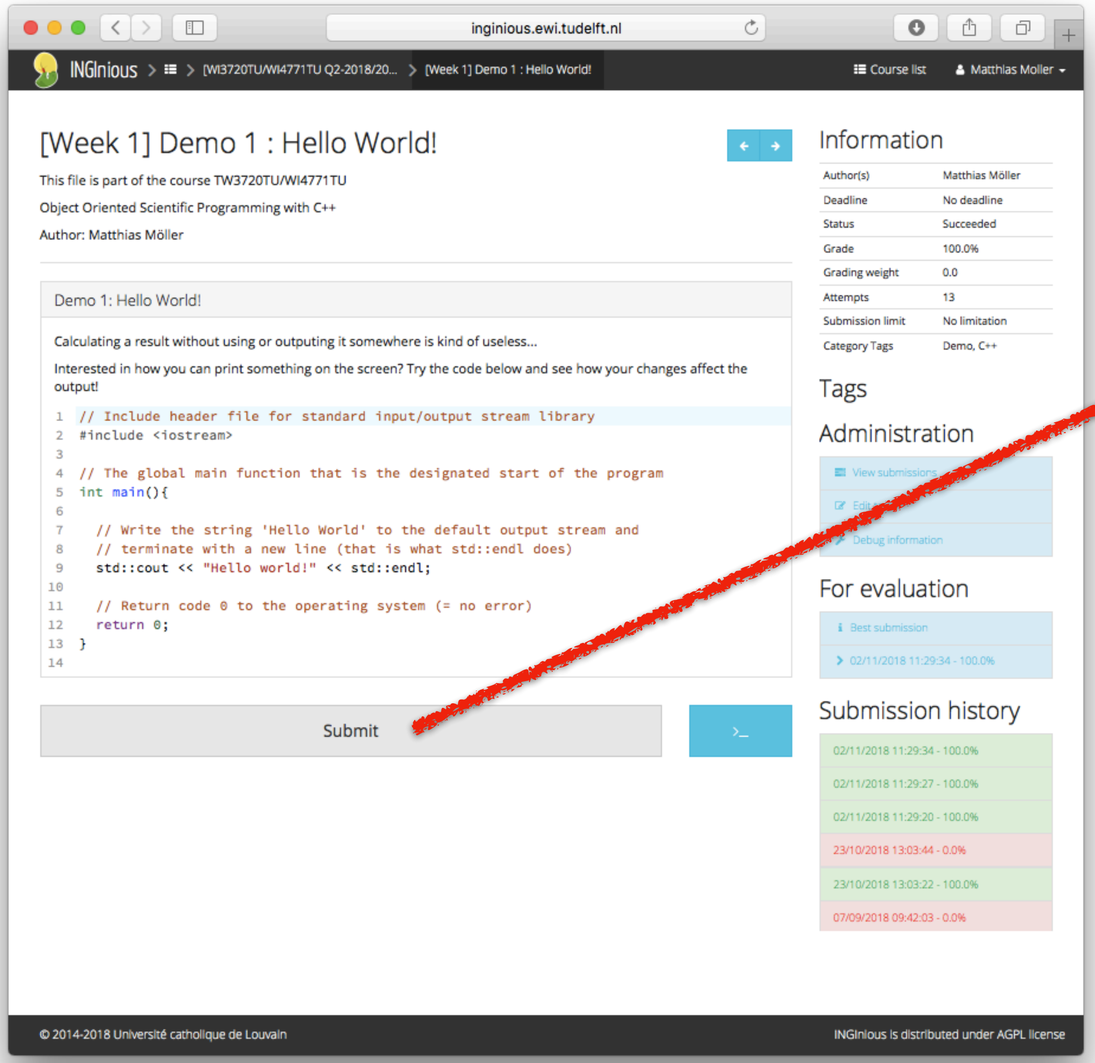


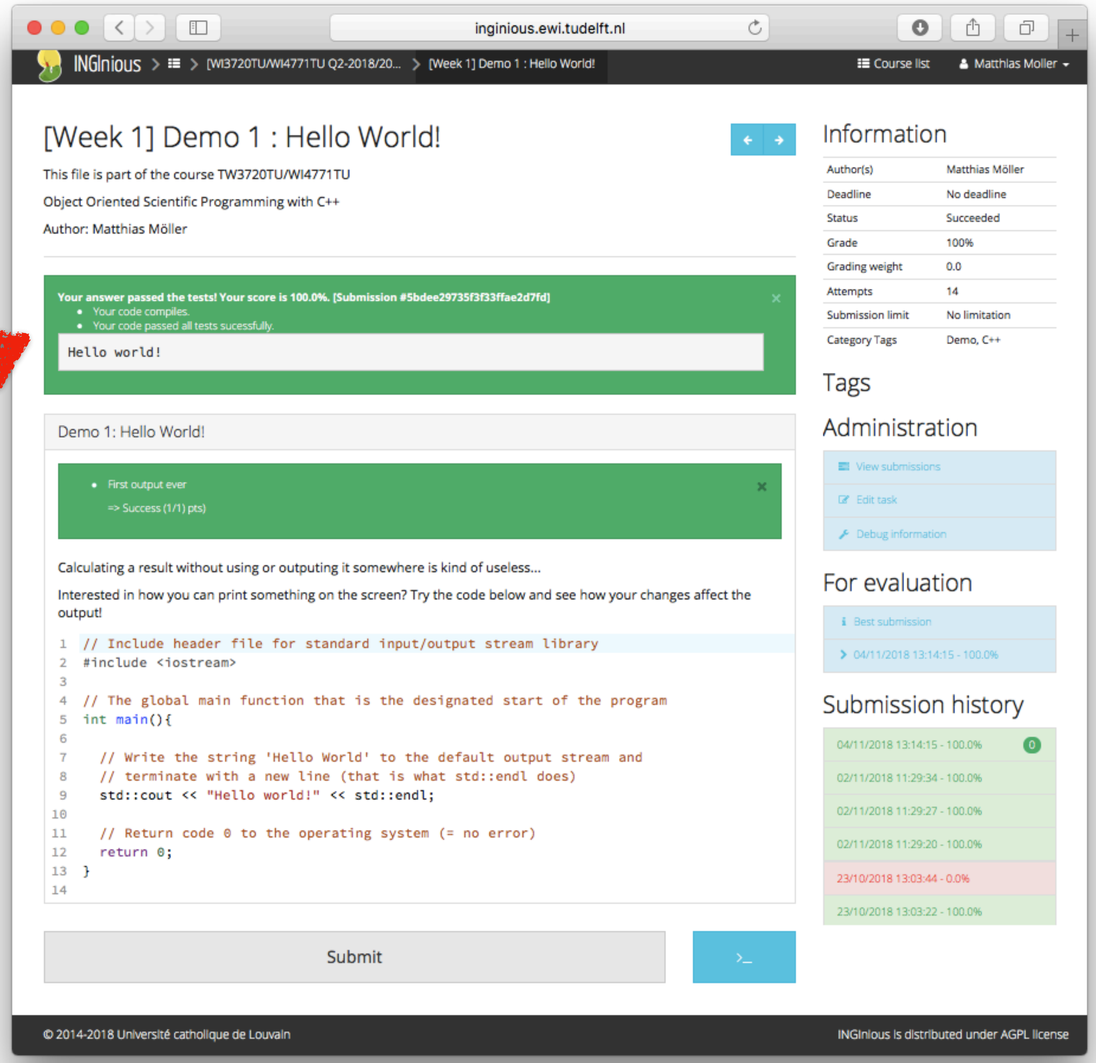
Web-based task editor



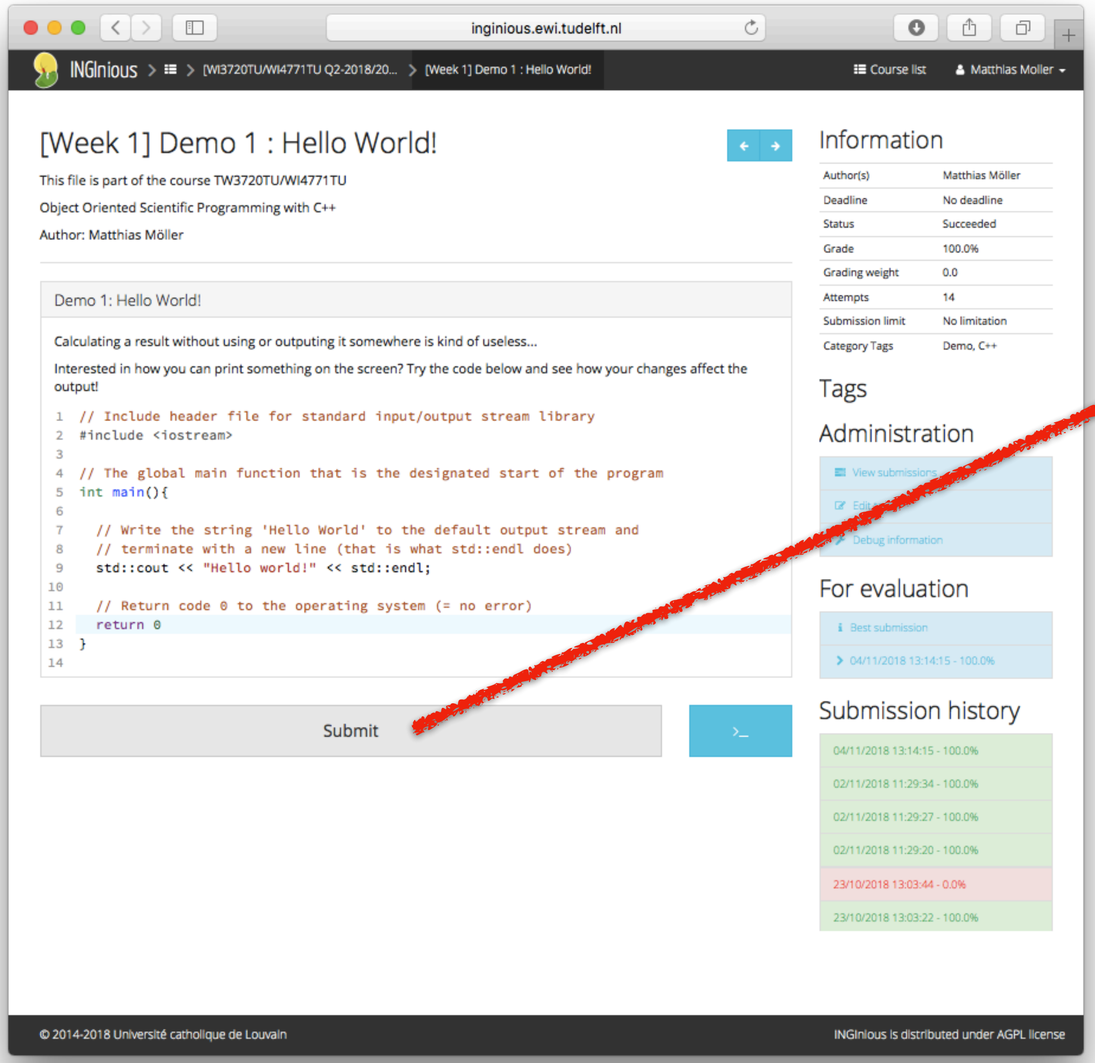


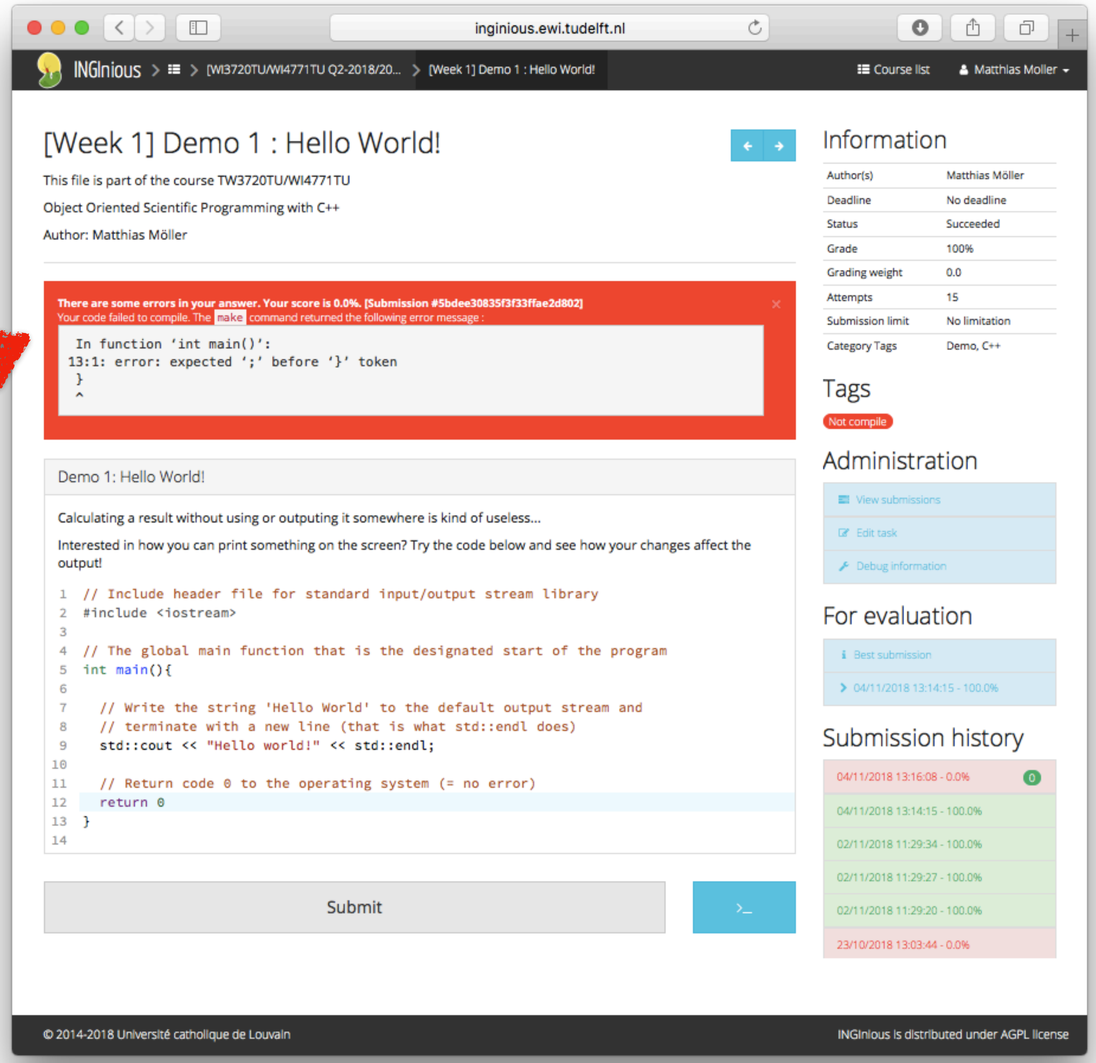
Task execution and feedback



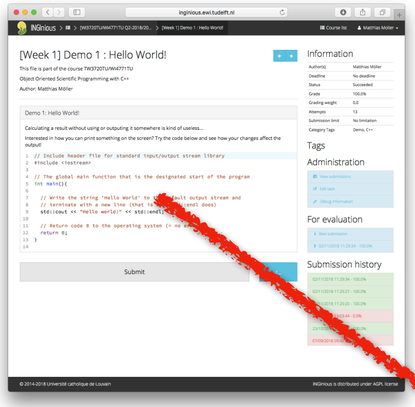


Task execution and feedback





Assessment workflow



Template.cxx

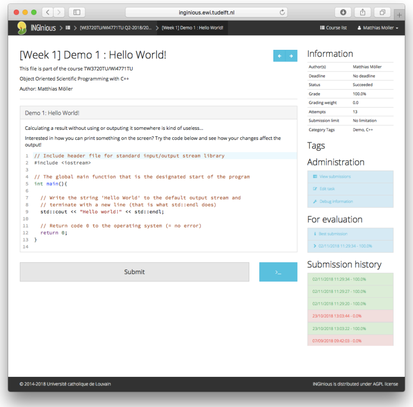
```
// Include header file for standard input/output stream library
#include <iostream>

// The global main function that is the designated start of the program
int main(){

  // Write the string 'Hello World' to the default output stream and
  // terminate with a new line (that is what std::endl does)
  std::cout << "Hello world!" << std::endl;

  // Return code 0 to the operating system (= no error)
  return 0;
}
```

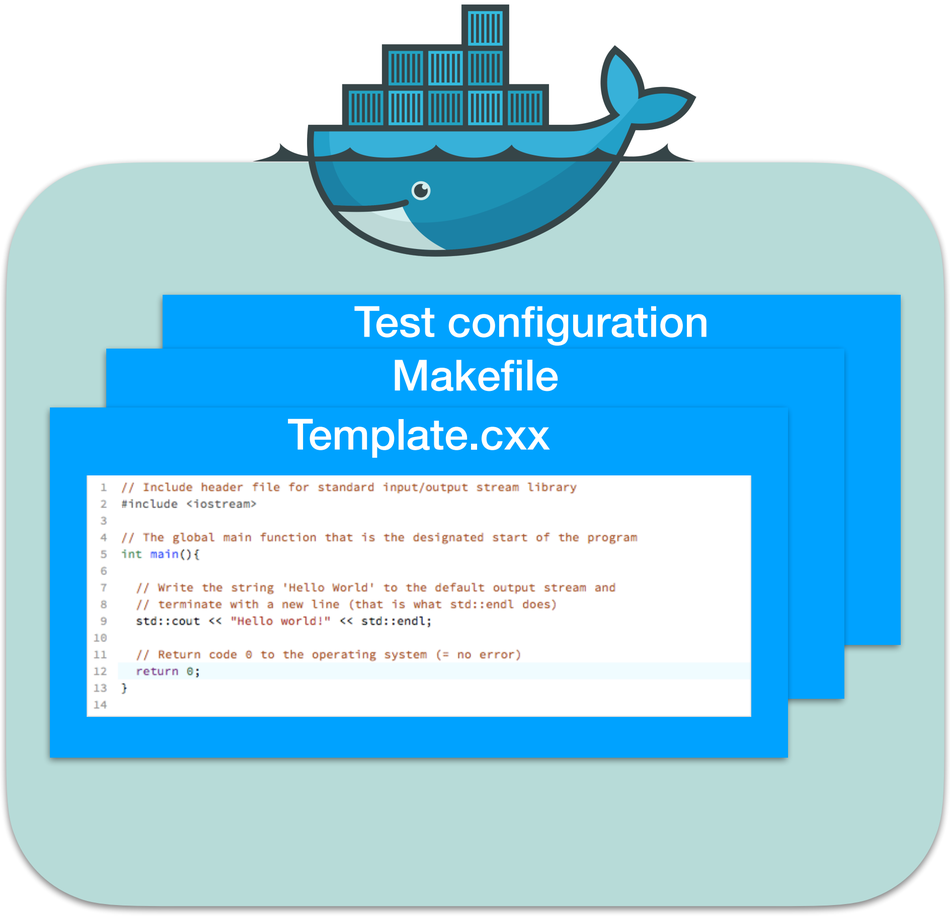
1. Smart copy-and-paste: Student input is parsed for banned ‘expressions’ and injected into template file.

File list | /run | /student/config/banned

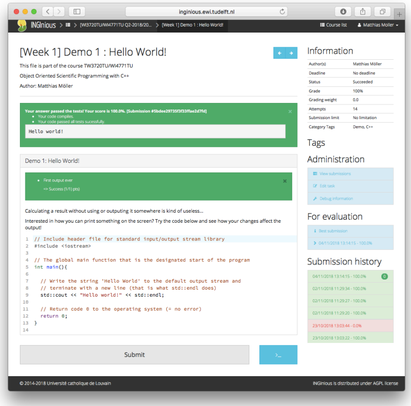
```
# Input here the banned include names:
bannedIncludeList = thread, mutex

# Input here the banned function names:
bannedFunctionList = open, fprintf

# Input here the banned words (Not recommended to use this):
bannedWordList = fstream

# Note: Don't change the layout too much, it is read with a custom function.
# Expected layout: <listName> = <element1>, <element2>, <element3>
# The amount of elements are not limited to 3, it is just an example.
```

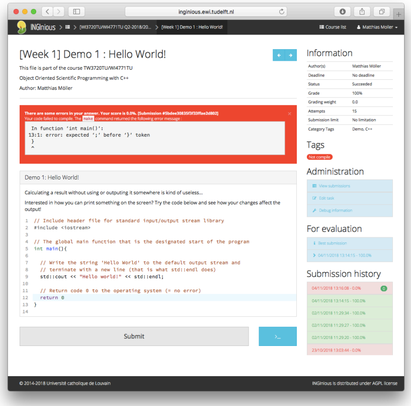
Assessment workflow



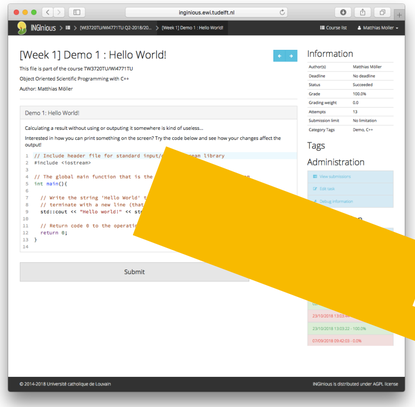


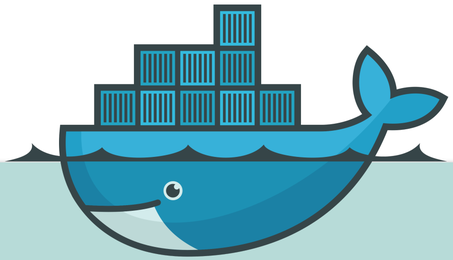
2. Docker-based execution: Input files are copied into new docker container that compiles submission based on Makefile. Compiler and execution output is captured, post-processed and presented to the student.





Assessment workflow



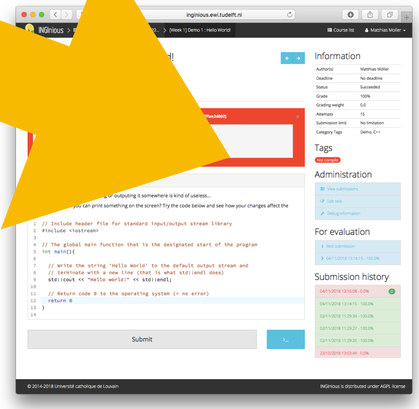


Test configuration

Makefile

```
1  // Include header file fo
2  #include <iostream>
3
4  // The global main function that is the des
5  int main(){
6
7    // Write the string 'Hello World' to the default output str
8    // terminate with a new line (that is what std::endl does)
9    std::cout << "Hello world!" << std::endl;
10
11   // Return code 0 to the operating system (= no error)
12   return 0;
13 }
14
```

Flexible Python-based workflow developed by BSc student Sybold Hijlkema



Python+Docker

RUN script:

- Shell scripts (from tutorial) for short answer and multiple-choice tasks
- Python script (by S. Hijlkema) for C++ (also for C) programming tasks

Docker container:

- Default containers with C/C++ (GCC 4.8.5), Python (2.7), …
- Customised containers with GCC 7.x and 8.x for C++14, 17 support

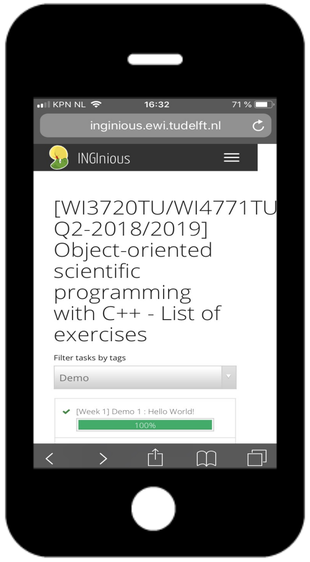
Example Dockerfile

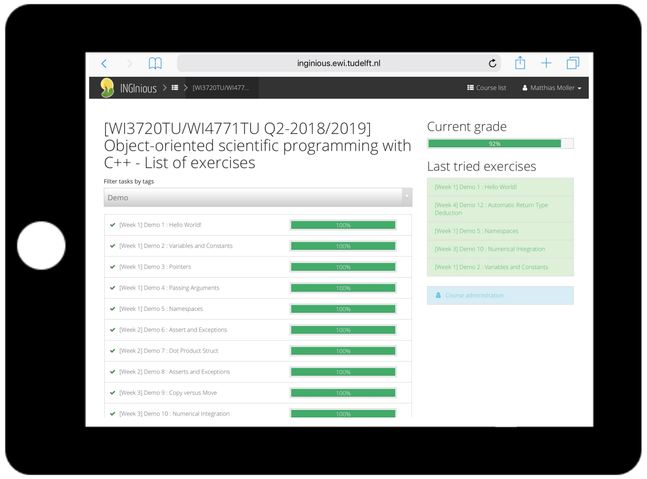
```
FROM    ingi/inginious-c-base
LABEL   org.inginious.grading.name="cpp-gcc7"
RUN     yum install -y centos-release-scl && \
        yum-config-manager --enable rhel-server-rhscl-7-rpms && \
        yum install -y yum install -y devtoolset-7-gcc devtoolset-7-
gcc-c++ devtoolset-7-gdb devtoolset-7-cpp devtoolset-7-make cmake
devtoolset-7-valgrind devtoolset-7-binutils libstdc++ clang clang-
analyzer clang-devel llvm automake check check-devel CUnit CUnit-
devel zlib-devel openssl-devel time jansson-devel graphviz graphviz-
devel cppcheck && \
        yum clean all

# Set PATH, LD_LIBRARY_PATH etcetera
ENV     PATH=/opt/rh/…
```

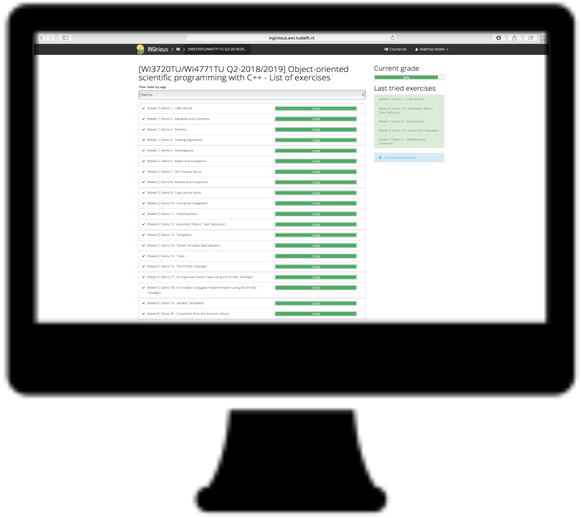
The sky is the limit

- Install commercial compilers, non-free test backends, simulators, …
- Integrate proprietary libraries/tools (in binary form) in Docker image
- Use customised Docker container as
  - abstraction to special hardware (NVIDIA-CUDA, Maxeler-FPGA, …)
  - communicator to external computer system (QuTech, Cluster, …)
  - drivers for software testing (fuzzing, regression, …)

It works on all platforms







Outlook

- First real-world test in Q2 2018/2019 with >130 participants (LObj. 1)
- Integration of web-based IDE and DevTools planned for 2019/20 (LObj. 2)
- If you are interested to give it a try for your course, let me know!

Acknowledgement: Sybold Hijlkema, Niels de Koeijer
Financial support by TU Delft Graduate School Kick-Start Funding