

Tutorial 1 - SOPRA

Lucio Canepa Joel Cedric Schmidt Luke Benjamin Fohringer

2026-02-06

Table of contents

1	Introduction	1
2	What do you need before starting	5
3	Clone repositories & run code locally	11
3.1	Client repository	12
3.2	Server repository	13
4	Git & Github workflows	14
4.1	Git short guide	14
4.2	Github workflows	26
5	Set-up Google Cloud and Vercel & deploy your project	28
5.1	Set-up Google Cloud and link it to the server repository	28
5.2	Set-up Vercel and link it to the client repository	63
5.3	Set-up Docker Hub and containerize your server application	76

1 Introduction

The goal of SOPRA and particularly Milestone 1 is to get familiar with the basics of Full-Stack web development with the chosen frameworks.

This guide offers step-by-step tutorials to set-up your local machine for development and deploy it. Conceptual explanations and recommandations are also included.

What's a Web Application? - Main concepts

Web Application components

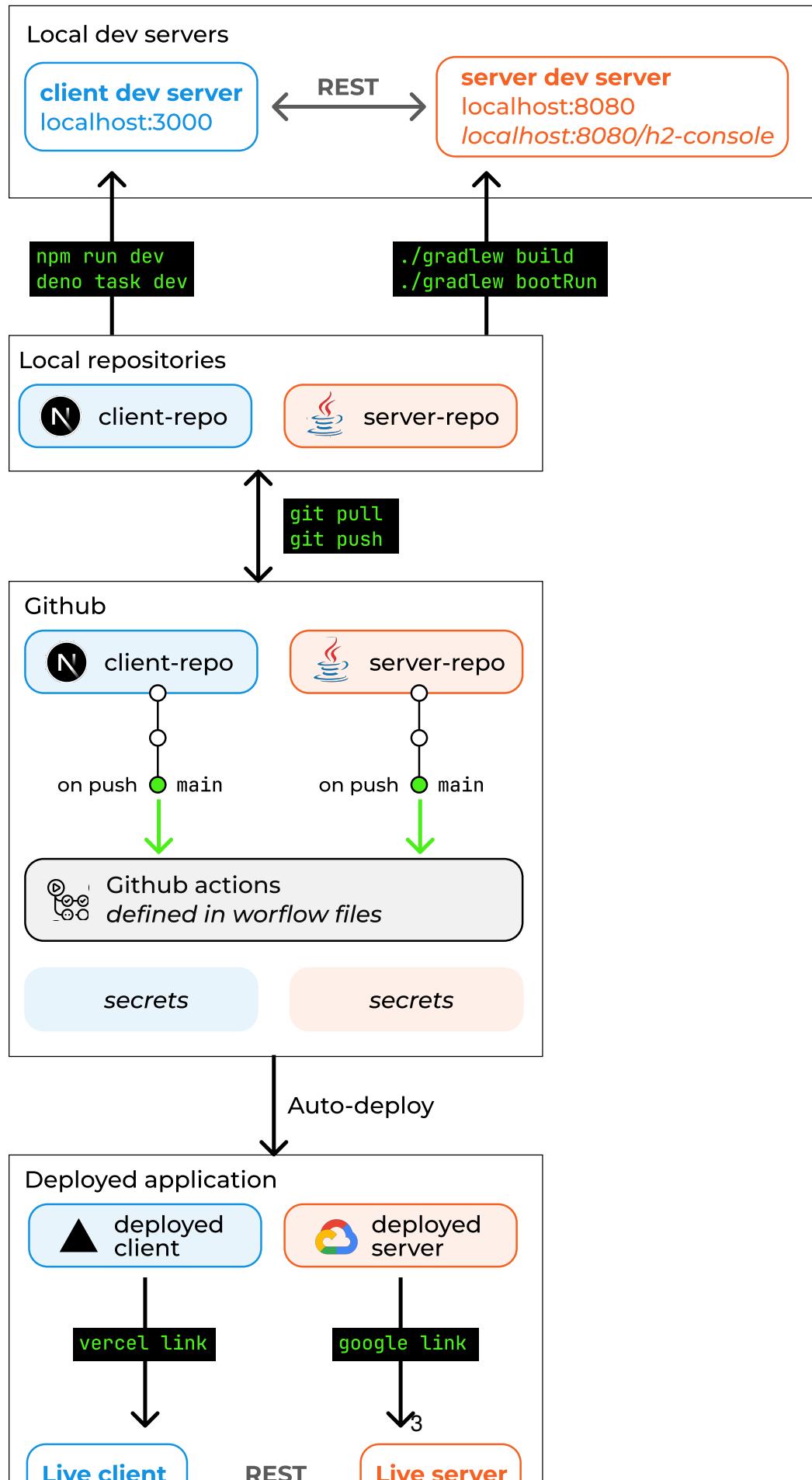
Web applications are usually built with 3 fundamental components:

- **Client:** This is what the users see when visiting a web application. It's responsible to fetch the data and display it to the user in an interactive way. The client can interact with the data through the **backend** and a defined API between them (in our and most cases

REST)

- **Backend (Server):** The backend has direct access to the database (can read and write) and offers specific access points (**endpoints**) to clients. It's responsible to manage the complexity of retrieving the needed data that clients request and check access permission.
- **Database:** The database store the actual data. There can be many types of databases and languages to interact with it. In this course, a semi-persistent database (only lives in memory, RAM) based on Postgress and accessible through SQL is set-up.

Diagram



REST

REST stands for REpresentational State Transfer and offers **CRUD** (Create, Read, Update, Delete) operations hitting a specific endpoint.

Endpoints

Endpoints are specific URI to describe a specific resource to retrieve: `/users/{id}`.

Used in combination with REST:

`GET /users/{id}`

header: Authorization token (a key used by the backend to grant access to the resource)

Database

The setup of the server repository in SOPRA already contains and spins up the in-memory database. This means that there is no external database to handle.

Deployment

The 2 repositories templates available for sopra are:

- `sopra-fs26-template-client`
- `sopra-fs26-template-server`

responsible for the client and server.

These repositories are hosted on Github. As a convention, the main branch represents the latest working instance (what should be deployed and accessible by the users). The 2 repositories are respectively deployed on:

- client on Vercel
- server (together with the in-memory database) on Google Cloud

This happens automatically when someone pushes code changes on the main branch: this is managed with Github Workflow to support continuous development (there is no need to manually deploy applications).

Development

We usually refer to 2 main environments:

- production: the one we just discussed and actually run the application accessible to the users
- development: where developers work on their local machine, change and share code while implementing new features and fixing bugs.

Developers clone the repositories hosted on Github on their local machine, work on the code and push the code (they share the changes to Github – with the other team members). If it's the case, the automatic CI/CD pipeline defined by Github workflows take care of it and redeploys the application.

To run and test the application locally, you need:

- local application running with the client
 - local application running with the backend (server)
-

Development flow

- clone application locally
- install needed dependencies to run it locally
- implement a new feature / fix a bug
- test locally that everything works correctly
- push the new code to the remote (Github) usually on a feature branch
- once the code is tested and ready to be integrated into production, the feature branch gets merged into the main branch, triggering the deployment

2 What do you need before starting

To clone the code on your machine and run it locally, you need to set-up / have installed:

- [git](#): version control system
- java 17+ (JDK) installed and JAVA_HOME environment variable set
- [Github account](#)
- ssh key to authenticate with Github easily

You will install other needed packages from set-up files provided by the repositories.

Warning

If you are on Windows, make sure to install WSL (Windows Subsystem for Linux) first.

Tip

If you are on a Mac, I highly recommend installing [Homebrew](#) as package manager:

- You can install packages with: `brew install <package_name>`
- You can install applications with: `brew install --cask <application_name>`

Setup ssh key

Note

If you are on Windows, first install Linux Subsystem for Windows (WSL) and then follow the instructions below.

In your terminal:

```
ssh-keygen -t ed25519 -C "<my_computer_name>"
```

It will prompt you for:

- choosing location (default is fine)
- choosing a passphrase (optional: can leave blank for no passphrase)

Then, add the ssh key to the ssh-agent:

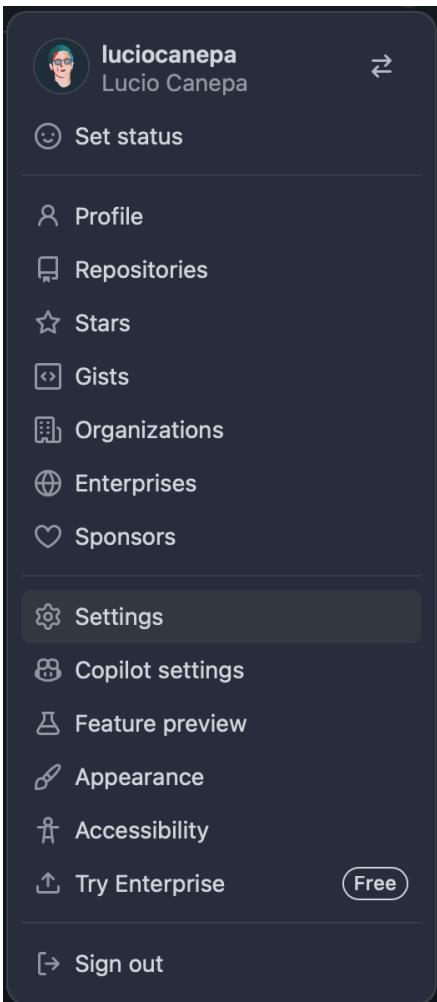
```
eval "$(ssh-agent -s)"  
ssh-add ~/.ssh/id_ed25519
```

You need the public key to add it to your Github account:

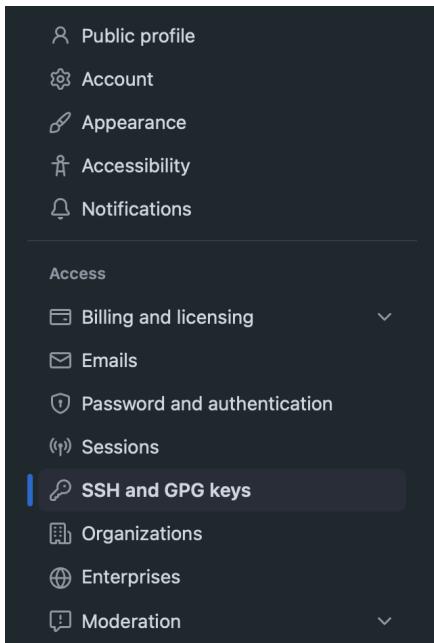
```
cat ~/.ssh/id_ed25519.pub
```

Link ssh key to Github account

1. Go to your Github settings



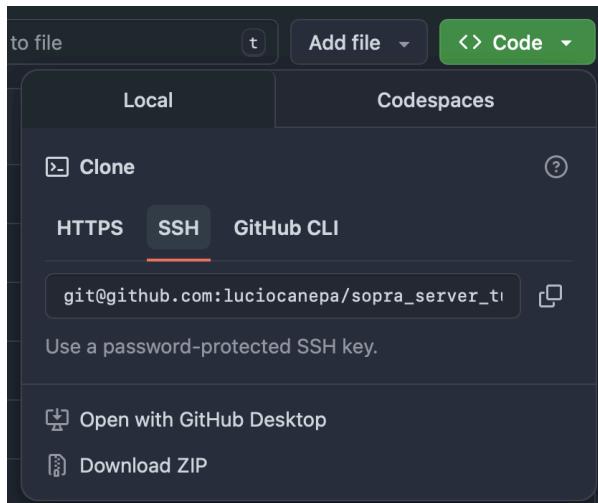
2. Go to “SSH and GPG keys”



3. Click on “New SSH key”, give it a title (e.g. “My Laptop”) and paste the public key you got from cat ~/ssh/id_ed25519.pub

A screenshot of the 'Add new SSH Key' form. It has a dark background. The form fields are: Title (lucio_mac_vm), Key type (Authentication Key), and a large text area for the Key (containing a long string of characters). At the bottom is a green 'Add SSH key' button.

4. Once you added it, you can use ssh to clone repositories from Github:



Now you are able to act on remote repositories hosted on Github without entering your credentials every time. Always use the ssh url when cloning repositories from Github.

Install Java 17

Mac

If you are on a Mac, you can install Java 17 with Homebrew:

```
brew install --cask temurin@17
```

Run:

```
/usr/libexec/java_home -v 17
/Library/Java/JavaVirtualMachines/temurin-
↳ 17.jdk/Contents/Home
```

to locate your Java 17 installation path.

Run:

```
which ${SHELL}
/bin/zsh
```

to find out which shell you are using.

Then, add the following line to your shell configuration file (`.zshrc` for zsh, `.bash_profile` or `.bashrc` for bash):

```
code ~/.zshrc
```

paste

```
export JAVA_HOME=$(/usr/libexec/java_home -v 17)
export PATH="$JAVA_HOME/bin:$PATH"
```

at the end of the file, save and close it.

Run `source ~/.zshrc` to reload the configuration file.

Linux / WSL

After making sure you have `git` and `curl` installed on your system:

```
sudo apt update
sudo apt install git curl
```

you can install Java 17 with:

```
sudo apt install openjdk-17-jdk
```

Depending on your package manager of choice.

Ensure you have Java 17 installed and configured correctly by running:

```
macvm@Macs-Virtual-Machine sopra_server_tutorial %
  ↵ java -version
openjdk version "17.0.17" 2025-10-21
OpenJDK Runtime Environment Temurin-17.0.17+10
  ↵ (build 17.0.17+10)
OpenJDK 64-Bit Server VM Temurin-17.0.17+10 (build
  ↵ 17.0.17+10, mixed mode, sharing)
```

Install WSL on Windows

If you are using **Windows**, you need to install **WSL (Windows Subsystem for Linux)** before continuing with the Linux/WSL installation steps (Git, Curl, Java, ...).

Tip

Save your work first: the installation may require a reboot.

1. Open **PowerShell as Administrator** Start Menu → search **PowerShell** → right-click → **Run as administrator**
2. Run:

```
wsl --install
```

3 Clone repositories & run code locally

Navigate to the folder where you want to store the project and clone the repository:

```
cd <path_to_your_folder>
git clone
↳ git@github.com:<user_name>/<repository_name>.git
```

where `<repository_name>` are the names of the server and client repositories you created from sopra templates.

Details

```
macvm@Macs-Virtual-Machine ~ % mkdir sopra
macvm@Macs-Virtual-Machine ~ % cd sopra
macvm@Macs-Virtual-Machine sopra % git clone
↳ git@github.com:<user_name>/sopra_serverTutorial.git
Cloning into 'sopra_serverTutorial'...
The authenticity of host 'github.com
↳ (140.82.121.4)' can't be established.
ED25519 key fingerprint is <your_SHA256_fingerprint>.
This key is not known by any other names.
Are you sure you want to continue connecting
↳ (yes/no/[fingerprint])? yes
Warning: Permanently added 'github.com' (ED25519)
↳ to the list of known hosts.
remote: Enumerating objects: 73, done.
remote: Counting objects: 100% (73/73), done.
remote: Compressing objects: 100% (46/46), done.
remote: Total 73 (delta 3), reused 72 (delta 3),
↳ pack-reused 0 (from 0)
Receiving objects: 100% (73/73), 71.61 KiB | 596.00
↳ KiB/s, done.
Resolving deltas: 100% (3/3), done.
macvm@Macs-Virtual-Machine sopra % git clone
↳ git@github.com:<user_name>/sopra_clientTutorial.git
Cloning into 'sopra_clientTutorial'...
remote: Enumerating objects: 52, done.
remote: Counting objects: 100% (52/52), done.
remote: Compressing objects: 100% (47/47), done.
remote: Total 52 (delta 0), reused 52 (delta 0),
↳ pack-reused 0 (from 0)
Receiving objects: 100% (52/52), 120.59 KiB |
↳ 620.00 KiB/s, done.
macvm@Macs-Virtual-Machine sopra % ls
sopra_clientTutorial    sopra_serverTutorial
```

Caution

Use the ssh url when cloning the repository.

At this point you can already look into the code, read the README.md file and run the project locally.

Tip

If you are using [VSCode](#) as code editor, you can open the folder directly from terminal with `code <repo_name>` after activating Shell Command: Install 'code' command in PATH from the Command Palette (Cmd+Shift+P / Ctrl+Shift+P).

Or `code .` to open the current folder.

3.1 Client repository

Tip

If while cd-ing into a repo folder this message shows up:

```
macvm@Macs-Virtual-Machine sopra % cd
  ↵ sopra_server_tutorial
direnv: error
  ↵ /Users/macvm/sopra/sopra_server_tutorial/.envrc
  ↵ is blocked. Run `direnv allow` to approve its
  ↵ content
```

run `direnv allow` to enable the automatic environment variable loading. If direnv is not installed, install it with your package manager of choice (or curl it).

Run `source setup.sh` as described in the README.md to install needed packages / dependencies.

Deno and npm (2 TS to JS runtimes) are available in the repository: use `npm run dev` to run the development server locally (at default port `localhost:3000`).

At `package.json` you can find available scripts:

```
"scripts": {
  "dev": "next dev --turbopack",
  "build": "next build",
  "start": "next start",
  "lint": "deno lint",
  "fmt": "deno fmt"
},
```

The code for your app lives under `app` / folder: there you can find the entry point to the application at `app/page.tsx`. Try to change something to see it automatically update in the browser on save.

3.2 Server repository

To build the application, use the provided Gradle wrapper. You first need to grant permission to the executable:

```
chmod +x gradlew  
./gradlew build
```

The first time you run this, the Gradle wrapper will download the necessary Gradle distribution.

If everything works correctly, you will see:

```
BUILD SUCCESSFUL in 2m 50s  
7 actionable tasks: 7 executed
```

Note

As suggested in the `README.md`, you can build the application continuously with:

```
./gradlew build --continuous -xtest
```

the two flags allow for:

- `--continuous`: to automatically recompile the code on changes
- `-xtest`: to skip running tests on every build

To run the server locally, use:

```
./gradlew bootRun
```

This runs on `localhost:8080` by default: there you will see The application is running.

At `localhost:8080/h2-console` you can access the H2 database console: Information needed to access the console are at `src/main/resources/application.properties`.

4 Git & Github workflows

4.1 Git short guide

To get started working with local / remote repositories and collaborate with your team members, it's important to get to know the most used git commands. [Videos](#), the [official documentation](#) and LLMs (with caution) are a good starting point.

Here we selected a subset of commands we consider relevant for this course.

Fundamentals

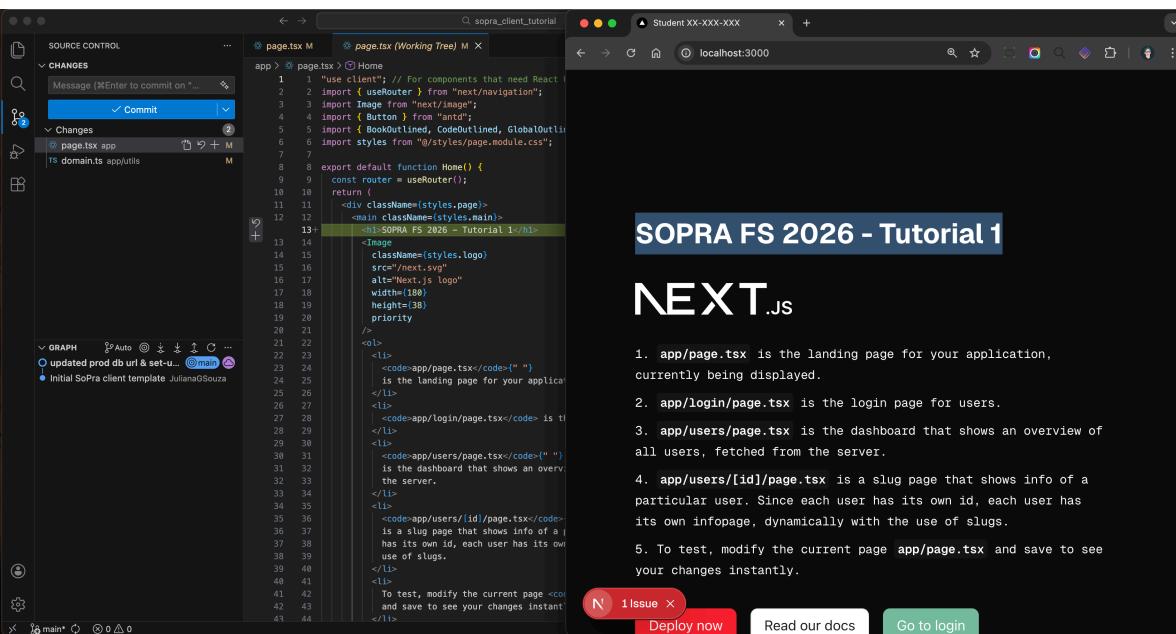
Git is a version control system based on branches that allows users to keep track of the file changes at a location. It allows to store incremental changes (commits) and to develop experimental features separately to the main code (branch and merge). All of this happens locally, on your machine. It's possible to connect your local repository to a remote origin (this happens automatically when you `clone` a repository from Github). Github simply acts as a second peer (the upstream) to whom you can push changes and from which you can pull changes (added by a team member).

The repositories you work on for Milestone 1 only have 1 branch, the `main` branch (standard entry point for an application).

- `git pull` tries to copy the changes of the branch you are currently on from the remote repository to your local one. This allows you to stay up-to-date with changes from other users.

Let's say you modified a file in your repository:

add title to main `page.tsx`



The screenshot shows a GitHub commit interface on the left and a browser window on the right. In the commit interface, a message is entered: "Message (Enter to commit on ...)" followed by a dropdown menu with options: "Commit", "Revert", "Squash", and "Cancel". Below this is a "Changes" section showing a diff for a file named "page.tsx". The diff highlights a new line of code: "13+ <h1>SOPRA FS 2026 - Tutorial 1</h1>". The browser window on the right displays a Next.js application at "localhost:3000". The page has a dark theme with a header containing the text "SOPRA FS 2026 - Tutorial 1". Below the header, there is some placeholder text: "NEXT.JS". At the bottom of the browser window, there are several buttons: "Issue", "Deploy now", "Read our docs", and "Go to login".

`git status` is a very useful command that gives you the snapshot (current state) of your branch:

- tells you on what branch you are and whether it's up-to-date with the remote branch
- lists all the files that were: modified, added, removed
- lists all the changes that are: untracked, unstaged, staged

You can move a change to the *staging area* (prepare them for a commit) with:

- `git add <file_name>`
- `git add .` to add all changes at once

Run `git status` to see what happened

Caution

Before committing the first time, ensure your `user.name` and `user.email` are properly set. In this way you can *sign* the commit: use your Github email and username to do so with:

```
macvm@Macs-Virtual-Machine
    ↵  sopra_client_tutorial % git config
    ↵  --local user.name "<username>"
macvm@Macs-Virtual-Machine
    ↵  sopra_client_tutorial % git config
    ↵  --local user.email "<email>"
macvm@Macs-Virtual-Machine
    ↵  sopra_client_tutorial % git config
    ↵  --local user.name
<username>
macvm@Macs-Virtual-Machine
    ↵  sopra_client_tutorial % git config
    ↵  --local user.email
<email>
macvm@Macs-Virtual-Machine
    ↵  sopra_client_tutorial %
```

You can choose between:

- the `--local` flag (recommended) that sets name and email for this specific repository (you need to do it both on the server and on the client repositories). This allows you to have multiple repositories on your computer linked to remotes where you are logged in with different accounts / you want to sign your commits differently
- the `--global` flag: all the commits on your machine are going to be signed with name and email you set

With `commit -m "<commit_message>"` you can create a commit with its commit message (where you described the changes / reference the issue / bug it solves)

You can see your commit with `git log`.

You can push the commit(s) to the remote repository with:

- `git push` if the branch already exist on the remote
- `git push --set-upstream origin <new_branch_name>` to register a new local branch with your commit(s) on the remote repository

Branching

A branch is a reference (pointer) to a commit, usually used to develop a new feature, fix a bug, etc. while maintaining the main branch (often – and in this course – continuously deployed) always working.

Access branches

- You can move your HEAD to an already existing branch with `git checkout <branch_name>`
- You can create a new branch pointing at current HEAD location with `git checkout -b <new_branch_name>`

You can fetch updates of a branch / branches in a repository with:

- `git fetch <branch-name>` gets you updates of that branch
- `git fetch --all` tells you what's new: new branch(es) has been created
- `git branch` lists all locally available branches (often useful to run together with `git fetch --all`)

Manage branches

Once your work in a branch is finished and you are ready to integrate your changes to the main branch (to deploy it). You want to merge your branch on the main branch:

- move to main: `git checkout main`
- merge your_branch to main: `git merge <your_branch>`
- this will create new commits *on top of* the main branch. Your branch and commits still exists at `git checkout <your_branch>`
- to update the remote repository with your changes: `git push`

Caution

before merging, always `pull` to get the latest changes on main

If you are interested in other ways to manage your branches and history, look into `rebase` and `cherrypick`.

Delete a branch

- You can delete a branch locally with `git branch -d <branch_name>`
- You can *publish* your deletion (delete the branch on the remote repository as well) with `git push origin --delete <branch_name>`

Recovery

If while working you realize you committed / pushed / lost something accidentally git offers some commands to rescue your code.

Reshape local history

If you didn't yet pushed your changes / commit to the remote repository, you are free to modify them as you wish:

- `git reset --soft HEAD~1` removes your last commit and leaves your changes in the staging area. This is useful if you want to add / remove something from that commit or change the commit message. (**Be careful!**: make sure to use the `--soft` flag)
- `git rebase -i <commit_hash>` lets you modify interactively the commits from your HEAD to the commit the hash references to. This is useful to squash commits together, modify their commit messages or drop them.

When needed (eg. when rebasing), git opens an interactive shell. By default is vi. If you wish to change it, you can configure the behavior with:

```
git config --global core.editor "code --wait"
```

- `--global` or `--local` flags to change the editor everywhere or just for this repository
- the `--wait` flag is needed for window-based editors: it waits until you close the editor to proceed. This is not needed with terminal-based editors.

Recover a lost commit / branch

A useful command in this scenario is `git reflog`. It shows all the recent *places* (hashes) you HEAD has been pointing to: you can checkout to the desired hash, branch off and recover that snapshot.

Reshape remote (shared) history

As a general advice, you should **never** change the history of an already pushed commit / branch. This can cause a lot of troubles to your team members and prevent you to recover specific changes or have a good sense of the rationale behind a change.

In an ideal world, each commit has the right size: not too small, not too big – contains one specific change described by a clear commit message that explains what the commit does and references the issue / bug it implements / solve. Unfortunately we don't live in an ideal world and sometimes the history of a repository is not as clean as we wish.

If you want a clean history on the main branch / on specific feature branches, it's advised to:

- rebase the feature branch before merging on the main branch: many changes appear as a single commit and the remote branch stays untouched (preserving the full history)
- create a new *cleaned* branch to keep working on / to merge, but keeping the old branch for future reference / recovery.

You should decide as a team what you want to prioritise and what approach / rules to follow in this project.

Resolve conflicts

When collaborating on a git repository, more developers may modify the same file / functionality. This divergence creates conflict:

git asks itself: what should I keep between the 2 options I see?

In most cases, git solves this conflicts automatically:

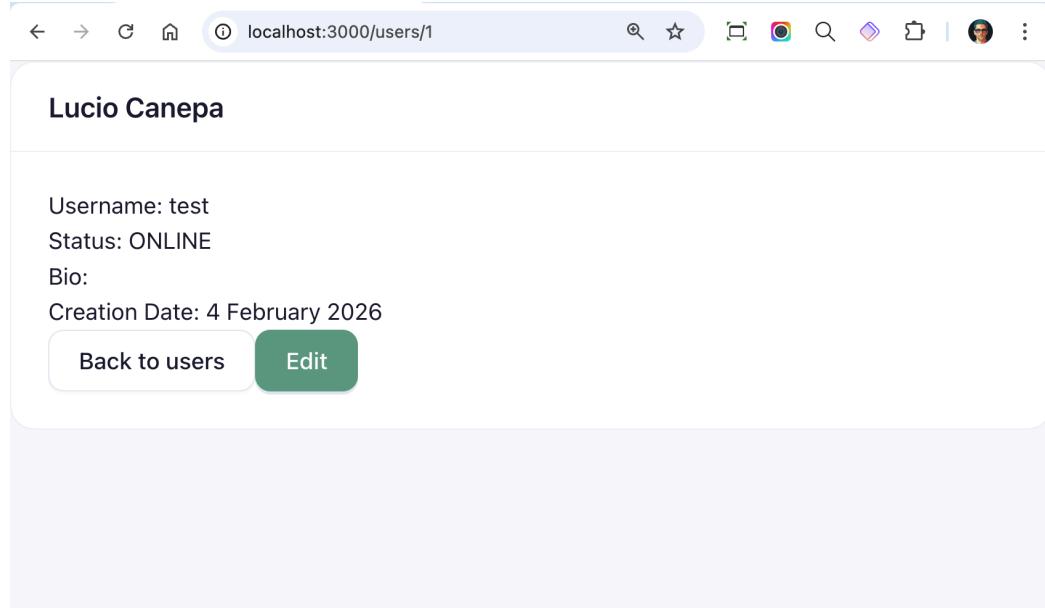
- *e.g.* when pulling, if you set `git config pull.rebase true`, git tries automatically to move your changes on top of the pulled commits).
- when changes are on same file / functionality but not conflicting: can keep both of them

git will prompt you to resolve conflicts only when it wasn't able on its own to decide what to keep and what to ignore. This often happens when you merge a `feature` branch into your main branch. Most code editors (VS code *eg.*) have built-in conflicts resolution tools we suggest to use.

Demonstration

Merge without conflicts

We want to modify how users/[id] looks



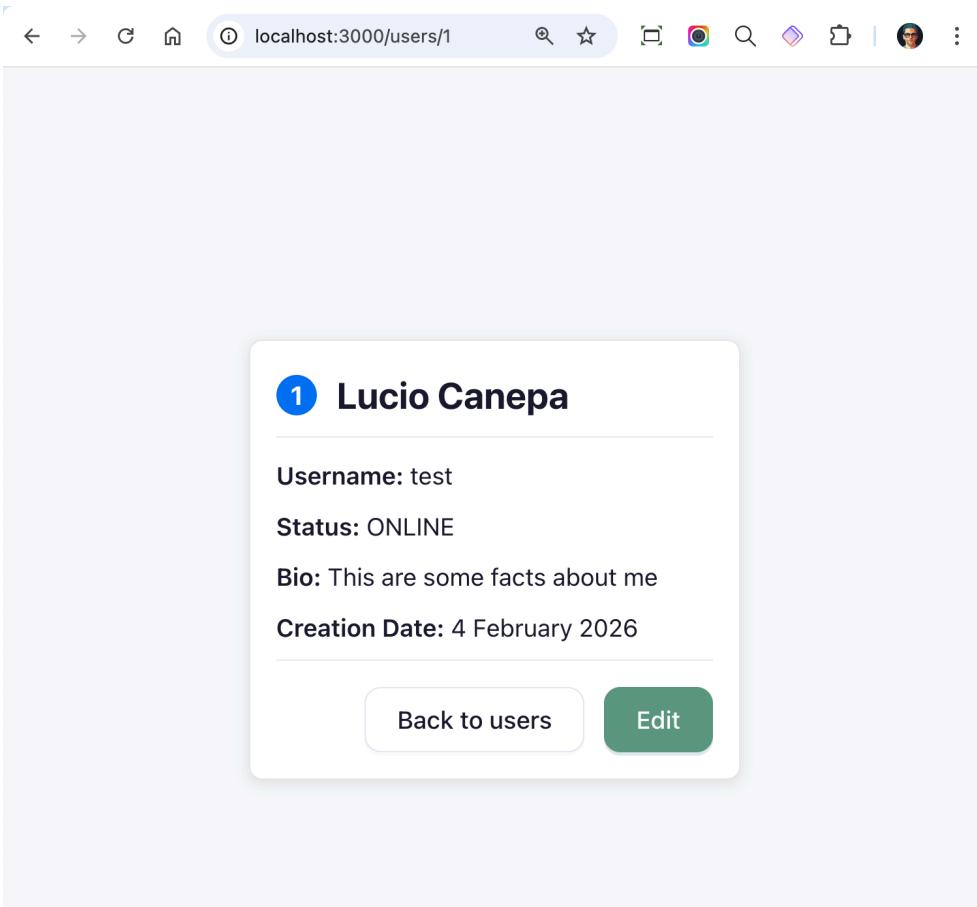
Username: test
Status: ONLINE
Bio:
Creation Date: 4 February 2026

Back to users Edit

For this let's create a branch to work on: #101_restyle_user

```
macvm@Macs-Virtual-Machine sopra_client_tutorial % git checkout -b "#101_restyle_user"
Switched to a new branch '#101_restyle_user'
macvm@Macs-Virtual-Machine sopra_client_tutorial % git status
On branch #101_restyle_user
nothing to commit, working tree clean
macvm@Macs-Virtual-Machine sopra_client_tutorial %
```

There we modify the code and get:



2 new files and 1 modified - we can commit and push (on
#101_restyle_user)

```
macvm@Macs-Virtual-Machine sopra_client_tutorial % git status
On branch #101_restyle_user
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   app/users/[id]/page.tsx

Untracked files:
  (use "git add <file>..." to include in what will be committed)
    app/components/CustomCardComponent.module.scss
    app/components/CustomCardComponent.tsx

no changes added to commit (use "git add" and/or "git commit -a")
macvm@Macs-Virtual-Machine sopra_client_tutorial % git add .
macvm@Macs-Virtual-Machine sopra_client_tutorial % git commit -m "new CustomCardComponent - styled users/[id] page"
[#101_restyle_user d90b2e7] new CustomCardComponent - styled users/[id] page
 3 files changed, 137 insertions(+), 20 deletions(-)
create mode 100644 app/components/CustomCardComponent.module.scss
create mode 100644 app/components/CustomCardComponent.tsx
macvm@Macs-Virtual-Machine sopra_client_tutorial % git push -u origin "#101_restyle_user"
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 5 threads
Compressing objects: 100% (9/9), done.
Writing objects: 100% (9/9), 1.71 KiB | 1.71 MiB/s, done.
Total 9 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
remote:
remote: Create a pull request for '#101_restyle_user' on GitHub by visiting:
remote:     https://github.com/luciocanepa/sopra_client_tutorial/pull/new/%23101_restyle_use
r
remote:
To github.com:luciocanepa/sopra_client_tutorial.git
 * [new branch]      #101_restyle_user -> #101_restyle_user
branch '#101_restyle_user' set up to track 'origin/#101_restyle_user'.
macvm@Macs-Virtual-Machine sopra_client_tutorial % |
```

when we checkout to main and merge everything works: git was able to resolve conflicts on its own

```
macvm@Macs-Virtual-Machine sopra_client_tutorial % git checkout main
Switched to branch 'main'
Your branch is up to date with 'origin/main'.
macvm@Macs-Virtual-Machine sopra_client_tutorial % git merge "#101_restyle_user"
Updating 92f92db..d90b2e7
Fast-forward
 app/components/CustomCardComponent.module.scss | 55 ++++++-----+
 app/components/CustomCardComponent.tsx          | 23 ++++++++
 app/users/[id]/page.tsx                      | 79 ++++++-----+-
 3 files changed, 137 insertions(+), 20 deletions(-)
 create mode 100644 app/components/CustomCardComponent.module.scss
 create mode 100644 app/components/CustomCardComponent.tsx
macvm@Macs-Virtual-Machine sopra_client_tutorial %
```

The new commit is added to the history of main

```
commit d90b2e7bf1defeba51966fb059deb7f2f41bf5ed (HEAD -> main, origin/#101_restyle_user, #101_restyle_user)
Author: username <email>
Date:   Wed Feb 4 17:16:26 2026 +0100

    new CustomCardComponent - styled users/[id] page

commit 92f92db27edb30b308d89c9c8e8a65353debe70a (origin/main, origin/HEAD)
Author: luciocanepa <lucio.canepa.6@gmail.com>
Date:   Sun Feb 1 12:03:37 2026 +0100

    Implementation of Bearer token

commit 061fdd9e15b9f24d9d7c05cf2b4e42c279fde28e
Author: luciocanepa <lucio.canepa.6@gmail.com>
Date:   Sat Jan 31 22:03:33 2026 +0100

    Users/id & users/id/edit pages
```

Merge with conflicts

Let's modify the `userHeader()` function on #101_restyle_user

```
41  41 | const userHeader = () => {
42  - |   const nameToDisplay = user?.name ? user?.name : user?.username;
42+ |   // MODIFIED LINE: Logic changed to prioritize Username over Name
43+ |   const nameToDisplay = user?.username || user?.name || "Unknown User";
43  44 |   return (
44  - |     <>
45  - |     <p>{user?.id}</p>
46  - |     <h2>{nameToDisplay}</h2>
47  - |   </>
45+ |   <div className="header-container">
46+ |     {/* MODIFIED LINE: Wrapped ID in a span and changed tag to h3 */}
47+ |     <span>ID: {user?.id}</span>
48+ |     <h3>{nameToDisplay}</h3>
49+ |   </div>
48  50 | );
49  51 | };
```

Commit and push (on #101_restyle_user)

```
macvm@Macs-Virtual-Machine sopra_client_tutorial % git status
On branch #101_restyle_user
Your branch is up to date with 'origin/#101_restyle_user'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
    modified:   app/users/[id]/page.tsx

no changes added to commit (use "git add" and/or "git commit -a")
macvm@Macs-Virtual-Machine sopra_client_tutorial % git add .
macvm@Macs-Virtual-Machine sopra_client_tutorial % git commit -m "new logic - will trigger conflict"
[#101_restyle_user d91475b] new logic - will trigger conflict
  1 file changed, 7 insertions(+), 5 deletions(-)
macvm@Macs-Virtual-Machine sopra_client_tutorial % git push
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 5 threads
Compressing objects: 100% (6/6), done.
Writing objects: 100% (6/6), 735 bytes | 735.00 KiB/s, done.
Total 6 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To github.com:luciocanepe/sopra_client_tutorial.git
  d90b2e7..d91475b #101_restyle_user → #101_restyle_user
macvm@Macs-Virtual-Machine sopra_client_tutorial % |
```

Let's modify the same function `userHeader()` on main in a different way - commit

```
41  41 | const userHeader = () => {
42  - |   const nameToDisplay = user?.name ? user?.name : user?.username;
42+ |   const nameToDisplay = user?.name ? user.name : "No Name";
43  43 |   return (
44  44 |     <>
45  45 |     <p>{user?.id}</p>
46  - |     <h2>{nameToDisplay}</h2>
46+ |     <h1>{nameToDisplay}</h1>
47  47 |   </>
48  48 | );
49  49 | };
```

Try to merge #101_restyle_user on main → conflict

```
macvm@Macs-Virtual-Machine sopra_client_tutorial % git merge "#101_restyle_user"
Auto-merging app/users/[id]/page.tsx
CONFLICT (content): Merge conflict in app/users/[id]/page.tsx
Automatic merge failed; fix conflicts and then commit the result.
macvm@Macs-Virtual-Machine sopra_client_tutorial %
```

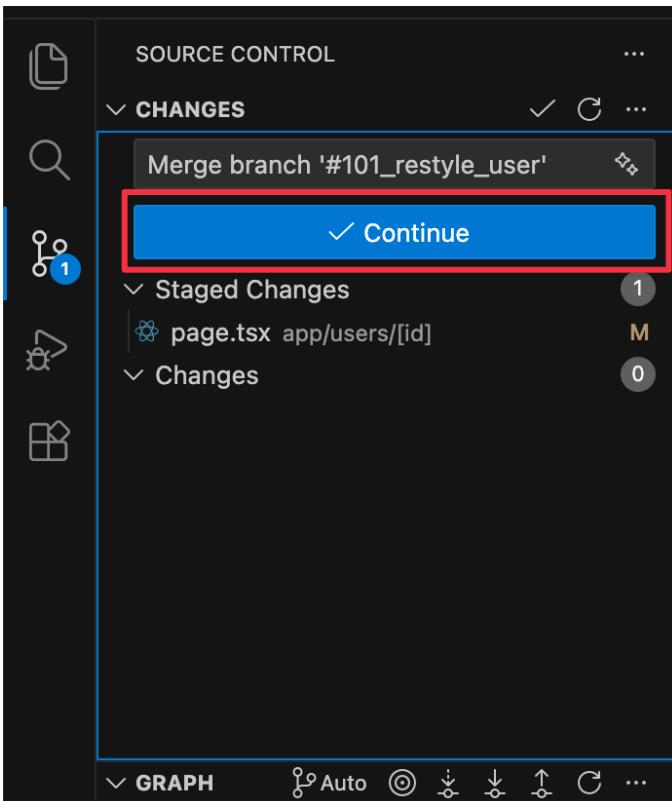
In VS code (under the git tab) you can see merge conflicts - click on **Resolve** in Merge Editor

The screenshot shows the VS Code interface with the Source Control tab selected. In the 'Changes' section, a merge conflict for 'page.tsx' is highlighted. The 'Merge Changes' section shows a 'Continue' button and a 'Resolve' button. The main code editor shows a conflict in line 43. The status bar at the bottom indicates 'Ln 46, Col 26'. A red box highlights the 'Resolve in Merge Editor' button.

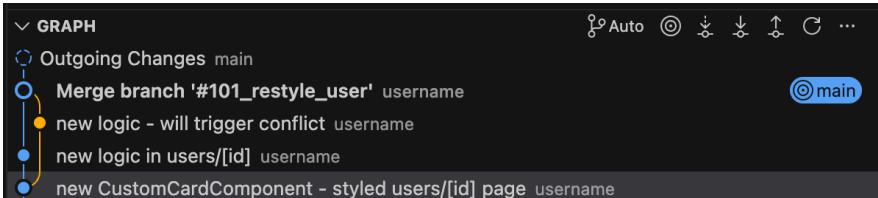
Here you can choose what to keep and what to discard (or manually update the file to a new resolution) - click **Complete Merge** when you are done (0 conflicts)

The screenshot shows the VS Code interface after the conflict has been resolved. The 'Merge Changes' section now displays '0 Conflicts Remaining'. The main code editor shows the merged code. The status bar at the bottom indicates 'Ln 1, Col 1'. A red box highlights the 'Complete Merge' button.

You can click **continue** to proceed with the merge with conflicts resolved



You can see the updated graph after merging and resolving conflicts



Best practices

Commits

- keep your commits small / reasonable size: each commit should implement and be responsible of a specific change (add a feature, solve a bug)
- give a title (short description) to your commit messages, then explain what you did and why. If possible, reference to an issue / bug number the commit is about.

Branches

- name your branches in a clear way: decide with your team what is the criteria for naming branches:
 - camelCase, snake_case, etc.
 - just issue number / issue number + name
 - naming convention when a feature branch is checked out

General

- always pull before you start working (fetch as well)
- always pull before you push

4.2 Github workflows

The SOPRA project relies on **CI/CD** (Continuous Integration / Continuous Deployment) to deploy (publish) the changes from the repositories hosted on Github to:

- Vercel for the client
- Google Cloud for the server

Instead of manually testing and deploying your code, **Workflows** are defined using GitHub Actions. These workflows are defined in YAML files located in `.github/workflows/` inside your repository. They define when they should run and allow granular and automated control on the deployment.

Environment Variables & Secrets

Applications require configuration that changes depending on where the code is running (e.g., a database URL that is `localhost` on your laptop but a Google Cloud URL in production) or sensitive data (passwords, API keys).

Locally (`.env`): On your machine, you store these in a `.env` file. The repository already includes a `.gitignore` that prevents this file from being uploaded to GitHub.

Security Risk

Never commit your `.env` file or hardcode API keys/passwords in your source code. If you push a key to a public repository, it is compromised immediately.

Remotely (GitHub Secrets): To let GitHub Actions access these values (to login to Google Cloud or Vercel), you must store them in the repository settings under `Settings > Secrets and variables > Actions`.

Workflow example (client)

Workflow files are usually intuitive enough to read and understand. If we take the client deployment to Vercel as an example (`.github/workflows/verceldeployment.yml`):

```

name: Deploy to Vercel

on:
  push:
    branches:
      - main

jobs:
  deploy:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout Repo
        uses: actions/checkout@v3

      - name: Setup Node.js
        uses: actions/setup-node@v3
        with:
          node-version: "18"

      - name: Install Dependencies
        run: npm install

      - name: Deploy to Vercel
        run: npx vercel --prod --token ${{ secrets.VERCEL_TOKEN }} --confirm
        env:
          VERCEL_ORG_ID: ${{ secrets.VERCEL_ORG_ID }}
        }
        VERCEL_PROJECT_ID: ${{ secrets.VERCEL_PROJECT_ID }}
        }

```

When new code is pushed to the `main` branch, 1 jobs is runned:

- defines the type of job (`deploy`) and the environment (latest Ubuntu)
- defined the steps: first it install needed resources
- then – in this case – reaches out to Vercel to deploy the frontend application
- to do so it needs environment variables (`secrets`) to connect and authenticate to Vercel.
These secrets are defined in the repository settings.

When deploying you need to make sure that the Environment variables are set correctly in the repository settings in Github. You can add / modify the content of these files to control their behavior.
eg. you might want to deploy a staging environment (`develop` branch) and test your changes in a production-like environment without making them accessible in production (`main`).

5 Set-up Google Cloud and Vercel & deploy your project

To make the application available, you need to deploy your changes:

Server repository

`main.yml` file runs 2 jobs when pushed on `main`:

- test (optional for now): runs the Jtest on the packaged version and make the results available at SonarQube
- deploy: deploys the application to the Google Cloud instance set-up as described below.

`dockerize.yml` creates a docker container and register it. This is optional for now, but will be needed later: a dedicated section explains what docker is and how to set it up.

Client repository

`verceldeployment.yml` file deploys the frontend code to vercel when pushed on `main`

`dockerize.yml` does exactly the same, but for the client (optional for now).

`sonarcloud.yml` runs an analysis (scan) of the code and publish the results to SonarQube (optional for now)

5.1 Set-up Google Cloud and link it to the server repository

To set up the backend infrastructure on Google Cloud, the following services and entities must be successfully configured and enabled (as illustrated in the step-by-step tutorials):

Core Infrastructure

- GCP Project: A dedicated project container named `sopra-fs26-[lastname]-[firstname]-server`.
- App Engine Application: The specific environment where the Java/Spring Boot server resides, localized in the `europe-west6` region.

Identity & Access Management (IAM)

- Service Account: A virtual identity used by GitHub Actions to interact with your Google Cloud resources.
- Editor Role: The specific permission level granted to the Service Account to allow resource modification.
- Service Account Key: A generated JSON key that serves as the “password” for GitHub to log into your Google Cloud project.

Cloud APIs (Must be “Enabled”)

- Cloud Build API: Handles the building of your application code into a deployable format.
- App Engine Admin API: Allows external tools (like GitHub Actions) to manage and deploy versions of your App Engine service.

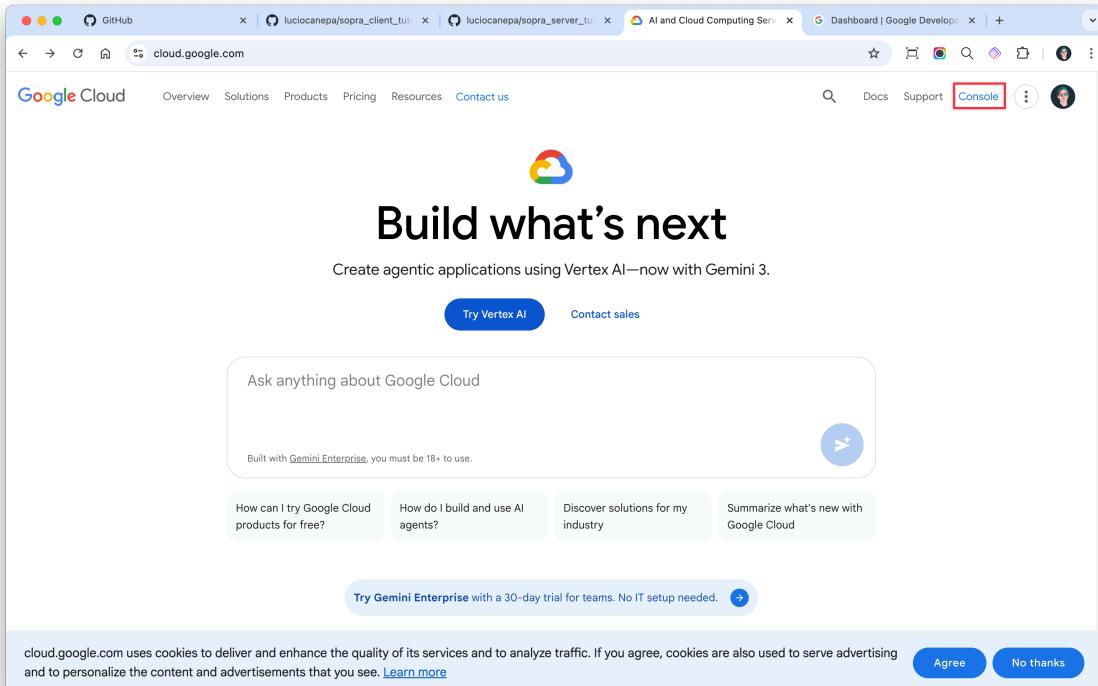
Secrets Management

- GCP_SERVICE_CREDENTIALS: The GitHub Actions secret containing the JSON key, which bridges the link between your code repository and the Google Cloud entities listed above.

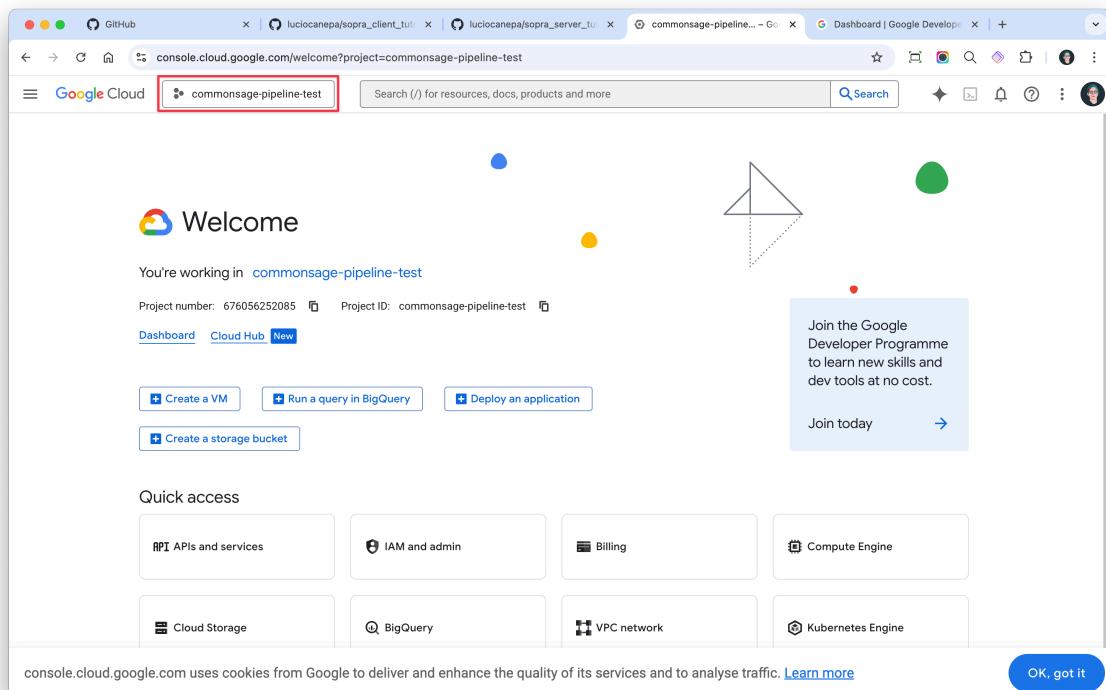
Set-up Google Cloud

Navigate to cloud.google.com and create an account / login

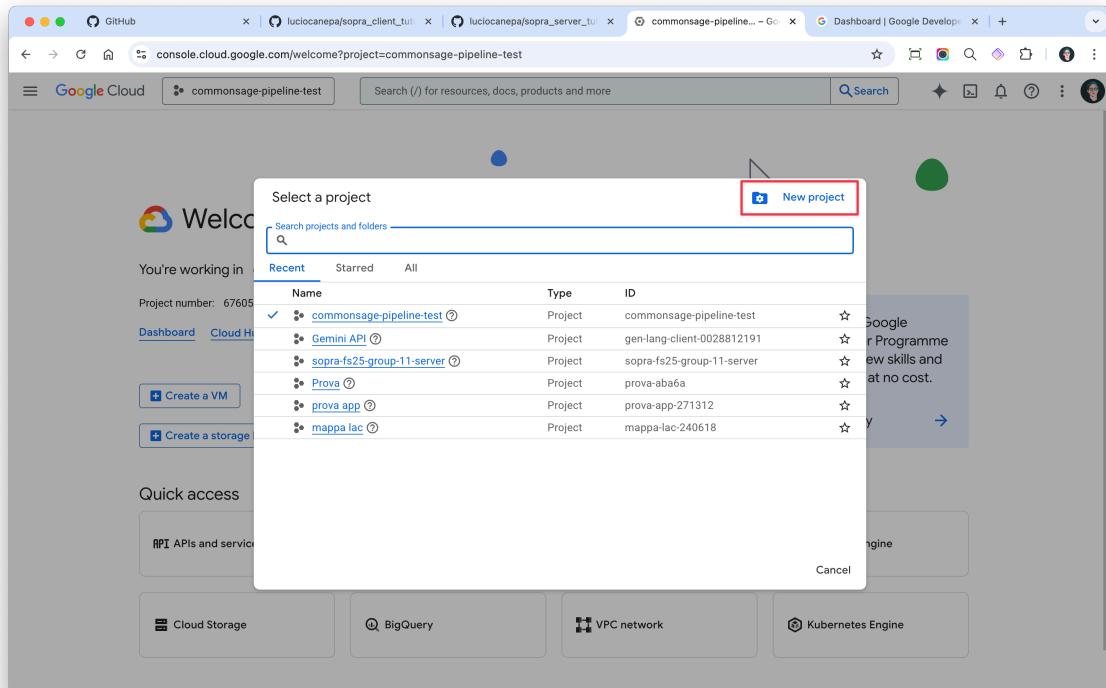
Click on console



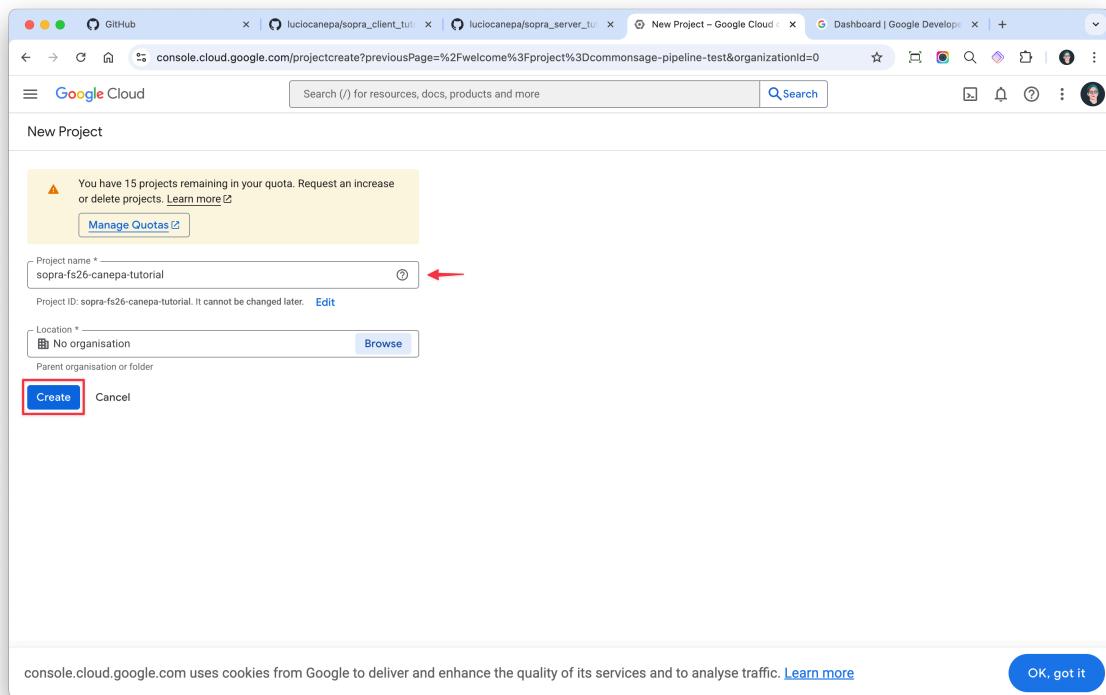
Click on the project selection



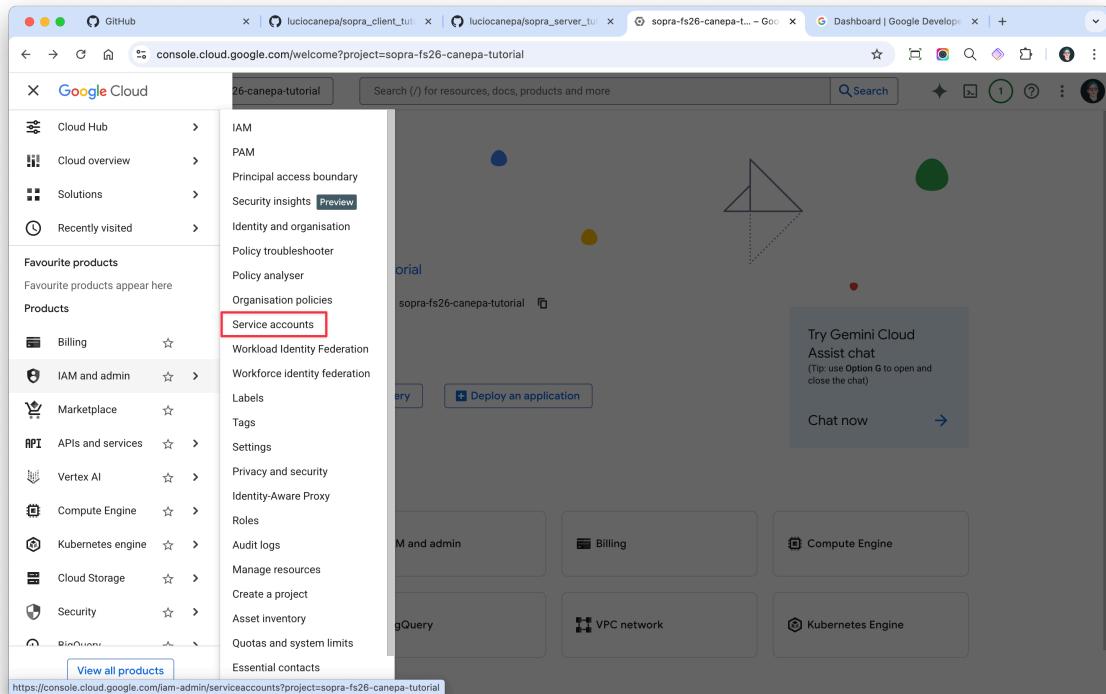
Create new project



Name it according to specification:
sopra-fs26-lastname-firstname-server & click create



Navigate to IAM and admin < Service accounts



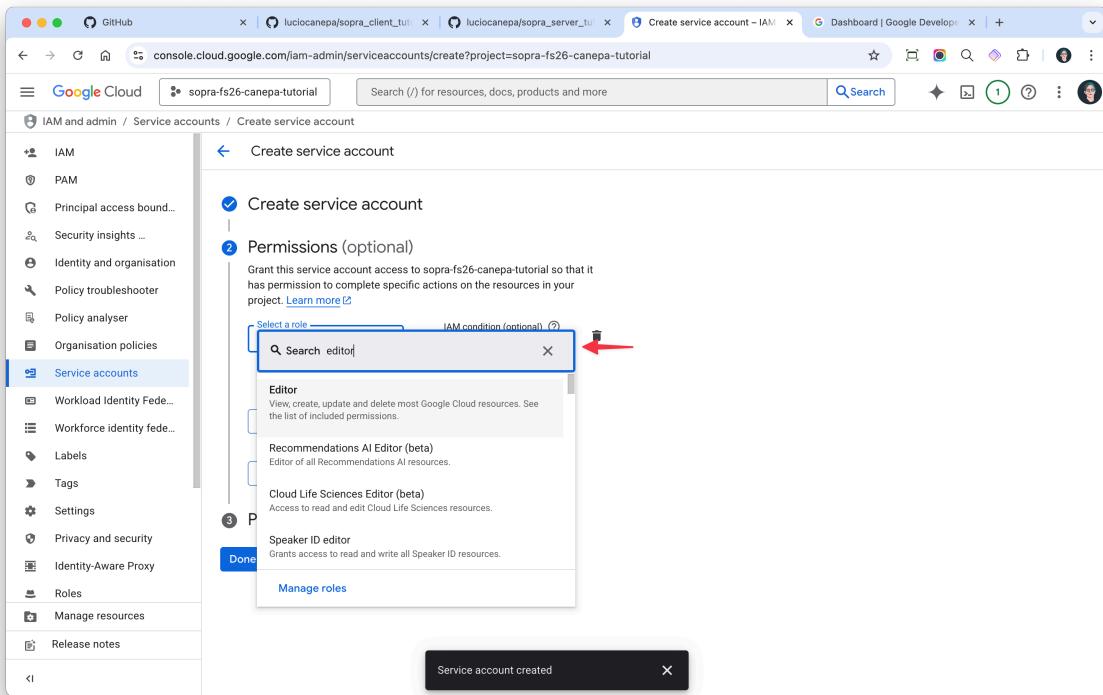
Create a new service account

The screenshot shows the Google Cloud IAM and admin / Service accounts page. The left sidebar is visible with various options like IAM, PAM, Principal access bound..., Security insights..., Identity and organisation, Policy troubleshooter, Policy analyser, Organisation policies, and Service accounts (which is selected and highlighted in blue). The main area shows a table titled 'Service accounts for project 'sopra-fs26-canepa-tutorial''. The table has columns: Email, Status, Name, Description, Key ID, Key creation date, OAuth 2 client ID, and Actions. A red box highlights the '+ Create service account' button at the top right of the table header.

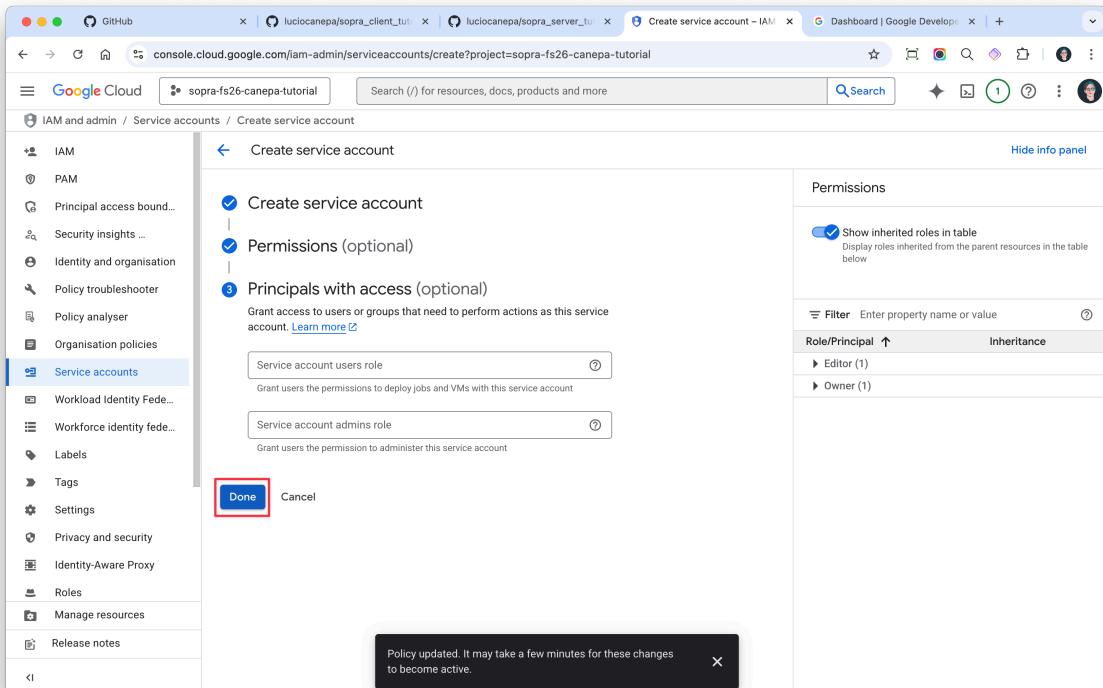
Name it as you wish (it will grant editor access) & click Create and continue

The screenshot shows the 'Create service account' dialog. The left sidebar is identical to the previous screenshot. The main form has a step-by-step guide: 1. Create service account (with a 'Service account name' field containing 'Editor' and a 'Display name for this service account' field below it), 2. Permissions (optional), and 3. Principals with access (optional). The 'Service account name' field is highlighted with a red box. The 'Create and continue' button at the bottom of step 1 is also highlighted with a red box.

add editor role in step 2 - permission (you can search for it) & click Continue



Click done



Next to the created service account, select more > Manage keys

The screenshot shows the Google Cloud IAM and admin Service accounts page. A context menu is open over a service account named "editor@sopra-fs26-canepa-tutorial.iam.gserviceaccount.com". The menu options are:

- Manage details
- Manage permissions
- Manage keys** (highlighted with a red box)
- View metrics
- View logs
- Disable
- Delete

A message at the bottom of the page says "No change – principal already exists on the policy".

Add a new key

The screenshot shows the Google Cloud IAM and admin Service account Keys page for the "Editor" service account. The "Keys" tab is selected. A button labeled "Create new key" is highlighted with a red box.

Other tabs visible include Details, Permissions, Metrics, Logs, and Principals with access.

A message at the bottom of the page says "No change – principal already exists on the policy".

in JSON format (default)

The screenshot shows the Google Cloud IAM and admin service account keys editor. A modal window titled 'Create private key for 'Editor'' is open. It contains instructions about the security risk of downloading private keys and two options for key type: 'JSON' (selected) and 'P12'. A red arrow points to the 'JSON' radio button. Below the modal, a message says 'No change – principal already exists on the policy'.

Key creation confirmation screen - the key as JSON file is saved on your computer

The screenshot shows the Google Cloud IAM and admin service account keys editor. A modal window titled 'Private key saved to your computer' displays a download link for a JSON file named 'sopra-fs26-canepa-tutorial-bf75058add46.json'. Above the modal, a notification bubble shows the same file path and size (2,384 B - Done).

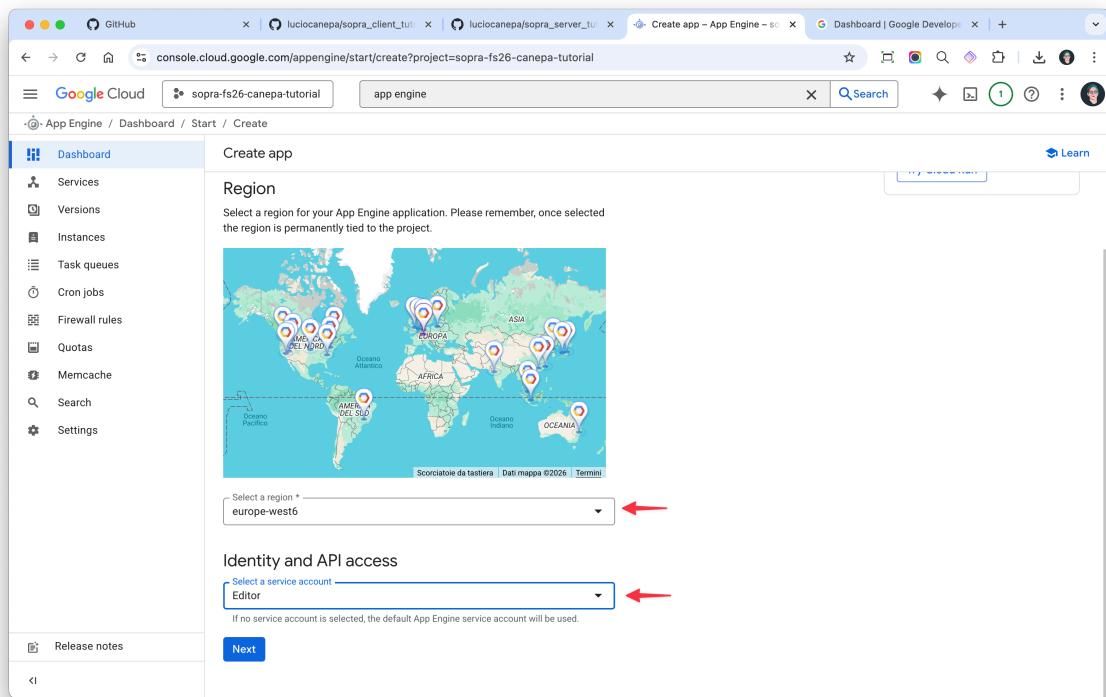
Use top bar to search for App Engine

The screenshot shows the Google Cloud IAM and Admin Service Accounts interface. A search bar at the top contains the query 'app engine'. Below the search bar, a 'Top results' section is displayed, with 'App Engine' being the first item and highlighted with a red box. Other results include 'Compute Engine', 'Cloud Run', and several entries for 'App Engine' under 'Products and pages' and 'Documentation and tutorials'.

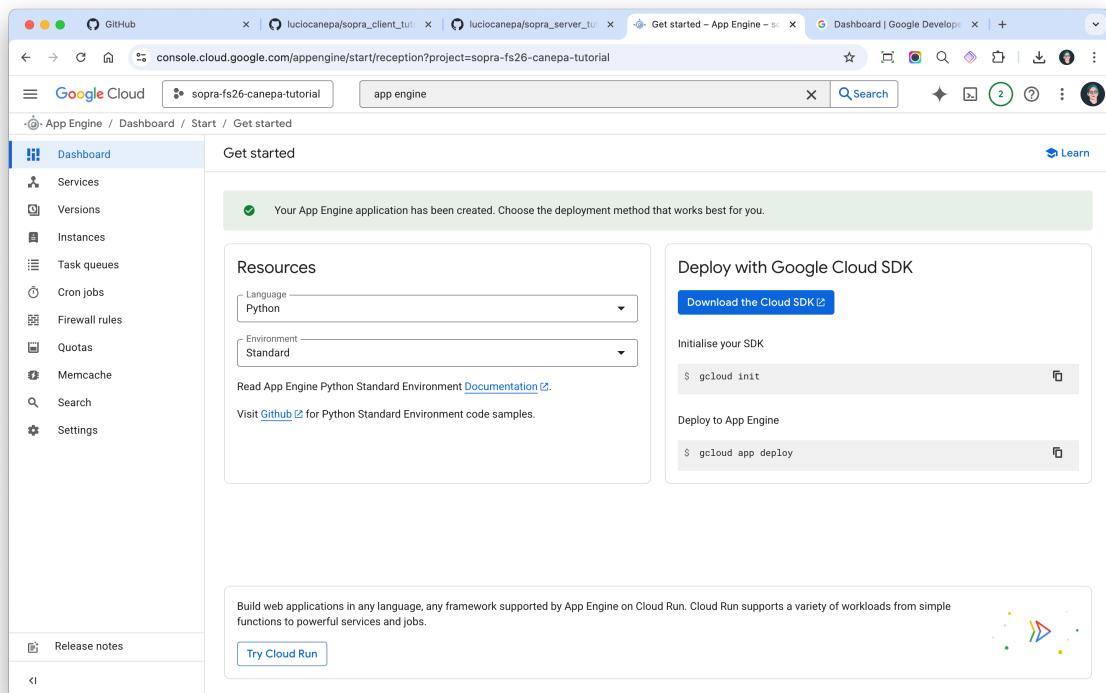
Create new application

The screenshot shows the App Engine dashboard. On the left, a sidebar menu includes options like Dashboard, Services, Versions, Instances, Task queues, Cron jobs, Firewall rules, Quotas, Memcache, Search, and Settings. The main area features a large 'Welcome to App Engine' message with the text 'Build scalable apps in any language on Google's infrastructure' and a prominent 'Create application' button, which is highlighted with a red box. Below this, there is information about Cloud Run and a 'Try Cloud Run' button.

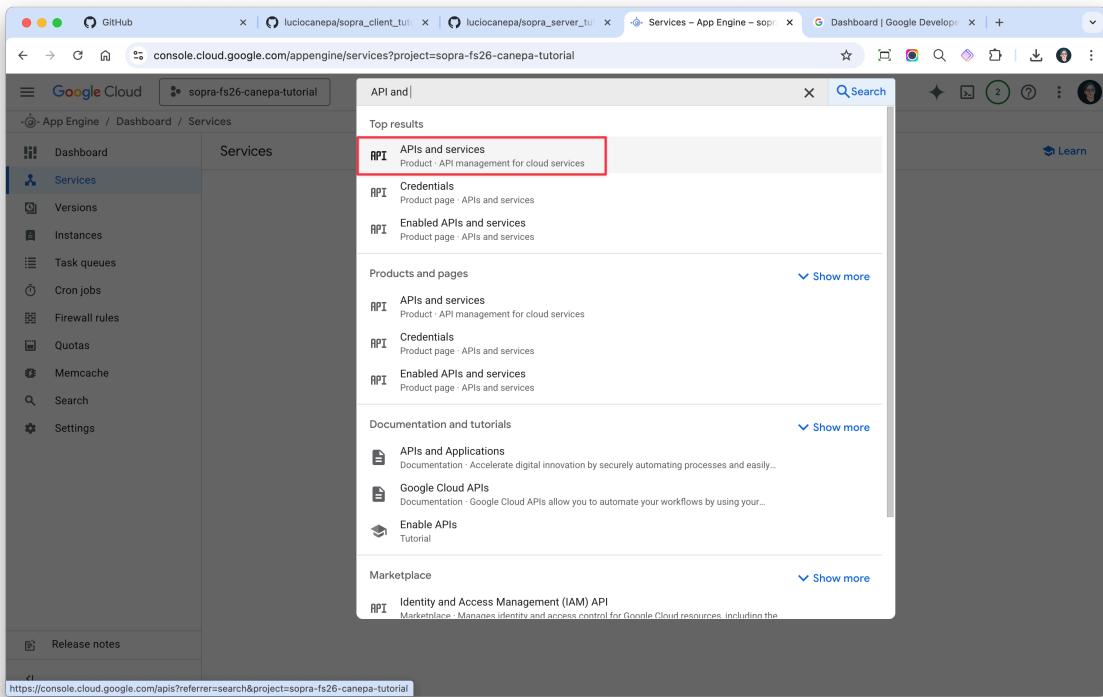
choose time zone europe-west6 and connect the editor service account created before



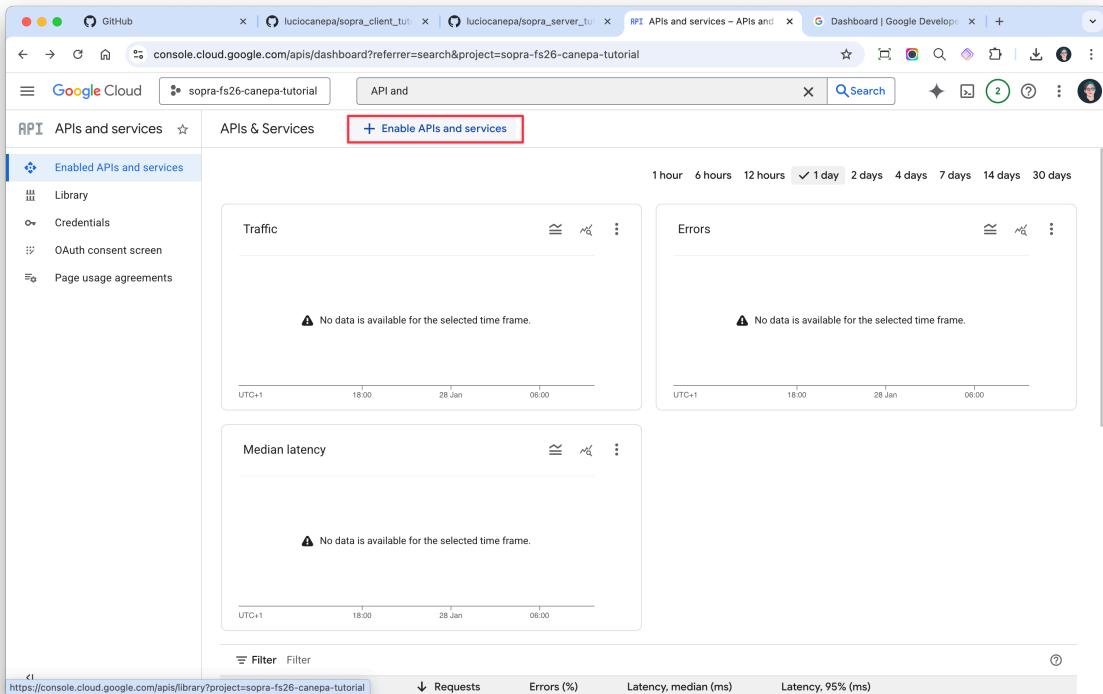
Confirmation screen



Use the top bar to search for APIs and services



Click + Enable APIs and services



Search for Cloud Build API

The screenshot shows the Google API Library interface. A search bar at the top contains the query "cloud build api". Below the search bar, a list of results is displayed under the heading "Maps". The results include:

- Maps SDK for Android
- Maps SDK for iOS
- Maps JavaScript API
- Places API

On the left side, there are filters for "Visibility" (Public: 502, Private: 2) and "Category" (Analytics: 11, Big data: 22, Databases: 6, Machine learning: 16). At the bottom right, there are buttons for "View all (32)" and "View all (16)".

Click on Cloud Build API

The screenshot shows the Google API Library interface with the search results for "cloud build api". The first result, "Cloud Build API", is highlighted with a red box. The details for the Cloud Build API are shown in a box:

Cloud Build API
Google Enterprise API ⓘ
Cloud Build, Google Cloud's continuous integration (CI) and continuous delivery (CD) platform, lets you build software quickly across all languages. Get complete control over defining custom workflows for building, testing, and deploying across multiple environments such as VMs, serverless, Kubernetes, or Firebase.

Below this, other API results are listed:

- Ad Exchange Buyer API II
- Cloud Location Finder API
- Gemini for Google Cloud

On the left side, there are filters for "Visibility" (Public: 15) and "Category" (Analytics: 1, Big data: 2, Databases: 3, Machine learning: 1, Maps: 1, DevOps: 3, Compute: 1, Advertising: 1, Google Enterprise APIs: 9). At the bottom right, there is a link: "https://console.cloud.google.com/apis/library/cloudbuild.googleapis.com?project=sopra-fs26-canepa-tutorial".

Enable it

The screenshot shows the Google Cloud API library interface. The URL in the address bar is `console.cloud.google.com/apis/library/cloudbuild.googleapis.com?project=sopra-fs26-canepa-tutorial`. The page displays the 'Cloud Build API' product details. Key elements include:

- Cloud Build API**: The main title with a blue hexagonal icon.
- Google Enterprise API**: Below the title.
- Enable**: A prominent red button with white text.
- Try this API**: A blue button with white text.
- Click to enable this API**: A small link below the 'Enable' button.
- Overview**: The active tab, followed by **Documentation** and **Related products**.
- Overview** section: Describes Cloud Build as a continuous integration (CI) and continuous delivery (CD) platform. It mentions building software quickly across all languages and environments like VMs, serverless, Kubernetes, or Firebase. Includes a [Learn more](#) link.
- Additional details** section: Lists the type as **SaaS & APIs**, last update as 30/04/2022, category as **DevOps, Google Enterprise APIs**, and service name as `cloudbuild.googleapis.com`.
- Tutorials and documentation** section: Includes links for [Quickstart](#) and [Documentation](#).

Confirmation screen: Cloud Build API is active

The screenshot shows the Google Cloud API library interface, specifically the 'API and services' section. The URL in the address bar is `console.cloud.google.com/apis/api/cloudbuild.googleapis.com/metrics?project=sopra-fs26-canepa-tutorial`. The page displays the 'API/Service details' for the Cloud Build API. Key elements include:

- API**: The active tab, followed by **APIs and services**.
- Enabled APIs and services**: A sidebar with options like Library, Credentials, OAuth consent screen, and Page usage agreements.
- Cloud Build API**: The main service entry, described as 'Creates and manages builds on Google Cloud Platform'.
- Type**: Public API.
- Status**: Enabled.
- Documentation**: Links to [Quickstart](#) and [Documentation](#).
- Explore**: Links to [Try in API Explorer](#).
- Metrics**: The active tab, showing metrics for the Cloud Build API.
- Quotas and system limits**: A tab for managing quotas.
- Credentials**: A tab for managing credentials.
- Cost**: A tab for managing costs.
- Metrics section**: Includes filters for 'Select graphs' (4 Graphs), time range (1 hour to 30 days), and 'Filters' (Versions v1 and v2, Credentials Editor, Unspecified, Ano..., Methods 68 options selected).
- Traffic by response code**: A chart area showing traffic distribution by response code, with a note: '⚠ No data is available for the selected time frame.'

Back to the API library, search for App Engine Admin API

The screenshot shows the Google Cloud API Library interface. A search bar at the top contains the query "app engine". Below the search bar, a dropdown menu lists three results: "app engine", "app engine admin api", and "app engine flexible environment". A red arrow points from the text "Click on App Engine Admin API" to the "app engine admin api" entry in the dropdown. The main content area displays a grid of API cards under the "Maps" category. On the left, there are filters for "Visibility" (Public: 502, Private: 2) and "Category" (Analytics: 11, Big data: 22, Databases: 6, Machine learning: 16). A "View all (32)" link is located in the top right corner of the grid.

Click on App Engine Admin API

The screenshot shows the Google Cloud API Library interface after selecting the "app engine admin api" from the previous screen. The search bar now shows "app engine admin api". The results grid has one item highlighted with a red border: "App Engine Admin API" by Google Enterprise API. This card includes a circular icon with a left-pointing arrow, the API name, its provider, and a brief description: "Provisions and manages developers' App Engine applications." The left sidebar remains the same as in the previous screenshot, showing visibility and category filters.

Enable it

The screenshot shows the Google Cloud API library interface. The main page displays the "App Engine Admin API" under the "Google Enterprise API" category. It provides a brief description: "Provisions and manages developers' App Engine applications." Below this are two prominent buttons: "Enable" (highlighted with a red box) and "Try this API". A note below the buttons says "Click to enable this API". Below these buttons are three navigation links: "Overview" (underlined), "Documentation", and "Related products".

Overview

Provisions and manages developers' App Engine applications.

Additional details

Type: SaaS & APIs
Last product update: 22/07/2022
Category: Compute, Google Enterprise APIs
Service name: appengine.googleapis.com

Tutorials and documentation

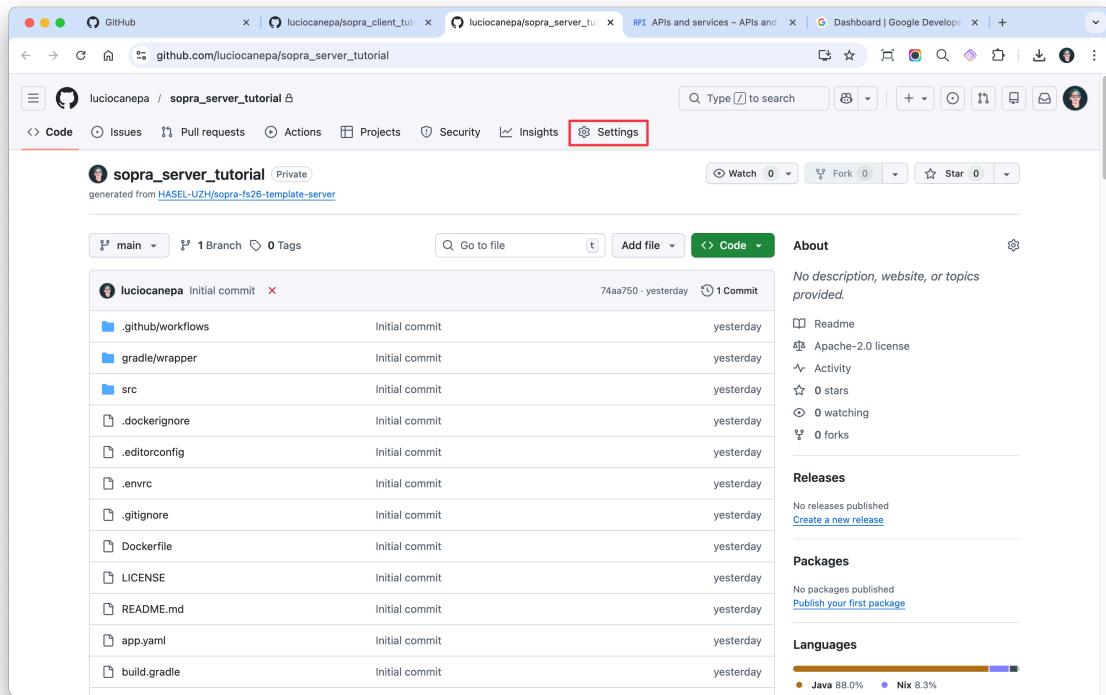
[Learn more](#)

Terms of Service

Confirmation screen: App Engine Admin API is active

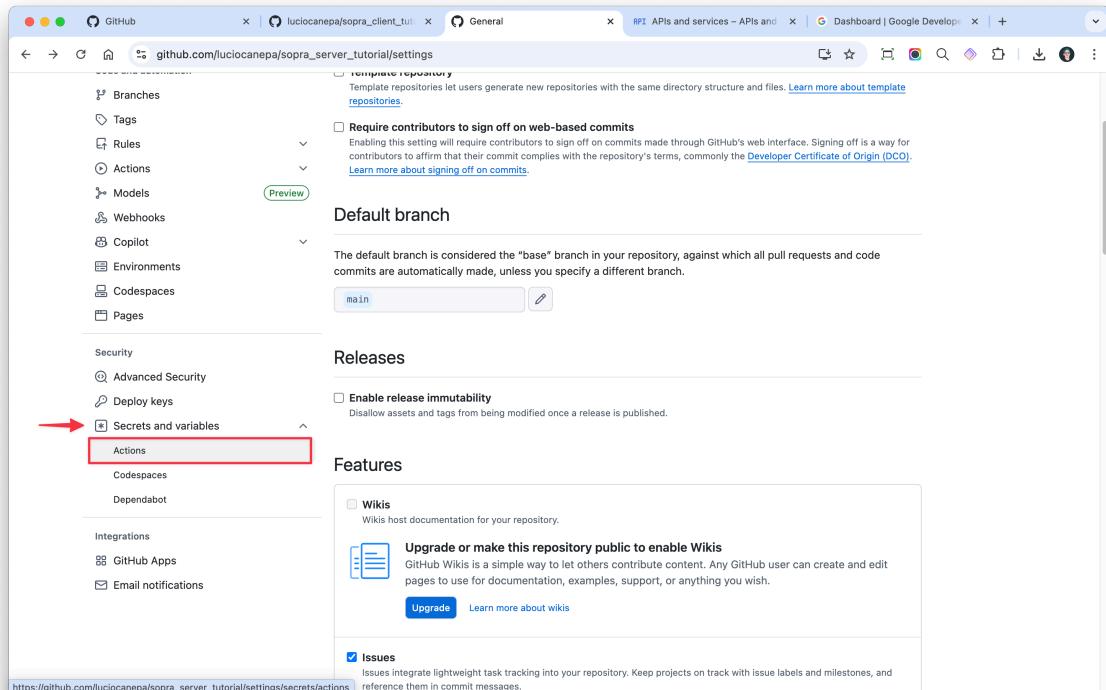
The screenshot shows the Google Cloud API library interface, specifically the "APIs and services" section. On the left, there's a sidebar with "Enabled APIs and services" listed: Library, Credentials, OAuth consent screen, and Page usage agreements. The main area displays the "App Engine Admin API" details. It includes a note: "To call this API from your own applications, you may need to create credentials." A "Create credentials" button is available. The API details show: Service name - appengine.googleapis.com, Type - Public API, Status - Enabled. There are "Documentation" and "Explore" links, along with "Try in API Explorer". Below this, there are tabs for "Metrics", "Quotas and system limits", and "Credentials". The "Metrics" tab is selected, showing a graph interface with filters for "Select graphs" (4 Graphs), "Filters" (Versions: v1, v1alpha and v1beta, Credentials: Editor, Unspecified, Ano..., Methods: 88 options selected), and a time range (1 hour, 6 hours, 12 hours, 1 day, 2 days, 4 days, 7 days, 14 days, 30 days). A message at the bottom states: "No data is available for the selected time frame."

Go to the server repository on Github & go to the Settings



A screenshot of a GitHub repository settings page. The repository is named 'sopra_server_tutorial' and is private. The 'Settings' tab is selected. The left sidebar shows repository settings categories: General, Branches, Tags, Rules, Actions, Models, Webhooks, Copilot, Environments, Codespaces, Pages, Security, Advanced Security, Deploy keys, Secrets and variables (highlighted with a red arrow), Actions, Codespaces, Dependabot, Integrations, GitHub Apps, and Email notifications. The main content area displays repository details like commit history, file structure, and statistics. A red box highlights the 'Actions' tab in the top navigation bar.

Look for Secrets and variables > Actions



A screenshot of the 'Actions' section within the GitHub repository settings. The 'Actions' tab is selected. The page includes sections for 'Template repository', 'Default branch' (set to 'main'), 'Releases' (with an 'Enable release immutability' checkbox), and 'Features'. Under 'Features', there are options for 'Wikis' (unchecked) and 'Issues' (checked). A red arrow points to the 'Actions' tab in the sidebar.

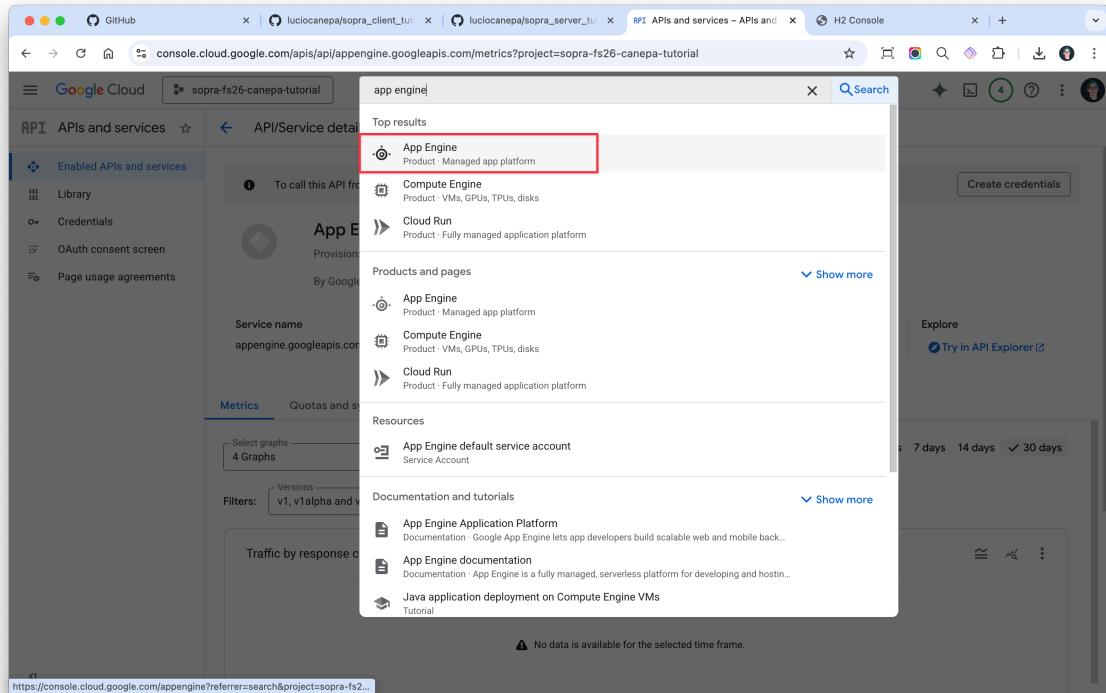
Add a New repository secret

The screenshot shows the GitHub Actions secrets settings page for the repository `sopra_server_tutorial`. The left sidebar lists various repository settings like General, Access, Collaborators, etc. The main area is titled "Actions secrets and variables". It has two sections: "Environment secrets" and "Repository secrets". Both sections show a message stating "This environment has no secrets." and "This repository has no secrets.". In the "Repository secrets" section, there is a red box around the "New repository secret" button.

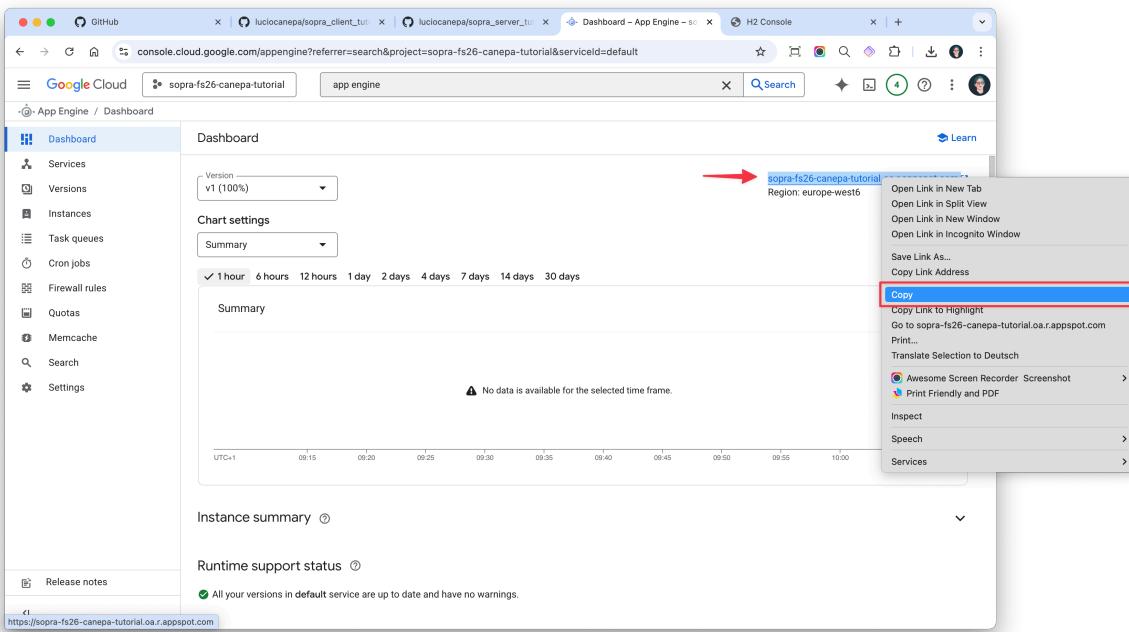
Name it `GCP_SERVICE_CREDENTIALS` and paste as value the content of the JSON file you download before from Google Cloud console

The screenshot shows the "New secret" creation page on GitHub. The "Name" field is filled with `GCP_SERVICE_CREDENTIALS`. The "Secret" field contains a large amount of binary or JSON data, which is heavily blurred. A red arrow points from the text "Back to Google Cloud console - search for App Engine" to this blurred secret value field. To the right of the browser window, a Mac OS X "Downloads" window is open, showing a file named `sopra-fs26-canepa-tutorial-bf75058add46.json`, which corresponds to the secret value being pasted.

Back to Google Cloud console - search for App Engine



Here you find the link where your backend (server) will be deployed - copy it



Push to deploy & verify changes

Google Cloud console is now setup and the secrets are loaded in Github: the next time you

push to the remote repository, the server application will be automatically deployed (based on the .github/workflows).

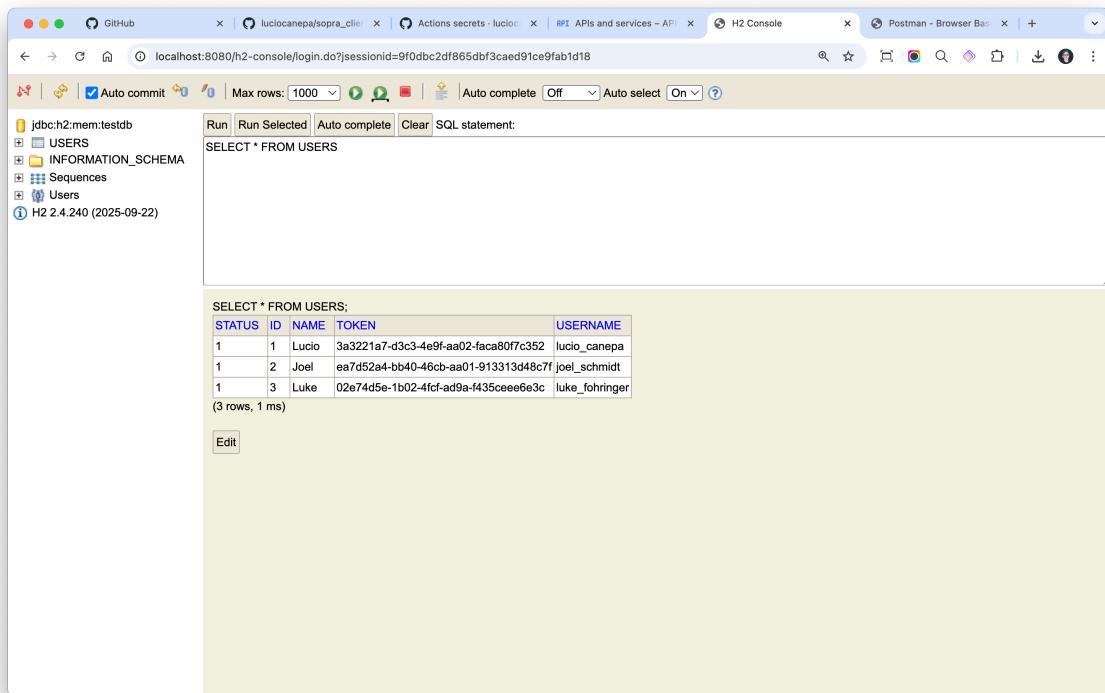
We can test this out by making some changes locally:

Local changes: add GET /users/{id} endpoint

new endpoint in users.controller (I have also modified the corresponding DTO and service files)

```
@GetMapping("/users/{id}")
@ResponseBody
@ResponseStatus(HttpStatus.OK)
public UserGetDTO getUserById(@PathVariable("id") Long id) {
    User user = userService.getUser(id);
    return DTOMapper.INSTANCE.convertEntityToUserGetDTO(user);
}
```

After creating 3 users, I can see the database locally



The screenshot shows the H2 Console interface running in a browser. The database is named 'testdb' and contains three users: Lucio, Joel, and Luke. The table structure is as follows:

STATUS	ID	NAME	TOKEN	USERNAME
1	1	Lucio	3a3221a7-d3c3-4e9f-aa02-faca80f7c352	lucio_canepa
1	2	Joel	ea7d52a4-bb40-46cb-aa01-913313d48c7f	joel_schmidt
1	3	Luke	02e74d5e-1b02-4fcf-ad9a-f435ceee6e3c	luke_fohringer

(3 rows, 1 ms)

Test locally my new endpoint

The screenshot shows the Postman application interface. On the left, the sidebar displays 'My Workspace' with a collection named 'sopra_tutorial' containing a POST request for 'post user' and a GET request for 'New Request'. The main area shows a 'New Request' dialog for a GET request to `{(base_url)}/users/2`. Below it, the 'Body' tab shows a JSON response with a red box highlighting the data:

```
1 {
2   "id": 2,
3   "name": "Joel",
4   "status": "OFFLINE",
5   "username": "joel_schmidt"
6 }
```

Commit and push the changes

```

macvm@Macs-Virtual-Machine sopra_server_tutorial % git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:  src/main/java/ch/uzh/ifi/hase/soprafs26/controller/UserController.java
      modified:  src/main/java/ch/uzh/ifi/hase/soprafs26/service/UserService.java

no changes added to commit (use "git add" and/or "git commit -a")
macvm@Macs-Virtual-Machine sopra_server_tutorial % git add .
macvm@Macs-Virtual-Machine sopra_server_tutorial % git commit -m "added endpoint: GET /users/:id"
[main 17357a6] added endpoint: GET /users/:id
 2 files changed, 13 insertions(+)
macvm@Macs-Virtual-Machine sopra_server_tutorial % git push
Enumerating objects: 27, done.
Counting objects: 100% (27/27), done.
Delta compression using up to 5 threads
Compressing objects: 100% (7/7), done.
Writing objects: 100% (14/14), 1.05 KiB | 1.05 MiB/s, done.
Total 14 (delta 4), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (4/4), completed with 4 local objects.
To github.com:luciocanepa/sopra_server_tutorial.git
 7kaa750...17357a6 main → main
macvm@Macs-Virtual-Machine sopra_server_tutorial %

```

We are interested in this deploy workflow

main · 1 Branch · 0 Tags

Go to file Add file · Code · About

luciocanepa added endpoint: GET /users/:id · 17357a6 · now · 2 Commits

No description, website, or topics provided.

.github/workflows

Some checks haven't completed yet 3 in progress checks

- Deploy Project to App Engine / Deploying to Google Cloud (push)** In progress - This check has started... [Details](#)
- Deploy Project to App Engine / Test and Sonarqube (push)** In progress - This check has started... [Details](#)
- Dockerize / build (push)** In progress - This check has started... [Details](#)

gradle/wrapper

src

.dockerignore

.editorconfig

.envrc Initial commit 2 days ago

.gitignore Initial commit 2 days ago

Readme

Apache-2.0 license

Activity

0 stars

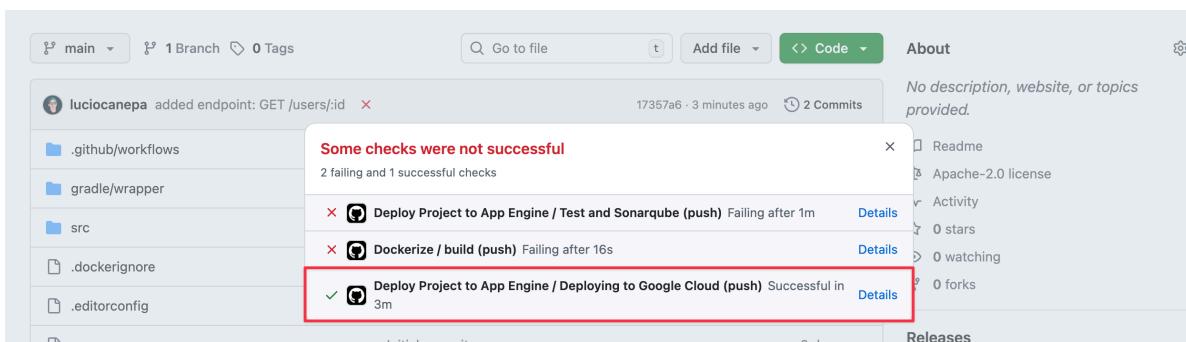
0 watching

0 forks

Releases

No releases published [Create a new release](#)

Confirmation: the Deploy to App Engine workflow succeeded - for now ignore the other 2



Now we can test the new endpoint in production (prod_url is the link we copied before from App Engine) I create 2 users (here you see how to POST a new user)

The screenshot shows a Postman collection named 'sopra_tutorial' with a 'post user' POST request. The URL is {{prod_url}}/users. The request body is a JSON object:

```

1 {
2   "name": "Lucio",
3   "username": "lucio_canepa"
4 }

```

The response is a 201 Created status with a JSON object:

```

1 {
2   "id": 2,
3   "name": "Lucio",
4   "status": "OFFLINE",
5   "username": "lucio_canepa"
6 }

```

from prod_url/h2-console, you can see the tables as you did locally (to access use the same authentication)

The screenshot shows the H2 Console interface. On the left, the database schema is displayed with tables like USERS and INFORMATION_SCHEMA. A SQL statement 'SELECT * FROM USERS' is run, and the results are shown in a table:

	STATUS	ID	NAME	TOKEN	USERNAME
1	1	Luke	a5eb620c-4c57-4b4b-a41e-97df03af8ee9	luke_fohringer	
1	2	Lucio	231c3afe-6324-4c93-a385-3b7ce6c7bd19	lucio_canepa	

(2 rows, 3 ms)

Test the new endpoint in production

The screenshot shows the Postman application interface. A new request is being created for the URL `http://sopra_tutorial`. The method is set to GET, and the URL is `((pred_url))/users/1`. An arrow points to the URL field. The response body is displayed as JSON:

```

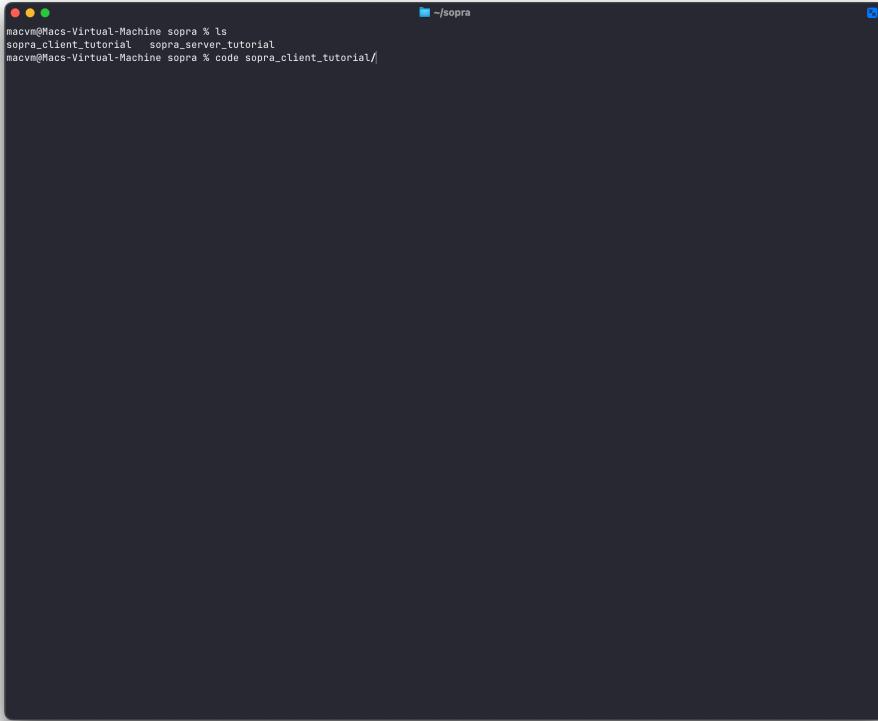
1 {
2   "id": 1,
3   "name": "Luke",
4   "status": "OFFLINE",
5   "username": "luke_fohringer"
6 }

```

Update fallback prodUrl in the client repository

Make sure to have the server production URL copied to the clipboard (you find it at Google Cloud console > App Engine)

Open the client repository in an IDE of your choice

A screenshot of a terminal window titled 'sopra'. The window shows a dark background with white text. At the top, there are three colored window control buttons (red, yellow, green) on the left and a close button on the right. The title bar contains the text 'sopra' and a small blue icon. The terminal output is as follows:

```
macvm@Macs-Virtual-Machine sopra % ls
sopra_clientTutorial  sopra_serverTutorial
macvm@Macs-Virtual-Machine sopra % code sopra_clientTutorial/
```

The terminal window has a thin black border and is centered on the page.

Look for app/utils/domain.ts and replace the hardcoded fallback value to the URL you just copied - you completed the TODO commented out there

```

EXPLORER    page.tsx  TS domain.ts M
SOPRA_CLIENT_TUTORIAL ...
  > .direnv
  > .github
  > .next
  > app
    > api
    > hooks
    > login
    > styles
  > types
  > users
  > utils
    > TS domain.ts M
      TS environment.ts
      TS uid.ts
      ★ favicon.ico
      ✎ layout.tsx
      ✎ page.tsx
      > node_modules
      > public
      < .dockerignore
      $ .envrc
      $ .gitignore
      $ .xcode.env.local
      $ contributions.md
      $ deno.lock
      ✎ Dockerfile
      JS eslint.config.mjs
      $ flake.lock
      $ flake.nix
      $ install-nix.sh
      TS next-env.d.ts
      TS next.config.ts
      (1) package-lock.json
      (1) package.json
    > OUTLINE
    > TIMELINE
  ✎ main* ⌂ 0 △ 0

```

```

app > utils > TS domain.ts > getApIdomain
1 import process from "process";
2 import { isProduction } from "@/utils/environment";
3 /**
4  * Returns the API base URL based on the current environment.
5  * In production it retrieves the URL from NEXT_PUBLIC_PROD_API_URL (or falls back to a hardcoded url).
6  * In development, it returns "http://localhost:8080".
7 */
8 export function getApIdomain(): string {
9   const prodUrl = process.env.NEXT_PUBLIC_PROD_API_URL ||
10     "sopra-fs26-canepa-tutorial.oss.r.appspot.com"; // TODO: update with your production URL as needed.
11   const devUrl = "http://localhost:8080";
12   return isProduction() ? prodUrl : devUrl;
13 }
14

```

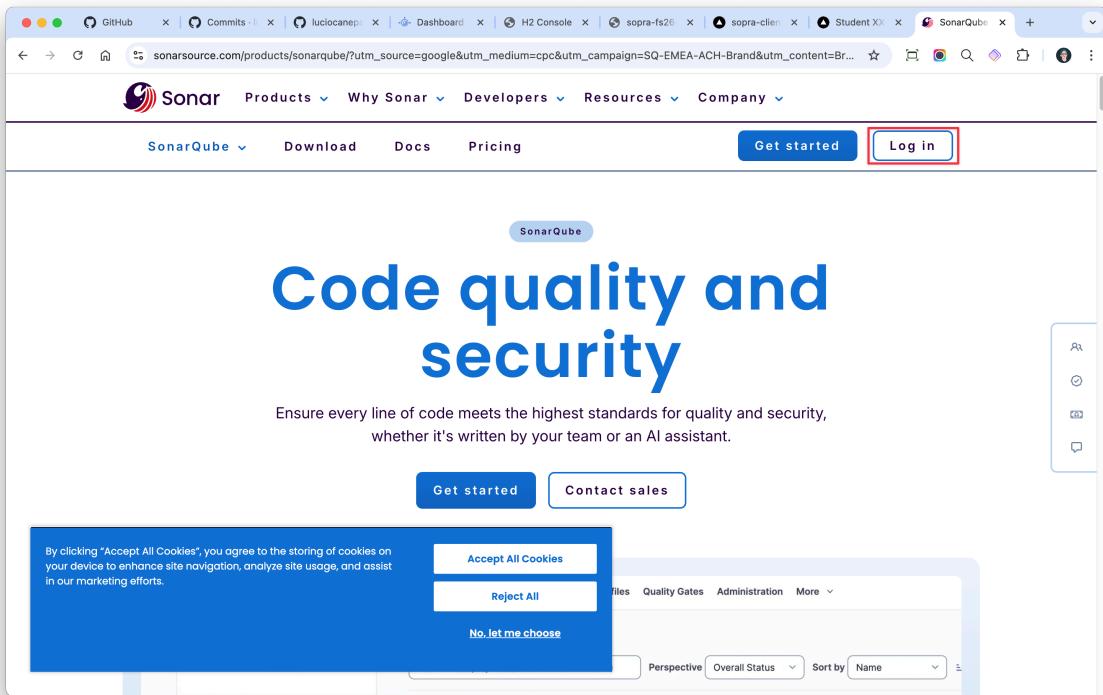
Not Committed Yet Ln 10, Col 104 (52 selected) Spaces: 2 UTF-8 LF () TypeScript

Ideally the server URL in production comes from NEXT_PUBLIC_PROD_URL that we are going to set-up later (in the dedicated Vercel part of this tutorial). This only serves as a fallback if, for some reason, the environment variable set in vercel is not retrieved correctly.

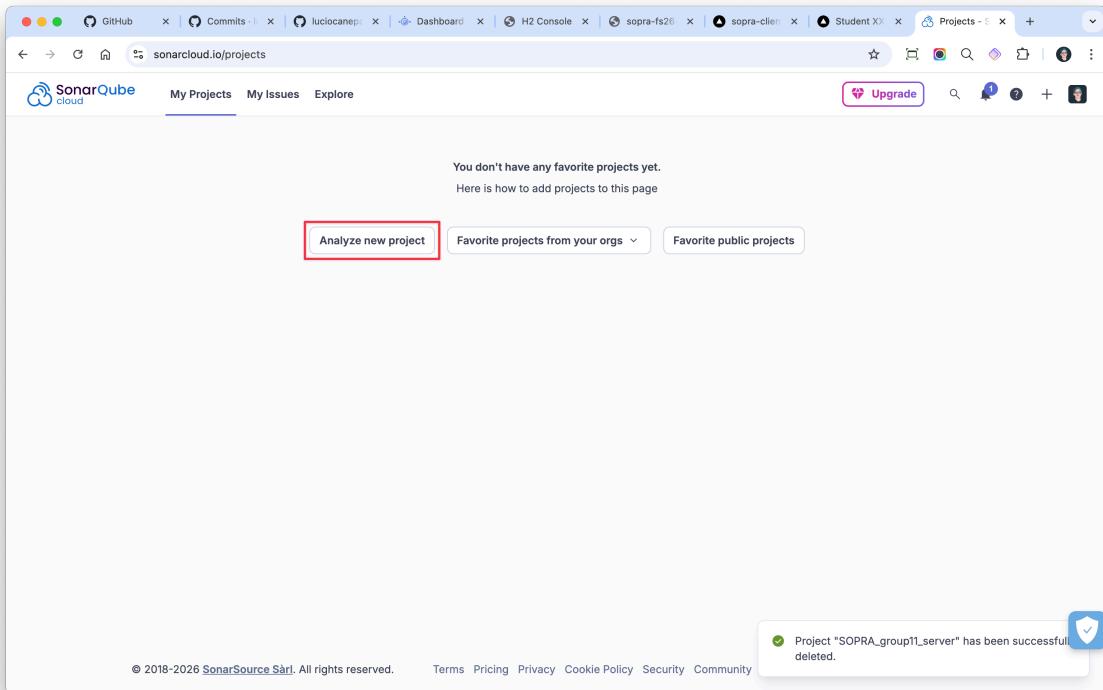
Set-up Sonar cloud and connect it to the sever repository (optional)

Go to <https://www.sonarsource.com/>

Register or login (best if with your Github account)



Click Analyze new project



Choose your organization and link the server repository

The screenshot shows the SonarQube Cloud interface for creating a new project. In the top navigation bar, there are tabs for GitHub, Commits, luciocanepa, Dashboard, H2 Console, sopra-fs2, sopra-client, Student, and Analyze project. The current page is 'sonarcloud.io/projects/create?organization=luciocanepa-1'. The main section is titled 'Analyze projects' and asks to 'Select repositories from one of your GitHub organization.' A dropdown menu labeled 'Organization' shows 'Lucio Canepa' selected. Below it is a link to 'Import another organization'. A search bar says 'Search for repositories' and shows '16 results'. A checkbox 'Select all on this page' is checked. A list of repositories includes 'sopra_client_tutorial' and 'sopra_serverTutorial' (Private), which is highlighted with a red box and has a red arrow pointing to it. To the right, a box says '1 repository selected' and '1 repository will be created as a private project on SonarQube Cloud'. It contains a 'Set Up' button. Another box asks 'Already have a coupon?' with a link to apply it. At the bottom, it says 'Just testing? You can [create a project manually](#). [Setup a monorepo](#)'.

Leave default Previous version toggle & click Create project

The screenshot shows the 'Set up new code for project' page. The title is 'Set up new code for project'. It explains that the new code definition sets criteria for what's considered new code. It allows project administrators to customize the new code definition for their project. A link to 'Learn more in documentation' is provided. A note states that 'Your organization's new code definition is not set' and provides a link to 'Lucio Canepa - Administration - New Code'. The 'New code definition for this project:' section has two options: 'Previous version' (selected) and 'Number of days'. A red arrow points to the 'Previous version' option. Below it is a 'Create project' button, which is highlighted with a red box. A note says 'You can change this at any time in the project administration'. At the bottom, it says '© 2018-2026 SonarSource Sàrl. All rights reserved.' and lists links for Terms, Pricing, Privacy, Cookie Policy, Security, Community, Documentation, Contact us, Status, and About.

Under Administration > Analysis Method turn off the toggle Automatic Analysis (since we are going to push it via CI)

sonarcloud.io/project/analysis_method?id=luicocanepa_sopra_server_tutorial

Analysis Method

Use this page to manage and set-up the way your analyses are performed.

Automatic Analysis ✓ Recommended

Even better analysis and results are available through SonarQube Cloud's CI-based analysis. [Learn more](#)

SonarQube Cloud automatically analyzes your default branch and Pull Requests. [Learn more](#)

Set up analysis via other methods

- With GitHub Actions
- With CircleCI
- With other CI tools

Manually

Use this for testing. Other modes are recommended to help you set up your CI environment.

Modify your `build.gradle` as in the diff (server repository)

```

41 34 dependencies {
42 42     implementation 'org.springframework.boot:spring-boot-h2console'
43 43     developmentOnly 'org.springframework.boot:spring-boot-devtools'
44 44     runtimeOnly 'com.h2database:h2'
45 45     testImplementation 'org.springframework.boot:spring-boot-starter-test'
46 46     testImplementation 'org.springframework.boot:spring-boot-starter-webmvctest'
47 47     testImplementation 'org.springframework.boot:spring-boot-starter-data-jpa-test'
48 48     testRuntimeOnly 'org.junit.platform:junit-platform-launcher'
49 49 }
50 50
51 51
52 52
53 53 bootJar {
54 54     archiveFileName = "${archiveBaseName.get()}${archiveExtension.get()}"
55 55 }
56 56
57 57 sonar {
58 58     properties [
59 59         property "sonar.projectKey", "TODO"
60 60         property "sonar.organization", "TODO"
61 61         property "sonar.projectKey", System.getenv("SONAR_PROJECT_KEY")
62 62         property "sonar.organization", System.getenv("SONAR_ORGANIZATION")
63 63     ]
64 64
65 65 jacocoTestReport {
66 66     reports {
67 67         xml.required = true
68 68     }
69 69 }
70 70
71 71 test {
72 72     useJUnitPlatform()
73 73     testLogging.showStandardStreams = true
74 74     maxParallelForks = 1
75 75 }
76 76
77 77 File secretPropsFile = file('../local.properties')
78 78 if (secretPropsFile.exists()) {
79 79     Properties p = new Properties()
80 80     p.load(new FileInputStream(secretPropsFile))
81 81     p.each { name, value ->
82 82         ext[name] = value

```

Ln 59, Col 1 Spaces: 4 UTF-8 LF () Groovy

Modify your `main.yml` workflow file as in the diff (server repository)

```

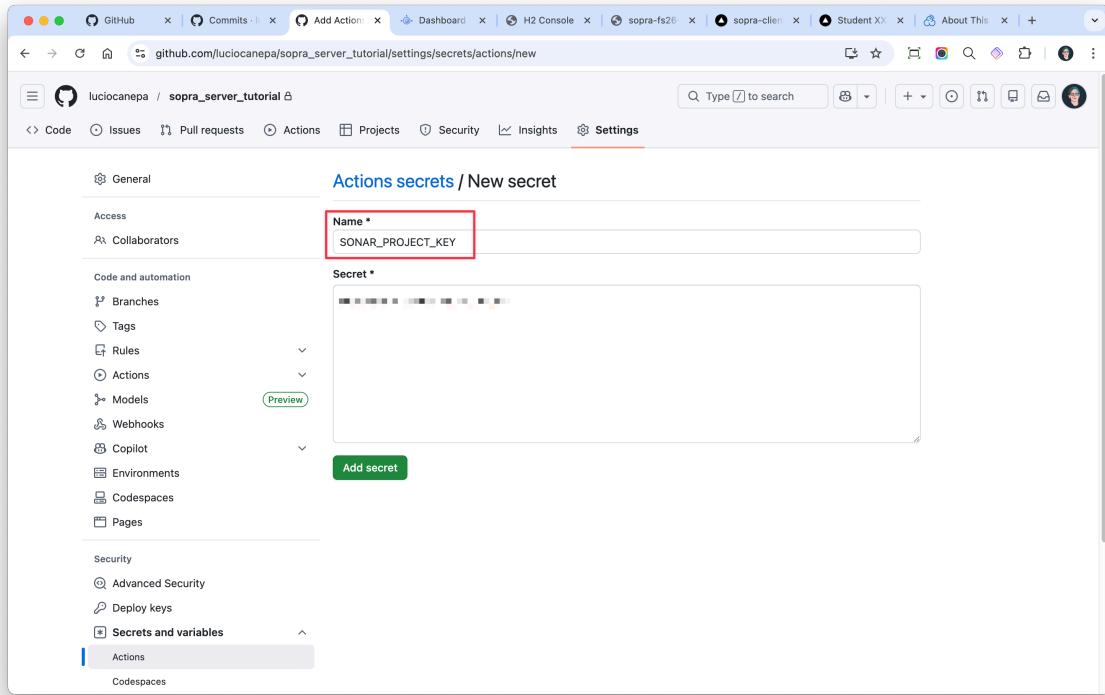
    ...
    - name: Checkout repository code
    - name: Install Java 17
      uses: actions/setup-java@v3
      with:
        distribution: "temurin"
        java-version: "17"
    - name: Make gradlew executable
      run: chmod +x gradlew
    - name: Test and analyze
      run: ./gradlew test jacocoTestReport sonar
      env:
        GITHUB_TOKEN: ${{ secrets.GITHUB_TOKEN }}
        SONAR_TOKEN: ${{ secrets.SONAR_TOKEN }}
        SONAR_PROJECT_KEY: ${{ secrets.SONAR_PROJECT_KEY }}
        SONAR_ORGANIZATION: ${{ secrets.SONAR_ORGANIZATION }}
    - name: Deploy
      name: Deploying to Google Cloud
      runs-on: ubuntu-latest
      # needs: test
    steps:
      - name: Checkout
        uses: actions/checkout@v2
      - name: Deploy to App Engine
        id: deploy
        uses: google-github-actions/deploy-appengine@v0.2.0
        with:
          deliverables: app.yaml
          version: v1
          credentials: ${{ secrets.GCP_SERVICE_CREDENTIALS }}
      - name: Test
        run: curl "${{ steps.deploy.outputs.url }}"

```

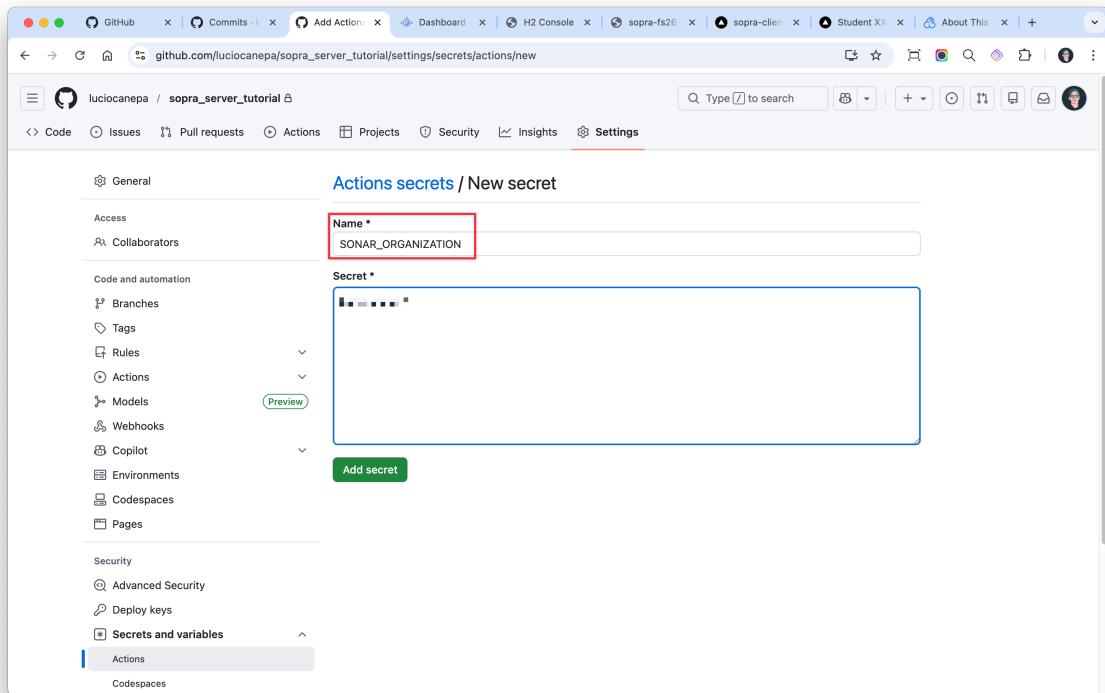
We now need to collect the secrets to connect the server repo to sonarcloud. You find the Project Key and Organization Key under Information - copy them

The screenshot shows the SonarCloud project information page for 'sopra_serverTutorial'. The left sidebar has 'Information' selected. The main area displays the 'Project Key' as 'luciocanepa_sopra_serverTutorial' and the 'Organization Key' as 'luciocanepa-1'. Both keys have a copy icon (a small square with a white 'C') to their right, which is highlighted with a red arrow.

Add a new secret to the server repository: SONAR_PROJECT_KEY (Project Key)



Add new secret: SONAR_ORGANIZATION (Organization Key)



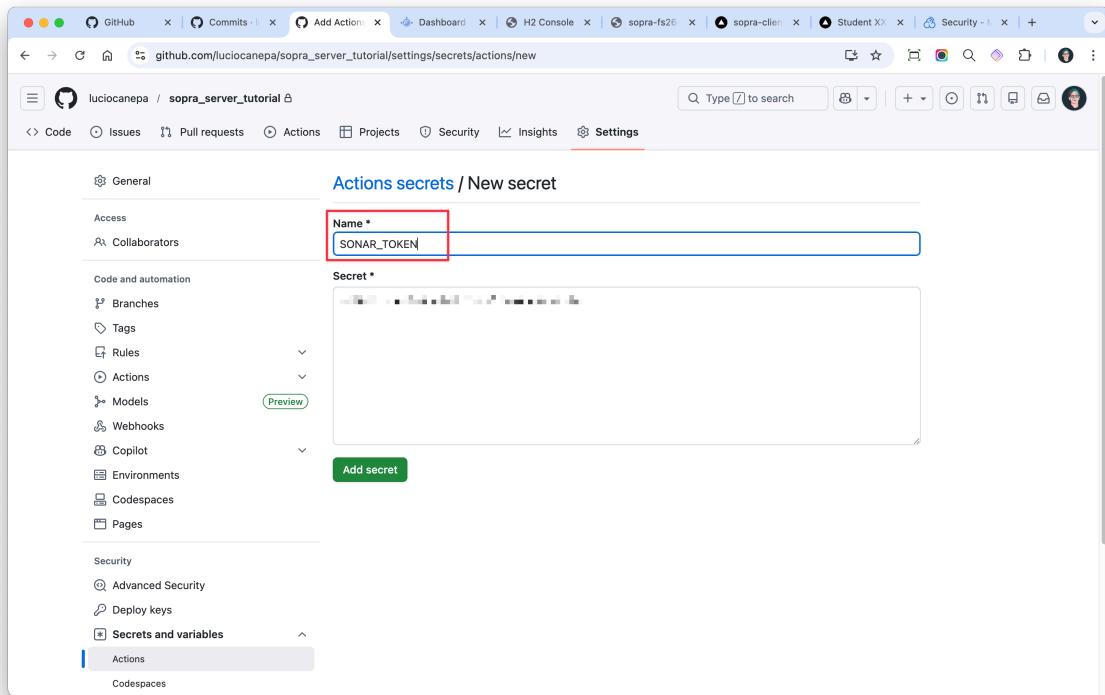
To get the token, click on your profile picture > My Account > Security - enter a token name (whatever you prefer) > click Generate Token

The screenshot shows the SonarCloud account security interface. In the top right corner, a user profile for 'Lucio Canepa' is visible with a dropdown menu open. The 'Security' tab is selected in the main navigation bar. On the left, there's a sidebar with 'Profile', 'Security', 'Notifications', 'Organizations', 'Enterprises', and 'Appearance'. The main content area is titled 'Security' and contains a section for generating tokens. A red box highlights the 'Enter Token Name' input field, and a red arrow points to the 'Generate Token' button next to it. Below this, there's a section for 'Existing Tokens' with a table header for 'Name', 'Last use', and 'Created'. A message 'No tokens' is displayed. To the right, there's a promotional box for upgrading to scan private projects, listing benefits like a 14-day free trial and support for private projects.

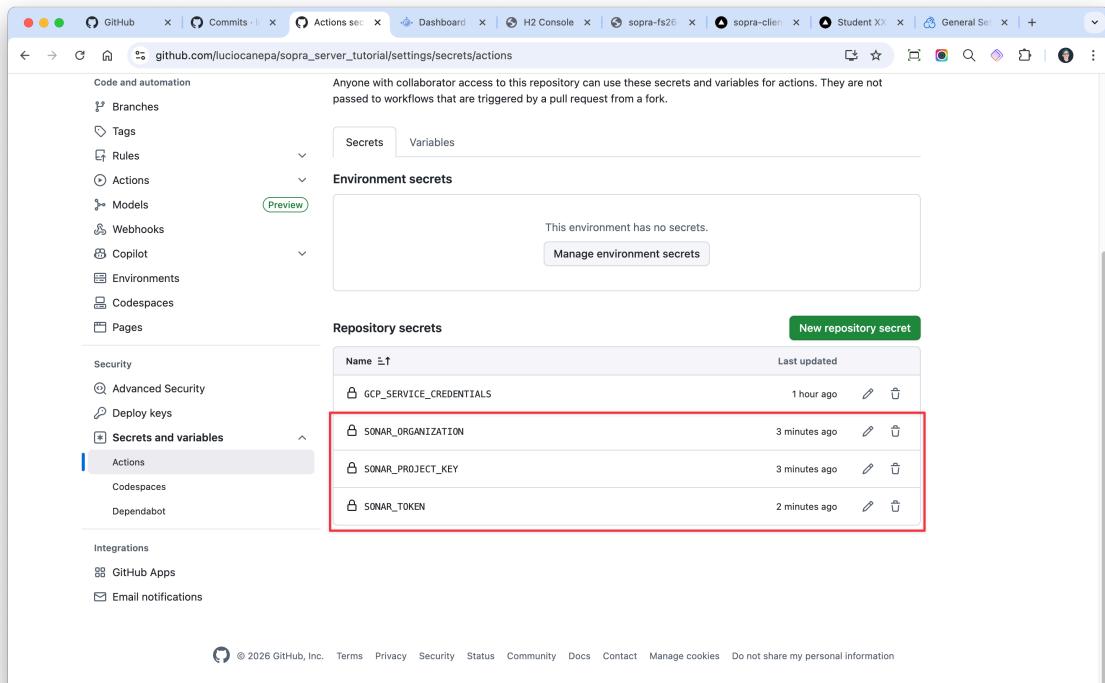
Copy the generated token

This screenshot shows the same SonarCloud security interface after a token has been generated. The 'Generate Tokens' section now displays a success message: 'New token "sopraTutorialServer" has been created. Make sure you copy it now, you won't be able to see it again!' A red box highlights the 'Copy' icon (a clipboard with a plus sign) next to the token name, and a red arrow points to it. The 'Existing Tokens' table now lists the new token: 'sopraTutorialServer' with 'Never' as the last use date and '28 January 2026' as the creation date. A 'Revoke' button is also visible next to the token row. The rest of the interface remains the same, including the upgrade offer and user profile sidebar.

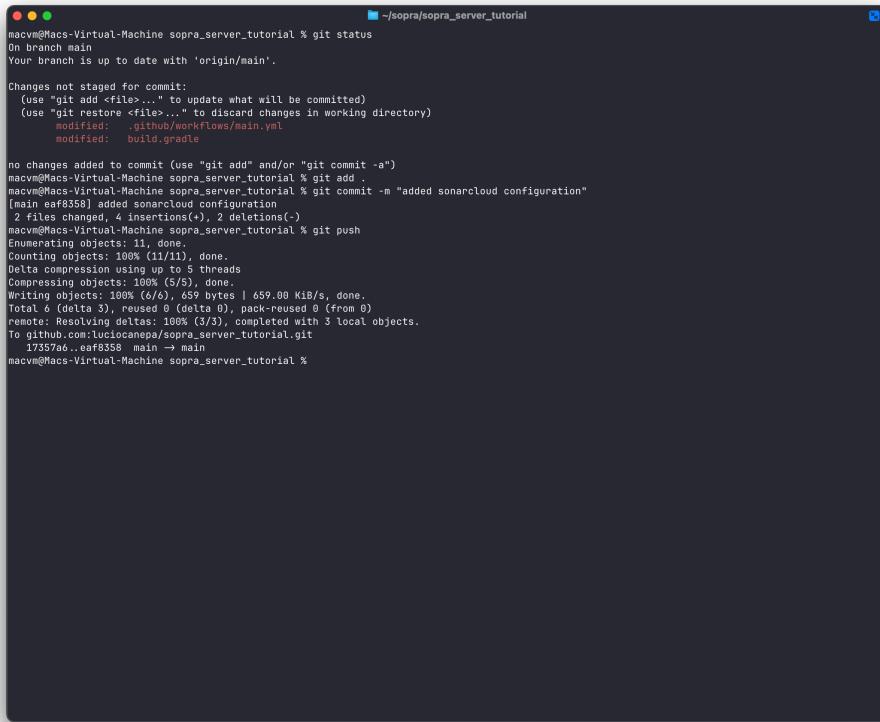
Add new secret: SONAR_TOKEN



At this point you should have added 3 new secrets to your server repository



Commit and push the changes to your server repository. Now both the Google cloud deploy and the Test and Sonarqube should be successfull



```
macvm@Mac-Virtual-Machine sopra_server_tutorial % git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
    (use "git restore <file>..." to discard changes in working directory)
      modified:   .github/workflows/main.yml
      modified:   build.gradle

no changes added to commit (use "git add" and/or "git commit -a")
macvm@Mac-Virtual-Machine sopra_server_tutorial % git add .
macvm@Mac-Virtual-Machine sopra_server_tutorial % git commit -m "added sonarcloud configuration"
[main eaf8358] added sonarcloud configuration
 2 files changed, 4 insertions(+), 2 deletions(-)
macvm@Mac-Virtual-Machine sopra_server_tutorial % git push
Enumerating objects: 11, done.
Counting objects: 100% (11/11), done.
Delta compression using up to 5 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 659 bytes | 659.00 Kib/s, done.
Total 6 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To github.com:luciocanepa/sopra.server.tutorial.git
  17357a6..eaf8358  main -> main
macvm@Mac-Virtual-Machine sopra_server_tutorial %
```

In SonarQube, under your project, you can see the results

The screenshot shows the SonarCloud interface for a project named 'soprafs26'. The main navigation bar includes links for GitHub, Commits, added sonar, Dashboard, H2 Console, sopra-fs26, sopra-client, Student, and Summary. The current page is 'sonarcloud.io/summary/new_code?id=luciochanepa_sopra_server_tutorial'. The left sidebar shows project details like 'Project: soprafs26', 'Status: Private', and branches ('Main Branch' is selected). The main content area displays the 'Main Branch Summary' for the 'main' branch. It indicates 386 Lines of Code, Version 0.01-SNAPSHOT, and a last analysis 11 minutes ago (commit eaf83589). A prominent red 'Failed' status is shown, with a note: 'The quality profile assigned to this project has changed. You can explore the change in the [Activity page](#)'. Below this, tabs for 'New Code' (1 failed) and 'Overall Code' are visible. The 'New Code' section shows 1 condition failed (0.0% Security Hotspots Reviewed) and 21 New Issues. The 'Overall Code' section shows 0 Accepted Issues and 21 Open issues. Coverage is at 86.4%.

and navigate between them

The screenshot shows the SonarCloud interface for the same project 'soprafs26'. The URL is 'sonarcloud.io/summary/overall?id=luciochanepa_sopra_server_tutorial&branch=main'. The left sidebar shows the 'Main Branch' is selected. The main content area displays the 'Overall Summary' for the 'main' branch. It includes sections for Security (0 Open issues), Reliability (0 Open issues), Maintainability (21 Open issues), Accepted Issues (0), Coverage (86.4%), Duplications (8.5%), and Security Hotspots (5). The overall status is marked as 'A' (green). The bottom of the page includes copyright information (© 2018-2026 SonarSource Sàrl. All rights reserved.) and links for Terms, Pricing, Privacy, Cookie Policy, Security, Community, Documentation, Contact us, Status, and About.

5.2 Set-up Vercel and link it to the client repository

To set up the frontend on Vercel, the following components must be configured (as in the step-by-step tutorial):

Platform & Project

- Vercel Project: Linked directly to your GitHub client repository.
- Deployment URL: The live production link for the web application.

GitHub Secrets (for CI/CD)

- VERCEL_TOKEN: Authorized access token for deployment.
- VERCEL_ORG_ID: Your Vercel account/organization identifier.
- VERCEL_PROJECT_ID: The specific ID for the client project.

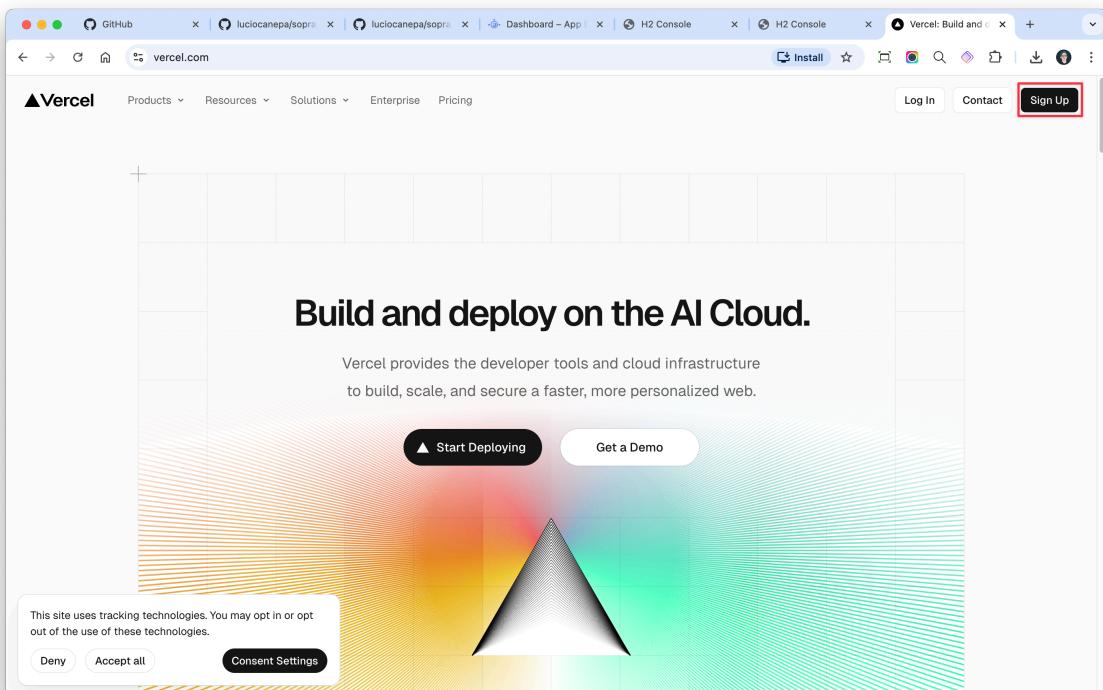
Environment Variables

- NEXT_PUBLIC_PROD_URL: The backend's Google Cloud URL, stored in Vercel settings to enable frontend-backend communication.

Set-up Vercel project and connect the client repository

Navigate to vercel.com

Create an account or login (best if with your Github account)



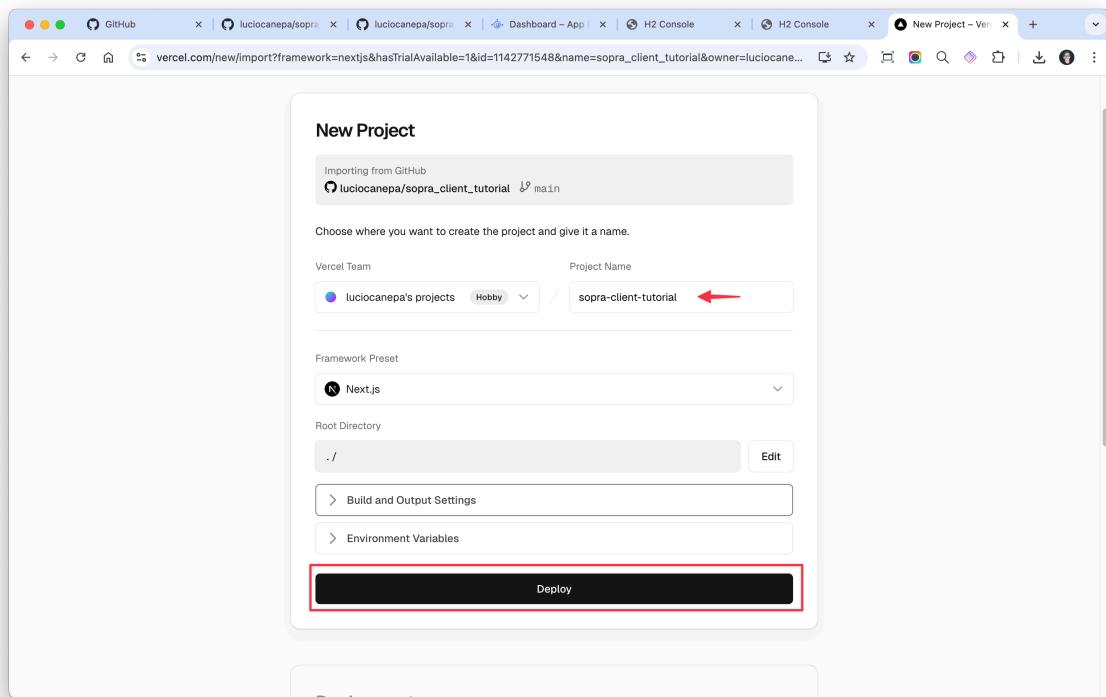
Create a new project

The screenshot shows the Vercel dashboard interface. On the left, there's a sidebar with sections like Overview, Integrations, Deployments, Activity, Domains, Usage, Observability, Storage, Flags, AI Gateway, Sandboxes, Agent, Support, and Settings. Below the sidebar is a search bar labeled 'Search Projects...'. The main area is titled 'Projects' and features a 'Deploy your first project' section with various templates: Next.js Boilerplate, AI Chatbot, Express.js, and Vite + React Starter. Each template has a 'Deploy' button. To the left of the main area, there's a 'Usage' section showing metrics for the last 30 days and an 'Alerts' section. A red box highlights the 'Project' button in the top right corner of the sidebar.

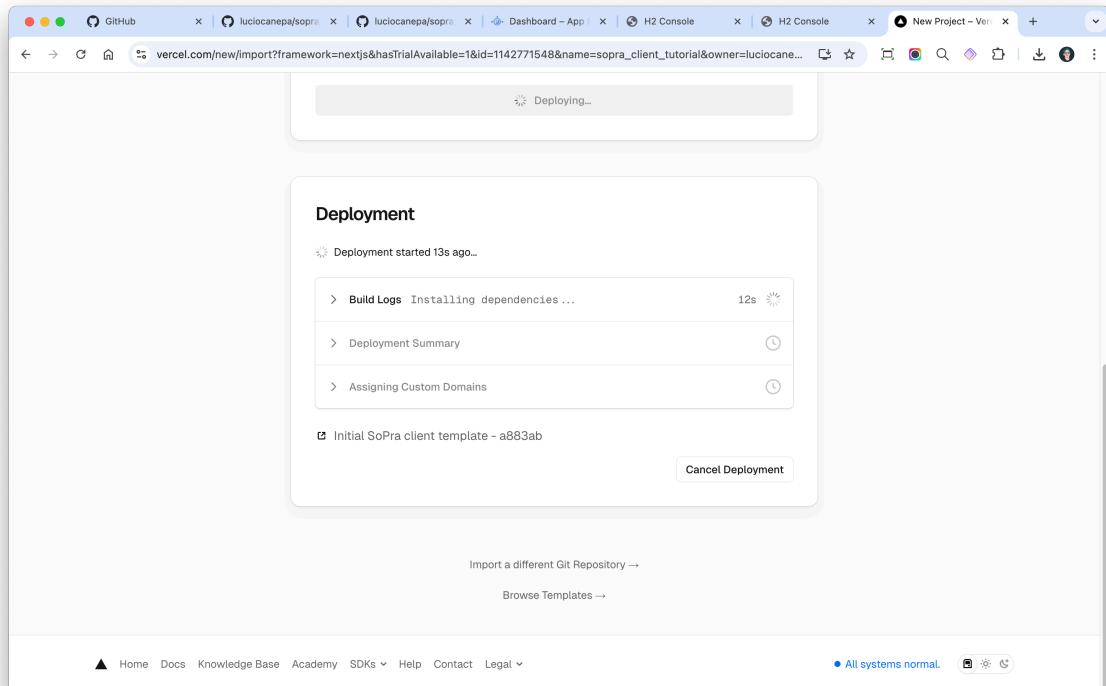
Connect the client repository (either you already connected your Github profile or you should do it now)

The screenshot shows the 'New Project' screen on Vercel. At the top, there's a search bar for 'Enter a Git repository URL to deploy...' and a 'Continue' button. Below that, there are two main sections: 'Import Git Repository' and 'Clone Template'. In the 'Import Git Repository' section, a dropdown shows 'luciocanepa' and a list of repositories, with 'sopra_client_tutorial' selected and highlighted by a red box. To the right, the 'Clone Template' section shows four templates: Next.js Boilerplate, AI Chatbot, Express.js on Vercel, and Vite + React Starter, each with a preview image and a 'Deploy' button. At the bottom, there's a navigation bar with links like Home, Docs, Knowledge Base, Academy, SDKs, Help, Contact, Legal, and a status indicator 'All systems normal.'

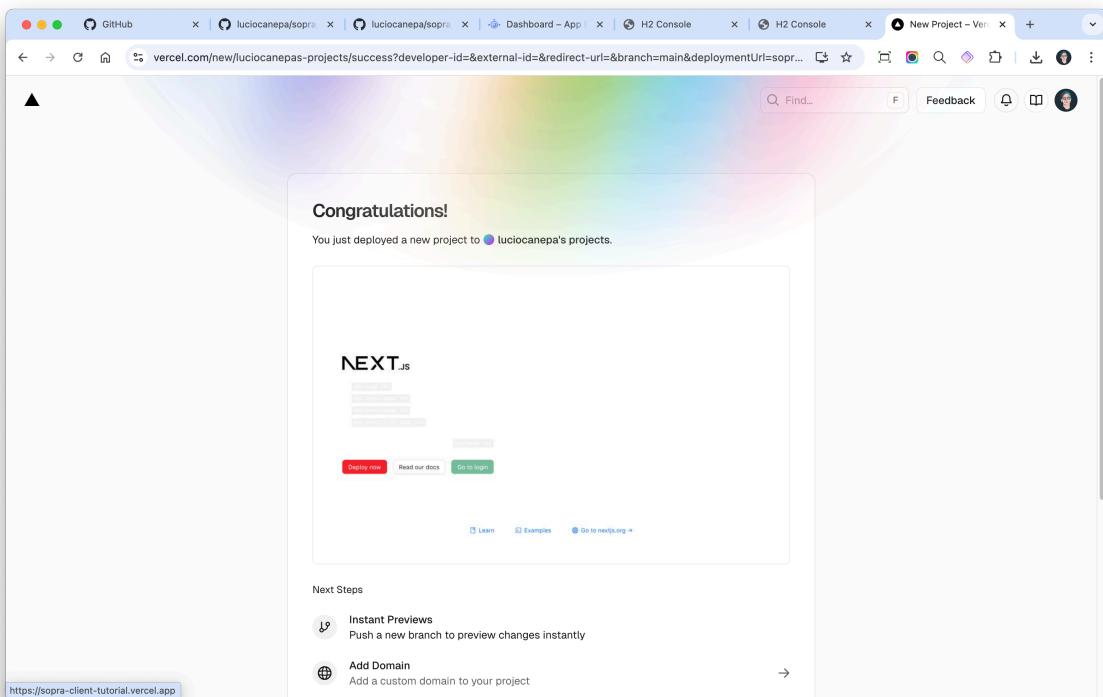
Name the project & click Deploy



The deployment will start



Confirmation: deployment was successful



This is the overview page of your Vercel project

A screenshot of the Vercel dashboard showing the project "sopra-client-tutorial". The top navigation bar includes GitHub, luciocanepa/sopra, luciocanepa/sopra, Dashboard - App, H2 Console, H2 Console, and sopra-client-tutorial. The main content area shows the project name "sopra-client-tutorial" and a "Production Deployment" card. The deployment card shows a preview of a Next.js app, deployment details (Deployment ID: sopra-client-tutorial-6vqoOf6pr-luciocanepas-projects.vercel.app, Status: Ready, 1m ago by JulianaGSouza), and source information (main branch, commit hash a883ab0). Below the deployment card are sections for "Deployment Settings" and "Recommendations". At the bottom, there are cards for "Firewall" (24h, Active - All systems normal), "Observability" (6h, Edge Requests), and "Analytics".

In the .github/workflows/verceldeployment.yml you can see that 3 secrets are needed: this should be added to the github client repository

```

version: '2'
name: Deploy to Vercel
on:
  push:
    branches:
      - main
  jobs:
    deploy:
      runs-on: ubuntu-latest
      steps:
        - name: Checkout Repo
          uses: actions/checkout@v3
        - name: Setup Node.js
          uses: actions/setup-node@v3
          with:
            node-version: "18"
        - name: Install Dependencies
          run: npm install
        - name: Deploy to Vercel
          run: npx vercel --prod --token ${secrets.VERCEL_TOKEN} --confirm
          env:
            VERCEL_ORG_ID: ${secrets.VERCEL_ORG_ID}
            VERCEL_PROJECT_ID: ${secrets.VERCEL_PROJECT_ID}

```

Do you want to install the recommended 'GitHub Actions' extension from GitHub for verceldeployment.yml?

[Install](#) Show Recommendations

In Vercel under Settings > General, you will find your Project ID - copy it

Project Name

Used to identify your Project on the Dashboard, Vercel CLI, and in the URL of your Deployments.

Save

Project ID

Used when interacting with the Vercel API.

1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
Copy

Learn more about Project ID

Vercel Toolbar

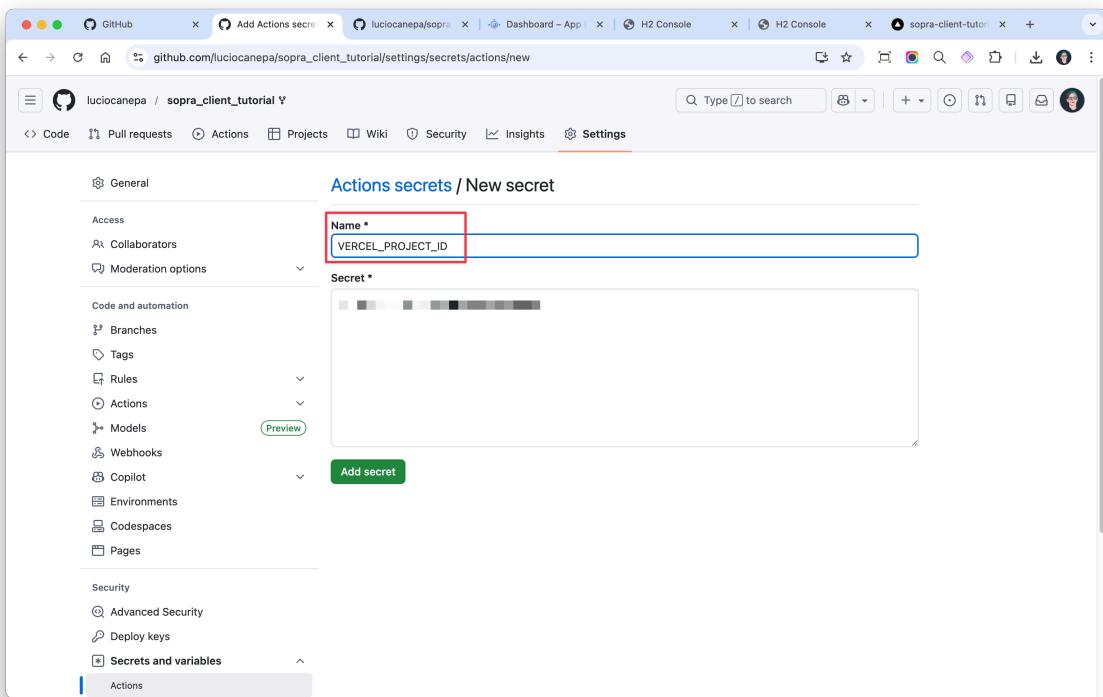
Enable the Vercel Toolbar on your Deployments.

Pre-Production Deployments: Default (controlled at the team level)

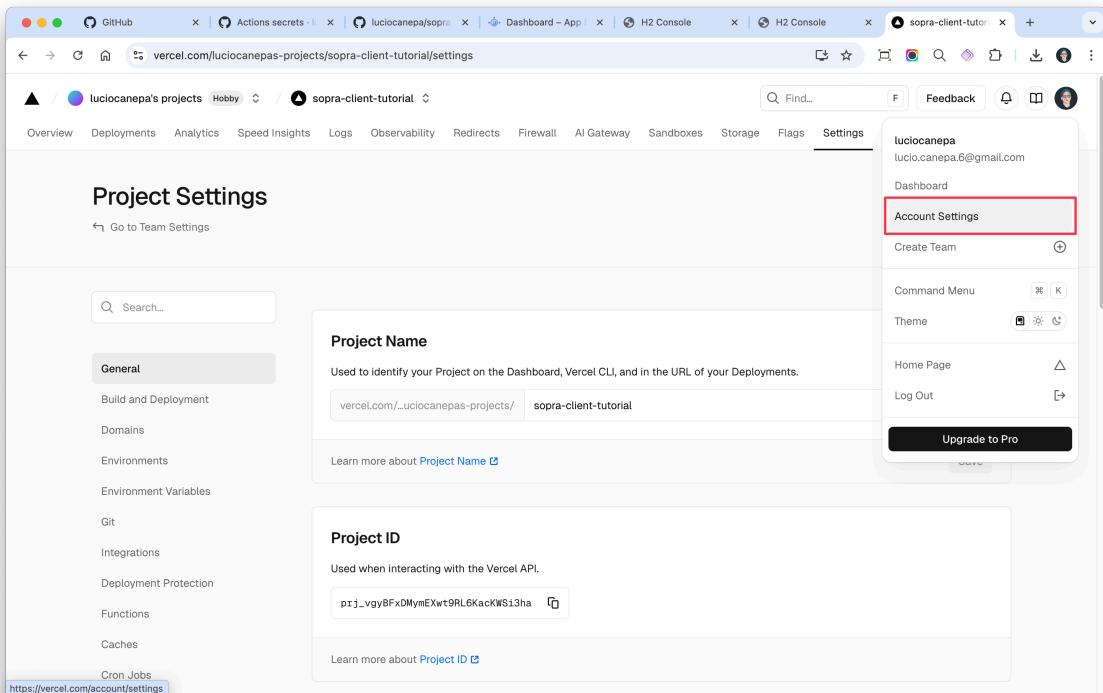
Production Deployments: Default (controlled at the team level)

To use the toolbar in production your team members need the [Chrome extension](#) or to enable the toolbar for that domain in

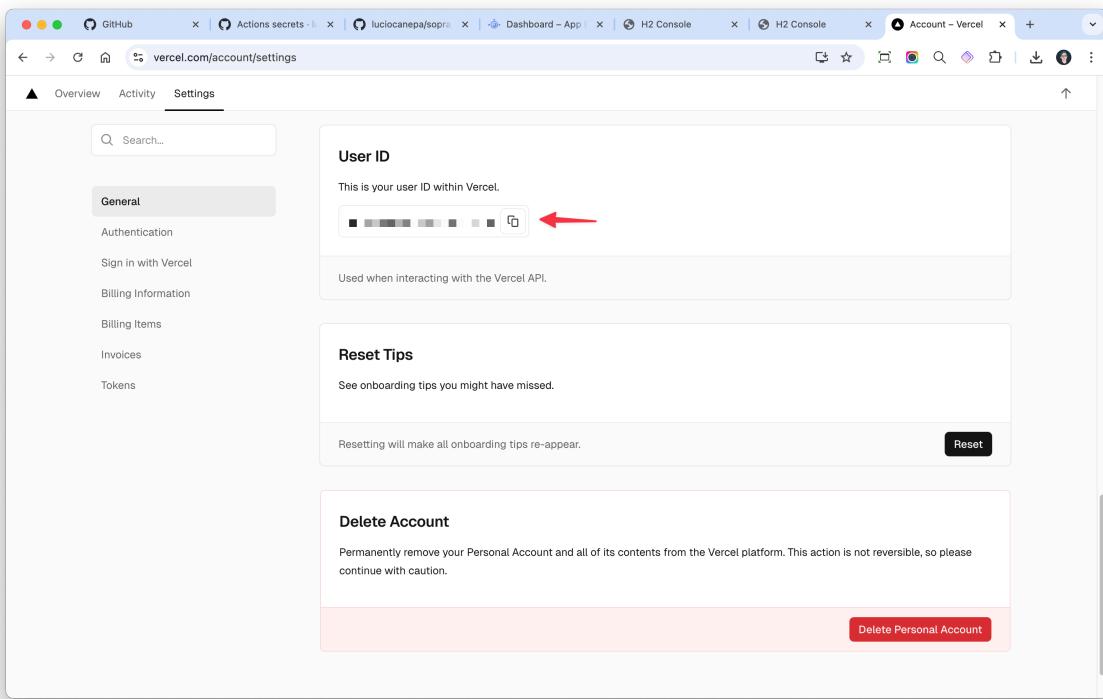
In your client repository, create a new repository secret matching precisely the name from your workflow file: VERCEL_PROJECT_ID



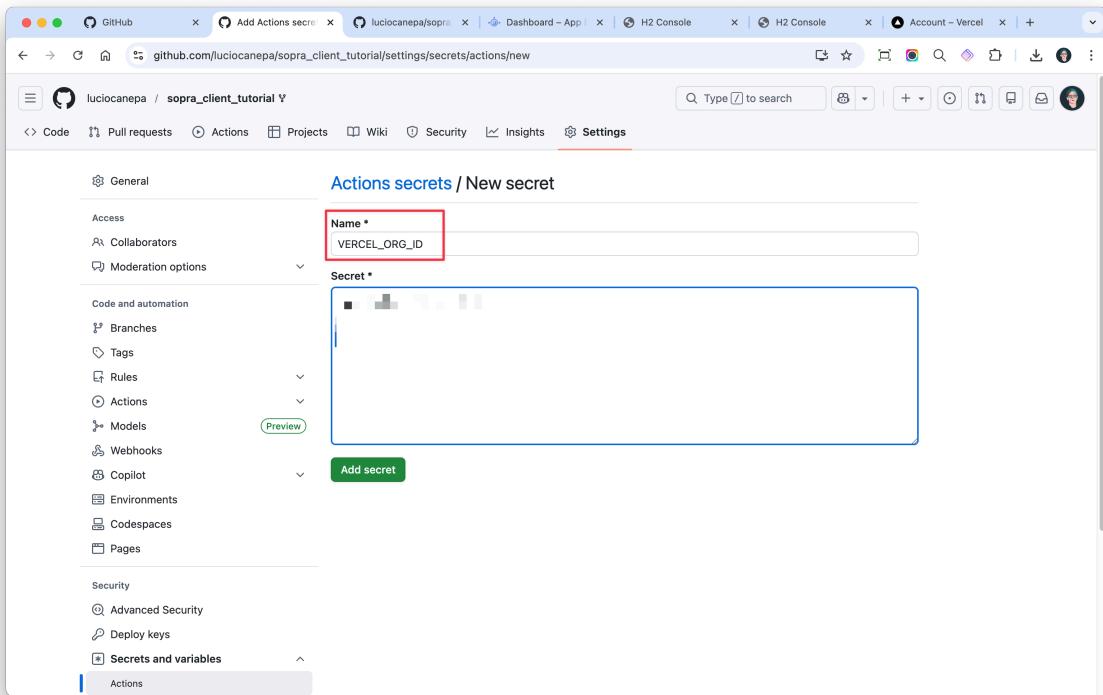
To find your User ID (= Organization ID), navigate to the Account Settings



Under General, you find the User ID - copy it



New repository secret VERCEL_ORG_ID - paste the copied User ID (= Organization ID)



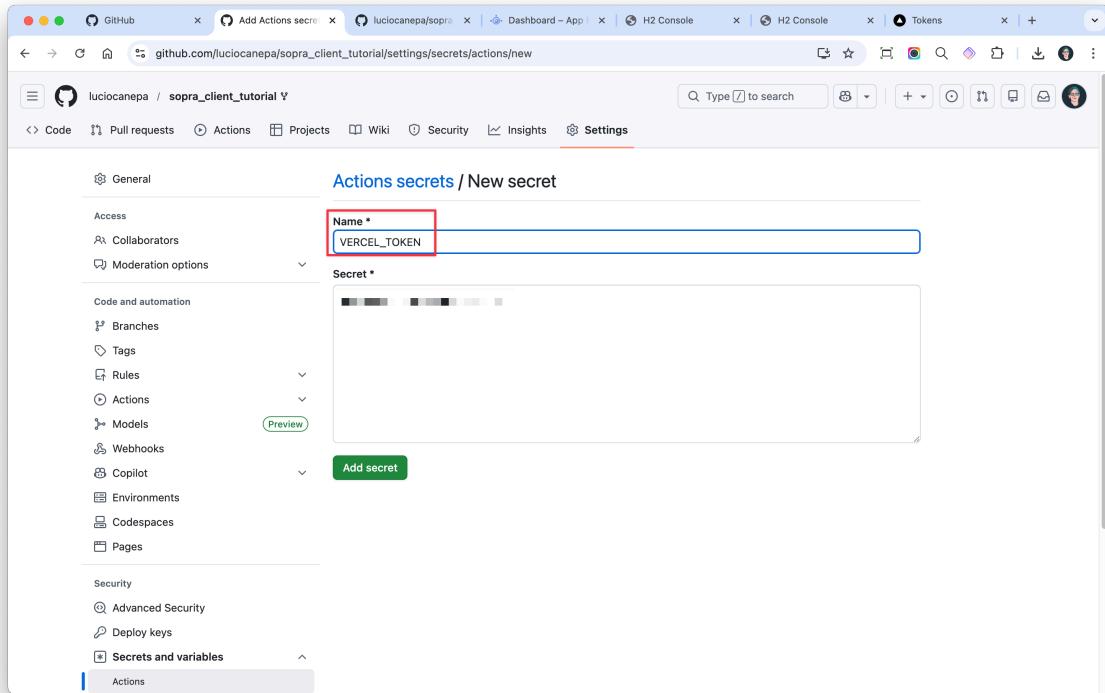
To generate a new Token, from the Account Settings > Tokens - give it a name (whatever you want), a scope (your organization) and an expiration date - click create

The screenshot shows the Vercel account settings page with the 'Tokens' tab selected. In the 'Create Token' section, a token named 'sopra_tutorial_client' is being created with a scope of 'luciocanepa's projects' and an expiration of '90 Days'. A red box highlights the 'Tokens' tab in the sidebar, and three red arrows point to the token name, scope, and expiration fields.

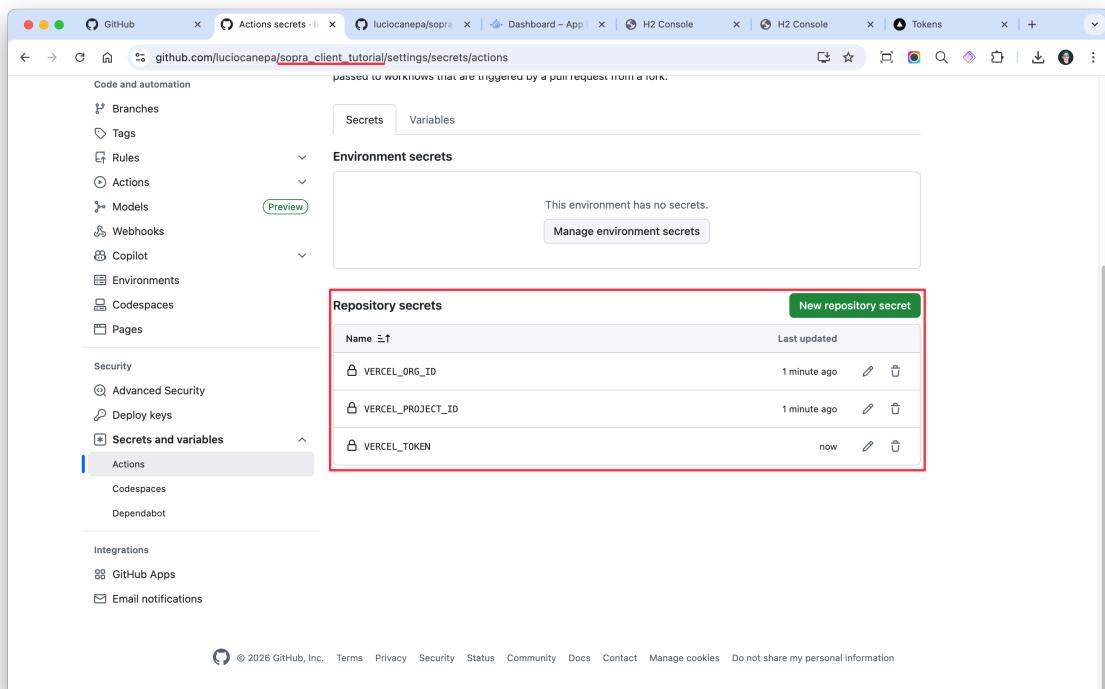
Confirmation: token is created - copy it

The screenshot shows the Vercel account settings page with the 'Tokens' tab selected. A 'Token Created' dialog box is open, prompting the user to copy the token. The token value is obscured by a red redaction box. A red arrow points to the 'Done' button at the bottom of the dialog. The background shows a list of existing tokens.

New repository secret: VERCEL_TOKEN - paste the copied token



The 3 secrets should be visible under Settings > Secrets and variables in your **client repository**



We need to add a last Environment Variables in Vercel > your client project > Settings > Environment Variables > Add Environment Variable

The screenshot shows the Vercel Project Settings page for the 'sopra-client-tutorial' project. The 'Environment Variables' section is highlighted with a red box. On the left sidebar, the 'Environment Variables' option is also highlighted with a red box. At the top right, there is a red box around the 'Add Environment Variable' button. The main content area displays a message: 'No Environment Variables Added' with a sub-instruction: 'Add Environment Variables to Production, Preview, and Development environments, including branches in Preview.'

Name NEXT_PUBLIC_PROD_URL and paste the server production URL from Google Cloud console > App Engine (the same we pasted into the client repository in domain.ts)

Now you can commit and push your changes from the client repository

```
macvm@Macs-Virtual-Machine sopra % cd sopra_clientTutorial
direnv: loading ~/sopra/sopra_clientTutorial/.envrc
direnv: using flake --no-warn-dirty
=====
If you are using code in Xcode, please install 16.2 or 15.3
You can install the latest version with xcode-select install 16.2
=====

direnv: export +AR +AS +C +COMPOSE_PROJECT_NAME +CONFIG_SHELL +CXX +DEVELOPER_DIR +HOST_PATH +HOST_PROJECT_PATH +IN_NIX_SHELL +LD +LD_OYLO_PATH +M +ACOSX_DEPLOYMENT_TARGET +NIX_APPLE_SDK_VERSION +NIX_BINTOOLS +NIX_BINTOOLS_WRAPPER_TARGET +HOST_aarch64.apple_darwin +NIX_BUILD_CORES +NIX_BUILD_TOP +NIX_CC +NIX_CC_WRAPPER_TARGET +HOST_aarch64.apple_darwin +NIX_CFLAGS_COMPILE +NIX_DONT_SET_RPATH +NIX_DONT_SET_RPATH_FOR_BUILD +NIX_ENFORCE_NO_NATIVE +NIX_HARDENING_ENABLE +NIX_IGNORE_LD_THROUGH_GCC +NIX_LDFLAGS +NIX_NO_SELF_RPATH +NIX_STORE +NM +NODE_PATH +OBJCOPY +OBJDUMP +PATH_LOCALE +RANLIB +S+ORKROOT +SIZE +SOURCE_DATE_EPOCH +STRINGS +STRIP +TEMP +TEMPDIR +TMP +ZERO_AR_DATE + _darwinAltLocalNetworking + _impureHostDeps + _propagateDirpureHostDeps + _propagatedsandboxProfile + _structuredAttrs +buildInputs +builder +cmakeFlags +configureFlags +depsBuildBuild +depsBuildPropagated +depsBuildTarget +depsBuildTargetPropagated +depsHostHost +depsHostPropagated +depsTargetTarget +depsTarg etTargetPropagated +doCheck +doInstallCheck +dontAddDisableDepTrack +mesonFlags +name +nativeBuildInputs +out +outputs +patches +phases +preferLoca lBuild +propagatedBuildInputs +propagatedNativeBuildInputs +shell +shellHook +stdevn +strictDeps +system -PATH ~TMPDIR -XDG_DATA_DIRS
macvm@Macs-Virtual-Machine sopra_clientTutorial % git status
On branch main
Your branch is up to date with 'origin/main'.

Changes not staged for commit:
  (use "git add <file>" to update what will be committed)
  (use "git restore <file>" to discard changes in working directory)
    modified:   app/utils/domain.ts

no changes added to commit (use "git add" and/or "git commit -a")
macvm@Macs-Virtual-Machine sopra_clientTutorial % git add .
macvm@Macs-Virtual-Machine sopra_clientTutorial % git commit -m "updated prod db url & set-up environment variables"
[main cc269eb] updated prod db url & set-up environment variables
1 file changed, 1 insertion(+), 1 deletion(-)
macvm@Macs-Virtual-Machine sopra_clientTutorial % git push
Enumerating objects: 9, done.
Counting objects: 100% (9/9), done.
Delta compression using up to 5 threads
Compressing objects: 100% (5/5), done.
Writing objects: 100% (5/5), 538 bytes | 539.00 KiB/s, done.
Total 5 (delta 3), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (3/3), completed with 3 local objects.
To github.com:luciocanepa/sopra_clientTutorial.git
 a883ab0..cc269eb main → main
macvm@Macs-Virtual-Machine sopra_clientTutorial % |
```

The Vercel deployment workflow will start

This screenshot shows the GitHub repository overview for 'sopra-client-tutorial.vercel.app'. The repository has 1 branch and 0 tags. It is 1 commit ahead of the 'main' branch. A message indicates that the branch is 1 commit ahead of 'HASEL-UZH/sopra-fs26-template-client:main'. There are 1 pending check and 1 commit from 'luciocanepa' dated 2 weeks ago. A prominent message says 'Some checks haven't completed yet' with '1 pending check'. Another message from 'Vercel Pending - Vercel is deploying your app' is also present. The repository has 0 stars, 0 forks, and 0 releases.

From the overview page of your project, you can Visit the deployed client side application

This screenshot shows a browser window displaying the deployed Next.js application at 'sopra-client-tutorial-jyiwt0by-luciocanepas-projects.vercel.app'. The page title is 'NEXT.js'. It lists several static files: 'app/page.tsx', 'www2020/index.tsx', 'wwwusers/index.tsx', and 'wwwusers/[id]/page.tsx'. Below these is another 'app/page.tsx'. At the bottom, there are three buttons: 'Deploy now' (red), 'Read our docs' (white), and 'Go to login' (green). At the very bottom of the page, there are links to 'Learn', 'Examples', and 'Go to nextjs.org'.

Verify it's connected with the backend (server repository deployed on Google Cloud) by navigating to /users - here we see the users we created before

A screenshot of a web browser window titled "sopra-client-tutorial.vercel.app/users". The page displays a table with the heading "Get all users from secure endpoint:". The table has three columns: "Username", "Name", and "Id". There are two rows of data:

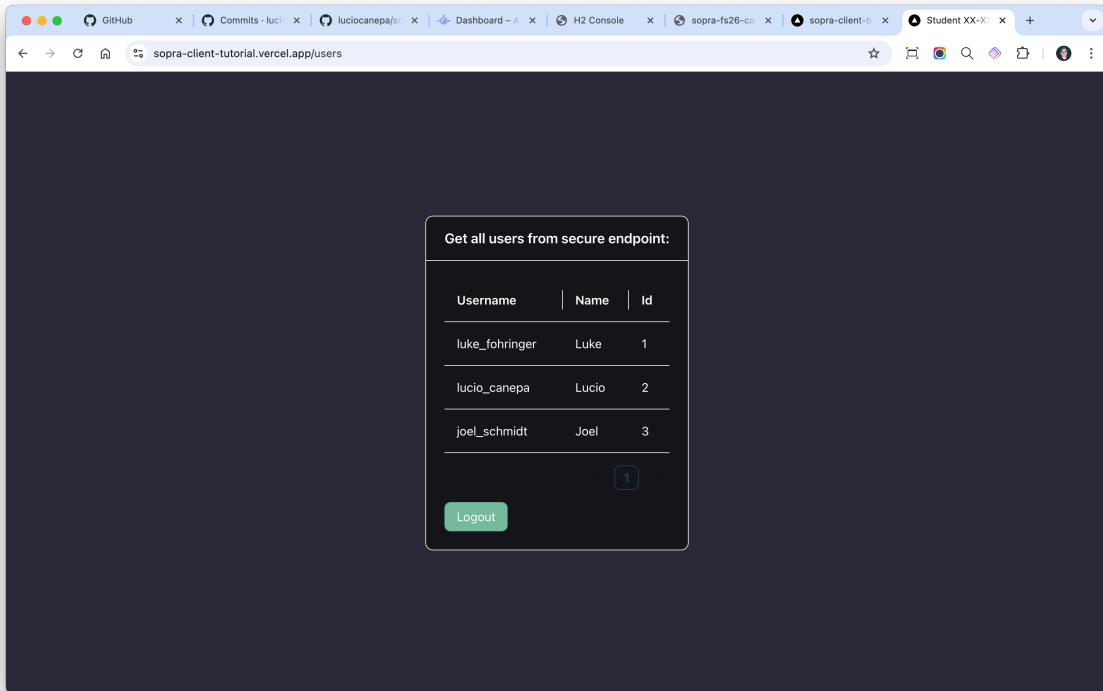
Username	Name	Id
luke_fohringer	Luke	1
lucio_canepa	Lucio	2

At the bottom right of the table is a small blue button with the number "1". Below the table is a green "Logout" button.

under /login you can create a new user

A screenshot of a web browser window titled "sopra-client-tutorial.vercel.app/login". The page features a simple form with two input fields and a "Login" button. The first field is labeled "Username" and contains the value "joel_schmidt". The second field is labeled "Name" and contains the value "Joe". A green "Login" button is positioned below the inputs.

and verify it has been created



5.3 Set-up Docker Hub and containerize your server application

Docker allows you to package your application into a *container* – a standardized unit that includes everything needed to run the software (code, runtime, libraries, dependencies). This makes it easy to run your application consistently on any machine.

Create Docker Account

1. Navigate to hub.docker.com and create an account
2. Incorporate your group number into the account name, for example `sopra_group_12`
3. After logging in, click Create repository
4. Name the repository *exactly the same* as your GitHub server repository name (e.g., `sopra-fs26-template-server`) – set visibility to Public & click Create

Adding Secrets

1. From Docker Hub, click on your profile picture > Account settings > Personal access tokens
2. Click Generate new token – give it a description (e.g., `github-actions`), select *Read & Write* access & click Generate
3. Copy the generated token immediately – you won't be able to see it again
4. Go to your *server repository* on GitHub > Settings > Secrets and variables > Actions

5. Click New repository secret and add the following 3 secrets:

Name	Value
dockerhub_username	Your Docker Hub username (e.g., sopra_group_12)
dockerhub_password	The PAT you generated (not your login password!)
dockerhub_repo_name	Your Docker Hub repository name

Caution

The secret names must match *exactly* what's in your workflow file, including lowercase letters and underscores.

Running Workflow

Push any change to your main branch to trigger the workflow:

```
bash git add . git commit -m "trigger docker build" git push origin main
```

Go to your GitHub repository > Actions tab – you should see the Dockerize workflow running. Once successful (green checkmark), go to your Docker Hub repository – you should see your image with the latest tag.