

## Section 2: About Your Project

### Project Details

**Project title**      Dockerize Oppia

**Project size**      Large (~350 hours)

<p><b>Why did you choose this project?</b></p>	<p>New contributors to Oppia currently face numerous challenges when setting up the development environment and the installation process can be quite complex and time-consuming, requiring careful attention to detail. This can discourage beginners and make the installation process more error-prone, leading to frustration and, in some cases, even causing contributors to leave the organization.</p> <p>To alleviate these issues, we plan to simplify and streamline the installation process using Docker Compose, which will package our application into multiple containers. With just a few straightforward commands, developers will be able to install, build, and run servers in the development environment. Additionally, we will optimize the test runs of GitHub actions by leveraging Docker images in the workflows. This approach will reduce the load on the hosted GitHub servers and speed up test runs, improving the overall efficiency of the development process.</p>
--	--

## Project Timeframe

**Note:** *Oppia will only be offering a single GSoC coding period timeframe this year, starting on **May 29**. All work for Milestone 1 must be completed and submitted by **July 14**, and all work for Milestone 2 must be completed and submitted by **Sept 15**. We will not be able to extend these deadlines.*

<p><b>Coding period</b></p>	<ul style="list-style-type: none"> <li>• I will adhere to the above deadlines.</li> </ul>
<p><b>Planned time commitment</b></p>	<p>I am having a Summer break from May to July, I will have approximately 8 weeks starting from 29th May to 26th July. During this period, I will be able to dedicate around ~30 hrs/week. (30*8 → 240 hrs.)</p> <p>From 27th July until 15th September, which is approximately 6 weeks, I will only be able to devote around ~20 hrs/week due to my college semester being in session. (20*6 → 120 hrs.)</p> <p>The allocated time frame is adequate for completing the project; however, I can adjust my work hours to meet the project's needs if necessary.</p>
<p><b>What other obligations might you need to work around during the summer?</b></p>	<ul style="list-style-type: none"> <li>• On 26th July, I will need to travel back to my college campus from my home, and it may take up to 2 days to complete the journey and settle down.</li> <li>• I am expecting my mid-semester exams to begin on 10th September, and I will need to adjust my work schedule accordingly. The exact date has not been announced by the university yet.</li> </ul>

## Communication Channels

**Note:** The Oppia team places a high emphasis on communication, and we have found that daily contact between contributors and mentors is important for helping keep projects on track. This is why we ask that contributors send short daily updates to their mentors explaining what they have done, where they are stuck, and what they plan to do next.

<b>I can commit to sending daily updates to my mentor by email, each day I work during the GSoC period.</b>	<ul style="list-style-type: none"><li>• Yes</li></ul>
<b>In addition to the above: how often, and through which channel(s), do you plan on communicating with your mentor?</b>	<p>I aim to provide daily updates to my mentor to maintain proper communication and ensure the project runs smoothly. For such discussions and addressing my small queries, my preferred mode of communication would be either Google Chat or Email.</p> <p>Meetings with mentor: 2 times a week (flexible) - on Google Meet or any other platform.</p>

## Section 3: Proposal Details

**Note:** This section is largely adapted from [Oppia TDD Template \(Dec 2021\)](#) -- PLEASE MAKE A COPY. **We strongly recommend reading that doc**, since it has detailed explanations of how to fill out the various sections. It also contains some links to sample TDDs, so that you can see what others have done in the past. Additionally, you might find the guidance for "how to plan a technical project" on this [Oppia wiki page](#) useful.

## Problem Statement

<b>Link to PRD</b>	N/A
<b>Target Audience</b>	All Developers, Testers/QA*, Maintainers
<b>Core User Need</b>	<ul style="list-style-type: none"><li>• Developers and testers who are installing Oppia must follow an extensive installation process to set up the development environment successfully. The installation process is complex and requires careful attention to detail, as well as a significant amount of time and effort to complete.</li></ul>

	<ul style="list-style-type: none"> <li>• Developers and testers may encounter inconsistency due to variations in development environments, such as differences in the versions of services, packages, and dependencies used in the application. This inconsistency can result in problems, including the well-known issue of "it works on my machine, but not on yours."</li> <li>• Developers face challenges in managing and configuring installed dependencies and setting up environments for those dependencies, which can be both difficult and error-prone.</li> <li>• As a Maintainer of Oppia's repository, it is a little long process for upgrading any dependency with a newer version.</li> </ul>
<p><b>What goals do we want the solution to achieve?</b></p>	<ul style="list-style-type: none"> <li>• Making the installation process quick, easy, and independent of the platform/OS.</li> <li>• Packing up the application dependencies/services into containers, developers can work in a consistent environment without worrying about the system packages being affected.</li> <li>• No extra setting up of environments or configuration of the dependencies is required.</li> <li>• The Setup should be quick and easy. It should follow the following steps <ul style="list-style-type: none"> <li>○ Install Docker Desktop</li> <li>○ Clone oppia/oppia</li> <li>○ Run '<b>make install-dependencies</b>'</li> <li>○ Run '<b>make run</b>'</li> </ul> </li> <li>• Any upgradations in the Oppia's dependencies will be smooth for both Maintainers and Developers.</li> <li>• Speeding up the test runs on GitHub Actions by using Docker Image to install dependencies in the workflows.</li> </ul>

\* here, Testers/QA can be also referred to as a non-technical person who may not have programming skills or are not proficient in writing codes, and tests the application(using the development server) to its limits.

## Section 3.1: WHAT

*This section enumerates the requirements that the technical solution outlined in "Section 2: HOW" must satisfy.*

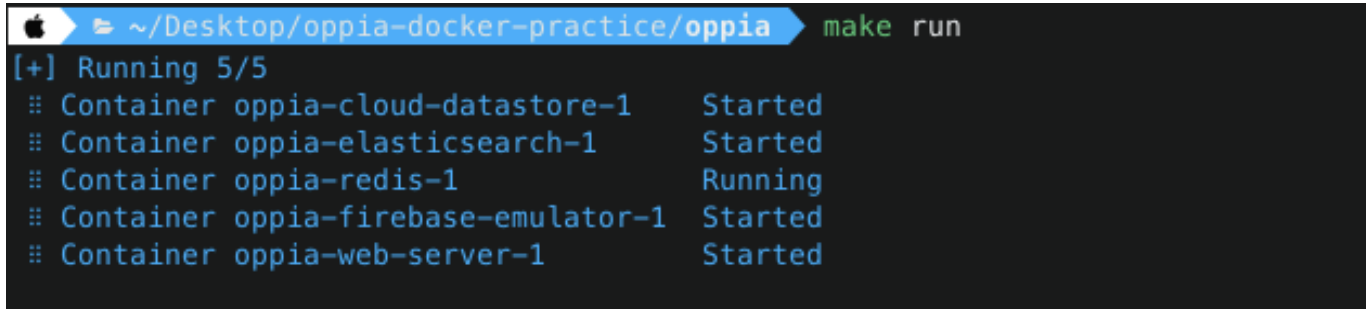
This project aims to reorganize the installation process for our application by containerizing our application using **Docker**. By packaging our application as **Docker containers (using several docker images)**, users will experience ease and efficiency in setting up a development environment. To setup the development environment, users must →

- 1) **Install Docker Desktop** on their machine

- 2) **Clone the Oppia repository**
- 3) run the "**make install-dependencies**" command to instantly install all required dependencies and set up the entire development environment.
- 4) run the "**make run**" command to start the Oppia development server.

With Docker's seamless setup of necessary services, users can expect a faster, smoother, and more error-resistant installation experience.

- **make run** command example using cached docker images →



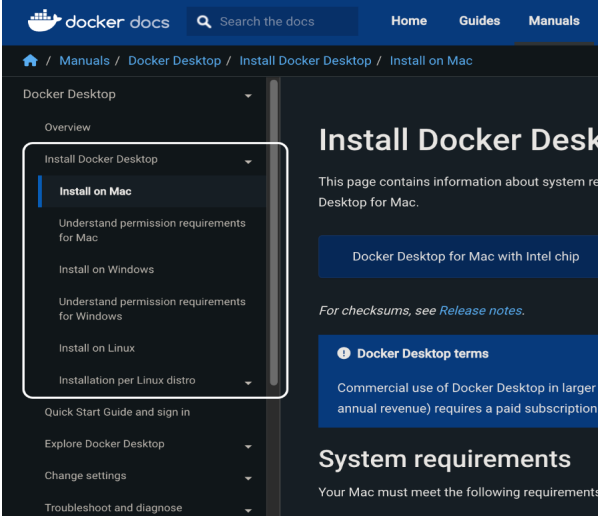
```
~/Desktop/oppia-docker-practice/oppia make run
[+] Running 5/5
  :: Container oppia-cloud-datastore-1    Started
  :: Container oppia-elasticsearch-1     Started
  :: Container oppia-redis-1             Running
  :: Container oppia-firebase-emulator-1 Started
  :: Container oppia-web-server-1        Started
```

Once the application has been containerized with Docker, it is essential to ensure that various types of tests, including backend, frontend, lints, e2e, and others, can be executed successfully within the webserver Docker container. To achieve this objective, Docker Compose can be employed, which enables all services within multiple containers to be connected. In addition, data consistency and persistence can be ensured by mounting shared volumes. To execute any test in the local development environment, the developer can leverage Docker Compose to spin up the necessary containers and execute the tests within the appropriate container. To run any test in the local development environment →

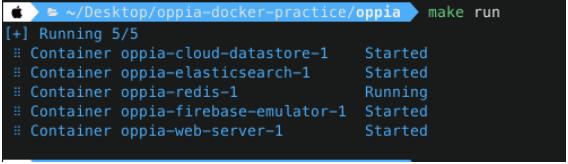
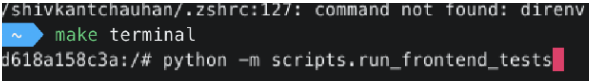
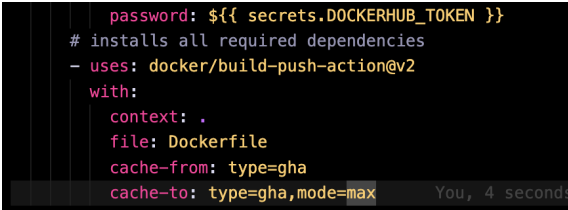
- 1) run "**make terminal**" to enter a terminal within the webserver container's environment.
- 2) use existing python scripts to run any test within the terminal session of the webserver container. example → "**python -m scripts.run\_lighthouse\_tests**"

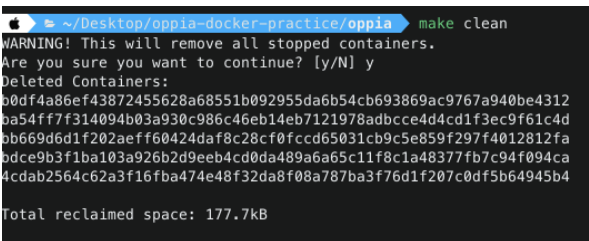
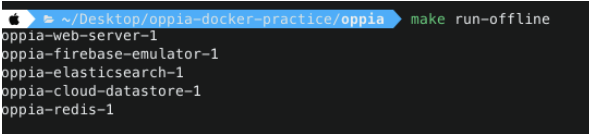
Additionally, we will consolidate all GitHub action workflows into a single build step that utilizes a Docker image to install all necessary dependencies. This approach will accelerate test runs on GitHub actions by caching the builds of Docker images.

# Key User Stories and Tasks

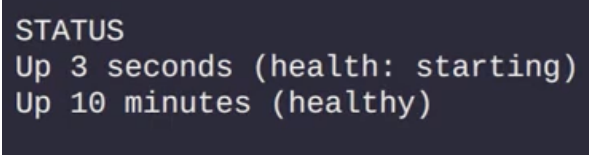
#	Title	User Story Description (role, goal, motivation) <i>"As a ..., I need ..., so that ...."</i>	Priority <sup>1</sup>	List of tasks needed to achieve the goal (this is the "User Journey")	Links to mocks / prototypes, and/or PRD sections that spec out additional requirements.
1	Setting-up development environment	As a developer, it is crucial to have a fully functional development server that is properly configured and can run all pages of the application without errors. This is essential for debugging, testing, and verifying the functionality of the application.	Must Have	User installs the Docker Desktop in the system.	 <p>Install Docker Desktop from <a href="#">here</a></p>
				Clone the 'oppia' repository	Clone the Oppia (web) from your forked repo using git commands in a parent folder (preferably name: opensource): <code>git clone https://github.com/{{GITHUB USERNAME}}/oppia.git`</code>
				Building and running the containers from the docker-compose file	<p>(building for the first time) →</p> <pre>[*] Building 24.34 (6/11) =&gt; [internal] load build definition from Dockerfile =&gt; =&gt; transferring dockerfile: 793B =&gt; [internal] load .dockerignore =&gt; =&gt; transferring context: 34B =&gt; [internal] load metadata for docker.io/library/ubuntu:20.04 =&gt; [auth] library/ubuntu:pull token for registry-1.docker.io =&gt; [1/6] FROM docker.io/library/ubuntu:20.04sha256:9fa38fce427e5e88c76bc41ad37d/cc573e1d79cec2b385e413c4be6476ab =&gt; =&gt; resolving docker.io/library/ubuntu:20.04sha256:9fa38fce427e5e88c76bc41ad37d/cc573e1d79cec2b385e413c4be6476ab =&gt; sha256:71281a4c55f72c33671fcb0076894f6a13a35f028f94f8c518967d052e1 4248 / 4248 =&gt; sha256:7ace798bb0cc2713e1a71ecdd634d0f2014f1b1298a8788355798332c4e9c9 2.32kB / 2.32kB =&gt; sha256:698ac8b3f45184648d74b08e8ba8f07ed1780c3994fa5c7e4980b8e3c488 25.97kB / 25.97kB =&gt; sha256:f1c28fcef4d32ce0e8706c41ad37d/cc573e1d79cec2b385e413c4be6476ab 1.12kB / 1.12kB =&gt; extracting sha256:698ac8b3f45184648d74b08e8ba8f07ed1780c3994fa5c7e4980b8e3c488 =&gt; [internal] load build context =&gt; =&gt; transferring context: 792.489B =&gt; [2/6] RUN apt update &amp;&amp; apt install -y curl git unzip &amp;&amp; apt install openjdk-8-jre &amp;&amp; apt =&gt; # publicsuffix unzip kauth =&gt; # &amp; upgraded, 35 newly installed, 0 to remove and 0 not upgraded. =&gt; # Total space used: 11.1MB of 14GB available</pre> <p>Run <code>make install-dependencies</code> to install the required dependencies</p>

<sup>1</sup> Use the **MoSCoW** system ("Must have", "Should have", "Could have"). You can read more [here](#).

				Run <code>'make run'</code> command to run the application in the development environment.	(running with the cached docker images) → 
2	Using git	As a developer, it is essential to have a properly executed version of Git and its associated commands to facilitate tasks such as pushing changes for opening pull requests and updating local forks with the origin repository.	Must Have	Run <code>'make terminal'</code>  Execute the standard Git workflow of staging, committing, and pushing the changes from the local to the remote Oppia repository using the appropriate Git commands.	
3	Running tests	As a developer, it is necessary to execute various tests (including end-to-end, backend, TypeScript tests, etc.) on the local development environment to ensure the correctness of code changes and avoid any test failures.	Must Have	Run <code>'make terminal'</code>  Run the standard commands to start the tests inside the docker terminal.	Users must run tests with existing Python scripts using the same commands as before. The only difference is to run those commands within the running webserver container. The command for this is: <code>'make terminal'</code> and then run the existing command to run the test. Example: <code>'python -m scripts.run_frontend_tests'</code>  
4	Migrating GitHub Action workflows	As a developer, I need to have all the checks passed on the GitHub actions so that the changes do not cause any failure in the tests.	Must Have	All GitHub action workflows utilize dockerfile and cache the image, in their build step (instead of the python scripts)	 <pre> password: \${ secrets.DOCKERHUB_TOKEN }} # installs all required dependencies - uses: docker/build-push-action@v2   with:     context: .     file: Dockerfile     cache-from: type=gha     cache-to: type=gha,mode=max </pre> Sample workflow file, having just 1 build step that uses a dockerfile, and cache the docker image,

				to install the required dependencies.	installing all the required dependencies for that workflow.
5	Clean the entire Oppia setup	As a developer, it is necessary to perform the task of cleaning up the Oppia setup in the event of data clearance or removal of locally created setups in multiple containers.	Must Have	Run the command <code>make clean</code> to clean the entire setup by deleting all containers.	 <pre> ~/Desktop/oppia-docker-practice/oppia make clean WARNING! This will remove all stopped containers. Are you sure you want to continue? [y/N] y Deleted Containers: b0df4a86ef43872455628a68551b092955da6b54cb693869ac9767a940be4312 ba54ff7f314094b03a930c986c46eb14eb7121978adbcced4dcd1f3ec9f61c4d bb669d6d1f202aeff60424daf8c28cf0ccd65031cb9c5e859f297f4012812fa bdce9b3f1ba103a926b2d9eeb4cd0da489a6a65c11f8c1a48377fb7c94f094ca 4cdab2564c62a3f16fba474e48f32da8f08a787ba3f76d1f207c0df5b64945b4 Total reclaimed space: 177.7kB </pre>
6	Run the Oppia development server in offline mode	As a developer, I many times need to start the Oppia development server without checking/building the dependencies	Must Have	Run command <code>make run-offline</code>  This will start the previously build and stopped containers for the Oppia development server, and thus no need for internet access.	 <pre> ~/Desktop/oppia-docker-practice/oppia make run-offline oppia-web-server-1 oppia-firebase-emulator-1 oppia-elasticsearch-1 oppia-cloud-datastore-1 oppia-redis-1 </pre>
7	Update the required dependencies	As a developer, it is necessary to install the dependencies specified in the dependency-specification files (package.json) with their exact version to maintain consistency and avoid any compatibility issues.	Must Have	Assuming that your forked repository is up-to-date with the upstream repository, executing the command <code>make update-pip-and-npm-packages</code> will install all the dependencies, as listed in the dependency specification file, with their exact versions.	



8	Healthcheck for the containers running for Oppia development server	As a developer, I need the server operating seamlessly, without any interruption or complications, for a smooth working experience. Thus, all the containers must run in a 'healthy' state.	Must Have	Run <code>docker ps</code> to check the health condition of the running containers. The configurations for the healthchecks in the compose file will allow users to check the health state in the 'Status' section after running the <code>docker ps</code> command.	 <p>similar to this, the health state can be checked from the 'STATUS' section after running <code>docker ps</code> command.</p>
---	---	---	-----------	--	--

## Other Requirements

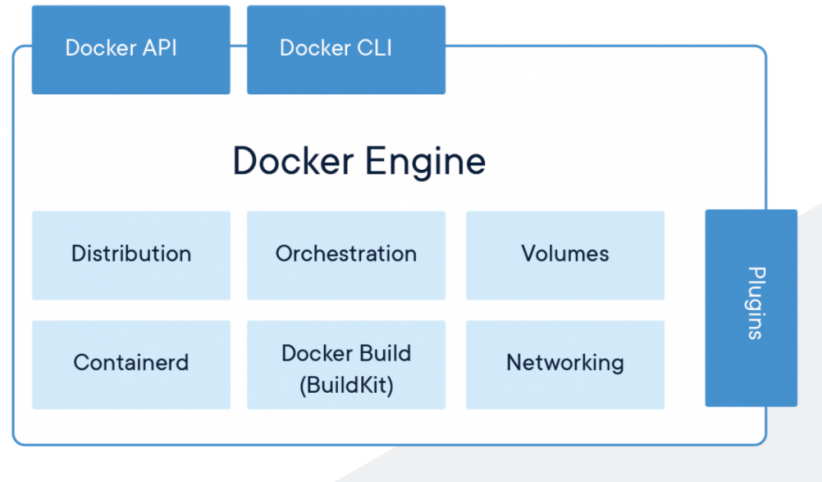
For the configuration of the Oppia development server, it is necessary to have Docker Desktop installed on our system. Additionally, within our docker-compose file, we will be retrieving certain images published on Docker Hub in order to execute the application in multiple containers.

### 1) Docker Desktop (preferably [4.16.2](#))

Docker Desktop is a tool for building, running, and managing Docker containers. It includes **Docker Engine and Docker Compose, pre-installed within it**. Docker Desktop simplifies the process of running Docker containers by providing a user-friendly interface and a complete development environment for building, testing, and deploying Docker containers on the local machine.

- **Docker Engine (v20.10.23)**

Docker Engine allows developers to create, deploy, and run applications in isolated containers. It provides a runtime environment for containers. Docker Engine uses a **client-server architecture** where the client provides a command-line interface for interacting with the server.



(image credits: [ducmanhphan](#))

- **Docker Compose (v2) – [compose file version (^3.4)]**

Docker Compose is a tool for defining and running multi-container Docker applications by defining the services and dependencies for the application in a YAML file and then starting them all with a single command “docker-compose up”. This makes it easier to manage applications that require multiple containers.

- **Dockerfile(s)**

A Dockerfile provides instructions for building a Docker image. It automates the process of creating a Docker image by defining the base image, application code, configurations, and dependencies required to run the application. We need to create the following dockerfiles →

- **Webserver**  
(dockerfile to install the dependencies and configure the main webserver)
- **Firestore Emulator**  
(dockerfile to configure firestore emulator - auth)

**The versions mentioned here (for Docker Desktop, Docker Engine, and Docker Compose file) are the base versions we can use, and it would be recommended to use the latest versions to ensure maximum compatibility and feature support.**

## 2) DockerHub Images

Docker Hub images are pre-built images that can be easily searched and downloaded using the "docker pull" command. Docker Hub also allows developers to create and publish their own Docker images, making it easier to share their applications with others.

List of DockerHub Images we will be using →

Service Name (with link to the Docker Hub image)	Image version to be used (according to what we use in Oppia's codebase)	Image Type
<a href="#">Redis</a>	redis: 7.0-alpine	Official Docker Image
<a href="#">Google Cloud SDK</a>	google/cloud-sdk: 364.0.0	Verified Docker Image by Google
<a href="#">ElasticSearch</a>	elasticsearch: 7.17.0	Official Docker Image

---

## Section 3.2: HOW

### Existing Status Quo

The current installation process for our application is a significant obstacle for users, often leading to frustration and difficulty due to the intricate set of prerequisites that must be followed with precision. Any minor error or oversight in the installation process can result in major problems and errors, making it especially challenging for new contributors to navigate and troubleshoot. The existing installation process requires users to carefully follow a series of steps, including:

- 1) Installing prerequisites packages (python3-pip, python3-setuptools, curl, openjdk-8-jre, git, python3-dev, python3-yaml, python3-matplotlib, unzip, libbz2-dev) and other pyenv pre-required packages.
- 2) Installing correct python versions (2 and 3.8.15)
- 3) Cloning Oppia's repo
- 4) Setting up the virtual environment (using pyenv)

- 5) Setting up the virtual-env configurations and adding pyenv environment variables.  
(These 2 steps are very error-prone and takes long debugging time if user made even a small mistake)
- 6) Start the development server by running up the **start.py** script

Currently, the use of pyenv to set up a virtualenv only encapsulates Python dependencies, which are stuck with the host OS. Unfortunately, this falls short of our requirements since we need many Python scripts to run seamlessly behind the scenes. As a result, these factors contribute to a challenging installation process, making it a painful experience to debug and not user-friendly to set up.

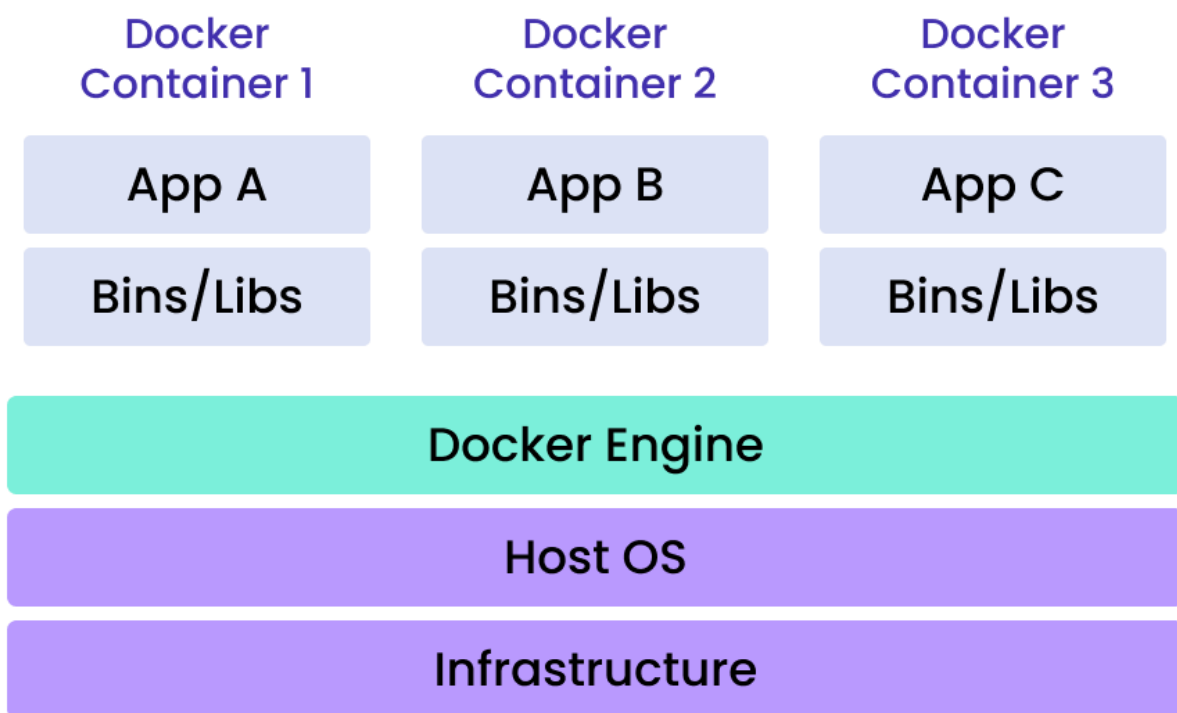
**Check out the current process for setting up the Oppia development server, with all the required dependencies and services – [link](#)**

## Solution Overview

To simplify and optimize our installation process and startup procedure for the local development server, we will use Docker Compose to bundle our application into multiple containers (microservices). Additionally, we will utilize the docker images in our GitHub workflows to accelerate testing in GitHub Actions, to install the required dependencies. This will ultimately increase our efficiency and reduce the overall load on the hosted Github servers.

### **Packaging the application - Docker**

Docker is an open platform for developing, shipping and running applications. Docker will enable us to separate the application from the infrastructure of the system delivering software quickly and in an error-resistant way, using OS-level virtualization.



## Docker Compose

Docker Compose runs the multi-container Docker application creating multiple Docker images for each service so developers can establish a consistent environment and streamline the installation process of the dependencies in both the development environment and Github Action workflows.

## Dockerizing Oppia Server (overview)

To run the application, we'll utilize Docker containers to build and start the development server. Each requisite service will be allocated **separate docker images**. Docker images will contain all necessary dependencies and libraries for each service, eliminating the need for prerequisite steps and external environment configuration. Using docker-compose, we can seamlessly combine, build, and launch all services with a single command: "docker-compose up -d". Docker

compose also makes it easy to manage communication between containers by sharing them on the same network. By **sharing the same network**, containers can communicate with each other using their unique IP addresses or container namespaces, making communication between containers both efficient and easy to manage. Data persistence can be achieved through the use of **shared directories or Volumes**. By storing data on the host machine, it becomes persistent and accessible to other containers in the future. Services can be configured to use shared volumes, allowing them to easily access and maintain the same data, ensuring consistency across all services.

Once the application has been containerized with Docker, the subsequent steps can be followed to setup the local development server →

- 1) Install Docker Desktop – [link](#)
- 2) Fork and Clone the Oppia (web) repo – [link](#)
- 3) Run `make install-dependencies``
- 4) Run `make run``

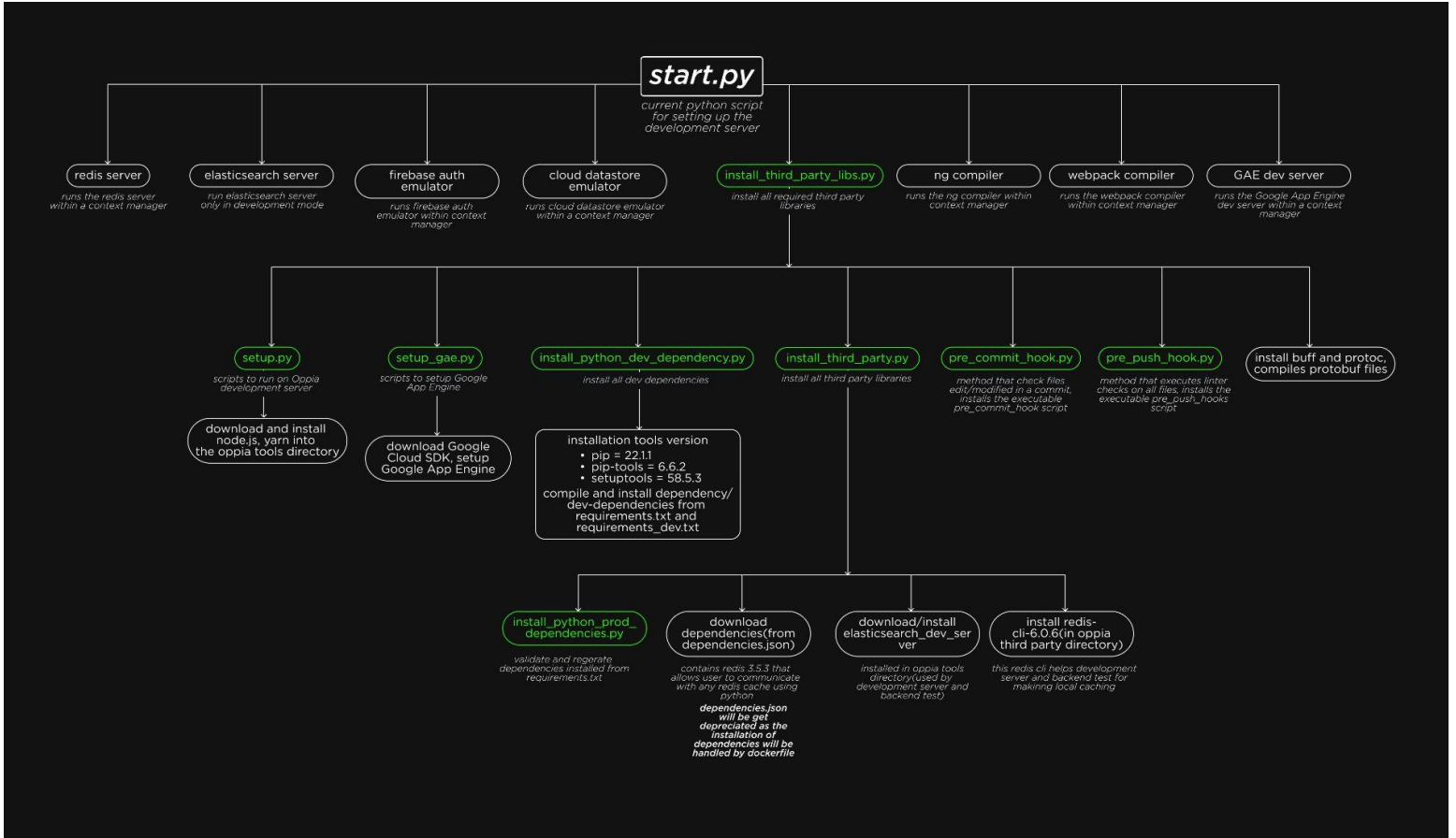
This greatly simplifies the installation process as compared to the current installation process, which is often arduous and troublesome to diagnose.

Docker and Makefile will manage all dependencies for the application. The existing dependency management logic, which is reliant on Python scripts, will become deprecated. Check the detailed explanation [here](#).

## **All required dependencies/services required for Oppia development server**

Check flow for the current Oppia development server installation, and all the required dependencies and services – [link](#)

(please refer to this link for better image resolution)



(Check the images on figma: [link](#))

This is the flow, how the current Oppia development server starts and installs all the required services and

## dependencies (listed in the flowchart and below)

### *prerequisites tools/packages*

```
python2
curl
git
openjdk-8-jre
python3-setuptools
python3-dev
python3-pip
python3-yaml
python-matplotlib
python3-matplotlib
pip==21.2.3
unzip
```

### *Third party libraries/dependencies*

#### *requirements.txt*

```
apache-beam[gcp]==2.38.0
beautifulsoup4==4.10.0
bleach==4.1.0
certifi==2021.5.30
deepdiff==5.8.1
defusedxml==0.7.1
elasticsearch==7.17.0
firebase-admin==5.2.0
futures==0.18.2
googledatastore==7.0.2
google-cloud-storage==2.1.0
google-auth==1.35.0
google-cloud-dataflow-client==0.3.1
google-cloud-logging==3.0.0
google-cloud-ndb==1.11.1
google-cloud-secret-manager==2.12.4
google-cloud-tasks==2.7.2
google-cloud-translate==3.6.1
gunicorn==20.1.0
html5lib==1.1
mailchimp3==3.0.15
mutagen==1.45.1
pillow==9.0.1
pylatexenc==2.10
pytest==6.2.5
PyYAML==6.0
redis==3.5.3
requests==2.26.0
requests-mock==1.9.3
requests-toolbelt==0.9.1
result==0.6.0
rsa==4.7.2
simplejson==3.17.5
six==1.16.0
soupsieve==2.3.1
typing-extensions==3.10.0.2
urllib3==1.26.7
webapp2==3.0.0b1
webencodings==0.5.1
```

#### *requirements\_dev.txt*

```
requirements_dev.txt -->
Pillow==9.0.1
PyGithub==1.55
certifi==2021.10.8
coverage==6.1.2
esprima==4.0.1
future==0.18.2
grpcio==1.41.1
isort==5.10.1
protobuf==3.13.0
psutil==5.8.0
pycodestyle==2.8.0
pylint==2.11.1
pylint-quotes==0.2.3
pyyaml==6.0
rcssmin==1.1.1
six==1.16.0
typing-extensions==4.0.1
webtest==3.0.0
xmldict==0.13.0
```

### *dependencies.json*

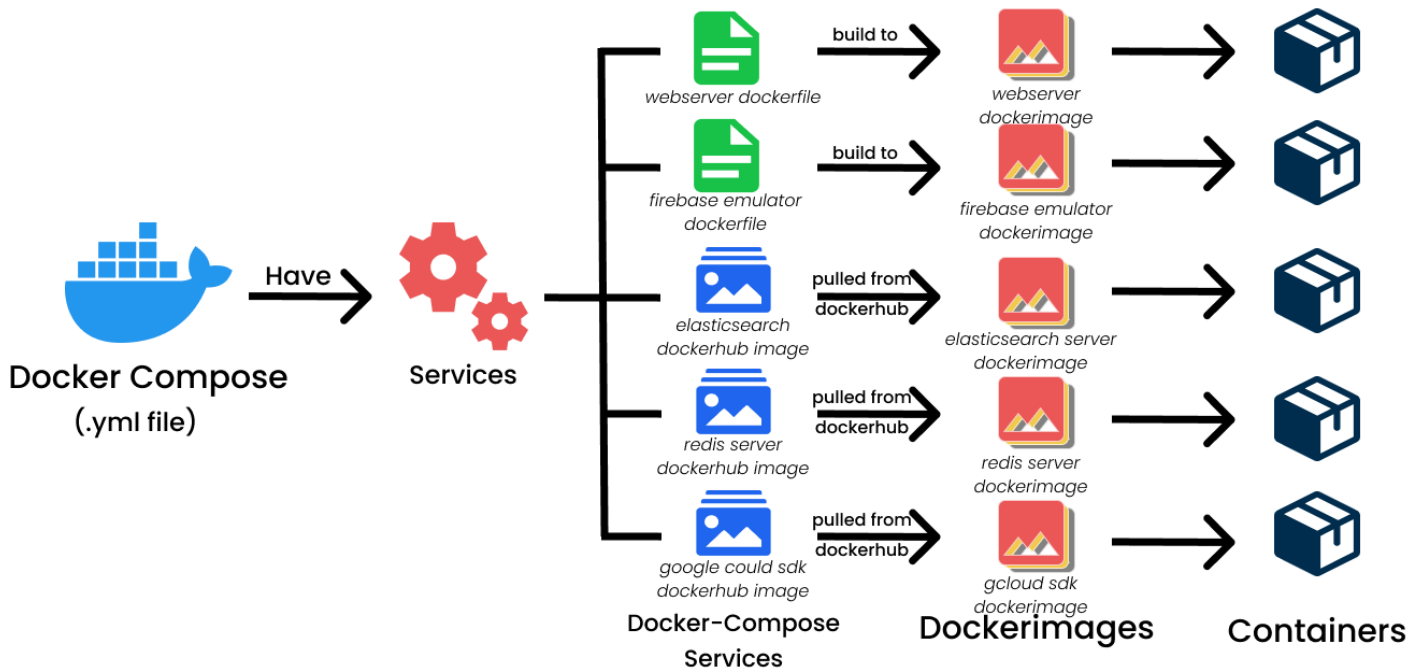
```
oppiaMIProto==0.0.0
angular==1.8.2
lamejs==1.2.0
angularTest==1.8.2
angularTranslate==2.18.1
angularTranslateLoader==2.18.1
angularDragAndDrop==2.1.0
angularStorageCookiesRev==2.18.1
messageFormat==2.0.5
angularTranslateInterpolationMessageFormat==2.18.1
popper.js==1.15.0
bootstrap==4.3.1
bowerAngularTranslateLoaderPartial==2.18.1
bowerMaterial==1.1.19
codemirror==5.17.0
diff-match-patch==1.0.0
fontAwesome==5.9.0
angularUILeaflet==1.0.3
leaflet==1.4.0
angular-simple-logger==0.1.7
guppy==f509e155dc66f6310737c328cbfbc35b85194be2
jquery==3.5.1
jqueryUI==1.12.1
jqueryUITouchPunch==0.3.1
mathJax==2.7.5
midi.js==c26ebb9baaf0abc060c8a13254dad283c6ee7304
select2==4.0.3
ckeditor==4.12.1
ckeditorBootstrapCK==1.0.0
uiBootstrap==2.5.0
skulpt-dist==1.1.0
uiCodemirror==5d04fa5c991f915e4578be5f1175e22d94696633
uiTree==
```

Through the process of dockerizing our application, we will be deprecating various Python scripts that are necessary for installing the required dependencies and libraries, as well as the dependencies.json file.

Thus, the packages from `dependencies.json` will be moved to `package.json` or will be added to the Oppia that can be downloaded while forking, this will do all the dependency management within a single file.



## docker-compose.yml structure:



Docker Compose is a tool for defining and running multi-container Docker applications. It allows developers to define a group of Docker containers as a single application and specify how they should interact with each other.

- **Version:** docker-compose file version to use. Preferably 3.0+ for docker-compose v2
- **Services:** The services that make up the application, each defined as a separate block with a unique name. Each service specifies the docker-image to use, the command to run, entrypoints, service depends-on, network, and any required environment variable, volumes and ports.
- **Volumes:** The volumes that the services use to store data, each defined as a separate block. Volumes persist the data between containers, and share the data between the services.

### -> **Services required:**

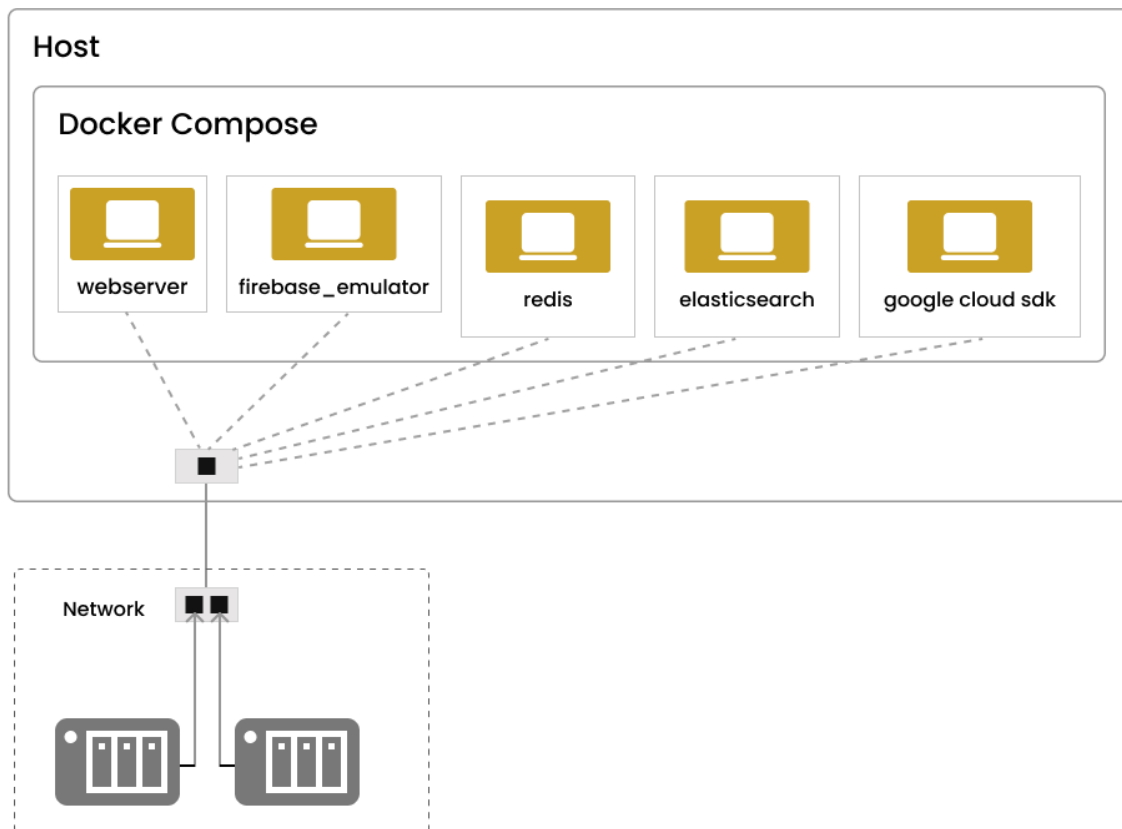
1. [Redis server - 7.0](#)
2. [Elasticsearch dev server - 7.17.0](#)
3. [Firebase emulator](#)

4. [Google Cloud SDK - 364.0.0](#)
5. [App development server](#)

The settings and environments can be configured within the image (or within in the docker-compose.yml file, if we use the official docker image), eliminating the need for developers to perform any additional configuration when setting up their development environment

## Docker Networking

Docker has a DNS server embedded with it that maintains a mapping of namespaces and IP addresses of the containers. When an application runs with docker-compose, it will automatically create a network and add containers to that network, so that the running containers can talk to each other over this virtual network bridge.



By default, docker compose creates a **bridge network** for the “docker-compose.yml” file, and **attaches all services to that network** – (this diagram depicts the container networking within a shared virtual network bridge). However, we can also define multiple networks in the docker compose file, and can specify which service should be connected. **But here, we will be handling the container interactions within the same virtual network** i.e. the default behavior of the compose file.

**To enable services running within Docker containers to be accessible on the host network, I will configure and publish the ports in the compose file. This will bind the services from the exposed ports of the containers to execute on the host network.**

Referring to our docker-compose.yml file ([link](#)), all of the services will be connected within a shared virtual network, so we can simply use our service name to communicate with other services. These services’ names are: webserver, redis, firebase-emulator, elasticsearch, and cloud-datastore.

## **Using git command for pushing changes:**

In the present scenario, when pushing changes from the local repository to the remote Oppia repository, a series of checks/tests are triggered, including frontend tests, lint checks, typescript checks, and other tests. The Git command is executed only if all of these tests pass successfully.

Once the application has been containerized with Docker, it is possible to run the standard Git commands within the Docker containers. It is essential to execute these Git commands within the Docker container, [as all the tests must run without failing](#) before any changes are pushed. The tests rely on all necessary dependencies and services, and executing the Git commands within the container ensures that these components are present and functional during testing.

The users must not execute the git command for pushing their changes outside the container. If they do, they should get a warning displayed in the terminal saying that they should run the git commands with the Docker Container environment using `make terminal` command. I will ensure that this happens by defining a new environment variable – `oppia-is-dockerized` (its

value will be true when inside the docker container environment) within the container to determine whether or not we are running inside the container environment. This environment variable can be subsequently accessed and checked within the `pre_push_hook` scripts. If the intended value of the environment variable is detected, the corresponding git command can be executed using the aforementioned hooks. Conversely, an error can be thrown, instructing users to execute git commands inside the container.

## Running all flags of `start.py` in the dockerized setup

Devs need to run all the flags that they currently have for the `start.py` in the dockerized setup of the Oppia development server. The flags we currently have with the `start.py` script –

- `save_datastore`
- `disable_host_checking`
- `maintenance_mode`
- `prod_env`
- `no_browser`
- `no_auto_restart`
- `source_maps`

All these flags will be working the same in the dockerized setup except the `no_browser` flag. This flag does not open a browser if specified. As we dockerize our application, it would be the default behavior (seen in all dockerized applications) that the browser window is not opened automatically for the localhost. So we will be dropping this `no_browser` flag in the dockerized setup.

For implementing all the other flags in our dockerized application, we can have all the flags of `start.py` as the environment variables for the webserver container environment. Devs will pass the env variable value in the make command, and this sets those values for the env variables that are passed. For example, if devs want to use `save_datastore` and `prod_env` flag, then command will be:

```
`make run prod_env=true save_datastore=true`
```

and the env variables will have the values passed by the dev (by default, all env variables will be false). We can access all these env variables in our webserver dockerfile, and can run the `build.py` script as per our requirements from the webserver dockerfile (NOTE: we are not

deprecating the build.py within this project),

example – if `prod\_env` flag is true, we will run a command `python -m scripts.build --prod\_env` in the webserver dockerfile that will execute the flag.

The environment variables we will be having for the webserver container will be specified in a `.env` file (inside the /docker directory), this file will be having the whole list of the environment variables that we will be using –

env variable	default value	description
oppia-is-dockerized	true (value shouldn't be changed by devs)	This boolean will specify whether we are inside the docker container environment or not.
save_datastore	false	Does not clear the datastore if specified
disable_host_checking	false	Disables the host checking if true so that the dev server can be accessed by any device on the same network using the host device's IP.
prod_env	false	Runs Oppia in prod environment if this flag is used.
maintenance_mode	false	Puts Oppia in maintenance mode
no_auto_restart	false	If flag is used, does not automatically recompile the webpack on file changes
source_maps	false	Builds webpack with source maps if this flag is specified.

**NOTE:** add comment in this .env file explaining why not to directly change the value of the variables in this file, especially for the `oppia-is-dockerized`, otherwise errors might occur within the dockerized Oppia setup. The actual comment can be as follows:

NOTE TO DEVELOPERS: Please do not directly change the values of the variables specified in this file, as this may lead to unexpected behavior of your local development server. Specifically:

- Setting ``oppia-is-dockerized`` to false will prevent the proper execution of the `git push` command and the local tests within the dockerized setup.

- Additionally, changing other variables (which are flags for running the local development server) to true will permanently enable the flag to your ``make run/run-offline`` command even you do not explicitly pass the flag.

## **How we will proceed with the overall project**

To achieve project completion, a series of detailed steps and activities will be employed, with each step thoroughly elaborated and documented →

1. To configure all the services required for our application, we will generate a `docker-compose.yml` file. This file will specify the various components of our application, along with their configurations and dependencies.
2. Create the required dockerfiles for the services (for the webserver, and for the services whose reliable docker hub images are not available), migrate all dependencies from `dependencies.json` to `package.json`
3. Add the configuration to verify the dependency installation through checksums in webserver dockerfile, migrate dependencies from ``dependencies.json``, add healthchecks for containers in `docker-compose.yml` file.
4. Configure a Makefile that provides users with commands to build, run, update, or clean the Oppia development server setup.
5. Making all tests run properly in the dockerized application, within a container environment.
6. Migrating all GitHub actions to use a single build step that utilizes docker image, which is cached for use of other runs.
7. Deprecating the existing python scripts used for installing dependencies, start services and configure the local development server.

# 1. Configuring docker-compose.yml file

```
docker-compose.yml
You, 1 second ago | 1 author (You)
1  version: "3.4"
2
3  services:
4    webserver:
5      platform: linux/amd64
6      container_name: oppia-webserver
7      build:
8        context: .
9        dockerfile: Dockerfile
10     ports:
11       - "8080:8080"
12     volumes:
13       - ./app/oppia
14       - /app/oppia/node_modules
15     environment:
16       - NODE_ENV=development
17     depends_on:
18       - redis
19       - google_cloud_sdk
20       - elasticsearch
21       - firebase_emulator
22
```

```

19
20  firebase_emulator:
21  build: | You, 14 seconds ago • Uncommitted changes
22  context: .
23  dockerfile: Dockerfile.firebase_emulator
24  ports:
25  - "9099:9099"
26  volumes:
27  - ./app/firebase_emulator_cache
28
29  redis:
30  image: redis:7.0-alpine
31  ports:
32  - "6379:6379"
33  volumes:
34  - ./app/firebase_emulator_cache
35
36  google_cloud_sdk:
37  image: google/cloud-sdk:364.0.0
38  environment:
39  - DATASTORE_PROJECT_ID=dev-project-id
40  - DATASTORE_LISTEN_ADDRESS=0.0.0.0:8089
41  ports:
42  - "8089:8089"
43  - "8080:8080"
44  - "8000:8000"
45  volumes:
46  - ./app/cloud_datastore_emulator_cache
47  command: gcloud beta emulators datastore start --consistency=1.0 --platform linux/amd64 --quiet
48
49  elasticsearch:
50  image: elasticsearch:7.17.0
51  ports:
52  - "9200:9200"
53  volumes:
54  - ./usr/share/elasticsearch/data
55  restart: unless-stopped
56

```

Check the configuration/setting up of webserver and firebase dockerfiles from here: [link](#)

A Docker Compose file describes the configuration of multiple Docker containers and how they interact with each other to form a single application. It allows to define and run a multi-container Docker application as a single entity, making it easier to manage complex applications. Services we are setting up in the docker-compose →

- a. **webserver:** This docker image will install and configure dependencies/libraries (listed [here](#)) required to compile the source code and to start the development server.
  - configured within a separate dockerfile



- exposed port: 8181

**b. firebase\_emulator:** The service we need to link the Firebase emulator to our application in order to enable authentication.

- configured within a separate dockerfile.
- exposed port: 9099

**c. redis:** The service we need to locally cache data from the development server and backend tests.

- configured using official docker hub image: [redis:7.0-alpine](#)
- exposed port: 6379

**d. google\_cloud\_sdk:** used for many services and tools in our application like gcloud (command-line tool for managing the GCP resources and services), Google App Engine (fully managed, serverless platform for developing and hosting web applications), Google Cloud Datastore (fully managed NoSQL database service).

- configured using verified docker hub image: [google/cloud-sdk:364.0.0](#)
- exposed ports:
  1. 8089 - cloud datastore
  2. 8080 - app engine
  3. 8000 - GAE admin server

**e. elasticsearch:** service providing distributed and analytics search engine allowing to store, search, and analyze huge volumes of data quickly in milliseconds.

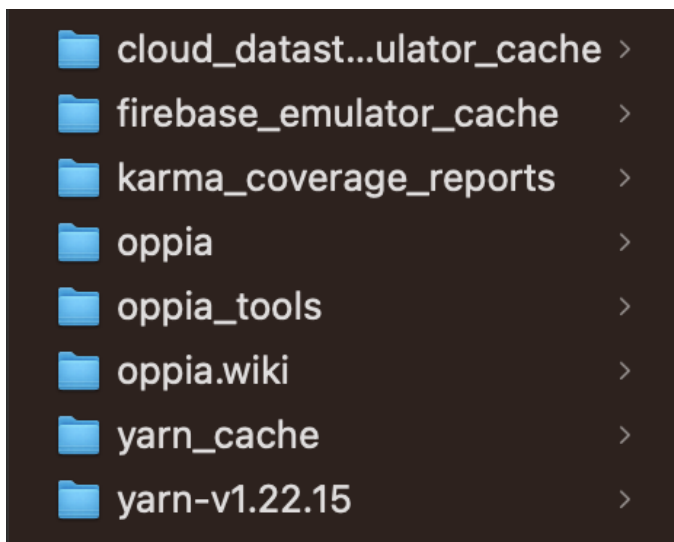
- configured using official docker hub image: [elasticsearch:7.17.0](#)
- exposed port: 9200

## Mounting Volumes → persisting and sharing data with multiple running containers

Volumes: is a way to store and manage persistent data outside of a container's file system.

Volumes provide a way for containers to share and persist data even after the containers are deleted or recreated. They can be used to store application data, configuration files, and logs, among other things.

Current file structure:



main folder (let's say: opensource)

---

```
opensource/  
├── cloud_datastore_emulator_cache  
├── oppia (main folder - cloned from https://github.com/oppia/oppia)  
├── firebase_emulator_cache  
├── karma_coverage_reports  
└── oppia_tools
```

---

Similar file structure we will be following with the dockerized containers within the `app` as the working directory... Such as:

---

containers shared working directory (WORKDIR): app

```
app/
|
|— oppia (root)
|   |
|   |—
|   |—
|   |—
|
|— cloud_datastore_emulator_cache
|— firebase_emulator_cache
|— karma_coverage_reports
```

---

### Volumes Mounted for different Services:

Services	directory mapped in container
1. Application files (source code)→	/app/oppia/
2. Firebase Emulator	→ /app/firebase_emulator_cache
3. Google Cloud Datastore	→ /app/cloud_datastore_emulator_cache
4. Redis Server	→ /app/redis_cache
5. Elasticsearch Server	→ /app/elasticsearch_emulator_cache

The file structure for the dockerfiles (Dockerfile and Dockerfile.firebase) and `docker-compose.yml` that will be integrated into our code base →

---

```
oppia/
|
|—
|— docker (new directory)
|   |— Dockerfile
|   |— Dockerfile.firebase
|   |— docker-compose.yml
|   |— .env
```

---

**NOTE:** We need to ignore the `docker` directory from being uploaded to production during releases. So, I will add this docker directory path to the `.gcloudignore` file in order to ignore this directory during releases. Refer [#here](#)

## 2. Setting up dockerfiles and consolidating dependencies into a single file for more efficient management

These are the dockerfiles we will use for configuring the webserver for installing all the dependencies and starting the local dev server, and the firebase emulator for authentication.  
(structure for the webserver dockerfile)

```
oppia > Dockerfile > ...
You, 1 second ago | 1 author (You)
1 FROM python:3.8
2
3 WORKDIR /app/oppia
4
5 # installing the pre-requisites
6 RUN apt-get update
7 RUN apt-get -y install curl
8 RUN apt-get -y install git
9 RUN apt-get -y install software-properties-common
10 RUN apt update
11 RUN add-apt-repository ppa:webupd8team/java
12 RUN apt-get -y install openjdk-8-jre
13 RUN apt-get -y install python3-dev
14 RUN apt-get -y install python3-setuptools
15 RUN apt-get -y install python3-pip
16 RUN apt-get -y install unzip
17 RUN apt-get -y install python3-yaml
18 RUN apt-get -y install python3-matplotlib
19 RUN pip install --upgrade pip==21.2.3
20
21 # installing python dependencies from the requirements.txt file
22 COPY requirements.txt .
23 COPY requirements_dev.txt .
24
25 RUN pip install --require-hashes -r requirements.txt
26 RUN pip install --require-hashes -r requirements_dev.txt
27
28 ## installing packages from the package.json file
29 COPY package.json .
30 RUN npm install --legacy-peer-deps
31
32 RUN apt-get -y install python2
33 COPY . .
34
35 EXPOSE 8181
36
37 CMD [ "node_modules/.bin/ng", "serve", "--host", "localhost" ]
38
```

**NOTE:** Here, we are dropping the python2 package for matplotlib as we are required to migrate to python3. Please refer [#here](#).

### Firestore-auth-emulator dockerfile structure:

```
Dockerfile-firebase > ...
1 FROM node:14.15.4-alpine3.10
2
3 RUN npm install -g firebase-tools
4
5 WORKDIR /app
6
7 COPY firebase.json .
8 COPY firestore.rules .
9 COPY firestore.indexes.json .
10
11 ENV FIREBASE_AUTH_EMULATOR_HOST="0.0.0.0:9099"
12 ENV OPPIA_PROJECT_ID="dev-project-id"
13
14 EXPOSE 9099
15
16 CMD ["firebase", "emulators:start", "--only", "auth"]
17
```

Additionally, we will migrate the dependencies from `dependencies.json` to `package.json`, This will ensure that dependency management is handled through a single file.

### Dependencies Management for the application

Developers can efficiently modify the local setup by updating the dependencies whenever necessary. This is facilitated by pinning all the dependencies to a specific version for consistency. Users can effortlessly update their forks by pulling changes from the upstream Oppia repository and running the `make update-pip-and-npm-packages` command. This command installs all the dependencies listed in the `package.json`, `requirements.txt`, or `requirements_dev.txt` files, along with their precise versions specified in these dependency-specification files.

### 3. Verifying checksum of dependencies, migrating dependencies and integrating healthchecks for containers

#### Verify the checksum of the dependencies

Checksum verification for dependencies is important because it helps ensure the security and reliability of the software you are developing or deploying. When you install a dependency, you trust that the code you are installing is legitimate and has not been tampered with. A checksum is a hash value calculated from the contents of the file, and any modification to the file would result in a different checksum. By comparing the expected and actual checksums, you can detect whether the file has been modified or tampered with.

For installing the dependencies from `package.json`, we already have [`yarn.lock`](#) file which have checksum for each dependency in the `package.json`, and the checksum verification can be done while installing the dependencies using yarn.

Other than that, we install python dependencies from `requirements.txt` and `requirements\_dev.txt` using pip. And pip-compile already have a flag

`--generate-hashes` which generates the hashes using sha256 for each dependency in the `requirements.txt` and `requirements\_dev.txt` files while compiling them. The command we can use to generate hashes is –

```
`pip-compile --generate-hashes requirements.in`  
`pip-compile --generate-hashes requirements_dev.in`  
After we have generated the hashes for each dependency, we need to verify the  
checksum for each dependency which we can verify while installing the dependencies  
from these files using `--require-hashes` flag, this will enable the hash checking  
mode while installing the dependencies –  
`pip install --require-hashes -r requirements.txt`  
`pip install --require-hashes -r requirements_dev.txt`
```

This process of performing checksum verification on the dependencies being installed guarantees that the packages being installed in Docker conform precisely to the Maintainer's specifications for other developers' installation, with no alterations or tampering occurring during transit.

For ensuring that the checksum verification methods works fine, I will pass a wrong checksum

for a dependency and will show that the installation is blocked if wrong hashes are there as a proof that the checksum verification mechanism works fine.

## Migrating dependencies from `dependencies.json`

We need to migrate the dependencies from the `dependencies.json` file, and add them to the `package.json` file. Within the `dependencies.json` file, there are 2 types of dependencies, one is those which are available via NPM, and the rest aren't available via NPM, we need to install them from their git repositories. Referring to the table below, we can easily identify which dependencies can be installed via NPM, or can be installed from their git commit hashes.

dependencies.json	Version	Source
angular	1.8.2	npm ▾
lamejs	1.2.0	npm ▾
angularTest	1.8.2	npm ▾
angularTranslate	2.18.1	npm ▾
angularTranslateLoader	2.18.1	npm ▾
angularDragAndDrop	2.1.0	npm ▾
angularStorageCookiesRev	2.18.1	npm ▾
messageFormat	2.0.5	npm ▾
angularTranslateInterpolation MessageFormat	2.18.1	npm ▾
popperJs	1.15.0	git commit hash ▾
bootstrap	4.3.1	npm ▾
bowerAngularTranslateLoader Partial	2.18.1	npm ▾
bowerMaterial	1.1.19	npm ▾
codemirror	5.17.0	npm ▾

diff-match-patch	1.0.0	npm ▾
fontAwesome	5.9.0	npm ▾
angularUiLeaflet	1.0.3	npm ▾
leaflet	1.4.0	npm ▾
angular-simple-logger	0.1.7	npm ▾
guppy		git commit hash ▾
jquery	3.5.1	npm ▾
jqueryUI	1.12.1	npm ▾
jqueryUITouchPunch	0.3.1	git commit hash ▾
mathJax	2.7.5	npm ▾
midiJs		git commit hash ▾
select2	4.0.3	npm ▾
ckeditor	4.12.1	git commit hash ▾
ckeditorBootstrapCK	1.0.0	git commit hash ▾
uiBootstrap	2.5.0	npm ▾
skulpt-dist	1.1.0	git commit hash ▾
uiCodemirror		git commit hash ▾
uiTree	2.22.6	npm ▾

Currently, the dependencies from `dependencies.json` file are installed within `/third_party/static/` folder, and after we migrate the dependencies to the `package.json`, the dependencies will be installed within the `/node_modules/`. So, we are moving these dependencies to a new location and thus we need to change the import statements everywhere we are importing the migrated dependencies.

There is no problem in migrating the packages which can be installed via NPM, we can simply add them to the `package.json` with the exact version we currently utilize in Oppia. (Need to change the import statements in the files where we will be importing the packages since we are moving the packages to a new location).



For the packages that need to be installed from their git repositories, instead of forking them to the Oppia organization, we can follow –

We can specify the GitHub repository path with the commit SHA in the package.json along with the package we need to install. For example –

```
"dependencies": {  
  ....  
  ....  
  "packageName": "git@github.com:{owner}/{project}.git#commitSHA"  
}
```

(Need to change the import statements in the files where we will be importing the packages since we are moving the installed packages to a new location - `/node_modules/`).

However, there is a tricky situation here. On checking the `dependencies.json` file, I observed that there are several packages where we only need to download specific file(s) rather than the entire GitHub repository for those packages. Unfortunately, NPM does not have built-in support for installing files from packages using the package.json file. To address this, the solution is to download the complete GitHub repository for these packages (from which we only need certain files) into the /node_modules/` directory. And this can be achieved similarly as we will do for the other packages by specifying the commit hashes in the package.json` file. There are many packages for which we just download some files –`

angularTest	angularTranslate	angularTranslateLoader
diff-match-patch	angularUiLeaflet	angularStorageCookiesRev
bowerAngularTranslateLoader Partial	leaflet	angularTranslateInterpolationM essageFormat
angular-simple-logger	jquery	guppy
jqueryUI	uiBootstrap	

The size of all these packages will be around 40MB, if we download whole packages instead of some files.

Then, we can specify the file path from the downloaded package in the import statements wherever the package's file is utilized.

## Healthchecks for the containers

In order to evaluate the operational status of Docker containers/services, it is necessary to establish healthchecks. These healthchecks can determine whether the containers for a given service are functioning properly or not. By configuring the healthchecks in the compose file, users can inspect the health status of containers in the "STATUS" section, accessible via the ``docker ps`` command.

The healthcheck configuration in a compose file looks like this →

```
13     redis
14     healthcheck:
15         test: curl --fail http://localhost || exit 1
16         interval: 60s
17         retries: 5
18         start_period: 20s
19         timeout: 10s | You, 1 second ago • Uncommitted
20
```

On running the ``docker ps`` commands to see the status of the running processes, we can check the health condition of the containers (under STATUS section) as →

```
STATUS
Up 3 seconds (health: starting)
Up 10 minutes (healthy)
```

## 4. Configuring docker commands into Makefile

We will be creating a Makefile that supports the following `make` commands, because these make commands will be easy to understand for the new developers:

<b>Makefile Command</b>	<b>Description</b>
<b>make run-offline</b>	Starts the Oppia server without checking dependencies. Should not require internet access. This can be done by starting the previously stopped containers (already built) of our application. Users will face unusual errors or blank pages or errors in the console, in the case when all dependencies are not installed properly
<b>make run</b>	Sets up and starts the Oppia server. Also runs the steps in "make update-pip-and-npm-packages". This will build and run the docker compose file to start the fully functional Oppia development server.
<b>make setup-devserver</b>	Installs all the necessary dependencies and services required for the devserver (redis, App Engine, elasticsearch, firebase, pip libraries, npm libraries, etc.), but does NOT start the server.
<b>make clean</b>	Cleans the entire setup. Deletes all the dependencies. This can be done by deleting the containers (and the setup of the development server can be done again using `make run`)
<b>make terminal</b>	Opens a terminal accessing the Oppia environment built using Docker. The tests will be run in the standard way (e.g. python -m scripts.run_frontend_tests) from the Docker terminal after running the "make terminal" command. This will start a terminal within the webserver container so that we can run the tests.

```

oppia > Makefile
You, 44 minutes ago | 1 author (You)
1  CONTAINER_NAME=oppia-webserver
2
3  run:
4    docker compose up --build
5
6  install-dependencies:
7    docker compose build
8
9  update-pip-and-npm-packages:
10   pip install -r requirements.txt
11   pip install -r requirements_dev.txt
12   npm install --legacy-peer-deps
13
14  clean:
15   docker rm -f oppia-webserver oppia-redis oppia-elasticsearch
16   docker rm -f oppia-firebase_emulator oppia-google_cloud_sdk
17
18  terminal:
19   docker exec -it $(CONTAINER_NAME) bash
20
21  run-offline:
22   docker compose up
23

```

(structure for Makefile)

## 5. Running Tests locally under Docker

Tests can be executed in a Docker Compose file or via commands in the terminal of the running webserver container. To access the container's terminal, use the command:

``make terminal``

which opens a Bash terminal in the container's environment. Once inside the terminal of the running web server container with all the dependencies installed to run the tests, existing python scripts can be used to run automated tests. For example, you can run:

``python -m scripts.run_frontend_tests`` to execute Frontend tests.

```
~/De/opp/oppia on master !4 ?3 make terminal
docker exec -it oppia-webserver-1 bash
root@c55708e405d1:/app/oppia# python -m scripts.run_frontend_tests
```

For now, we have the following tests in our application:

#### Automated Tests

1. Backend Tests : `python -m scripts.run_backend_tests`
2. Frontend Tests : `python -m scripts.run_frontend_tests`
3. Frontend test Coverage: `python -m scripts.check_frontend_test_coverage`
4. Backend test Coverage: `python -m scripts.check_overall_backend_test_coverage`
5. End-to-end Tests : `python -m scripts.run_e2e_tests --suite="suiteName"`
6. Typescript Tests : `python -m scripts.typescript_checks`
7. Lighthouse Tests : `python -m scripts.run_lighthouse_tests`
8. Acceptance Tests : `python -m scripts.run_acceptance_tests --suite="suiteName"`
9. Mypy Checks : `python -m scripts.run_mypy_checks`
10. Backend Associated Tests: `python -m scripts.check_backend_associated_test_file`

#### Custom Lint Tests

1. Pylint checks : `python -m scripts.run_backend_tests --test_target=scripts.linters.pylint_extensions_test --verbose`
2. ESLint checks : `python -m scripts.run_custom_eslint_tests`

All tests can be executed in the running web server container using python scripts. The dependencies and services required to run the tests are already installed and running in the container (containers can also interact if required as they are connected within the same virtual network bridge), enabling seamless test execution. Therefore, tests can be executed in the same way as they are currently executed.

The tests will run within the container, and to ensure proper test execution, the volumes have already been mapped [here](#) for the running services. This will allow tests to have access to the required dependencies and files within the container, ensuring a smooth test execution process.

Considering that we will be having a grace period from 15th July from 10th Sept, during which both the old setup (using python scripts) and the new setup (using Docker containers) will be functional (for ensuring a smooth transition for devs), so in this period, **the test scripts will be modified** by adding an if-else condition that if the `oppia-is-dockerized` env variable is true, then developer will be inside the docker container environment and they will use the docker build step to execute, otherwise, the test scripts will simply use the existing python scripts for their execution.

With PR#2.3, the if-else conditions that support both python as well as docker setup for the test scripts will be removed, and the tests will only run using the Docker setup. Instead, the modified test scripts will solely focus on running the tests.

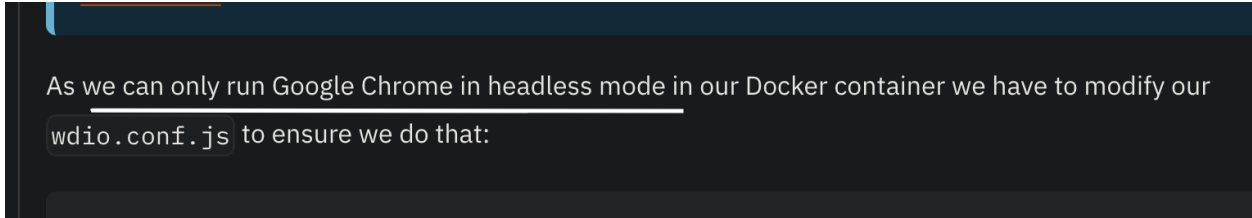
## **Problem: Running e2e tests and acceptance tests after Dockerizing the application**

Currently, the end-to-end (e2e) tests and the acceptance tests are executed in a non-headless mode, which means that the tests are run in a visible browser window. This setup allows developers to monitor the progress of the tests in real time and quickly identify any potential issues.

### **PROBLEM**

When running end-to-end tests or acceptance tests in a Docker container, it is **necessary to run them in the headless mode** because there is no graphical user interface.

The WebDriverIO documentation explicitly states that end-to-end tests can only be run in headless mode when they are dockerized (refer [here](#)).



As we can only run Google Chrome in headless mode in our Docker container we have to modify our `wdio.conf.js` to ensure we do that:

However, this can make it difficult for developers to debug issues that may arise due to changes in the code.

## SOLUTION

To facilitate end-to-end (e2e)/acceptance testing in a non-headless mode, we require a virtual display server that provides an in-memory display server. One such package that offers this functionality is **xvfb** or **Xvfb – X virtual framebuffer**, which offers a display server based on the X11 display server protocol. In the e2e\_tests.yml file (a GitHub action workflow for e2e tests), we utilize this package to enable non-headless mode testing in the Continuous Integration (CI) environment. Similarly, we will use this package to run e2e tests in non-headless mode within the docker container in the local development environment, thus addressing the aforementioned problem through the utilization of the ``xvfb`` package.

The new command to run e2e/acceptance tests in the dockerized application to facilitate non-headless mode testing →

```
`xvfb-run -a --server-args="-screen 0, 1285x1000x24" python -m scripts.run_e2e_test`
```

```
`xvfb-run -a --server-args="-screen 0, 1285x1000x24" python -m scripts.run_acceptance_tests -suite={{suiteName}}`
```

We need to use the server-args flag to specify the screen resolution for the chrome instance where the tests would be running, as we have specified this already in the e2e tests config file - [refer here](#).

In addition, we must include the **xvfb** package as a new dependency essential for our local development server.

## 6. Integrating with GitHub Actions

Finally, to speed up the execution of the GitHub tests and overall reduce the load on the hosted servers provided by GitHub, it is possible to install the necessary dependencies within the GitHub workflows using the docker-image. This approach ensures that the required dependencies are already present within the image, which will significantly reduce the time required to build the images and execute tests.

Further, caching the docker image will improve the test execution process by avoiding rebuilding the same image repeatedly. This will not only save time but also reduce the overall load on the hosted servers. Installing dependencies with docker-image and caching them will result in faster and more efficient test execution.

Sample workflow (modifying e2e tests workflow file)

`oppia/.github/workflows/e2e\_tests.yml`

```
13
14 jobs:
15   build-with-docker:
16     runs-on: ${{ matrix.os }}
17     strategy:
18       matrix:
19         os: [ubuntu-22.04]
20     steps:
21     - name: Checkout
22       uses: actions/checkout@v3
23     - uses: docker/setup-buildx-action@v1
24     # installs all required dependencies
25     - uses: docker/build-push-action@v2
26       with:
27         context: .
28         file: Dockerfile
29         cache-from: type=gha
30         cache-to: type=gha,mode=max
31
32 e2e_test:
```

We utilize our docker containers to install the dependencies and start the services required for running the workflow on Github actions.

Most commonly, in all of the action workflows, we install the dependencies using a single action script that is: [.github/actions/install-oppia-dependencies/action.yml](https://github.com/oppia/oppia/blob/master/.github/actions/install-oppia-dependencies/action.yml)

So we can simply use `make install-dependencies` and `make update-pip-and-npm-packages` in the build step of the workflow for installing the required dependencies and start services that will be required for the execution of the workflow. The docker image will only be set up once, and then stored and reused (cached the docker image) for all the remaining CI/CD tests in that run.



## 7. Deprecating the existing scripts

The current installation process involving `start.py`, `servers.py`, and associated scripts depicted in the [flowchart](#) will be deprecated. The current setup process utilizes `start.py` to invoke other scripts for installing the necessary dependencies, configuring required services, and initiating the local development server. **These operations and all configurations will be entirely managed by Docker and Makefile, facilitating the smooth execution of the development server.**

Therefore, we intend to discontinue the usage of such scripts that carry out the installation of dependencies or the initiation of different services, such scripts are →

1. `start.py`
2. `servers.py`
3. `install_third_party.py`
4. `install_third_party_libs.py`
5. `install_python_dev_dependencies.py`
6. `install_python_prod_dependencies.py`
7. `install_chrome_for_ci.py`
8. `setup_gae.py`
9. `setup.py`

(additionally, I will also deprecate the test scripts of the above mentioned scripts)

## **Prototype of Dockerized Oppia**

I have successfully created a functional prototype that incorporates the Oppia development server and dockerization. The prototype accurately reflects the development server's structure when dockerized. Please do checkout the repository and the repository's Readme file →

<https://github.com/Shivkant-Chauhan/prototype---dockerizing-oppia>

(repository)

<https://github.com/Shivkant-Chauhan/prototype---dockerizing-oppia#dockerizing-oppia>

(Readme)

Other useful links from the prototype repository –

[Dockerfile](#)

[Dockerfile.firebase\\_emulator](#)

[docker-compose.yml](#)

[Makefile](#)

As this was merely a prototype, I utilized pre-existing build artifacts and avoided webpack compilation, along with some other adjustments that I plan to address during GSoC. It is important to note that not all tasks can be completed within the scope of the prototype, as those things are intended to be done during GSoC!

# Execution of the Dockerized Oppia prototype on different platforms

(mac M1, Windows, Linux) →

Recordings of the screen displaying the prototype being run on different platforms:

[drive-link](#)

## 1) macOS with M1chip – [repo Readme for mac M1](#)

- Screen recording of the dockerized development server → [drive-link](#)
- Installing dependencies using `make install-dependencies`

```
~/Desktop/oppia on master ?1 make install-dependencies took 4s shivi 3.8.15 at 10:14:33
docker compose build
[+] Building 969.7s (22/23)
=> [internal] load build definition from Dockerfile 0.1s
=> => transferring dockerfile: 2.10kB 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/python:3.8 5.6s
=> [auth] library/python:pull token for registry-1.docker.io 0.0s
=> [internal] load build context 163.0s
=> => transferring context: 5.02GB 158.2s
=> [ 1/17] FROM docker.io/library/python:3.8@sha256:f9af3c8c4614d955495d23a665760abe3b37d4310bf9c33ea1ebf9f18991d8fe 104.6s
=> resolve docker.io/library/python:3.8@sha256:f9af3c8c4614d955495d23a665760abe3b37d4310bf9c33ea1ebf9f18991d8fe 0.0s
=> sha256:670730c27c2eac07897a6e94fe55423ea50b884d9c28161a283bbbf064d1124 10.88MB / 10.88MB 12.2s
=> sha256:68a71c865a2c34678c6dea55e4b0928f751ee3c0ca91cace6e4e0578c534d6cf 5.17MB / 5.17MB 10.2s
=> sha256:f9af3c8c4614d955495d23a665760abe3b37d4310bf9c33ea1ebf9f18991d8fe 1.86kB / 1.86kB 0.0s
=> sha256:00af0878e19a32802037964268982b52425d0e8395cb61483baec3a28f89eeb5 2.22kB / 2.22kB 0.0s
=> sha256:a865bc2999a48902fe4d5f3b64ed2ab2d0ba18ece4b0310808058ce9404aee39 7.94kB / 7.94kB 0.0s
=> sha256:3e440a7045683e27f8e2fa04000e0e078d8dfac0c971358ae0f8c65c13321c8e 55.05MB / 55.05MB 15.4s
=> sha256:5a7a2c95f0f8b221d776ccf35911b68eec2cf9414a44d216205a6f03e381d3d7 54.58MB / 54.58MB 45.4s
=> sha256:6d627e120214bb28a729d4b54a0ecba4c4aea0295ca2d1f129480145fad2af6 196.81MB / 196.81MB 71.8s
=> sha256:f8c6dc6780819f5eb5a6ee84ce72dd613d5e6c406585211a064b6ef641c8a09a 6.29MB / 6.29MB 18.4s
=> extracting sha256:3e440a7045683e27f8e2fa04000e0e078d8dfac0c971358ae0f8c65c13321c8e 4.5s
=> sha256:b64f7311a27303009c36d4f43548974848bf281c8a5e1d914531f758edbece2d 15.53MB / 15.53MB 26.1s
=> extracting sha256:68a71c865a2c34678c6dea55e4b0928f751ee3c0ca91cace6e4e0578c534d6cf 0.6s
=> extracting sha256:670730c27c2eac07897a6e94fe55423ea50b884d9c28161a283bbbf064d1124 0.7s
=> sha256:e851df9f4b615b7cfb0e8e42f30ba57f3827005cc9385b6893c9017cf949a877 244B / 244B 26.5s
=> sha256:aa941cbc6a4c4b1d076f1d6206e2f64c2c5c58d5f5d250ae068fe38d998f5319 2.91MB / 2.91MB 28.9s
=> extracting sha256:5a7a2c95f0f8b221d776ccf35911b68eec2cf9414a44d216205a6f03e381d3d7 6.8s
=> extracting sha256:6d627e120214bb28a729d4b54a0ecba4c4aea0295ca2d1f129480145fad2af6 21.2s
=> extracting sha256:f8c6dc6780819f5eb5a6ee84ce72dd613d5e6c406585211a064b6ef641c8a09a 1.0s
=> extracting sha256:b64f7311a27303009c36d4f43548974848bf281c8a5e1d914531f758edbece2d 3.7s
=> extracting sha256:e851df9f4b615b7cfb0e8e42f30ba57f3827005cc9385b6893c9017cf949a877 0.1s
=> extracting sha256:aa941cbc6a4c4b1d076f1d6206e2f64c2c5c58d5f5d250ae068fe38d998f5319 0.6s
=> [ 2/17] WORKDIR /app/oppia 1.5s
=> [ 3/17] RUN apt-get update 42.5s
=> [ 4/17] RUN apt-get -y install curl 12.2s
=> [ 5/17] RUN apt-get -y install git 7.8s
=> [ 6/17] RUN apt-get -y install python3-dev 13.4s
=> [ 7/17] RUN apt-get -y install python3-setuptools 10.4s
=> [ 8/17] RUN apt-get -y install python3-pip 12.0s
=> [ 9/17] RUN apt-get -y install unzip 3.9s
=> [10/17] RUN apt-get -y install python3-yaml 7.1s
=> [11/17] RUN pip install --upgrade pip==21.2.3 18.8s
=> [12/17] COPY requirements.txt . 0.0s
=> [13/17] COPY requirements_dev.txt . 0.0s
=> [14/17] RUN pip install -r requirements.txt 391.6s
=> [15/17] RUN pip install -r requirements_dev.txt 143.3s
=> [16/17] RUN apt-get -y install python2 25.0s
=> [17/17] COPY . . 158.9s
=> exporting to image 10.6s
```

- Starting development server with `make run`

```

~/De/opp/oppia on master ?1 make run
docker compose up --build
[+] Building 393.8s (23/23) FINISHED
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 32B                                              0.0s
=> [internal] load .dockerignore                                                 0.0s
=> => transferring context: 2B                                                  0.0s
=> [internal] load metadata for docker.io/library/python:3.8                    4.8s
=> [auth] library/python:pull token for registry-1.docker.io                    0.0s
=> [ 1/17] FROM docker.io/library/python:3.8@sha256:f9af3c8c4614d955495d23a665760abe3b37d4310bf9c33ea1ebf9f18991d8fe  0.0s
=> [internal] load build context                                               19.2s
=> => transferring context: 33.24MB                                           17.0s
=> CACHED [ 2/17] WORKDIR /app/oppia                                           0.0s
=> CACHED [ 3/17] RUN apt-get update                                           0.0s
=> CACHED [ 4/17] RUN apt-get -y install curl                                  0.0s
=> CACHED [ 5/17] RUN apt-get -y install git                                    0.0s
=> CACHED [ 6/17] RUN apt-get -y install python3-dev                           0.0s
=> CACHED [ 7/17] RUN apt-get -y install python3-setuptools                    0.0s
=> CACHED [ 8/17] RUN apt-get -y install python3-pip                           0.0s
=> CACHED [ 9/17] RUN apt-get -y install unzip                                  0.0s
=> CACHED [10/17] RUN apt-get -y install python3-yaml                           0.0s
=> CACHED [11/17] RUN pip install --upgrade pip==21.2.3                       0.0s
=> CACHED [12/17] COPY requirements.txt .                                        0.0s
=> CACHED [13/17] COPY requirements_dev.txt .                                    0.0s
=> CACHED [14/17] RUN pip install -r requirements.txt                           0.0s
=> CACHED [15/17] RUN pip install -r requirements_dev.txt                       0.0s
=> CACHED [16/17] RUN apt-get -y install python2                                0.0s
=> [17/17] COPY . .                                                            213.1s
=> exporting to image                                                         156.3s
=> => exporting layers                                                         156.1s
=> => writing image sha256:75a7118ed64046d4a5ff059e65e7ad8ef6e9b525acd2347cf8023f97433a59a  0.0s
=> => naming to docker.io/library/oppia-webserver                             0.0s
[+] Running 2/2
  # Network oppia_default      Created                                0.2s
  # Container oppia-webserver  Created                                98.9s
Attaching to oppia-webserver
oppia-webserver | INFO      2023-03-27 05:12:34,331 devappserver2.py:316] Skipping SDK update check.
oppia-webserver | WARNING  2023-03-27 05:12:36,187 simple_search_stub.py:1196] Could not read search indexes from /tmp/appengine.None.root/search_indexes
oppia-webserver | INFO      2023-03-27 05:12:36,204 <string>:383] Starting API server at: http://localhost:36199
oppia-webserver | INFO      2023-03-27 05:12:37,203 instance_factory.py:156] Detected python version "Python 3.8.16
oppia-webserver | " for runtime "python38" at "python3".
oppia-webserver | INFO      2023-03-27 05:13:07,051 instance_factory.py:313] Using pip to install dependency libraries; pip stdout is redirected to /tmp/tmpdAq208
oppia-webserver | INFO      2023-03-27 05:13:07,128 instance_factory.py:335] Running /tmp/tmpopjvIy/bin/pip install --upgrade pip
oppia-webserver | INFO      2023-03-27 05:13:26,101 instance_factory.py:335] Running /tmp/tmpopjvIy/bin/pip install -r requirements.txt
oppia-webserver | WARNING  2023-03-27 05:17:14,501 file_watcher.py:147] Could not create InotifyFileWatcher; falling back to MTimeFileWatcher: [Errno 38] ENOSYS
oppia-webserver | INFO      2023-03-27 05:17:14,660 dispatcher.py:281] Starting module "default" running at: http://0.0.0.0:8080
oppia-webserver | INFO      2023-03-27 05:17:14,683 admin_server.py:150] Starting admin server at: http://localhost:8000
oppia-webserver | /app/oppia/oppia_tools/google-cloud-sdk-364.0.0/google-cloud-sdk/platform/google-appengine/google/appengine/tools/devappserver2/mtime_file_watcher.py:183: User
Warning: There are too many files in your application for changes in all of them to be monitored. You may have to restart the development server to see some changes to your files

```

## 2) Linux (ubuntu 22.04) – [repo Readme for Linux](#)

- Screen recording of the dockerized development server → [drive-link](#)
- Installing dependencies using `make install-dependencies`

```
Activities  Terminal  Wed Mar 29 1:31 AM  100%
make install-dependencies
> make install-dependencies
docker compose build
[+] Building 1643.7s (20/23)
=> [internal] load build definition from Dockerfile                                0.0s
=> => transferring dockerfile: 2.10kB                                           0.0s
=> [internal] load .dockerignore                                                 0.0s
=> => transferring context: 2B                                                    0.0s
=> [internal] load metadata for docker.io/library/python:3.8                    17.3s
=> [auth] library/python:pull token for registry-1.docker.io                    0.0s
=> [ 1/17] FROM docker.io/library/python:3.8@sha256:f9af3c8c4614d955495d23a665760abe3b37d4310bf9c33ea1ebf9f18991d8fe 692.3s
=> => resolve docker.io/library/python:3.8@sha256:f9af3c8c4614d955495d23a665760abe3b37d4310bf9c33ea1ebf9f18991d8fe 0.0s
=> => sha256:3e440a7045683e27f8e2fa0400e0e078d8dfac0c971358ae0f8c65c13321c8e 55.05MB / 55.05MB 589.2s
=> => sha256:68a71c865a2c34678c6dea55e4b0928f751ee3c0ca91cace6e4e0578c534d6cf 5.17MB / 5.17MB 55.1s
=> => sha256:670730c27c2eac07897a6e94fe55423ea50b884d9c28161a283bbb1064d1124 10.88MB / 10.88MB 64.5s
=> => sha256:f9af3c8c4614d955495d23a665760abe3b37d4310bf9c33ea1ebf9f18991d8fe 1.86kB / 1.86kB 0.0s
=> => sha256:00af0878e19a32602037964268982b52425d0e8395cb61483banc3a28f89eeb5 2.22kB / 2.22kB 0.0s
=> => sha256:a805bc2909a489027e4d5f3b64ed2ab2d0ba18ece4b0310808058ce9404aee39 7.94kB / 7.94kB 0.0s
=> => sha256:5a7a2c95f0f8b221d776ccf35911b68eec2cf9414a44d216205a6f03e381d3d7 54.58MB / 54.58MB 643.2s
=> => sha256:6d627e120214bb28a729d4b54a0ecba4c4aea0295ca2d1f129480145fad2af6 196.81MB / 196.81MB 687.0s
=> => sha256:f8c6dc6780819f5eb5a6ee84ce72dd613d5e6c406585211a064b6ef641c8a09a 6.29MB / 6.29MB 593.6s
=> => extracting sha256:3e440a7045683e27f8e2fa0400e0e078d8dfac0c971358ae0f8c65c13321c8e 0.3s
=> => extracting sha256:68a71c865a2c34678c6dea55e4b0928f751ee3c0ca91cace6e4e0578c534d6cf 0.1s
=> => extracting sha256:670730c27c2eac07897a6e94fe55423ea50b884d9c28161a283bbb1064d1124 0.1s
=> => sha256:b64f7311a27303009c36d4f43548974848bf281c8a5e1d914531f758edbec2d 15.53MB / 15.53MB 598.8s
=> => sha256:e851df9f4b615b7cfb0e8e42f30ba57f3827005cc9385b6893c9017cf949a877 244B / 244B 681.3s
=> => sha256:aa941c6c6a4c4b1d076f1d6206e2f64c2c5c58d5f5d250ae068fe38d998f5319 2.91MB / 2.91MB 603.0s
=> => extracting sha256:5a7a2c95f0f8b221d776ccf35911b68eec2cf9414a44d216205a6f03e381d3d7 1.0s
=> => extracting sha256:6d627e120214bb28a729d4b54a0ecba4c4aea0295ca2d1f129480145fad2af6 3.2s
=> => extracting sha256:f8c6dc6780819f5eb5a6ee84ce72dd613d5e6c406585211a064b6ef641c8a09a 0.2s
=> => extracting sha256:b64f7311a27303009c36d4f43548974848bf281c8a5e1d914531f758edbec2d 0.3s
=> => extracting sha256:e851df9f4b615b7cfb0e8e42f30ba57f3827005cc9385b6893c9017cf949a877 0.0s
=> => extracting sha256:aa941c6c6a4c4b1d076f1d6206e2f64c2c5c58d5f5d250ae068fe38d998f5319 0.1s
=> [internal] load build context
=> => transferring context: 5.02GB                                               120.9s
=> [auth] library/python:pull token for registry-1.docker.io                    0.0s
=> [ 2/17] WORKDIR /app/oppia                                                    0.1s
=> [ 3/17] RUN apt-get update                                                    168.8s
=> [ 4/17] RUN apt-get -y install curl                                          0.7s
```

- Starting development server with `make run`

```
Activities Terminal Wed Mar 29 2:00 AM
prakash@prakash-Laptop:~/Desktop/oppia-docker-practice/oppia

> make run
docker compose up --build
[+] Building 1534.5s (22/22) FINISHED
=> [internal] load build definition from Dockerfile 0.0s
=> => transferring dockerfile: 32B 0.0s
=> [internal] load .dockerignore 0.0s
=> => transferring context: 2B 0.0s
=> [internal] load metadata for docker.io/library/python:3.8 5.0s
=> [ 1/17] FROM docker.io/library/python:3.8@sha256:f9af3c8c4614d955495d23a66578@abe3b37d4318bf9c33ca1ebf9f18991d8fe 0.0s
=> [internal] load build context 0.0s
=> => transferring context1: 32.89MB 8.4s
=> CACHED [ 2/17] WORKDIR /app/oppia 0.0s
=> CACHED [ 3/17] RUN apt-get update 0.0s
=> CACHED [ 4/17] RUN apt-get -y install curl 0.0s
=> CACHED [ 5/17] RUN apt-get -y install git 0.0s
=> CACHED [ 6/17] RUN apt-get -y install python3-dev 0.0s
=> CACHED [ 7/17] RUN apt-get -y install python3-setuptools 0.0s
=> CACHED [ 8/17] RUN apt-get -y install python3-pip 0.0s
=> CACHED [ 9/17] RUN apt-get -y install unzip 0.0s
=> CACHED [10/17] RUN apt-get -y install python3-yaml 0.0s
=> CACHED [11/17] RUN pip install --upgrade pip==21.2.3 0.0s
=> CACHED [12/17] COPY requirements.txt . 0.0s
=> CACHED [13/17] COPY requirements_dev.txt . 0.0s
=> [14/17] RUN pip install -r requirements.txt 1024.8s
=> [15/17] RUN pip install -r requirements_dev.txt 213.2s
=> [16/17] RUN apt-get -y install python2 92.7s
=> [17/17] COPY . . 118.2s
=> exporting to image 71.5s
=> => exporting layers 71.3s
=> => writing image sha256:c58290f67f0c1adc50c2418d291ece9e4cdc26dad67c64af88ec7f7808d1cabf 0.0s
=> => naming to docker.io/library/oppia-webserver 0.0s
[+] Running 2/2
  # Network oppia_default Created 0.4s
  # Container oppia-webserver Created 77.5s
Attaching to oppia-webserver
Error response from daemon: failed to create shim task: OCI runtime create failed: runc create failed: unable to start container process: exec: "./oppia_tools
/google-cloud-sdk-364.0.0/google-cloud-sdk/bin/dev_appserver.py": permission denied: unknown
make: *** [Makefile:4: run] Error 1
```

### 3) Windows 11 – [repo Readme for Windows](#)

- Screen recording of the dockerized development server → [drive-link](#)
- Starting development server with ``make run``

```
Windows PowerShell
PS C:\Users\yogesh\Desktop\oppia-docker-practice\oppia> docker compose up --build
[+] Building 49.3s (22/22) FINISHED
=> [internal] load build definition from Dockerfile                                0.1s
=> => transferring dockerfile: 32B                                              0.1s
=> [internal] load .dockerignore                                                0.1s
=> => transferring context: 2B                                                 0.1s
=> [internal] load metadata for docker.io/library/python:3.8                    2.1s
=> [internal] load build context                                                46.9s
=> => transferring context: 32.89MB                                           44.8s
=> [ 1/17] FROM docker.io/library/python:3.8@sha256:f9af3c8c4614d955           0.0s
=> CACHED [ 2/17] WORKDIR /app/oppia                                           0.0s
=> CACHED [ 3/17] RUN apt-get update                                           0.0s
=> CACHED [ 4/17] RUN apt-get -y install curl                                  0.0s
=> CACHED [ 5/17] RUN apt-get -y install git                                    0.0s
=> CACHED [ 6/17] RUN apt-get -y install python3-dev                          0.0s
=> CACHED [ 7/17] RUN apt-get -y install python3-setuptools                    0.0s
=> CACHED [ 8/17] RUN apt-get -y install python3-pip                          0.0s
=> CACHED [ 9/17] RUN apt-get -y install unzip                                  0.0s
=> CACHED [10/17] RUN apt-get -y install python3-yaml                          0.0s
=> CACHED [11/17] RUN pip install --upgrade pip==21.2.3                       0.0s
=> CACHED [12/17] COPY requirements.txt .                                        0.0s
=> CACHED [13/17] COPY requirements_dev.txt .                                    0.0s
=> CACHED [14/17] RUN pip install -r requirements.txt                          0.0s
=> CACHED [15/17] RUN pip install -r requirements_dev.txt                      0.0s
=> CACHED [16/17] RUN apt-get -y install python2                               0.0s
=> CACHED [17/17] COPY . .                                                     0.0s
=> exporting to image                                                         0.1s
=> => exporting layers                                                         0.0s
=> => writing image sha256:326f7e74707c8906d93ecb8e83a68c985a18411de         0.0s
=> => naming to docker.io/library/oppia-webserver                            0.0s
[+] Running 1/0
- Container oppia-webserver Created                                          0.0s
Attaching to oppia-webserver
oppia-webserver | INFO     2023-03-28 06:17:48,241 devappserver2.py:316] Skipping SDK update check.
oppia-webserver | INFO     2023-03-28 06:17:49,038 <string>:383] Starting API server at: http://localhost:44693
oppia-webserver | INFO     2023-03-28 06:17:49,044 instance_factory.py:156] Detected python version "Python 3.8.16
oppia-webserver | " for runtime "python38" at "python3".
oppia-webserver | INFO     2023-03-28 06:17:51,744 instance_factory.py:313] Using pip to install dependency libraries; pip stdout is redirected to /tmp/tmp
_KcJz0
oppia-webserver | INFO     2023-03-28 06:17:51,748 instance_factory.py:335] Running /tmp/tmpze0Kvx/bin/pip install --upgrade pip
oppia-webserver | INFO     2023-03-28 06:17:57,886 instance_factory.py:335] Running /tmp/tmpze0Kvx/bin/pip install -r requirements.txt
oppia-webserver | Requirement already satisfied: pip in /tmp/tmpze0Kvx/lib
```

- Logs of the Oppia development server under Docker

```

Windows PowerShell
=> => writing image sha256:326f7e74707c8906d93ecb8e83a68c985a18411de 0.0s
=> => naming to docker.io/library/oppia-webserver 0.0s
[+] Running 1/0
 - Container oppia-webserver Created 0.0s
Attaching to oppia-webserver
oppia-webserver | INFO      2023-03-28 06:17:48,241 devappserver2.py:316] Skipping SDK update check.
oppia-webserver | INFO      2023-03-28 06:17:49,038 <string>:383] Starting API server at: http://localhost:44693
oppia-webserver | INFO      2023-03-28 06:17:49,044 instance_factory.py:156] Detected python version "Python 3.8.16
oppia-webserver | " for runtime "python38" at "python3".
oppia-webserver | INFO      2023-03-28 06:17:51,744 instance_factory.py:313] Using pip to install dependency libraries; pip stdout is redirected to /tmp/tmp
_KcJz0
oppia-webserver | INFO      2023-03-28 06:17:51,748 instance_factory.py:335] Running /tmp/tmpze0Kvx/bin/pip install --upgrade pip
oppia-webserver | INFO      2023-03-28 06:17:57,886 instance_factory.py:335] Running /tmp/tmpze0Kvx/bin/pip install -r requirements.txt
oppia-webserver | Requirement already satisfied: pip in /tmp/tmpze0Kvx/lib/

oppia-webserver | INFO      2023-03-28 06:29:39,238 dispatcher.py:281] Starting module "default" running at: http://0.0.0.0:8080
oppia-webserver | INFO      2023-03-28 06:29:39,246 admin_server.py:150] Starting admin server at: http://localhost:8000
oppia-webserver | INFO      2023-03-28 06:29:40,247 instance.py:561] Cannot decide GOOGLE_CLOUD_PROJECT, using "test" as a fake value
oppia-webserver | [2023-03-28 06:29:40 +0000] [36] [INFO] Starting gunicorn 20.1.0
oppia-webserver | [2023-03-28 06:29:40 +0000] [36] [INFO] Listening at: http://0.0.0.0:16151 (36)
oppia-webserver | [2023-03-28 06:29:40 +0000] [36] [INFO] Using worker: sync
oppia-webserver | [2023-03-28 06:29:40 +0000] [39] [INFO] Booting worker with pid: 39
oppia-webserver | INFO      2023-03-28 06:29:41,264 instance.py:294] Instance PID: 36
oppia-webserver | INFO      2023-03-28 06:29:47,451 module.py:883] default: "GET /_ah/warmup HTTP/1.1" 200 -
oppia-webserver | INFO      2023-03-28 06:30:33,549 module.py:883] default: "GET / HTTP/1.1" 200 2581
oppia-webserver | INFO      2023-03-28 06:30:33,658 module.py:883] default: "GET /third_party/generated/webfonts/fa-solid-900.woff2 HTTP/1.1" 200 75440
oppia-webserver | INFO      2023-03-28 06:30:33,679 instance.py:561] Cannot decide GOOGLE_CLOUD_PROJECT, using "test" as a fake value
oppia-webserver | INFO      2023-03-28 06:30:33,692 module.py:883] default: "GET /third_party/generated/css/third_party.css HTTP/1.1" 200 279244
oppia-webserver | INFO      2023-03-28 06:30:33,693 module.py:883] default: "GET /third_party/generated/webfonts/fa-brands-400.woff2 HTTP/1.1" 200 74508
oppia-webserver | INFO      2023-03-28 06:30:33,713 module.py:883] default: "GET /dist/oppia-angular/runtime-es2015.js HTTP/1.1" 200 9915
oppia-webserver | INFO      2023-03-28 06:30:33,718 module.py:883] default: "GET /dist/oppia-angular/polyfills-es2015.js HTTP/1.1" 200 192414
oppia-webserver | INFO      2023-03-28 06:30:33,733 module.py:883] default: "GET /dist/oppia-angular/styles.css HTTP/1.1" 200 489821
oppia-webserver | INFO      2023-03-28 06:30:33,772 module.py:883] default: "GET /dist/oppia-angular/main-es2015.js HTTP/1.1" 200 385045
oppia-webserver | [2023-03-28 06:30:33 +0000] [63] [INFO] Starting gunicorn 20.1.0
oppia-webserver | [2023-03-28 06:30:33 +0000] [63] [INFO] Listening at: http://0.0.0.0:21888 (63)
oppia-webserver | [2023-03-28 06:30:33 +0000] [63] [INFO] Using worker: sync
oppia-webserver | [2023-03-28 06:30:33 +0000] [66] [INFO] Booting worker with pid: 66
oppia-webserver | INFO      2023-03-28 06:30:33,971 module.py:883] default: "GET /dist/oppia-angular/vendor-es2015.js HTTP/1.1" 200 4690110
oppia-webserver | INFO      2023-03-28 06:30:34,137 module.py:883] default: "GET /dist/oppia-angular/favicon.ico HTTP/1.1" 404 -
oppia-webserver | INFO      2023-03-28 06:30:34,155 module.py:883] default: "GET /assets/favicon.ico HTTP/1.1" 200 318
oppia-webserver | ERROR:root:Exception raised at http://localhost:8080/csrfhandler: Error 99 connecting to localhost:6379. Cannot assign requested address.

```



## Opting Docker Setup instead of python scripts

When running the Python scripts for the first time, it takes approximately 25-27 minutes or even longer. Additionally, there are prerequisite steps to follow for setting up the development server and resolving any issues that may arise during the setup process. However, after dockerizing the process, the setup time significantly reduced compared to using Python scripts, making the overall setup of the development server quicker and smoother. Docker made it effortless to set up the Oppia development server as there was no need for additional configurations or environment settings.

While setting up the development server for the first time using Docker, it builds the docker image for the web-server container, so I think it took around 10-12 minutes. Running the development server (using the cached docker images), took 2-3 minutes. But the actual dockerized server will take some more time (8-10 minutes more in the building docker images process) because I was having some internet connectivity problems at that time in my college campus so for running the prototype, I don't build the other services(redis, firebase, and google cloud SDK) used in local development server in that prototype.

## Challenges that may arise from the Docker Hub images we are using

We will be using the following Docker Hub images for dockerizing our application →

<b>Service Name (with link to the Docker Hub image)</b>	<b>Image version to be used (according to what we use in Oppia's codebase)</b>	<b>Image Type</b>
<a href="#">Redis</a>	redis: 7.0-alpine	Official Docker Image
<a href="#">Google Cloud SDK</a>	google/cloud-sdk: 364.0.0	Verified Docker Image by Google
<a href="#">ElasticSearch</a>	elasticsearch: 7.17.0	Official Docker Image

The Redis and Elasticsearch images available on Docker Hub are the official Docker Hub Images, and as such, can be relied upon for their stability and sustainability. Additionally, the Google Cloud SDK Image has been verified by Google and can be considered a trustworthy

resource for integration into our application.

## PROBLEM that may arise

It is important to anticipate potential contingencies in the event that the Google Cloud SDK image is removed from Docker Hub in accordance with the new policies of Docker.

## SOLUTION

In such a scenario, a prudent solution would be to create a separate Dockerfile to configure the Google Cloud SDK by downloading it directly from the official website. This would ensure that the functionality of the application is not affected and that it continues to run smoothly. The Google Cloud SDK dockerfile would look like this →

```
dockerfile.google-cloud-sdk > ...
FROM ubuntu:22.04

RUN apt-get install -y -qq wget python unzip

RUN wget https://dl.google.com/dl/cloudsdk/channels/rapid/google-cloud-sdk.zip
RUN unzip google-cloud-sdk.zip && rm google-cloud-sdk.zip
RUN google-cloud-sdk/install.sh --usage-reporting=true --path-update=true --bash-completion=true --rc-path=/.bashrc --additional-components app-engine-python

RUN mkdir -p /usr/local/gcloud/google-cloud-sdk/bin

ENV PATH="/usr/local/gcloud/google-cloud-sdk/bin:${PATH}"
```

(this is the sample Dockerfile for configuring the Google Cloud SDK)

Note: This is no longer required as this was a big concern when docker announced that they will be deleting many dockerhub images according to their new policies. But after few days, another announcement was made to not to worry about the dockerhub images and all the images will remain as they are! So this problem will not be concerning for us now.

## Running the local development server in offline mode→

Oppia contains certain components that are not available offline, and they need to be loaded on specific instances when we visit that page/component. For example - we are using iframe components in many places which loads another component within the page. Rendering iframes in offline mode is not possible because iframes require external resources to load, such as images, videos, and scripts. Without internet, they cannot be loaded, and the iframe content may not load/display correctly or at all. In this case, the pages or components that use these

components will not function as intended, in the offline mode. Note that this is the same scenario that we currently have, and would remain the same as we dockerize the development server setup.

## Problems we can face during the project →

### 1) Stop/Remove containers if not necessary

The simultaneous execution of multiple containers on a single machine can lead to a substantial surge in memory utilization, particularly if the containers require a significant amount of memory or are resource-intensive. Inadequate memory availability may compel the operating system to initiate memory swapping to disk, which can significantly impair the performance of the overall application and the containers.

In specific instances, there may be pre-existing containers that are already operational, and their existence can be confirmed by executing the ``docker ps`` command. While it is possible to establish the **Oppia development server successfully with 8GB of RAM**, there may be scenarios where executing other memory-intensive applications simultaneously may not be feasible. In such circumstances, the Oppia development server may fail to start due to insufficient available memory or vice-versa, when the Oppia server is running, and a memory-intensive application is launched.

To tackle such situations, it may be necessary to stop or delete the running containers.

The ``make clean`` command can be employed to stop and delete the Oppia setup.

Within the GSoC project, I will provide explanations and possible solutions for the issues that developers might face, which means I will provide a detailed section on "how to debug setup issues in Docker" under the Troubleshooting wiki page. Developers will be shown the log message – "Please go to our troubleshooting page at `{{URL}}` if you are facing any issue with the oppia development server" whenever they run the ``make run/run-offline/setup-devserver`` commands in their terminal, pointing to our troubleshooting page in the wiki for any setup-related issues.

References from: [Official Docker documentation](#), [Project Idea Doc](#), [docker layer caching in github actions](#), [docker-healthcheck](#), [Lighthouse integration with Docker](#)

## Impact on Other Oppia Teams

Prospective developers, QA testers, and new contributors intending to contribute to Oppia's codebase and utilize the Oppia development server will benefit from an improved installation process and streamlined startup procedure for the local development server, that will be more streamlined, efficient and minimizes the likelihood of errors.

## Key High-Level and Architectural Decisions

Decision 1:

Using the Verified Docker Hub image for [Google Cloud SDK \(364.0.0\)](#)

An alternative approach would be to create a distinct Dockerfile for configuring the Google Cloud SDK and its associated services (such as Google App Engine, Google Cloud Datastore, etc.) that our application uses.

Utilizing a Docker Hub image for the Google Cloud SDK, which has been verified by Google, can provide greater reliability and efficiency due to the following reasons:→

1. Using this Docker Hub image can be considered a more reliable and secure source, as it undergoes verification by Google to ensure its authenticity and integrity.
2. Reduce the required time and effort (for configuring separate dockerfile)
3. Benefit from robust community support.
4. Developed in accordance with established industry-standard best practices and guidelines.
5. Additional level of security to the Docker image.

## Decision 2:

### Asking devs to download the [Docker Desktop](#) in their local system

Another approach could involve specifying a particular version of Docker Desktop and instructing developers to install that specific version. However, it's important to note that Docker Desktop will be installed system-wide on the developer's computer, and they might utilize it for tasks unrelated to Oppia as well.

Including a link to the Docker Desktop downloader page pointing to the official Docker downloader page within our wiki pages will allow developers to conveniently download the Docker Desktop App installer with a single click. Furthermore, since Docker regularly releases updates, developers can easily download the latest version of Docker from the downloader page of Docker.

## Documentation changes

The Wiki pages will be updated to include comprehensive instructions for →

(Note: We will have a single wiki page instructing the installation process for Oppia development server for all platforms - as we will be dockerizing our application and that will make the development server platform independent)

- **Installing Oppia**

(Devs can set up Oppia's development server by following the Docker installation and configuration procedures, as well as executing the relevant Makefile commands)

- Setting up Oppia development server using Docker.  
(Include instructions for setting up the fresh local development server and launching it using appropriate make commands) –
  - Download and install the latest version of [Docker Desktop](#) in your system.
  - Fork and [Clone oppia](#) (web) repo.  
(above 2 steps will be required for the first time only for setting up the fresh development server)
  - To start the Docker Engine in your system, launch the Docker Desktop App from your 'Applications' menu.

(NOTE: To verify the Docker Engine has launched successfully, run `docker version` in your terminal to ensure that both the Client and Server are active.

It is not necessary to run this command every time you start the development server, this is just so you can verify the successful launch of the docker engine.)

- In the root oppia/ directory:
  - Run `make setup-devserver` command to set up the local development server.
  - Run `make run` command to start the local development server.
- Migrate to Docker Setup  
(ask folks to delete the current setup and move to the dockerized setup)
- Overview for the new Oppia setup.  
(brief overview for the dockerized Oppia development server with information regarding all the services we specified in the docker compose file)
- **Troubleshooting**
  - **Debugging dockerized Oppia setup**  
(information related to resolution of debugging issues that arise within the Docker environment, the sharing of Docker images, and other considerations for utilizing Docker with Oppia)

## Testing Plan

### Docker Action

I am planning to include a Docker Action that would serve to test the setup and configuration of Docker within the application's environment. The purpose of this action would be to verify that Docker is working correctly and that the application can be built and run as intended within a Docker container and also that the other devs will not break the changes of my PR after merging.

A Docker action that tests the Docker setup in the application will involve building a Docker container image with application configuration or code, and then running the container to ensure that it is functioning as expected. This Docker action can be used to verify that Docker is properly installed and configured on the host machine, and that the application can be deployed and run within a Docker container. It will be added as part of a Continuous Integration (CI) pipeline in the GitHub Action Workflows to ensure that changes to the application's codebase do not affect the functionality of the Docker setup.

**Note:** This Docker Action will be merged into `develop` branch with PR#1.3 but will be dropped with the PR#2.2 where I will migrate the GitHub actions to use Docker. The reason to drop this docker action with PR#2.2 is that the testing of this docker action will be automatically handled with the other github actions as they will utilize Docker in their build step.

The Docker Setup Action (`docker_setup_test.yml`) will look like →

```
oppia > .github > workflows > docker_setup_test.yml
1  name: Docker Setup Test
2
3  on:
4    merge_group:
5      types: [checks_requested]
6    push:
7      branches:
8        - develop
9        - release-*
10   pull_request:
11     branches:
12       - develop
13       - release-*
14
15   jobs:
16     build:
17       runs-on: ${ matrix.os }
18       strategy:
19         matrix:
20           os: [ubuntu-22.04]
21       steps:
22         - name: Checkout code
23           uses: actions/checkout@v2
24
25         - name: Install dependencies
26           run: make install-dependencies && make-update-pip-and-npm-packages
27
28         - name: Run Dockerized application
29           run: make run
```

**Note:** We need to remove this action, once we have the docker setup being used for the GitHub action tests.

## E2e testing plan

There is no explicit requirement for end-to-end (e2e) testing.

### **TESTING THE NEW SETUP**

It is imperative to conduct thorough testing of the entire Oppia development server setup across all platforms, including Linux, Windows, and macOS (with Intel chips, M1, and M2), to ensure the smooth functioning of every page and functionality, without encountering any errors.

Additionally, the testing must ensure that developers can continue to use Git commands as they presently do, while the various services utilized in the application, such as Redis, Elasticsearch, Cloud Datastore, and Firebase, perform their intended functions as expected.

**SOLUTION:** Given that I possess a Mac with an M1 chip, I am capable of conducting thorough tests of the development server setup, as well as all its functionalities on this platform.

However, for testing on other systems, I plan to seek help from my friends, as well as I can ask for support in the Oppia community.

We will have a grace period from around 15th July to 10th Sept. In this period, both the old setup (using python scripts) and the new setup (using Docker) will be functional, and this will ensure the smooth transition to the new development setup for the devs.

### **CONFLICT WITH OTHER PROJECTS**

The "Make CI faster" project involves making changes to the GitHub Workflows and pre-push Python scripts, as well as other scripts that may require modifications. However, these modifications may conflict with the changes that I intend to introduce in this project.

**SOLUTION:** To resolve any potential conflicts arising in such a situation, I will engage in a discussion with the contributor regarding their approach and ideas. This communication can help identify any areas of potential conflict with the ongoing projects and facilitate prompt resolution of any issues that may arise.



# Implementation Plan

## Milestone 1

**Key objective for this milestone:** Developers are able to set up and start the Oppia development server using appropriate Make commands on all platforms utilizing Docker, without errors, and with smooth operation of all Oppia webpages. There will be a Docker Action that will test the setup and ensure anyone won't break my changes. The `dependencies.json` file should be completely removed from the codebase, and developers can refer to updated wiki pages regarding setting up of Oppia development server using Docker. Developers are able to run all the tests within the docker container environment.

No.	Description of PR / action	Prereq PR numbers	Target date for PR creation	Target date for PR to be merged
1.1	Approximately half of the dependencies from dependencies.json file are migrated to the package.json. The dependencies migrated within this PR – angular, lamejs, angularTest, angularTranslate, angularTranslateLoader, angularDragAndDrop, angularStorageCookiesRev, messageFormat, angularTranslateInterpolationMessageFormat, popperJs, bootstrap, bowerAngularTranslateLoaderPartial, bowerMaterial, codemirror.	-	1 June	5 June
1.2	The dependencies.json file must be removed from the codebase, and all the dependencies from the dependencies.json file must be migrated to package.json.	-	7 June	10 June
1.3	Developers on all platforms are able to set up the fully functional local development server using the Make commands (run-offline, run, setup-devserver, clean, terminal). The `make	1.1, 1.2	20 June	25 June

	run` and `make run-offline` commands can be executed with all the flags (save_datastore, disable_host_checking, prod_env, maintenance_mode, no_auto_restart and source_maps) functioning we currently have for the start.py. A checksum verification mechanism must verify the installation of the dependencies. There is a Docker Action that tests the Docker setup in our application.			
1.4	Developers can run all the tests, including backend, frontend, lint, e2e, etc., successfully within the Docker container environment.	1.3	4 July	10 July
1.5	The new "Installing Oppia" wiki page will be created and published, and the existing per-OS "Installing Oppia" wiki pages will be removed. Refer <a href="#">#here</a>	1.3, 1.4	11 July	14 July

## **Milestone 2**

**Key objective for this milestone:** All contributors must have updated wiki pages with all relevant information regarding migrating to the new Oppia development server setup using Docker, how to debug Docker setup-related issues, and guidance related to the basic understanding of Docker. Testing of the new dockerized setup in all platforms (MacOS, Linux, Windows) will be done. Developers see CI test runs on GitHub sped up because of using cached Docker Images in their build step. The docker action is removed from the workflow. Developers run tests locally using Docker only, the if-else conditions in test scripts for supporting both python and docker setup will be removed and tests can only execute with Docker setup.

No.	Description of PR / action	Prereq PR numbers	Target date for PR creation	Target date for PR to be merged
2.1	The "Debugging dockerized Oppia setup" wiki page will be created and published. Refer <a href="#">#here</a> . Wiki pages will keep updating based on the	-	10 Aug	17 Aug

	<p>issues faced by the developers.</p> <p><b>Ask all the developers to create an alternative setup using Docker, and report issues they face while setting up the Docker setup and within the development server.</b></p> <p><b>Testing the entire dockerized setup in all platforms (MacOS with Intel/M1/M2 chips, Linux, Windows) will start including the testing of all docker-compose services, git commands, running tests.</b></p>			
2.2	<p>Developers can experience sped-up CI test runs on GitHub Actions because of utilizing cached Docker Images in the build step of the runs. The Docker Action is removed from the workflows.</p>	-	3 Sept	9 Sept
2.3	<p>The python scripts - start.py, servers.py, install_third_party.py, install_third_party_libs.py, install_python_dev_dependencies.py, install_python_prod_dependencies.py, install_chrome_for_ci.py setup_gae.py and setup.py are deprecated, together with their corresponding test scripts. All tests will run using Docker only (this includes removing the if-else conditions that supported both python and docker setup, and test scripts will only support docker setup for their execution).</p> <p><b>Now, since the new Docker setup is completed and thoroughly tested, we can force devs to switch to the Docker setup for the development server.</b></p>	2.1, 2.2	11 Sept	14 Sept

## Demo Plan to the Product Manager

I plan to demo the working parts of the project during the milestones to the Product Manager, and here are the proposed timelines –

DATE	Working Parts that can be showed
27 June	Dockerized Oppia setup (local tests will not be functional under the Docker setup)
12 July	Local tests fully functional within the dockerized setup
11 Sept	Final product with all project requirements completed

### Future Work

**Note:** This section is mainly for reference (since it is understood that items in this section will not be part of the GSoC project). Proposals will primarily be evaluated based on the implementation plan above.

- Create Dockerfiles for the production environment as well (docker-compose.prod.yml, Dockerfile.prod, etc..)
- Run Docker Hub actions on a Docker Hub repository - Oppia, where users can simply pull the published images, allowing quick deployments.