Assignment 2

This assignment will be done in teams, to which you have been assigned as shown in this sheet. You should familiarize yourself with the class protocol, and make sure your team has settled on a TL for this assignment.

Your team will build a very simple web server that echoes HTTP requests it receives into a well-formed HTTP response. Your team will then deploy this server to the Google Cloud platform.

Each student should submit this assignment by 11:59PM on April 15, 2025 into the submission form.

TABLE OF CONTENTS

Team resource setup

Before starting code on this assignment, your team will need to set up a team repository in Gerrit and set up a project in the **cs130.org** organization on Google Cloud. The TL should complete this portion of the assignment.

Gerrit

First, choose a creative name for your team. Make it unique, maybe a bit fun, maybe a bit clever. Choose a name that meets the following rules:

- Starts with a letter

- Contains only lowercase letters, numbers, or dashes
- No punctuation other than "-"
- No spaces
- Is *not* just a collection of your first or last initials... that's not creative

Then create a group with that name for your team in Gerrit. Change the **Group Options** to *Make group visible to all registered users*, and **Save Group Options**.

Next, create a repository for your team in Gerrit with the same name. Be sure to set the **Owner** to your team group.

Cloud

Finally, your team should create a project in Google Cloud:

- Register for a coupon code as instructed via e-mail using your *@g.ucla.edu* account. The billing account will be created in the *g.ucla.edu* organization but can be used by your project in *cs130.org*.
- Navigate to the New Project page. Make sure you are logged in with your *@g.ucla.edu* account as shown in the Google account selector in the upper right corner.
- Enter your team name as **Project Name**. The generated **Project ID**, which must be unique across all of Google Cloud, may end up with a numeric suffix.
- The **Billing account** should match the name of your recently created account (e.g. *Billing Account for Education*).
- Set the **Organization** to *cs130.org*.
- In **Location**, click **Browse** and select *2024* (under *cs130.org*) in the dialog.
- Click **Create** to create your project.
- You should be taken to the **Dashboard** with your project selected from the drop-down in the blue title bar at the top of the page. It may take a few seconds for your project to finish being initialized.
- Navigate to the **IAM & Admin** section from the hamburger menu (≡ icon) in the upper left and select **IAM**.
- At the top you will find a button to **Grant Access** for each of your team members by e-mail address (@g.ucla.edu) as *New principals* with the *Owner* role, then **Save**.
- Verify that your entire team is listed in the **Permissions** table with *Owner* in the **Role** column. New Owners may need to accept an e-mail invitation sent by the system, so make sure your team members accept the invites and any orange warning signs

disappear.

With that, your project should be set up. You will use this project throughout the class to build, store, and deploy your applications on servers running in Google's data centers, i.e. the cloud.

At this point you should initialize the Google Cloud SDK tool, `gcloud`. Follow these instructions to login and enable APIs for future use with `gcloud`.

Add initial skeleton code

At this point the TL should hand over work to a team member to handle coding tasks.

As you did in Assignment 1, start by uploading some skeleton code to your team repository. You will be using the Boost library (documentation) to help create your server. The official Boost examples include a simple echo server that would make a reasonable starting point for your web server. (Yes, the examples also include a simple HTTP server, but let's not jump there yet. Start with the echo server.) We've packaged the echo server into a file so that you can use it to get started.

```
$ git clone ssh://${USER}@code.cs130.org:29418/${REPO}
$ cd ${REPO}
$ git checkout -b skeleton
$ git review -s
$ curl http://static.cs130.org/src/boost-server.tar.gz | tar -zxv
```

You should once again generate config files from our project templates to get you started. You will need to edit `CMakeLists.txt` to compile `server_main.cc` and link it with `Boost::system`. Since you do not have any unit tests (yet), you can comment out the test executable and `gtest_discover_tests` rule, as well as all the lines at the end relating to code coverage.

You should be able to compile the server without making changes to the code, using CMake followed by `make`. Launch the server with a test port, and then test that the server is functioning correctly by typing text and checking for a response, using netcat in another terminal:


```
$ cd build
```

```
$ cmake ..
$ make
$ bin/server 8080
```

In another terminal:

```
$ nc localhost 8080
```

Assuming all goes well, send this initial code out for review on Gerrit and submit.

Write an echoing web server

Next you will take that example echo server and turn it into a configurable echoing web server. This is not super useful on its own, but will provide the foundation for a future fully-fledged web server.

You should start by refactoring `server_main.cc` into several source files, creating a separate source and header file for each class. The refactoring alone (with no behavior change) should be sent out for review.

The echoing web server should listen on a configurable port, and respond to HTTP 1.1 GET requests by echoing the request back to the client. You will have to:

- Detect when the request is complete
- Send an HTTP 200 response code
- Set the content type to `text/plain`
- Send the request in the body of the response

For reference you can consult the official HTTP/1.1 spec for request and response or search the web for examples.

Your web server should read its configuration from a file in Nginx format. Use the code (and unit tests) from from one of your team members' submissions for Assignment 1 as a starting point. Copy the code into your shared repository and update `CMakeLists.txt` to include it. The only configuration parameter you should need (so far) is the port number. Do not just use `example_config` as-is from Assignment 1, which contains other unnecessary parameters.

The server should take a path to the config file on the command line, as such:

```
$ bin/webserver my_config
```

../config/config

Once your server is running as you expect, send it out for review and submit it.

Create a Docker container

Next you will create a Docker container for your webserver. Start by addressing all the `TODO(!)` comments in the `docker/Dockerfile` generated from the project templates. In the deploy stage, you can add command-line parameters (e.g. your server configuration file name) for the `ENTRYPOINT` binary with the `CMD` statement. Make sure you `COPY` your configuration file from the `builder` stage so it's accessible in the `deploy` stage (see Docker docs on multi-stage builds).

You should build and run your server with Docker on your local machine using your edited `Dockerfile`. Note, you will need to build and tag your `:base` image before building the builder/deploy image since `builder` depends on `:base`. Run the server, mapping your server's port to a port on your loopback network interface (`127.0.0.1`). Verify that the server responds as expected. To shut down the container, run `docker container stop` in a separate terminal.

Make sure you submit your updated `Dockerfile` to your repository.

Deploy your web server to Google Cloud

In this step you will deploy your web server on Google's cloud platform for the world to see. To make things easier later, create a new server configuration file using port 80, and use that configuration file specifically for your Docker container (not for local development).

Next you will use your Docker container definition to create a container image on Google Cloud. Follow the guide for building a container, and then visit Cloud Build to see the results of your build. Click on a *Build* ID in the **Build history** table to see the **Build details**. From there you can view the **Build Logs** from each of the **Steps** that were run and, if successful, see links to the generated **Images** in the **Build Artifacts** tab. Note the full name of your generated image, e.g. `gcr.io/.../...:latest`.

Next you will deploy your container image to a server on Compute Engine. From the **VM instances** page, click **Create Instance**, and then fill in the following settings:

- Enter a descriptive name for **Name**, such as *web-server*
- For **Region** choose *us-west1 (Oregon)* and leave the default **Zone**. Do not choose *us-west2 (Los Angeles)*, it's more expensive. Cost of living and computing are less in Oregon.
- For **Machine configuration**, choose *E2* for **Series** and *e2-micro* for **Machine type**. It sounds small, but your server probably will not be seeing heavy loads. And, being written in C++, it's fairly lightweight. In any case, it's not too expensive and you can create a new faster instance later if you need to. At this point your monthly estimated cost should be $7-8.
- Under **Container**, click **Deploy Container**. For **Container image**, enter the full name of *your* generated image, which looks like `gcr.io/${PROJECT}/${REPO}:latest`. Click **Select** to close the container dialog.
- Under **Firewall**, click to **Allow HTTP traffic**.
- Click **Networking, disks, security, management, sole-tenancy** and edit some additional settings:
  - Within **Networking**, click the *default* under **Network interfaces**, and under **External IPv4 address** select *Create IP Address*. In the dialog that appears, enter a name (such as *web-server-ip*), a reasonable description, and click **Reserve**. Click **Done** to close the **Edit network interface** editor.
  - **NOTE:** External IP addresses are only free as long as they are attached to a running GCE machine. Try not to create create more than one or two external addresses, and always keep them attached to running machines. You can view and release IPs you have reserved here. Creating unused external IPs without cleaning them up will be viewed as a sign of poor project health, and graded accordingly.
  - Within **Management**, find **Metadata** and **Add item** with *Key* of `google-logging-enabled` and *Value* of `true`. This will allow your Docker container logs to propagate to Google Cloud's Logs Viewer.
- Click **Create** to create your instance.

Once your instance finishes starting, you can access your server on port 80 of the **External IP** shown in the table by clicking the linked IP adddress in the table. If something isn't working, you can start troubleshooting by connecting to the instance over *SSH*:

```
$ gcloud compute ssh web-server
web-server ~$ docker ps
```

If your server is up and running, returning responses, you're done! Submit any config changes, and have the TL fill out the team submission form with your running server's IP address (you can make DNS entries in a future assignment).

Grading criteria

The minimum requirements for this assignment are drawn from the instructions above. The team grading criteria includes:

- Correct refactoring
- Successful update of Dockerfile
- Server builds without failure
- Successful deployment of server on GCP
- Deployed server properly echoes client requests
- Server can be run with a simple command (e.g. `./webserver <config_file>` or similar)
- The port is configurable via config file

Individual Contributor criteria includes:

- Code submitted for review (follows existing style, readable, testable)
- Addressed and resolved all comments from TL

Tech Lead criteria includes:

- Kept assignment tracker complete and up to date
- Maintained comprehensive meeting notes
- Gave multiple thoughtful (more than just an LGTM) reviews in Gerrit

General criteria includes how well your team follows the class project protocol. Additional criteria may be considered at the discretion of graders or instructors.

Submit your assignment

Everyone should fill out the submission form before 11:59PM on the due date. We will only review code that was submitted to the `main` branch of the team repository before the time of the TL's submission. You may update your submission by re-submitting the form, which will update the submission time considered for grading purposes if you are the TL.

Late hours will acrue based on the submission time, so TL's should avoid re-submitting the form after the due date unless they want to use late hours.

"You may think using Google's great, but I still think it's terrible." —Larry Page

Page last modified: April 9, 2025.