Assignment 2 – IDG2001

Dockerised website

Table of Contents

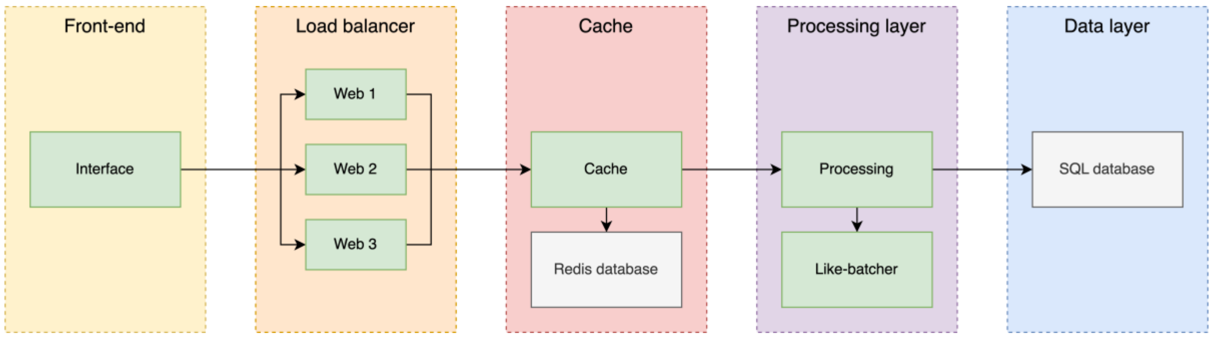
In this assignment, we will create a simple-ish website inspired by Reddit-like forum sites. We will create it on OpenStack (Chameleon Cloud), and we will use Docker and Docker-Compose to build it.

Deadline: See StudentWeb/Inspera for accurate deadline. Your group **MUST** deliver by the deadline. Inspera has zero chill. You will fail the course if you do not.

NB! Groups should be the same as for assignment 1. Talk to the teacher if this would be a big problem for you. The assignment should be delivered as a group, though double check this with the Inspera delivery site.

System description

Reddit allows users to make posts by category. So, there will be a feed in reddit/r/cats and another feed in reddit/r/science. They allow other users to like these posts. You can also create users and stay logged in during the session. Posts does not have to support images.



- Gray blocks should use external Docker images, from DockerHub. They can mainly be defined in the docker-compose file.
- Green blocks should be made by you. That being said, the green blocks can use external modules, e.g., load balancer using an nginx Docker image.

The following blocks should be their own Docker images:

- Web server (so it can be load-balanced)
- Cache
- Processing + Like-batcher
- SQL database (external image)
- Redis database (external image)

The rest, you can do with as seems fit. The system should be build using Docker-Compose, so re-building should only require a few commands when upgrading. E.g., something like the following.

- git push
- [ssh into VM]
- git pull
- docker-compose down
- docker-compose up --build

I recommend using either FastAPI (what I have shown during class) or Flask for the APIs. I except you to use Python for the processing and cache layers. For the web and data layer, feel free to use the technologies you want (for example NodeJS or PHP). The database should be an RDB, but whether you use MySQL, PostgreSQL, or similar systems, is up to you.

Usage/functionality

The website should support the following actions.

- Read posts by category/“sub-reddit”.

- Read posts by user.
- Create a post in a category (linked to your user).
- Create a user.
- Like a post

Like you would expect from a Reddit-like forum website. There may be more things you need to implement which are not listed. You will need at least the following interfaces.

- Site showing the last (up to) 10 posts, per category.
    - This must show the messages and show a clickable like-button.
    - It should also include a “Create new post” link/button.
- Site allowing the creation of new post. Can be the same as the point above.
- Site showing specific users (e.g., reddit/u/user123).
    - If you are logged in as this user: Show all info, including a delete button.
    - If not, show only username and email.

The categories can be hardcoded. I.e. add two or three hardcoded categories to your site.

A post should include the following data (though all this data does not have to be visible to the end-user).

- ID
- User
- Category
- Text
- Date
- Number of likes

And maybe more. Normally, a “like” should include which user likes it, so you can unlike, and so users won’t be able to like posts multiple times. In this case, you do not have to care about this. “Like” means increment a number in the database, regardless of whether *you* have liked it previously. A created user would require the following data (and maybe more).

- Username
- Email
- Password (preferably hashed and salted)
- Date of account creation
- ID (not necessary if username is unique)

Testing

I want you to do basic tests for the cache and processing layer. Add test files with some unit tests. They do not need to run automatically, or in GitHub, but they should be runnable locally. I.e., running the “pytest” command (or “python -m pytest” or similar) should run the tests. You should also follow PEP-8 principles (to an extent). So, the commands "pytest", "flake8 .", and "mypy ." should run without errors.

Web/front-end

You stand relatively free to make the web/front-end the way you want it. I would however recommend creating it the way it "should" be. You're here to learn, after all.

Load balancer

This should work on the front-end web server. I would recommend using external tools here, instead of reinventing the wheel. Check YouTube if there are any tutorials on how to do this. I do not expect you to do any scaling based on load. Only to spawn e.g., 3 web servers for the front-end. "3" can be hardcoded.

Cache

The cache module should be replaceable and exists only as a tool to help with large loads. Remove it, and the system should work as usual, but with more load on the processing layer. Include it, and many of the requests will be returned without accessing the processing layer.

Processing

The processing API should receive requests from the web server. Things like POST at site/r/cats or GET at site/r/science. It should follow REST principles to the best of your ability. See lecture about REST to find out how to model the data you will get from this site.

Like-batcher

Ref. Tom Scotts video from a previous lecture, we want the processing layer to batch likes. Instead of spamming the database with one post-like at the time, we want it to batch them up. You are free to decide by which method this is done (time-based, count-based, etc.) by my suggestion is to simply update the database whenever the size of the batch exceeds a set number of likes. E.g., when batched likes are above 10, send to database. The like-batcher does not have to be its own Docker container/Image. It can be a part of the processing API, but it should be in its own Python file, and thus be imported.

Database

I want you to use a relational database system (RDB), for example MySQL, and I would recommend using an external Docker image. This will require you to do some simple database modelling, but the quality of the modelling will not be important for the project. (Though I would recommend you following proper principles, just because you'll learn more.)

Report

Like last report, I except the report to include:

- Brief document your systems
- Discussion of some of the choices you had to take as part of the planning and development.
- Some basic modelling of your system and database.
- A very brief “How to” guide for building such a system.

Bonus points and extra

Subtracted points may be readded if you pass some of the following extra steps.

- A good front-end, a good report and good code.
- Automatic config of OpenStack VM, so creating a new VM would be much less work. (Also useful for you.)

Some groups did not finish all the requirements from assignment 1. This will drop your score and possibly grade. But, you will get this second chance to make up for it. If you add some of the following steps, assignment 1 will be corrected as if you did them as part of assignment 1.

- Automatic unit testing, tox and GitHub actions. Can be added to the processing layer, for example testing the like batcher.
- Adding automatically generated docs for parts of the project, for example using Sphinx.

Deliverables

Deliverables are a brief report and the source code. Download the repo as a zip file. The report can be part of the repo.

The deliverables should also include the first assignment. My suggestion: Make the following file structure:

- Assignments-idg2001-N/
    - Assignment-1/
        - Repo/
        - Report.pdf
    - Assignment-2/

- Repo/
- Report.pdf

Where N is your group number (see Blackboard).

Suggested project plan

You have almost 6 weeks before the deadline. Since Inspera has zero F-s to give, I would suggest delivering at least the day before the deadline (Sunday). That way, we may be able to solve issues if there are any on Monday.

My suggested project plan – which you can feel free to follow or not – would be something like the following.

Week	Work tasks
1	Set up CC/OpenStack config scripts and work environment. • Set up a git repo and put it on GitHub. • Set up necessary CC/OpenStack VM, volume, and network settings. • Install necessary tools, like Docker, Docker-Compose in your VM. • Clone your git repo to the VM. • Set up an initial Docker or Docker-Compose, just to make sure it actually works. Create locally, push to GitHub, pull to VM and run. • Make sure SSH also works on the VM.
2	Database and modelling. • Model the database (and set it up using Docker-Compose and an existing Docker Images on DockerHub). • Plan the dataset, API endpoints, etc. for the processing API (and the rest of the system).
3	Make the core elements of the system. • Build the processing API/layer. • Build the web server/front-end. • Connect all the systems and ensure they work.
4	Make the “optional” sub-systems. • Make the cache, the load balancer, and the like-batcher, and add them to the rest of the system. • Test that the whole systems work as it should.
5	Report and deliver. • Write the report, make diagrams, etc. • Deliver!

The cache and load balancer should be modules which can relatively easily be added or removes. E.g., by modifying ports or environmental variables in the Docker-Compose file. Thus, making the core system, and then adding this later, should work fine.