

Practical Lab

Cloud Systems Engineering

(cloud-lab)

Chair of Decentralized Systems Engineering

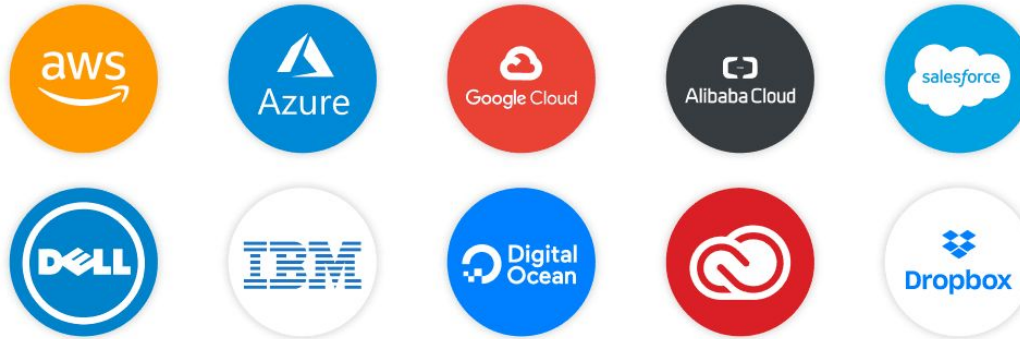
<https://dse.in.tum.de/>



Welcome to the cloud lab!

Cloud computing


- Cloud computing is powering the Internet
 - Large-scale computing resources
 - On-demand and cost effective
 - Geo-distributed data centers



- Cloud systems
 - Modern cloud systems handle millions of users and TBs of data
 - Cloud software systems employ large geo-distributed data centers
- How can we build cloud systems that ...
 - ... scale seamlessly?
 - ... are highly available?
 - ... are fault tolerant?
 - ... are easily configurable?
 - ... are easily maintained?
- Cloud systems aim to achieve all the above in a cost-effective manner

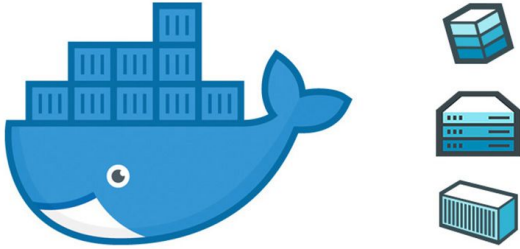
Our focus: Learning goals

- **Part I:** Cloud systems workflow
 - Container: How to build applications using containers?
 - Cluster orchestrators: How to deploy jobs?
- **Part II:** Distributed systems system architecture
 - Sharding / re-configuration of servers
 - Fault tolerance / replication
 - Consistent hashing
 - Consistency
 - Transactions / data management
 - Distributed locking / synchronization
 - Concurrency and high-performance architectures
 - Fault detection
 - Configuration management



Learn by building an end-to-end system!

Technologies



kubernetes



- A set of **four** programming tasks:
 - Each related to a different aspect of distributed systems
 - Built on top of each other, like a stack
- For each task, we will provide
 - Necessary background via a lecture
 - Q&As: after lecture and online via Slack
- Submitted tasks will be evaluated by
 - Automated grading system (only the last commit before the deadline counts)
 - Instructors

Layered architecture

#4: Distributed TXs: w/ and w/o replication

#3: Replicated distributed KVS

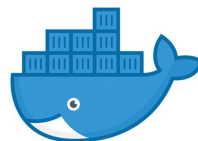
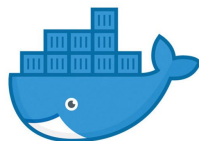
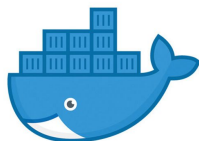
#2: Distributed KVS

#1: Single-node
KVS

#1: Single-node
KVS

...

#1: Single-node
KVS



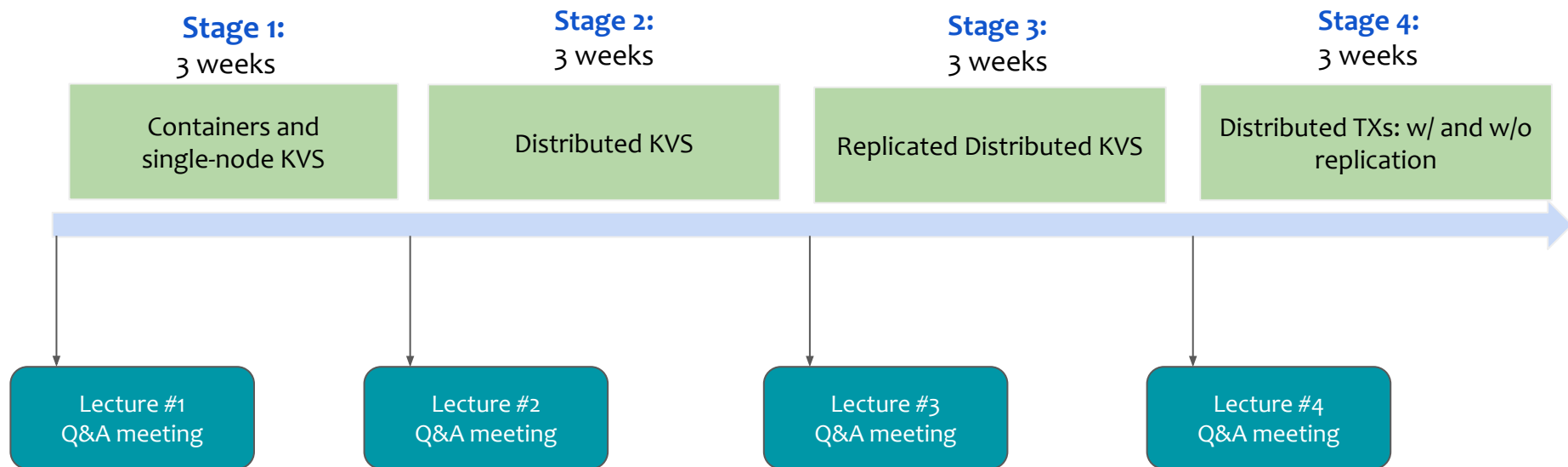
kubernetes



APACHE
ZooKeeper™

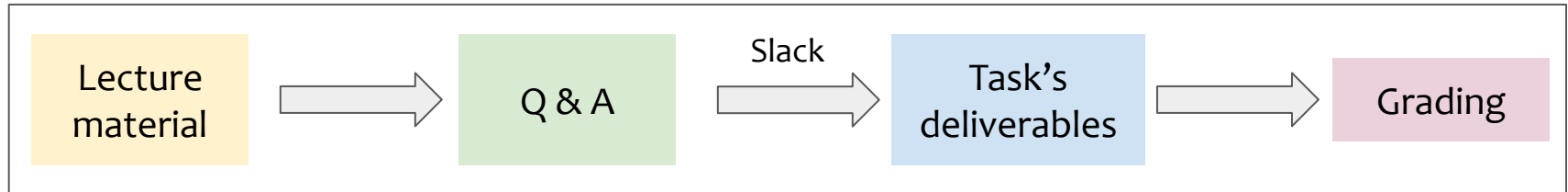
Lecture	Category	Details	Grade
#1	Single-node KVS setup	Build and deploy a single node KVS	25%
#2	Distributed KVS	Shard the keys across multiple nodes: fault detection and server reconfiguration	25%
#3	Replicated distributed KVS	Replicate the KVS instances across these nodes for fault tolerance	25%
#4	Distributed Transactions	Support distributed transactions across keys and nodes: w/ and w/o replication	25%

Timeline



- Four lectures
 - One for each task – [Recorded videos](#)
 - Necessary material and deliverables will be explained
 - Q&A – [Online on Thursdays via Zoom](#)
- Online help
 - Slack channel will be monitored by the instructors/tutors
- Format:

Life of a task



The assignments

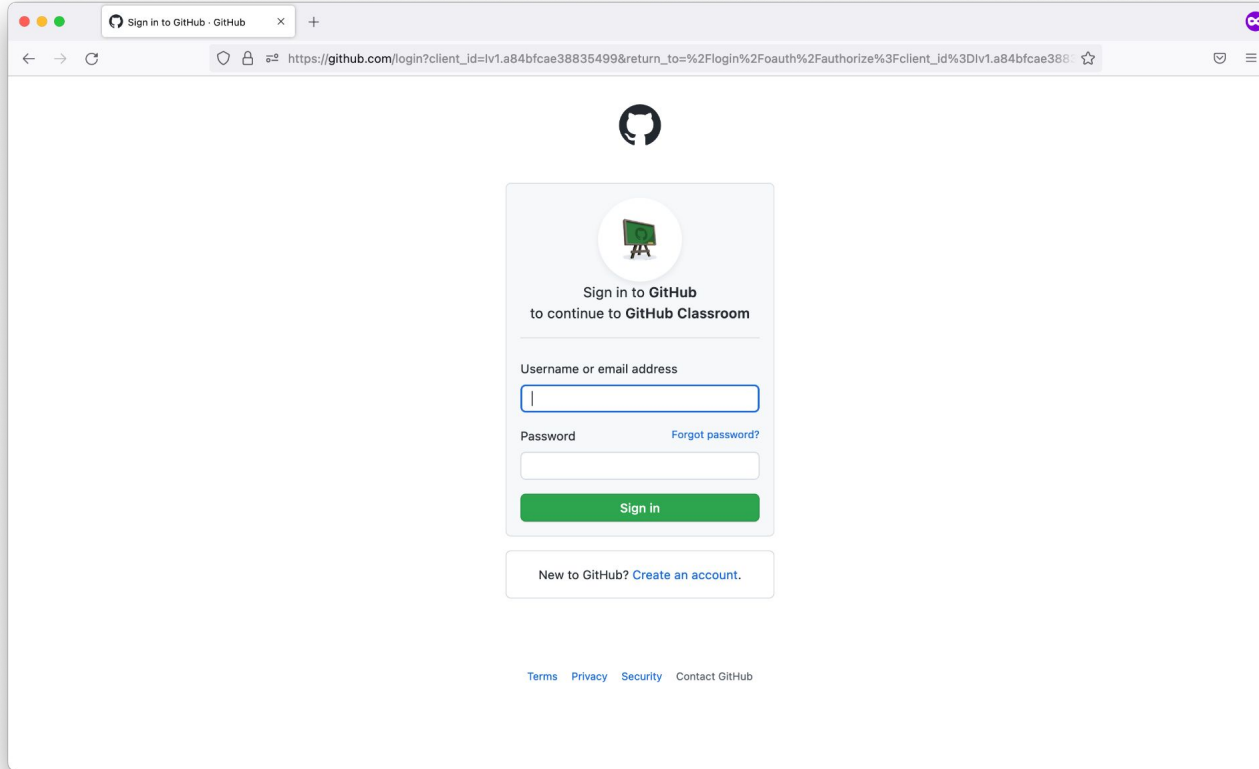
- Code should be preferably written in C++
 - Other system programming languages like Rust may be used as well but there is no support by us
- We provide a detailed task description in the [README.md](#) of the task
- Code is hosted on GitHub and will be auto-graded via GitHub classroom
 - You will need to create a GitHub account
 - We will send you invitation links for each assignment via Slack
- Use Linux for the tasks

Our CI runners

- Fancy kubernetes cluster
- Three servers, each equipped with
 - 2x Intel Xeon 5215
(10 cores / 20 threads each)
 - 128 GiB RAM
 - 10G NICs




Getting the assignment code




The screenshot shows a web browser window with the GitHub login page. The address bar displays the URL: `https://github.com/login?client_id=lv1.a84bfcae38835499&return_to=%2Flogin%2Foauth%2Fauthorize%3Fclient_id%3Dlv1.a84bfcae38835499&return_to=%2Flogin%2Foauth%2Fauthorize%3Fclient_id%3Dlv1.a84bfcae38835499`. The page features the GitHub logo at the top center. Below it, a circular profile picture placeholder with a green chalkboard icon is shown. The text "Sign in to GitHub to continue to GitHub Classroom" is centered. The login form includes a "Username or email address" field, a "Password" field, and a "Forgot password?" link. A green "Sign in" button is positioned below the password field. At the bottom of the form, a link for "New to GitHub? Create an account." is provided. The footer contains links for "Terms", "Privacy", "Security", and "Contact GitHub".

Sign in to GitHub · GitHub

← → ↻ 🔒 📄 https://github.com/login?client_id=lv1.a84bfcae38835499&return_to=%2Flogin%2Foauth%2Fauthorize%3Fclient_id%3Dlv1.a84bfcae38835499&return_to=%2Flogin%2Foauth%2Fauthorize%3Fclient_id%3Dlv1.a84bfcae38835499





Sign in to GitHub
to continue to GitHub Classroom

Username or email address

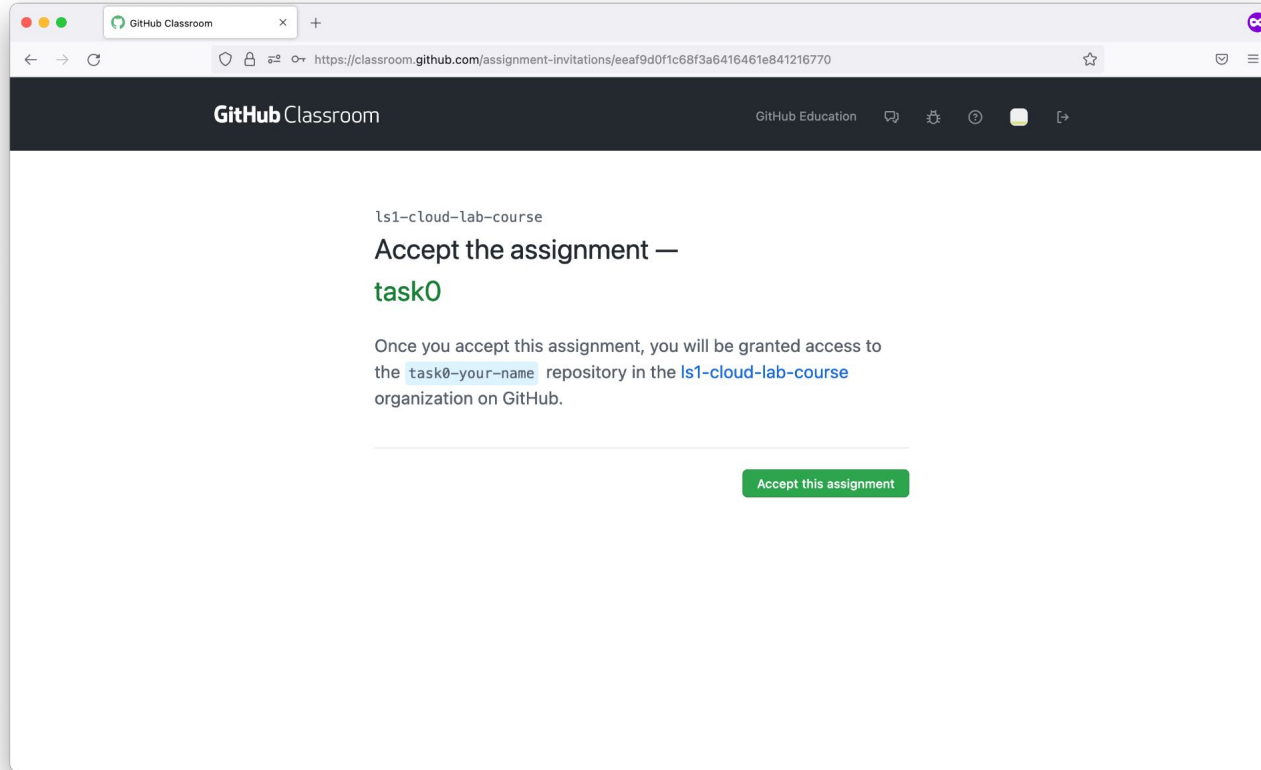
Password [Forgot password?](#)

[Sign in](#)

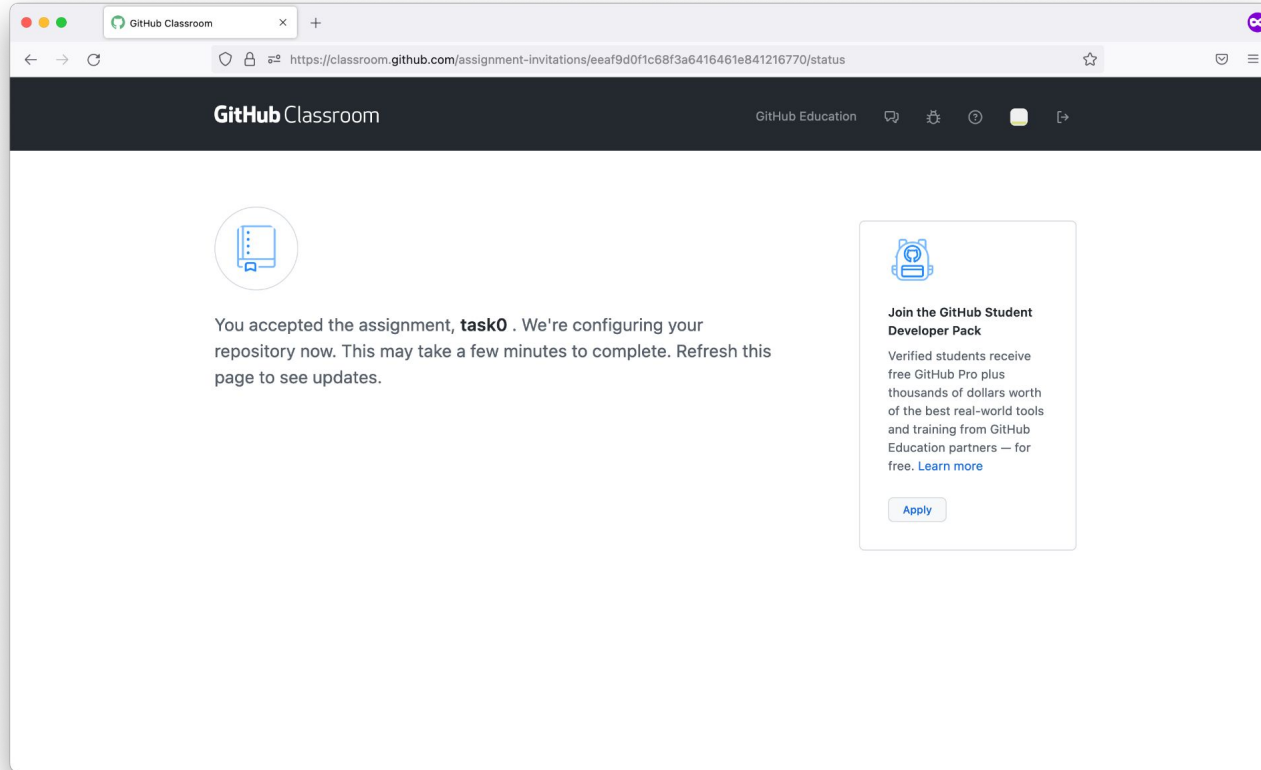
New to GitHub? [Create an account.](#)

[Terms](#) [Privacy](#) [Security](#) [Contact GitHub](#)

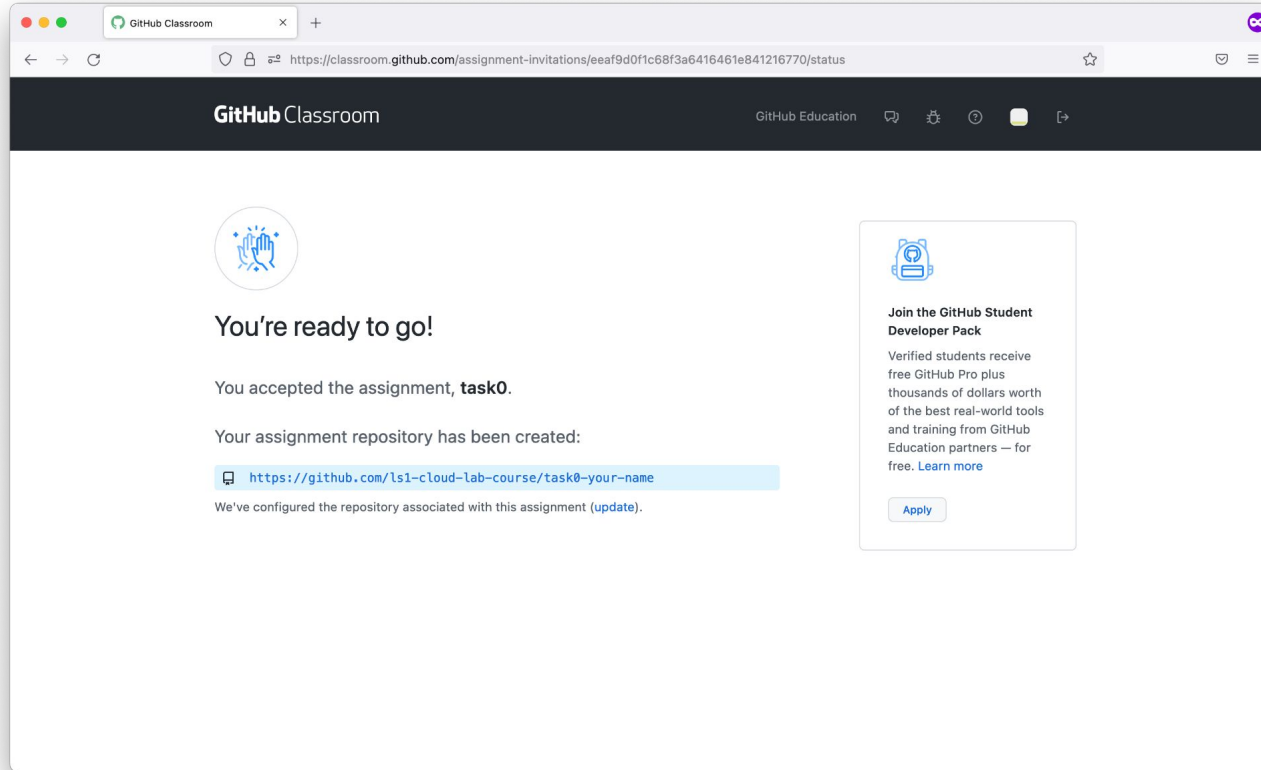
Getting the assignment code



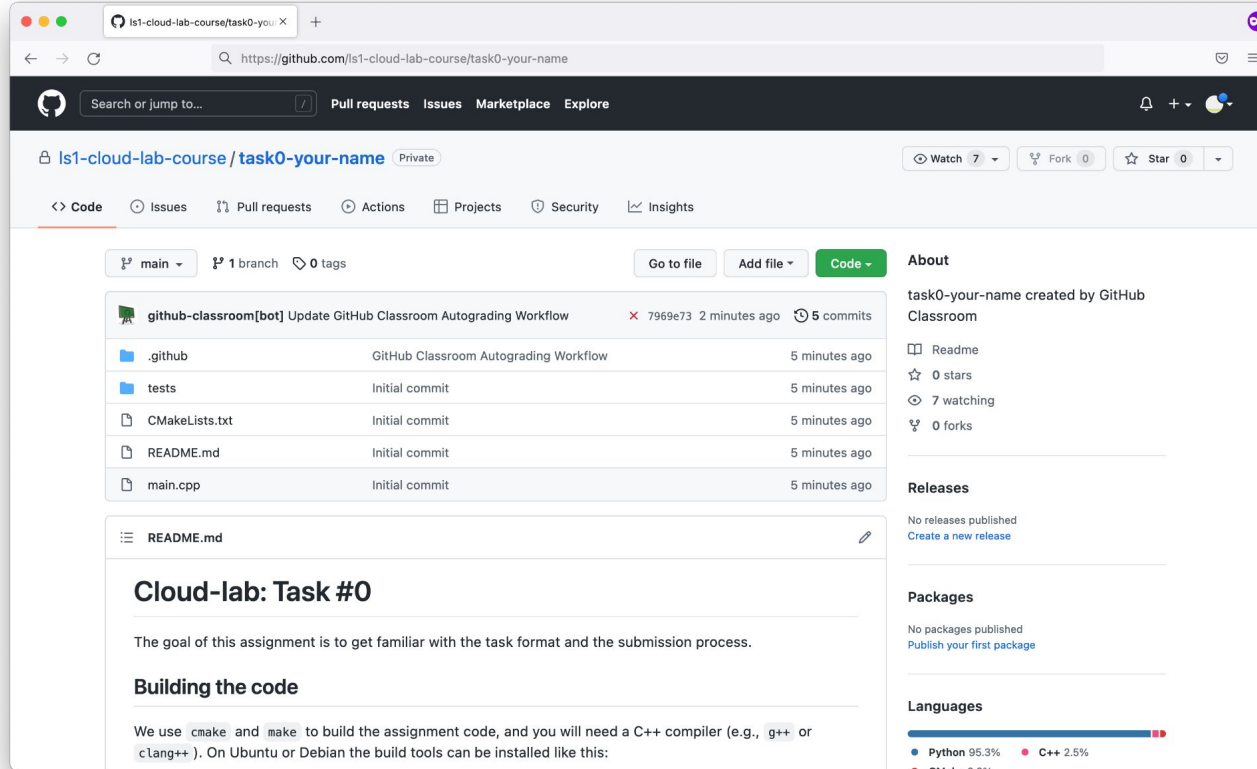
Getting the assignment code



Getting the assignment code

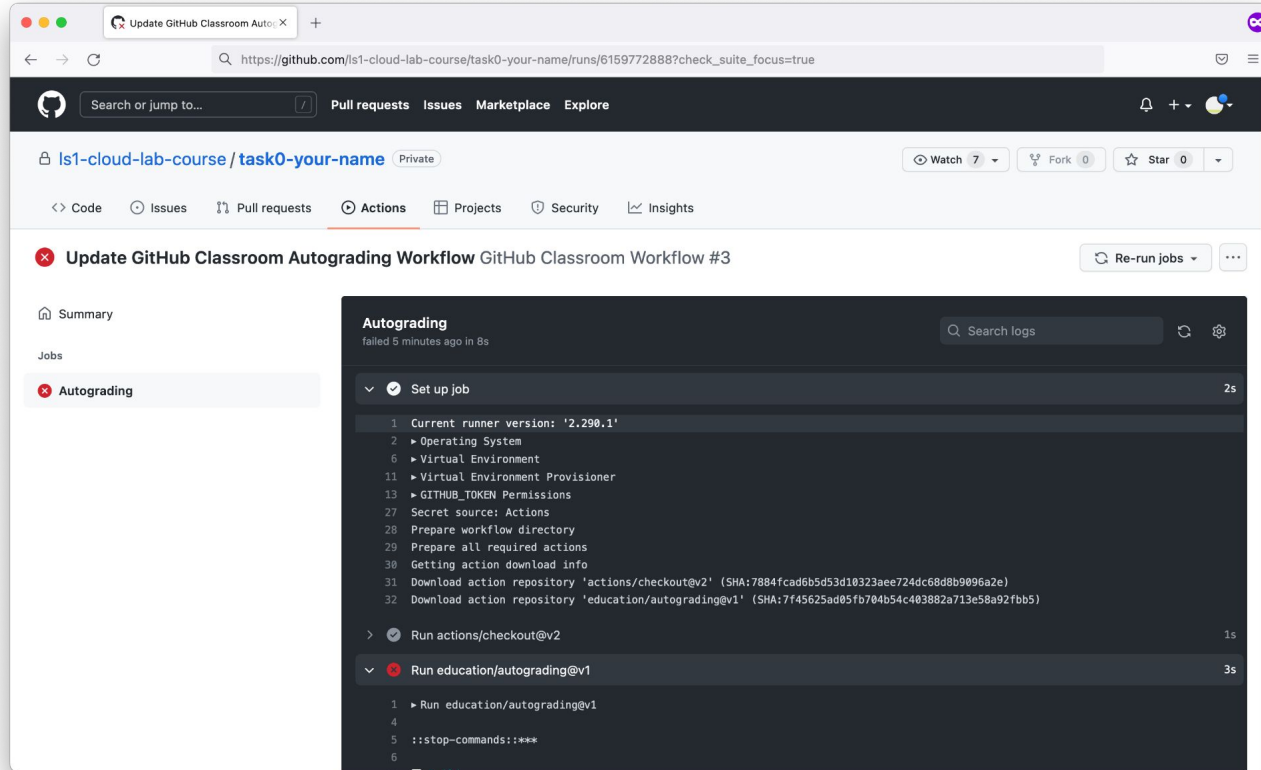


Getting the assignment code



The screenshot shows a web browser displaying a GitHub repository page. The address bar shows the URL `https://github.com/ls1-cloud-lab-course/task0-your-name`. The repository name is `ls1-cloud-lab-course / task0-your-name` and it is marked as `Private`. The page includes navigation tabs for `Code`, `Issues`, `Pull requests`, `Actions`, `Projects`, `Security`, and `Insights`. Below the repository name, there are buttons for `Watch` (7), `Fork` (0), and `Star` (0). The `Code` tab is selected, showing a list of files and a commit history. The commit history shows a single commit by `github-classroom[bot]` titled "Update GitHub Classroom Autograding Workflow" with a SHA of `7969e73` and 5 commits. The file list includes `.github`, `tests`, `CMakeLists.txt`, `README.md`, and `main.cpp`, all with an initial commit 5 minutes ago. The `README.md` file is selected, showing the title **Cloud-lab: Task #0** and the text: "The goal of this assignment is to get familiar with the task format and the submission process." Below this, the section **Building the code** states: "We use `cmake` and `make` to build the assignment code, and you will need a C++ compiler (e.g., `g++` or `clang++`). On Ubuntu or Debian the build tools can be installed like this:" The right sidebar contains sections for **About** (task0-your-name created by GitHub Classroom), **Releases** (No releases published), **Packages** (No packages published), and **Languages** (Python 95.3%, C++ 2.5%, CMake 2.2%).

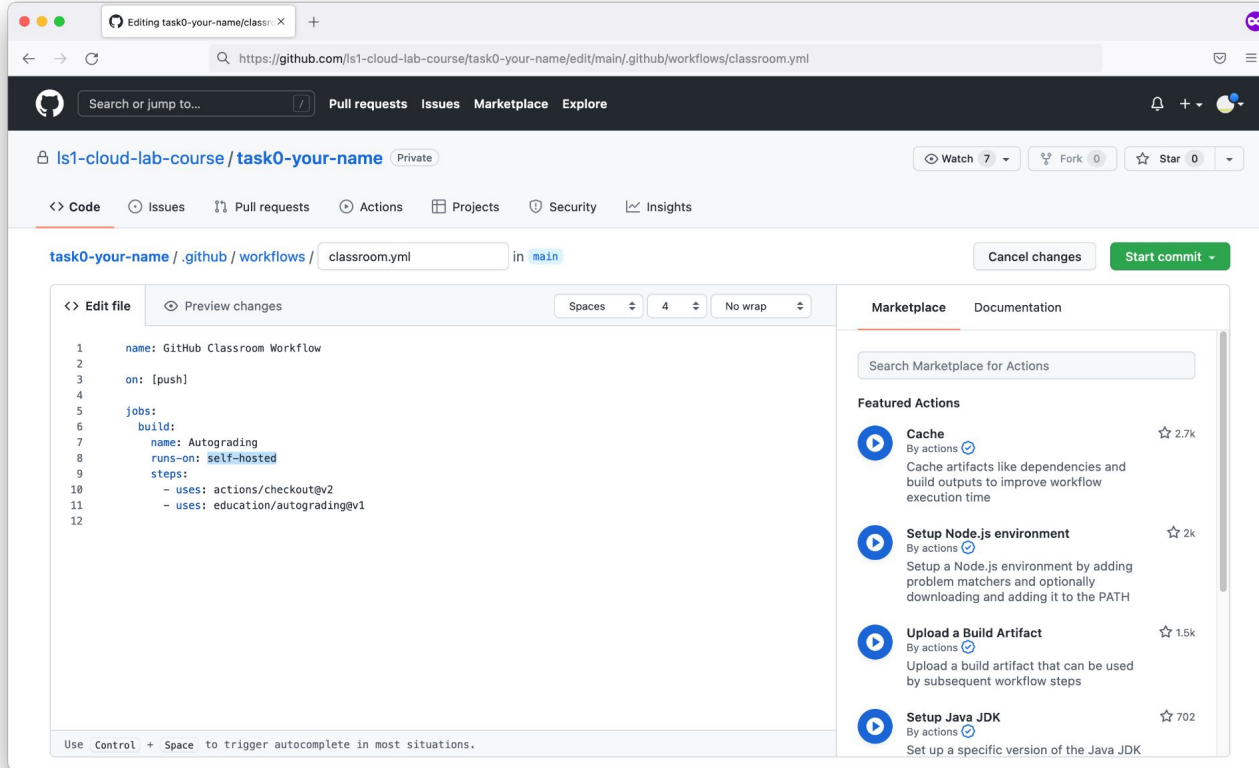
Submitting your code



The screenshot shows a web browser window displaying a GitHub Actions workflow run. The URL in the address bar is `https://github.com/lis1-cloud-lab-course/task0-your-name/runs/6159772888?check_suite_focus=true`. The page title is "Update GitHub Classroom Autograding Workflow GitHub Classroom Workflow #3". The workflow is named "Update GitHub Classroom Autograding Workflow" and is in a "Failed" state, indicated by a red 'x' icon. The "Autograding" job is highlighted in the left sidebar, showing it failed 5 minutes ago in 8s. The main content area displays the job's log, which includes the following steps:

- Set up job** (2s):
 - 1 Current runner version: '2.290.1'
 - 2 Operating System
 - 6 Virtual Environment
 - 11 Virtual Environment Provisioner
 - 13 GITHUB_TOKEN Permissions
 - 27 Secret source: Actions
 - 28 Prepare workflow directory
 - 29 Prepare all required actions
 - 30 Getting action download info
 - 31 Download action repository 'actions/checkout@v2' (SHA:7884fcad6b5d3d10323aee724dc68d8b9096a2e)
 - 32 Download action repository 'education/autograding@v1' (SHA:7f45625ad05fb704b54c403882a713e58a92fbb5)
- Run actions/checkout@v2** (1s)
- Run education/autograding@v1** (3s):
 - 1 Run education/autograding@v1
 - 4
 - 5 ::stop-commands::***
 - 6

Submitting your code



The screenshot shows a web browser window displaying a GitHub repository page for editing a workflow file. The browser's address bar shows the URL: `https://github.com/lis1-cloud-lab-course/task0-your-name/edit/main/.github/workflows/classroom.yml`. The GitHub interface includes a search bar, navigation links for Pull requests, Issues, Marketplace, and Explore, and repository statistics (7 watches, 0 forks, 0 stars). The repository name is `lis1-cloud-lab-course / task0-your-name` and it is marked as Private. The file path is `task0-your-name / .github / workflows / classroom.yml` in the `main` branch. The workflow file content is as follows:

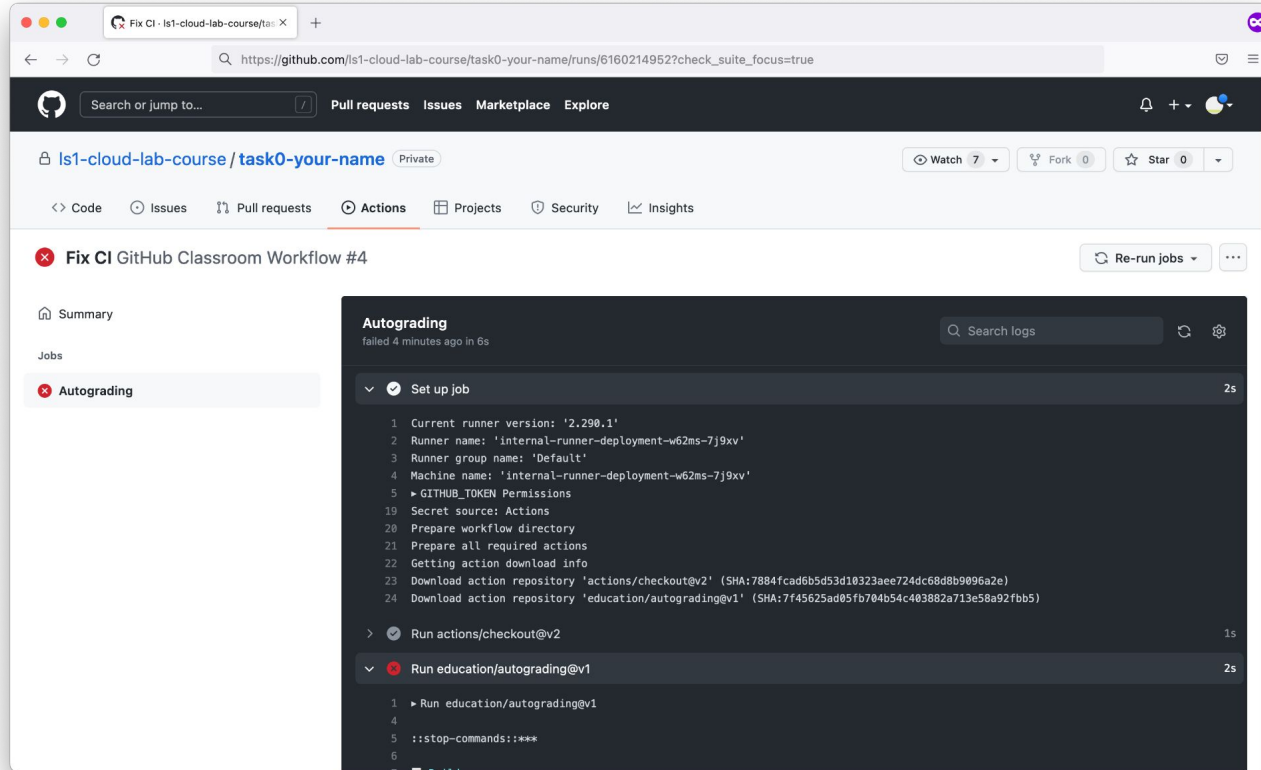
```
1 name: GitHub Classroom Workflow
2
3 on: [push]
4
5 jobs:
6   build:
7     name: Autograding
8     runs-on: self-hosted
9     steps:
10       - uses: actions/checkout@v2
11       - uses: education/autograding@v1
12
```

On the right side, the 'Marketplace' tab is active, showing a search bar and a list of 'Featured Actions':

- Cache** (2.7k stars): Cache artifacts like dependencies and build outputs to improve workflow execution time.
- Setup Node.js environment** (2k stars): Setup a Node.js environment by adding problem matchers and optionally downloading and adding it to the PATH.
- Upload a Build Artifact** (1.5k stars): Upload a build artifact that can be used by subsequent workflow steps.
- Setup Java JDK** (702 stars): Set up a specific version of the Java JDK.

At the bottom of the editor, a hint states: 'Use Control + Space to trigger autocomplete in most situations.'

Submitting your code



The screenshot shows a GitHub repository page for 'ls1-cloud-lab-course / task0-your-name'. The 'Actions' tab is selected, displaying a workflow named 'Fix CI GitHub Classroom Workflow #4'. The workflow has a red status icon and a 'Re-run jobs' button. On the left sidebar, the 'Autograding' job is highlighted with a red status icon. The main panel shows the logs for the 'Autograding' job, which failed 4 minutes ago. The logs are divided into three sections: 'Set up job' (2s), 'Run actions/checkout@v2' (1s), and 'Run education/autograding@v1' (2s). The 'Run education/autograding@v1' section shows a command to stop the process.

Fix CI GitHub Classroom Workflow #4

Summary

Jobs

- Autograding

Autograding
failed 4 minutes ago in 6s

Search logs

Set up job 2s

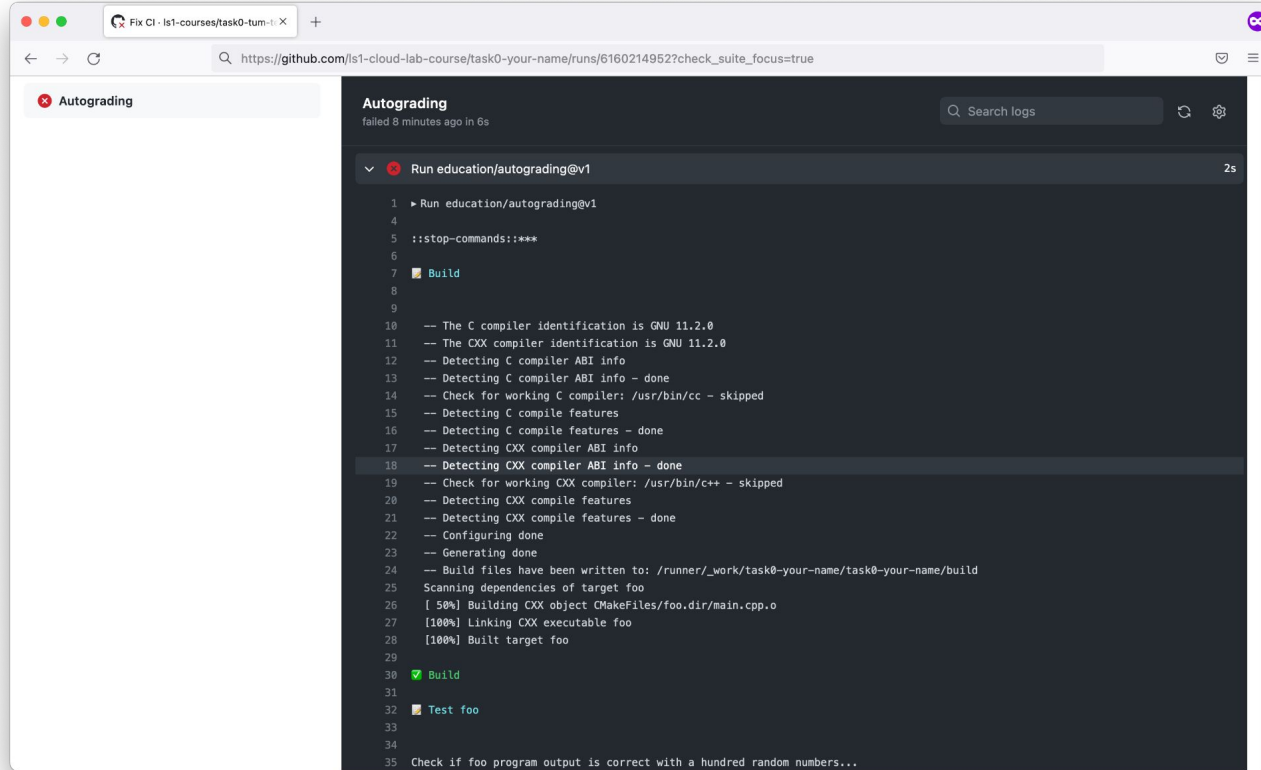
```
1 Current runner version: '2.290.1'
2 Runner name: 'internal-runner-deployment-w62ms-7j9xv'
3 Runner group name: 'Default'
4 Machine name: 'internal-runner-deployment-w62ms-7j9xv'
5 ▶ GITHUB_TOKEN Permissions
19 Secret source: Actions
20 Prepare workflow directory
21 Prepare all required actions
22 Getting action download info
23 Download action repository 'actions/checkout@v2' (SHA:7884fcad6b5d3d18323aee724dc68d8b9096a2e)
24 Download action repository 'education/autograding@v1' (SHA:7f45625ad05fb704b54c483882a713e58a92fbb5)
```

Run actions/checkout@v2 1s

Run education/autograding@v1 2s

```
1 ▶ Run education/autograding@v1
4
5 ::stop-commands::***
6
```

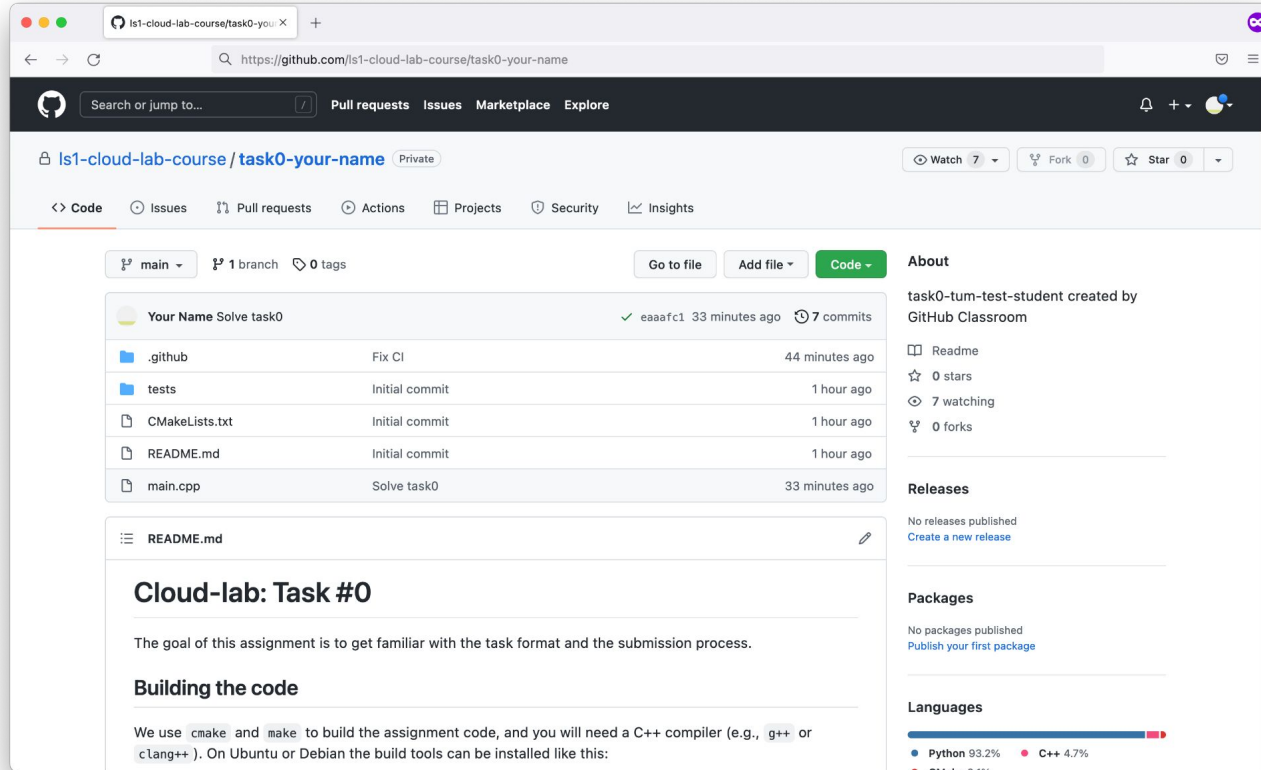
Submitting your code



The screenshot shows a web browser window displaying a GitHub Actions workflow run. The browser's address bar shows the URL: `https://github.com/lis1-cloud-lab-course/task0-your-name/runs/6160214952?check_suite_focus=true`. The page title is "Autograding" with a red status icon and the text "failed 8 minutes ago in 6s". On the left sidebar, there is a tab labeled "Autograding" with a red status icon. The main content area shows the workflow steps for "Run education/autograding@v1". The steps are listed with line numbers 1 through 35. The "Build" step (lines 7-30) is highlighted with a green checkmark, indicating it passed. The "Test foo" step (lines 32-35) is highlighted with a red status icon, indicating it failed. The output of the "Build" step shows the compilation of a C++ program named "foo".

```
1  ▶ Run education/autograding@v1
2
3  ::stop-commands::***
4
5  📦 Build
6
7  -- The C compiler identification is GNU 11.2.0
8  -- The CXX compiler identification is GNU 11.2.0
9  -- Detecting C compiler ABI info
10 -- Detecting C compiler ABI info - done
11 -- Check for working C compiler: /usr/bin/cc - skipped
12 -- Detecting C compile features
13 -- Detecting C compile features - done
14 -- Detecting CXX compiler ABI info
15 -- Detecting CXX compiler ABI info - done
16 -- Check for working CXX compiler: /usr/bin/c++ - skipped
17 -- Detecting CXX compile features
18 -- Detecting CXX compile features - done
19 -- Configuring done
20 -- Generating done
21 -- Build files have been written to: /runner/_work/task0-your-name/task0-your-name/build
22 Scanning dependencies of target foo
23 [ 50%] Building CXX object CMakeFiles/foo.dir/main.cpp.o
24 [100%] Linking CXX executable foo
25 [100%] Built target foo
26
27 🟢 Build
28
29 📦 Test foo
30
31 Check if foo program output is correct with a hundred random numbers...
```

Submitting your code



The screenshot shows a GitHub repository page for 'ls1-cloud-lab-course/task0-your-name'. The repository is private and was created by 'task0-tum-test-student created by GitHub Classroom'. It has 7 commits, 7 watchers, and 0 forks. The repository contains a file tree with the following files and their commit times:

File	Commit Time
.github	Fix CI 44 minutes ago
tests	Initial commit 1 hour ago
CMakeLists.txt	Initial commit 1 hour ago
README.md	Initial commit 1 hour ago
main.cpp	Solve task0 33 minutes ago

The README.md file is selected and displays the following content:

Cloud-lab: Task #0

The goal of this assignment is to get familiar with the task format and the submission process.

Building the code

We use `cmake` and `make` to build the assignment code, and you will need a C++ compiler (e.g., `g++` or `clang++`). On Ubuntu or Debian the build tools can be installed like this:

The right sidebar shows the repository's statistics and links to create a new release or package.

Task #1:

Single-node KVS

Learning goals



In this task you will learn about:

- Client-server architecture(s)
- Single node key-value stores (KVSs), e.g., RocksDB

Background

Containers

Motivation

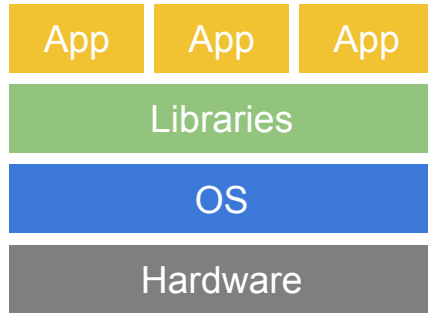
Share resources (IaaS, PaaS, ...) in a flexible and cost-effective way

→ split applications into microservices

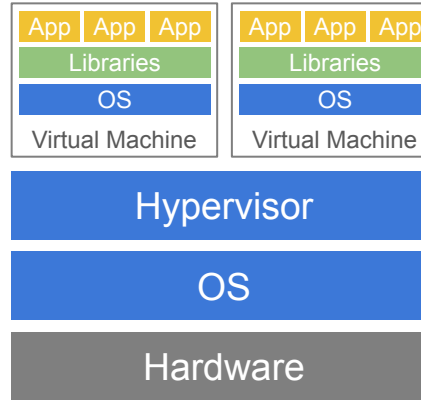
Advantages:

- Greater hardware resource-utilization
- Simply maintainable
- Scalable

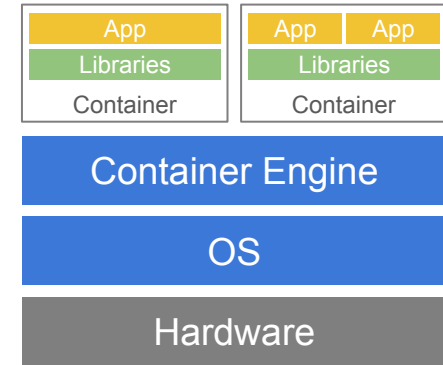
Containers



Traditional system



Virtual machines

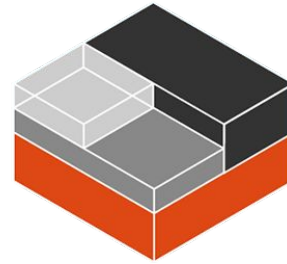
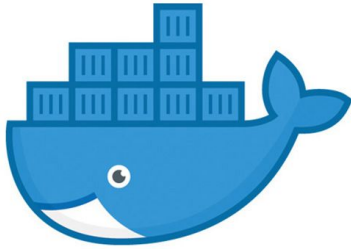


Containers

Unlike virtual machines ...

- containers often contain only one application
- containers do not include an operating system (OS)
- containers limit access to resources through host OS mechanisms

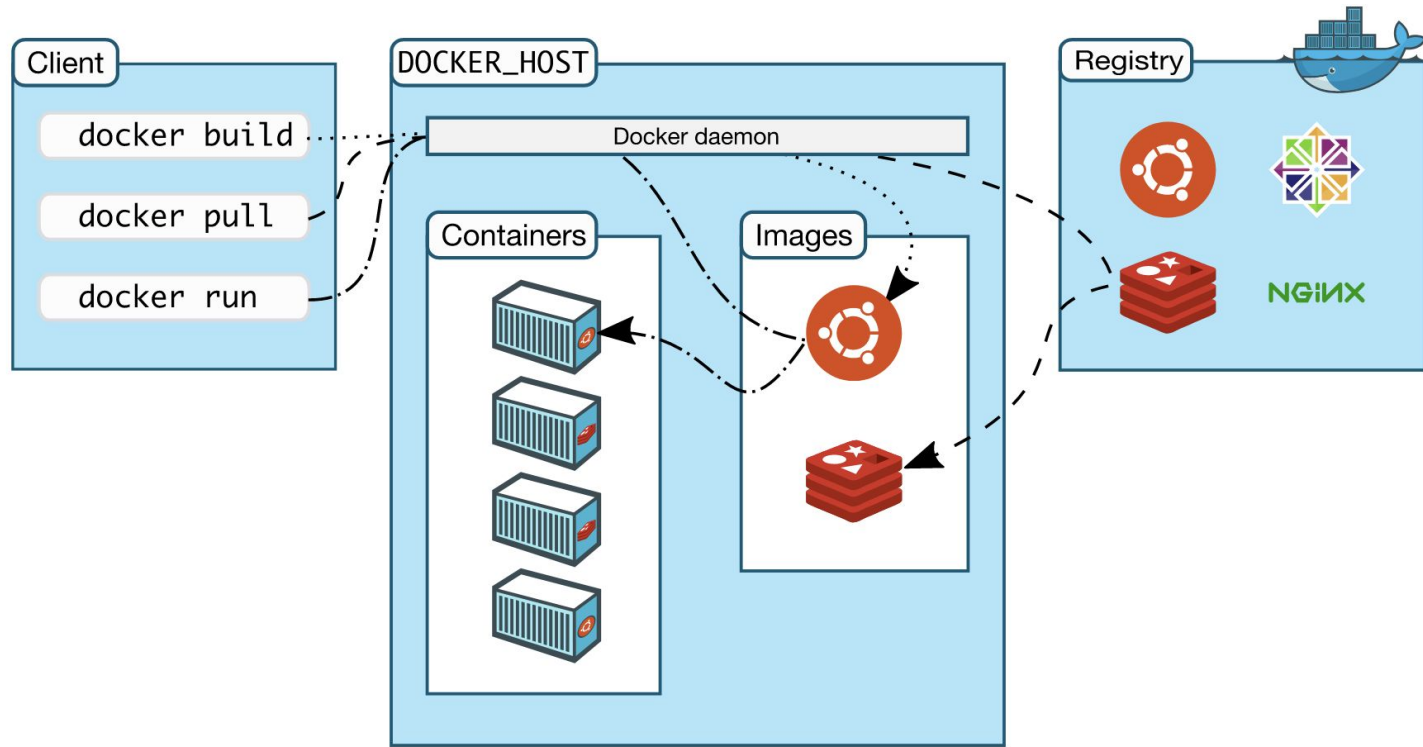
Container solutions



LXC

- **Application** container engine
 - Unlike LXC which is a **system** container engine
- Uses the resource isolation mechanisms of the Linux kernel
 - Namespaces
 - Cgroups
 - Layered filesystems
- Consists of three components: Software, objects and registry

Docker



Getting started

1. Install Docker on your system, e.g.: `# apt install docker-ce`
2. Run some container: `$ docker run -it ubuntu /bin/bash`

Hints:

- List all containers: `$ docker ps -a`
- Get a shell for a running container: `$ docker exec -it <container name> /bin/sh`

Getting started

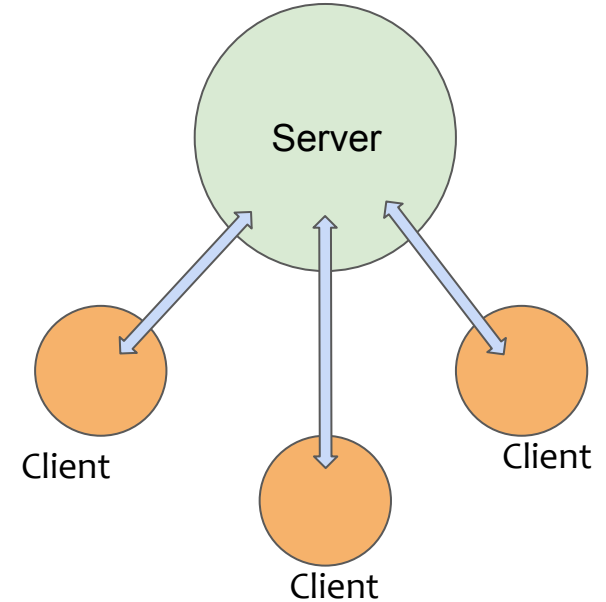
1. Create a simple Dockerfile (see [Docker's docs](#))
2. Build an image based on the Dockerfile: `$ docker build -t <image name> .`
3. Run the container: `$ docker run -d <image name>`
4. Create and run multiple containers using [Docker compose](#)

Hints:

- Force clean rebuild of image: `$ docker build --no-cache -t <image name> .`
- To prevent a container from exiting early when debugging, add `CMD tail -f /dev/null` to the end of the Dockerfile

Client-server architecture

- Server
 - Usually a long running process (*daemon process*)
 - Manages some resources
 - Receives and process requests
- Client(s)
 - Sends one or more requests to the server
 - Wait for the server's reply
- Transport layer
 - Network medium
 - Transfers the data

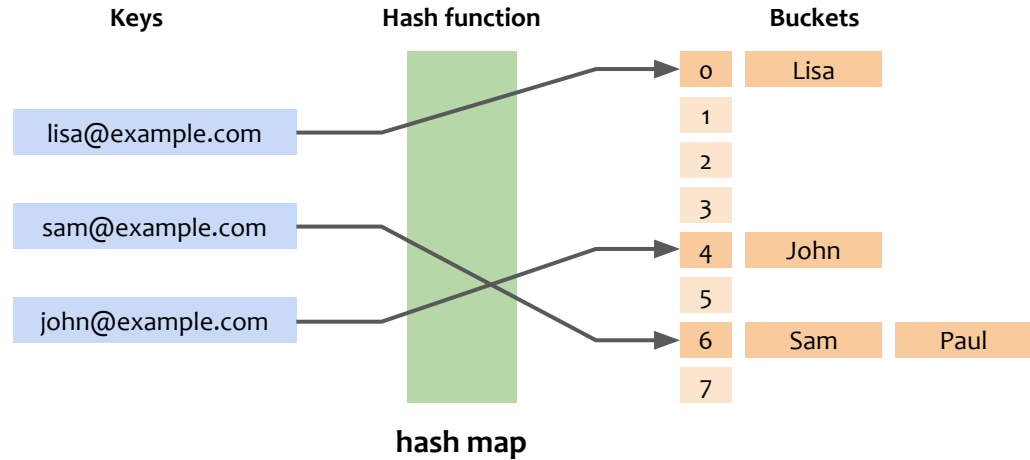


Key-Value store (KVs)

- Data structure
 - stores, retrieves and manages data
 - e.g., dictionaries, hash-tables

Key	Value
K1	AAA,BBB,CCC
K2	AAA,BBB
K3	AAA,DDD
K4	AAA,2,01/01/2022
K5	3,ZZZ,5623

dictionary



Motivation

Key-value stores play an important role at tech giants:

memcached

Facebook
Twitter
Zynga



Redis

GitHub
Digg
Blizzard Interactive



redis

Voldemort

LinkedIn

Dynamo

Amazon



Key-value stores play an important role in the scientific community:

- [FASTER: A Concurrent Key-Value Store with In-Place Updates](#)
[SIGMOD '18]
- [KVell: Design and Implementation of a Fast Persistent Key-value Store](#)
[SOSP '19]
- [Nova-LSM: A Distributed, Component-based LSM-tree Key-value Store](#)
[SIGMOD '21]

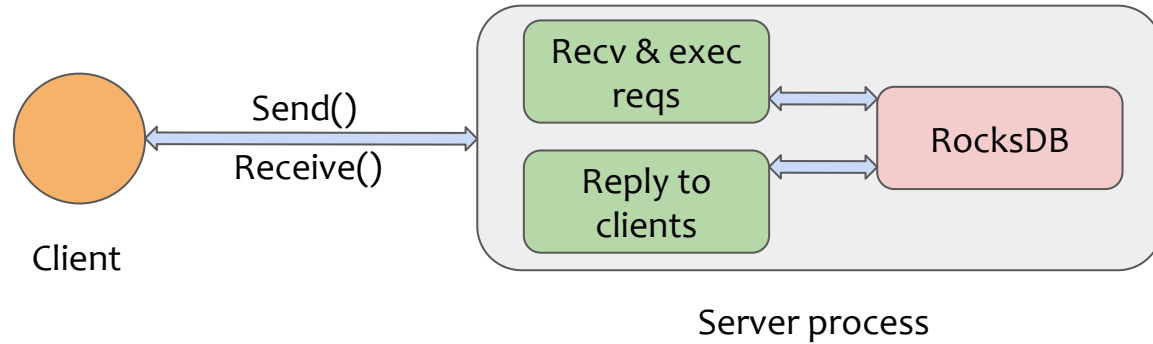
Key-value stores typically implement a set of operations:

- GET
 - Retrieve a value by key
- PUT
 - Insert or update a key-value pair
- DELETE
 - Deletes a key-value pair (if it exists)
- Range Queries
 - Queries applying to a range of KV pairs

- Performance
 - lock contention, significant write-traffic, complex memory management
 - low-latency operations and high throughput (I/O, batching)
 - parallelism (e.g., keys hashing)

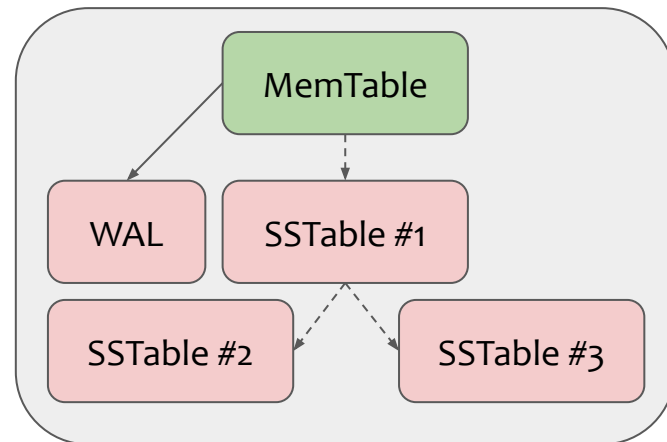
- Data properties
 - Persistency, e.g., persistent KVs or in-memory KVs
 - Consistency, e.g., linearizability or sequential consistency
 - Durability or crash consistency (for persistent KVs)

Task #1: Client/server arch + RocksDB



RocksDB architecture

- LSM-data structure
 - In-memory skiplist (MemTable)
 - SSTable files (persistent) organized on levels with (sorted) KV pairs
 - Compaction (background, multithreaded)
- Data properties
 - Linearizable reads, a read always "sees" the latest write
 - Durability, SSTables are persistent
 - Crash-consistency through Write-Ahead-Log (WAL)
- API
 - supports PUT, GET, DELETE queries



RocksDB

References

Talks & documentation:

- Rocksdb: <https://github.com/facebook/rocksdb>
- Rocksdb 101: https://www.youtube.com/watch?v=V_C-T5S-w8g
- Rocksdb talk: <https://www.youtube.com/watch?v=tgzkgZVXKB4>
- Protobufs: <https://developers.google.com/protocol-buffers>

Useful tutorials:

- Rocksdb tutorial: <https://rocksdb.org/docs/getting-started.html>
- Protobufs tutorial: <https://developers.google.com/protocol-buffers/docs/cpp tutorial>
- Socket tutorial: https://www.linuxhowtos.org/C_C++/socket.htm
- Libevent tutorial: <http://www.wangafu.net/~nickm/libevent-book/>

Task #1



Your task for the next three weeks:

- Implement the network connection between client-server in a KVS architecture
- Implement the KVS operations
- Use
 - (1) RocksDB as the KV store
 - (2) kernel sockets for the networking
 - (3) google protobufs as the serialization protocol

- Prof. Pramod Bhatotia
 - pramod.bhatotia@in.tum.de
- Dimitrios Stavrakakis
 - dimitrios.stavrakakis@tum.de



Workspace: <http://ls1-courses-tum.slack.com/>

Website: <https://dse.in.tum.de/>

GitHub: <https://github.com/TUM-DSE/cloud-lab/>

Channel: #ws-22-cloud-lab

Join us with TUM email address (@tum.de)

Thank you for listening!

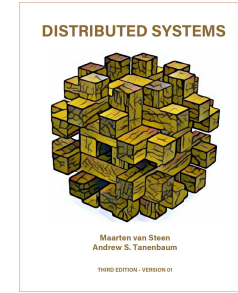
See you in the Q&A session

- University plagiarism policy
 - <https://www.in.tum.de/en/current-students/administrative-matters/student-code-of-conduct/>
- Decorum
 - Promote freedom of thoughts and open exchange of ideas
 - Cultivate dignity, understanding and mutual respect, and embrace diversity
 - Racism and bullying will not be tolerated

Recommended readings

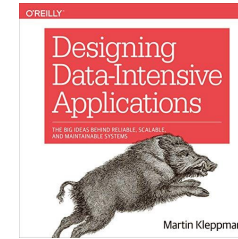
“Distributed Systems”

— Maarten van Steen and Andrew s. Tanenbaum



“Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems”

— Martin Kleppmann



“Designing Distributed Systems: Patterns and Paradigms for Scalable, Reliable Services Book”

— Brendan Burns

