



GSoC 2022 Proposal

Mentors:

- Mr. Willem Van Iseghem (@canihavesomecoffee)
- Mr. Shivam Jha (@thealphadollar)

Sample Platform

Tarun Arora

Table of contents

GSoC 2022 Proposal

1. Personal Information	...4
2. Commitments	...5
3. Introduction to the Project	...6
❖ Technologies used at present	
❖ Scopes of Improvement	
❖ My Primary Goals	
❖ Why Google Cloud Platform?	
❖ Why need Sample Platform?	
❖ Monthly Cost Prediction for the Platform	
4. Implementation of the Project Goals	...12
<u>Migrate website to a GCP Instance</u>	
• Creation of Bucket for Regression Tests	
• Linking GitHub with GCP	
• Migrating platform to an n1-standard-4 instance	
<u>Rewrite mod_ci module</u>	
• KVM model	
• Modify start_platforms()	
• Replace kvm_processor()	
• Modify progress_type_request()	
• Modify cron.py	
<u>Adding tests for current and updated modules</u>	
• Full rewrite	
<u>Current Code Refactoring</u>	
• Removal of unused imports.	
• Formatting of code.	
<u>Updating GitHub Actions for Check of Build Success</u>	
• Add check for GitHub Linux and Windows Build both	
<u>Updating mod_sample module</u>	
• Redirecting fetch sample information, uploading, editing, deleting sample to and from the samples bucket.	
<u>Reference Links</u>	

5. Timeline	...22
❖ Pre-GSoC Period	
❖ Community Bonding Period	
❖ Week 1 - 6 Targets & Plan	
❖ GSoC'22 Phase 1 Evaluations	
❖ Week 6-11 Targets & Plan	
❖ Final Week (Week 12)	
❖ Final GSoC'22 Evaluation Week	
❖ GSoC'22 Results	
❖ GSoC'22 Extended Timeline	
❖ Submission of Final Work Product	
6. My Past Contributions to CCExtractor	...24
Issues Involved	
PRs created	
7. Why are you the best person to execute this proposal?	...25
8. Prior Experience	...25
9. Post GSoC plans	...25

Personal Information

Name:	Tarun Arora
University:	Indian Institute of Technology (BHU), Varanasi
Field of Study:	Computer Science & Engineering
Date of Enrollment:	November 2020
Expected Graduation date:	July 2024
Degree:	Bachelor of Technology
Year:	Sophomore
Github:	https://github.com/Tarun-Arora
LinkedIn:	https://www.linkedin.com/in/03-Tarun/
Slack:	Tarun Arora
Email Address:	tarun.arora.cse20@itbhu.ac.in tarun.arora.030402@gmail.com
Phone number:	(+91) 6307964780
Timezone:	Indian Standard Time (UTC +5:30)

Commitments

★ **How many hours will you work per week on your GSoC project?**

I plan to spend 40 - 50 hours on the project per week.

★ **Do you have access to both Windows and Linux for development?**

Yes.

★ **Other Commitments, if any?**

I have no other commitments during the GSoC period.

★ **Do you plan to apply for any other organization for GSoC'22?**

I am only applying to CCExtractor for GSoC'22 and have no plans to contribute to any other organization.

★ **If you get selected as a GSoC student, would you like to work on other tasks besides the projects of your choice?**

Yes, I would love to work on other tasks unrelated to my GSoC project.

★ **If you do not get selected as a GSoC student, would you like to work on the projects as a general contributor?**

Yes, even if I'm not selected as a GSoC participant, I would happily continue working as a general contributor.

★ **Would you like to contribute to CCExtractor in the long term, after the GSoC program ends?**

Yes, I would like to contribute to CCExtractor even after GSoC ends.

★ **What motivated you the most towards applying for GSoC?**

There are various reasons for which I wanted to apply for GSoC, and I love the fundamental concept of Open Source Contributions; therefore, my motive is to get recognized as a GSoC participant, which provides learners like us to give hands-on-experience with it. Also, the stipend was not a motivating factor, but an opportunity to work with a big organization like CCExtractor indeed is.

Introduction To The Project

Sample Platform is CCExtractor's platform to manage tests, upload samples, run regression tests, and much more. The platform's primary purpose is to test every pull request of the CCExtractor repository with the samples uploaded (which currently have 180GB of samples) and report status.

Technologies used at present

- Sample Platform is built over Python and uses Bash Scripting for its installation on Linux.
- For testing the platform, Python Unit Testing Framework is used.
- The Platform is triggered via GitHub webhooks whenever a PR is created on CCExtractor.
- Rendering of its templates follows the basic HTML, CSS, and JavaScript architecture.

Scopes of Improvement

- **Long Runtime Issue**
Currently, the platform runs a test using KVM instance, so it can run only one test at an instant which is a major drawback and sometimes leads to the creation of a large queue of tests.
- **Comparison with Multiple Versions**
We should be able to have multiple approved versions and tell the user if the result deviates from those known ones so that it would be clear what the tests would be against the version of CCExtractor.
- **Updating GitHub Actions for Check of Build Success**
Currently, the CCExtractor workflows trigger the platform as soon as any PR is created, this would also start VM instances for those PRs which might fail the build on Linux and/or Windows, which would lead to unnecessary tests, therefore we would trigger the platform only when build succeeds on both the operating systems. This would reduce wasting time when a build fails anyway.

My Primary Goals

- **Migrate website to a GCP Instance**
 - The current website should be migrated to a GCP instance; this would replace the current KVM setup and would allow us to launch multiple disposable machines as per need and, therefore, would help to eliminate the queue that sometimes builds up.
- **Rewrite mod_ci module**
 - The current mod_ci module launches a KVM instance when triggered and determines whether to queue the test or to run it. With the migration of platform to cloud, the queuing process can be eliminated by launching instances when required.

- The current module launches an instance with the help of the method `kvm_processor()`. This method can be replaced by adding a new method which is defined by `launch_gcp_instances()`. The task of `launch_gcp_instances()` is to launch Google cloud instances whenever triggered, using Google Cloud Compute Python API.
- **Adding tests for current and updated modules**
 - After rewriting the `mod_ci` module, unit tests need to be updated. This will ensure the CodeCov test coverage of the platform doesn't go down.
- **Current Code Refactoring & Documentation Update**
 - There are quite a few modules having unused imports included in the code, which is not required, hence removing them would improve code readability.
 - The Sample platform uses docstrings heavily to document modules and methods. After rewriting the `mod_ci` module and updating the `mod_sample` module we need to update any related documentation so that it would be easy to understand and maintain the code in the future.
- **Updating GitHub Actions for Check of Build Success**
 - Currently, the CCExtractor workflows trigger the platform as soon as any PR is created, this would also start VM instances for those PRs which might fail the build on Linux and/or Windows, which would lead to unnecessary tests, therefore we would trigger the platform only when build succeeds on both the operating systems.
 - Also, this would stop the deployment for the corresponding PR, whenever the build process fails anyways.
- **Update `mod_sample` module**
 - The `mod_sample` is concerned with uploading, downloading, and generating media information for the same. After migrating the samples to a bucket, we need to update the module to allow it to mount the bucket to the platform and allow the platform read-write access to it for the module to work. This makes it possible to upload and download samples from the bucket.

Why Google Cloud Platform?

The migration of the current website to a GCP instance would help us resolve the current issues in the platform, and the parallelization would save us time, but here are some of the reasons for choosing Google Cloud Platform specifically, for the platform:

1. Compute Engine

Any commit or a PR in CCExtractor would trigger the platform to run the tests by creating a GCP instance for it, and therefore the speed of creation would be a vital factor and GCP instances are much faster than other cloud service providers.

Google's instances start up much faster than Amazon's, by a factor about 60-80%. And in a DaCapo benchmark suite run in March 2014, GCE instances completed the 14-test suite in a score of 575 seconds compared to Amazon's 719 seconds. In fact, a Google machine had the fastest time in 13 of the 14 tests.

References: <https://shorturl.at/Q0238>

2. Uptime

Google Cloud Infrastructure-as-a-Service (IaaS) is the top-performing public IaaS cloud. The Amazon Web Services (AWS) Elastic Compute Cloud (EC2) recorded 2.41 hours of downtime across 20 outages in 2014, while the Google Cloud Platform storage service experienced 14 minutes of downtime for a 99.9996 uptime percentage.

Google Compute Engine	Microsoft Azure	AWS Simple Storage Service (S3)
Experienced 72 outages for 4.42 hours of downtime in 2014.	Experienced 92 outages totaling 39.77 hours of downtime in 2014, and its storage platform had 141 outages totaling 10.97 hours of downtime.	Experienced 23 outages and 2.69 hours of downtime in 2014. Also, about 10 percent of AWS EC2 instances had to be rebooted in 2014.

References: <https://shorturl.at/dhjbB>

3. Pricing

Launching a VM instance on GCP is the fastest among all cloud service providers and takes around 35 seconds to boot up and apart from this Google allows its users to pay per second which is very useful for the Platform as any triggering of the platform would launch VM instances which would last merely 10 minutes, hence this would reduce our cost consumption as we would not be paying on the hourly basis billing on other platforms.

General Comparison for VM instances with 4CPUs and 16 GiB Memory

Instance Type	AWS	Azure	Google	AWS Pricing(per hour)	Azure Pricing(per hour)	Google Pricing(per hour)
General Purpose	m6g.xlarge	B4MS	e2-standard-4	\$0.154	\$0.166	\$0.134

Why need Sample-Platform?

The Platform helps to manage tests, upload samples, run regression tests, and much more, and report its status for every new version of CCExtractor.

The platform ensures that changes proposed in the pull request are safe and secure. This helps to maintain the stability of the CCExtractor.

Sample Platform aims to ensure that CCExtractor and other organizations can use it as part of their development process.

Monthly Cost Prediction for the Platform

Below will be the metric used to find the cost prediction for a month and this is considered for the worst-case scenario.

$$\max(\text{no: of tests}) = \max(\text{no: of pull requests in a month}) + \max(\text{no: of commits merged to master})$$

$$\rightarrow \max(\text{no: of commits merged to master}) = \max(\text{no: of pull requests in a month})$$

$$\rightarrow \max(\text{no: of tests}) = 2 * \max(\text{no: of pull requests in a month})$$

The maximum number of commits merged in a month is equal to the maximum number of pull requests in a month, since to the very extent there are chances that every pull request will be merged to the master branch.

Now if we browse to the CCExtractor repository and try to calculate the number of PRs per month, by applying filters like [applied here](#), we would find that on average we would be getting nearly 5 PRs a month. i.e 20 (5*2*2) tests to run.

But, to be on the safe side we would make it 50 PRs a month to get the maximum possible cost of the platform. So the number of tests executed in a month won't exceed 100. Since there are two platforms i.e. Linux and Windows, the total number of tests executed will be at most 200.

Google Compute Engine (GCE) is the Infrastructure as a Service (IaaS) component of Google Cloud Platform which enables users to launch virtual machines (VMs) on demand.

Data storage: The amount of data stored in your buckets. Storage rates vary depending on the storage class of your data and location of your buckets.

Data processing: The processing done by Cloud Storage, which includes operations charges, any applicable retrieval fees, and inter-region replication.

Network usage: The amount of data read from or moved between your buckets.

Operation charges apply when you perform operations within Cloud Storage. An operation is an action that makes changes to or retrieves information about resources such as buckets and objects in Cloud Storage.

Operations are divided into three categories: Class A, Class B, and free.

API or Feature	Class A operations	Class B operations	Free Operations
JSON API	storage.*.insert1 storage.*.patch storage.*.update storage.*.setIamPolicy storage.buckets.list storage.buckets.lockRetentionPolicy storage.notifications.delete storage.objects.compose storage.objects.copy storage.objects.list storage.objects.rewrite storage.objects.watchAll storage.projects.hmacKeys.create storage.projects.hmacKeys.list storage.*AccessControls.delete	storage.*.get storage.*.getIamPolicy storage.*.testIamPermissions storage.*AccessControls.list storage.notifications.list Each object change notification	storage.channels.stop storage.buckets.delete storage.objects.delete storage.project.hmacKey.delete

Google Cloud Service	Service Type	Pricing (per month)	Our Average Usage	Minimum Possible Price	Maximum Possible Price	Aggregate Price Expected
Data Storage	Standard storage (Bucket)	\$0.02/GB	180 GB (samples + tester)	\$3.60/mo	\$3.60/mo	\$3.60/mo
Network	Standard storage	\$0.12/GB	0.5GB (passing metadata such as XML files)	\$0/mo	\$0.6/mo	\$0.06/mo
Class A Operations	Standard storage	\$0.05 per 10,000 operations	100 writes (upload samples)	\$0/mo	\$0.005/mo	\$0.0005/mo
Class B Operations	Standard storage	\$0.004 per 10,000 operations	20*(200) reads (20 tests, Each test has access to around 200 samples)	\$0/mo	\$0.016/mo	\$0.0016/mo
VM instances	n1-standard-1(memory 3.75GB , 1 vCPU)	\$0.04749975 per hour	20 instances launched in a month and each test takes around 40 minutes	\$0/mo	$(40/60)*200*0.04749975 = \$6/\text{mo}$	$(40/60)*20*0.04749975 = \$0.6/\text{mo}$
Platform (Server)	n1-standard-4(memory 15GB, 4 vCPU)	\$97.09 per month	24*30 = 720 hours per month	\$97.09/mo	\$97.09/mo	\$97.09/mo
Total				\$100.69/mo	\$102.811/mo	\$100.9021/mo
Safe Deviation Assumptions				\$0/mo	\$10/mo	\$5/mo
Total Cost				\$100.69/mo	\$117.311/mo	106.3521/mo

Implementation of the Project Goals

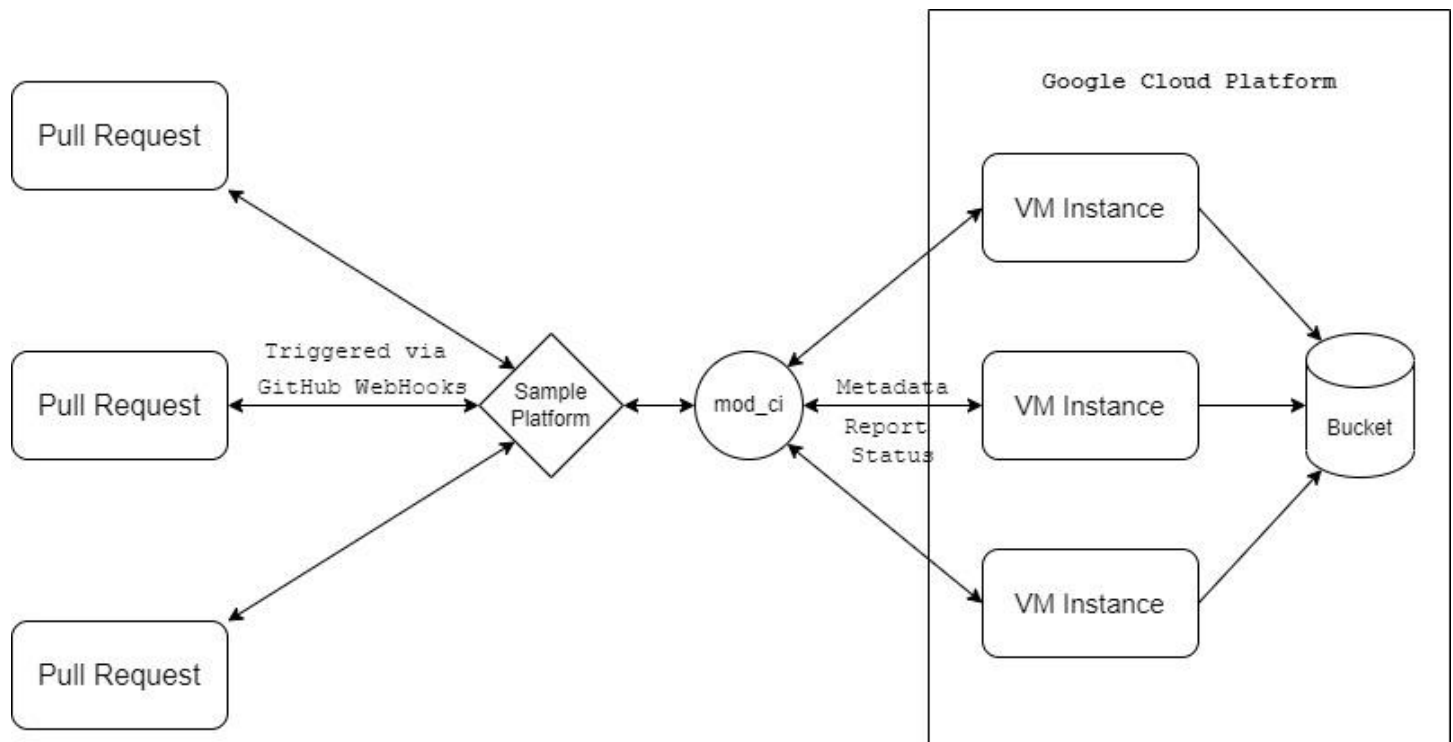
Migrate website to a GCP Instance

- Creation of Bucket for Regression Tests.

The first step is to make a bucket to hold all of the regression test samples, as well as the tester and runCI file. The VM instances would be able to mount the bucket and access the samples, tester, and runCI file due to this.

This would involve the creation of a 180GiB of standard cloud storage, location "lowa (us-central1)" on the google cloud console. Fetching the bucket credentials to allow further authenticated operations through the bucket.

- Migrating platform to an n1-standard-4 instance



The next step involves the creation of an n1-standard-4 instance of Google Cloud Compute Engine, setting up the project on the GCP instance, and then further setting up the GitHub webhooks for continuous deployment.

For Automated Deployment of the platform, the following procedure would be followed:-

Every commit on the master branch would trigger a script that would first connect to the GCP instance through ssh using gcloud compute ssh command as shown as follows:

```
gcloud compute ssh example-instance --zone=us-central1-a --command="bash -i -c env" -- -t
```

Reference Links:

<https://cloud.google.com/sdk/gcloud/reference/compute/ssh>

Secondly, after connecting to the instance we would use our mod_deploy module to fetch and pull the latest code and restart the server.

Reference Links:

https://github.com/CCExtractor/sample-platform/tree/master/mod_deploy

Useful Links:

<https://cloud.google.com/python/docs/getting-started/getting-started-on-compute-engine>

<https://www.section.io/engineering-education/deploy-flask-to-gce/#deploying-the-application>

<https://medium.com/google-cloud/run-shell-commands-and-orchestrate-compute-engine-vms-with-cloud-workflows-e345e616a24>

Rewrite mod_ci module

For a google cloud instance to be launched using the mod_ci module, changes are to be done. The changes would include removing the existing KVM-based method and writing a new method to launch google cloud compute instances. This would require a GoogleAPIClient module, so the very first step would be to import it and build and configure a compute instance-based machine.

→ Modify start_platforms (removal of python multiprocessing module)

This function is used to start a new test on both platforms in parallel.

Currently, we use a multiprocessing module that bypasses Python GIL to make use of multiple cores of the processor. But now we can start the tests for both the operating systems parallelly on different compute instances. Therefore we no longer need this module to start our VM instances parallelly. Currently, it calls the kvm_processor() function for starting the instances but we would replace it with a gcp_instance() function.

But first, we need to authorize our connection to the google cloud platform to access our services, which would authenticate us with our GCP connection.

The credentials that we need are called "Service Account JSON Key File". These are created in the Google Cloud Console under IAM & Admin / Service Accounts. We would need to create a service account and download the key file.

The Sample Authentication would be as follows using a service-account.json file.

```
from googleapiclient.discovery
from google.oauth2 import service_account

scopes = ['https://www.googleapis.com/auth/cloud-platform']
sa_file = 'service-account.json'
zone = 'us-central1-a'
project_id = 'my_project_id' # Project ID, not Project Name

credentials = service_account.Credentials.from_service_account_file(sa_file, scopes=scopes)

# Create the Cloud Compute Engine service object
service = googleapiclient.discovery.build('compute', 'v1', credentials=credentials)
```

Now our start_platforms function would require the following modifications:-

```
if platform is None or platform == TestPlatform.linux:
    gcp_instance(current_app._get_current_object(), db, TestPlatform.linux, repository, delay,)

if platform is None or platform == TestPlatform.windows:
    gcp_instance(current_app._get_current_object(), db, TestPlatform.windows, repository, delay,)
```

In the above block of code, the method which launches the VM is execute() and we would create a gcp_instance() function that would return a VM instance by replacing the kvm_processor() function.

→ Replace kvm_processor()

Some changes must be made before a Google Cloud instance could be launched using the mod_ci module. The modifications include eliminating the existing kvm_processor() method and replacing it with a new way of launching Google Cloud instances. Because the GoogleAPIClient module is required, the first step would be to import it and create a compute service object.

```
import googleapiclient.discovery
compute = googleapiclient.discovery.build('compute', v1)
```

The following step is to select an image family and to set up the config. Depending on the TestPlatform, the image family will change. The n1-standard-instance machine type is being chosen.

For Linux Operating System we currently choose:



```
NAME: ubuntu-minimal-2004-focal-v20220325
PROJECT: ubuntu-os-cloud
FAMILY: ubuntu-minimal-2004-lts
DEPRECATED:
STATUS: READY
```

For Windows Operating System we currently choose:



```
NAME: windows-server-2019-dc-v20220314
PROJECT: windows-cloud
FAMILY: windows-2019
DEPRECATED:
STATUS: READY
```

Therefore we now have an overall blueprint to start our compute instances:

```

def gcp_instance(app, db, platform, repository, delay) -> None:
    ##Some Code Above
    # Code To Fetch Config Details
    # Code To Start A GCP instance
    if platform == TestPlatform.linux:
        image_response = compute.images().
            .getFromFamily(project='ubuntu-os-cloud',
                           family='ubuntu-minimal-2004-lts').execute()

    if platform == TestPlatform.windows:
        image_response = compute.images().
            .getFromFamily(project='windows-cloud',
                           family='windows-2019').execute()

    source_disk_image = image_response['selfLink']

    # Configure the machine
    machine_type = "zones/%s/machineTypes/n1-standard-1" % zone
    startup_script = open(
        os.path.join(os.path.dirname(__file__), 'startup-script.sh'), 'r').read()

    config = {
        'name': name,
        'machineType': machine_type,

        # Specify the boot disk and the image to use as a source.
        'disks': [
            {
                'boot': True,
                'autoDelete': True,
                'initializeParams': {
                    'sourceImage': source_disk_image,
                }
            }
        ],

        # Specify a network interface with NAT to access the public
        # internet.
        'networkInterfaces': [{
            'network': 'global/networks/default',
            'accessConfigs': [
                {'type': 'ONE_TO_ONE_NAT', 'name': 'External NAT'}
            ]
        }],

        # Allow the instance to access cloud storage and logging.
        'serviceAccounts': [{
            'email': 'default',
            'scopes': [
                'https://www.googleapis.com/auth/devstorage.read_write',
                'https://www.googleapis.com/auth/logging.write'
            ]
        }],

        # Metadata is readable from the instance and allows you to
        # pass configuration from deployment scripts to instances.
        'metadata': {
            ## Would Be Explained in Further Sections
        }
    }

    compute.instances().insert(project=project, zone=zone, body=config).execute()
    # Some Other Code to Merge Code to Repo

```


References:

https://github.com/GoogleCloudPlatform/python-docs-samples/blob/main/compute/api/create_instance.py

We've started the VM, but it needs to be acknowledged whether it's started successfully or not. We verify the status of the VM when it has completed its initialization.

```
print('Waiting for operation to finish...')
while True:
    result = compute.zoneOperations().get(project=project, zone=zone, operation=operation).execute()
    if result['status'] == 'DONE':
        print("done.")
        if 'error' in result:
            raise Exception(result['error'])
```

Although we have successfully launched a VM instance, more prerequisites are required for the VM to mount the bucket and execute tests. A startup script must be given as metadata from the mod_ci module to install these dependencies.

```
'metadata': {
'items': [{
    'key': key,
    'value': startup_script
}, {
    'key': 'reportURL',
    'value': report_url
}],
}

for category in categories:
    {
        # XML file is first read and then passed as value
        # XML value can also be passed directly, this would
        # eliminate storing XML files even on the platform
        'key': category.name,
        'value': open(os.path.join(os.path.dirname(__file__), f'{category.name}.xml'), 'r').read()
    }
}
```

For Windows VM instances the startup script can be executed only by using unique, Windows-specific metadata keys. Metadata Keys such as windows-startup-script-cmd can be used to launch the startup script.

```

if platform == TestPlatform.linux:
    key = 'startup-script'
if platform == TestPlatform.windows:
    key = 'windows-startup-script-ps1'

```

The startup script will install necessary dependencies, mount the bucket and execute the runCI file.

```

#!/bin/bash
sudo apt-get install gcsfuse
mkdir bucket
gcsfuse samples bucket
./root/bucket/runCI

```

We would configure the startup scripts to run different runCI files for both the operating systems.

→ Modify upload_type_request()

The XML files, Google credentials, and the reportURL are all available. The startup script mounts the bucket and runs the runCI script. However, if two tests, triggered by two different pull requests, produce the same result and try to upload the same result file to the platform's TempFiles directory, a concurrency issue will occur, resulting in the corrupt content being uploaded back to the server. Changing the temp files directory to TempFiles/token/ in the method upload_type_request() is the quickest and easiest way to solve this issue. A similar modification is required in other similar methods like upload_log_type_request(args).etc.

```

def upload_type_request(log, test_id, repo_folder, test, request) -> bool:
    #temp_path = os.path.join(repo_folder, 'TempFiles', filename)
    temp_path = os.path.join(repo_folder, 'TempFiles' + token, filename)

```

→ Modify progress_type_request()

This method currently utilizes the KVM model to get the current progress of the tests passed. For replacing the current KVM model with the Cloud Instances Model, the function would require only some minor modifications because a very similar architecture would be followed for the new model.

→ Modify cron.py

Currently, this file utilizes the function kvm_processor(), which is to be replaced by gcp_instance(), which would then look like this:

```
...
if testing is True:
    gcp_instance(current_app._get_current_object(), db, TestPlatform.linux,
repository, delay)
...
```

Adding tests for current and updated modules.

It's important to update the unit tests now that the mod_ci module has been updated so the platform's CodeCov test coverage doesn't suffer. It will be easier to work with GCP instances if we mock newly developed methods. These would basically involve updating tests for the following features:

- Mocking for GCP instances.
- Remove Mocking for KVM machines
- Remove Mocking for Python Multiprocessing module
- Mocking for GCloud Authorization

Also we would have to update other tests or rewrite them if they fail due to our changes in mod_ci module, to maintain our test coverage.

```
@mock.patch('google.oauth2.service_account.Credentials.from_service_account_file')
def test_gcp_authorization(self):
    """
    Test that GCP doesn't launch without gcloud credentials.
    """
    from googleapiclient.discovery
    from google.oauth2 import service_account

    scopes = ['https://www.googleapis.com/auth/cloud-platform']
    sa_file = 'service-account.json'
    zone = 'us-central1-a'
    project_id = 'my_project_id' # Project ID, not Project Name

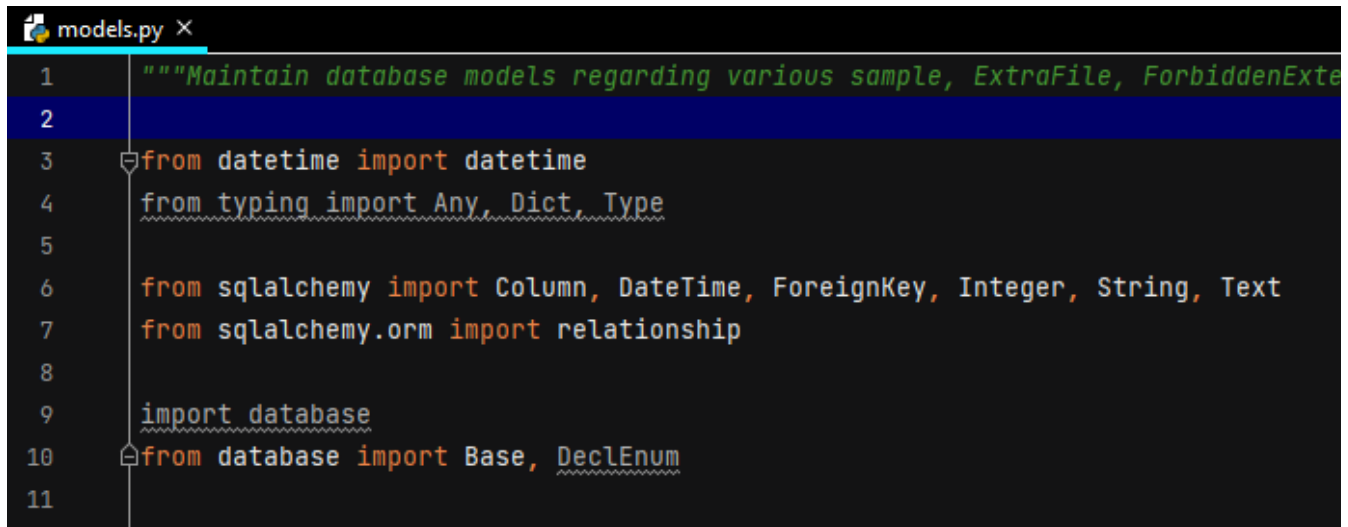
    credentials = service_account.Credentials.from_service_account_file(sa_file,
scopes=scopes)
    self.assertIsNone(credentials)

    # Create the Cloud Compute Engine service object
    service = googleapiclient.discovery.build('compute', 'v1', credentials=credentials)
    self.assertIsNone(service)
```

Current Code Refactoring

→ Removal of unused imports.

There are quite a few modules that have unused imports, which could be removed, and this would improve the readability of the code.



```

1  """Maintain database models regarding various sample, ExtraFile, ForbiddenExt
2
3  from datetime import datetime
4  from typing import Any, Dict, Type
5
6  from sqlalchemy import Column, DateTime, ForeignKey, Integer, String, Text
7  from sqlalchemy.orm import relationship
8
9  import database
10 from database import Base, DeclEnum
11

```

→ Formatting of code.

Sample-platform follows certain etiquettes which include docstrings, annotation, import sorting, etc.

Sample-platform uses docstrings heavily to document modules and methods.

Therefore, documentation of newly created methods and replacing all the KVM-related documentation with Google Cloud Platform information would be done.

Updating GitHub Actions for Check of Build Success

→ Add check for GitHub Windows & Linux Build before triggering the platform

Currently, the CCExtractor workflows trigger the platform as soon as any PR is created, this would also start VM instances for those PRs which might fail the build on Linux and/or Windows, which would lead to unnecessary tests, therefore we would configure the GitHub workflows to hit the webhooks after the build succeeds on both platforms. This might be a bit time-consuming but this is cost-effective and would save resources.

References:

<https://www.freecodecamp.org/news/how-to-use-github-actions-to-call-webhooks/>

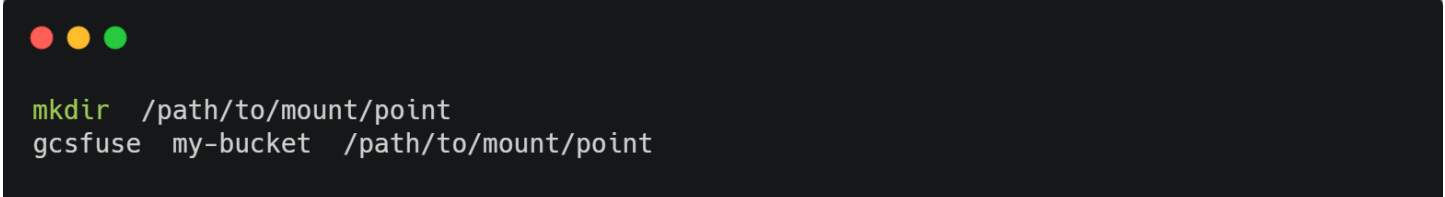
Updating mod_sample module

→ Redirecting fetch sample information, uploading, editing, deleting samples to and from the samples bucket.

It is essential to mount the bucket on the platform and grant read-write access to it in order for the existing upload and download methods to work. Users will be able to upload, download, and update samples as a result of this. Google Cloud Fuse can be used to mount the bucket.

Google Cloud Storage FUSE is an open-source FUSE adapter that allows you to mount Cloud Storage buckets as file systems on Linux or macOS systems. It also provides a way for applications to upload and download Cloud Storage objects using standard file system semantics. Cloud Storage FUSE can be run anywhere with connectivity to Cloud Storage, including Google Compute Engine VMs or on-premises systems.

We need to set the mount path. By default, files are currently stored in /repository directory.



```
mkdir /path/to/mount/point
gcsfuse my-bucket /path/to/mount/point
```

Reference Links:-

<https://github.com/googleapis/google-cloud-cpp>

<https://cloud.google.com/storage/pricing>

Timeline

April 19 - May 20, 2022 <u>Pre-GSoC Period</u>	<p>Before the GSoC period start, I should be completely comfortable with majorly all the services of GCP planned to be used in the platform. Therefore these weeks, I'm going to look into the Google Cloud Platform services. (Getting acquainted with Compute Engine, Cloud Storage, and other Google Cloud services).</p>
May 20 - Jun 13, 2022 <u>Community Bonding Period</u>	<p>It'd be easier for me to join the community since I've been contributing to Sample Platform for the past three months. The main goal for the first week of this period will be to have a better understanding of Google Cloud services, create a blueprint for them, and clarify my doubts with my mentors. From the second week on, I'll be working on finalizing the platform's blueprint/design of the new mod_ci module, with frequent inputs from the mentors.</p>
June 13 - July 25, 2022 <u>Coding officially begins!</u>	<p>The first week of this period will be spent migrating the platform to a Google Cloud n1-standard-4 instance, setting up a bucket, and storing samples test suites, and modifying the runCI files. This week will be spent on micro tasks and getting underway on rewriting the mod_ci module.</p> <p>The next step after migrating the server to a Google cloud instance is to rewrite the mod_ci module and configure the newly deployed VM instances. This time period will be critical to the project's success. During this time, work such as rewriting the mod ci module, configuring both Linux and Windows VM instances, writing startup scripts, passing metadata, retrieving meta-data, and altering the runCI file will be performed.</p>
July 29 <u>Phase 1 Evaluation</u>	<p>Submission of the report of work done in the Coding Period.</p> <ul style="list-style-type: none"> • Migrating server to n1-standard-4 instance • Initializing bucket • Rewriting mod_ci module • Configuring launched VM instances
July 25 - September 4, 2022 <u>Work Period</u>	<p>The first week of this period will be used to finish any outstanding work from Coding Period 1, if any, before moving on to the next set of promised deliverables.</p> <p>In the subsequent weeks, I'll be rewriting unit tests for the mod ci module, making necessary changes to mod_sample, and mounting the bucket on the server so that samples can be posted and downloaded.</p> <p>Each modularized activity will have documentation supplied, making it simple for contributors to set up the new project.</p>

September 5-September 12, 2020 <u>Final week</u>	Submission of the report of work done in the Coding Period. <ul style="list-style-type: none"> • Rewrite unit tests for mod_ci module • Updating GitHub Actions • Updating mod_sample module • Code Refactoring • Documentation Updation
September 12-September 19	Final GSoC contributor evaluations
September 20	Initial results of GSoC 2022
September 12 - November 13 <u>Extended timelines</u>	Because 12 weeks is plenty of time, there are very few chances that certain tasks will go unfinished, but if that happens in any case, I'll finish them during this period.
November 21	Final date of submission of final work product

My Past Contributions to CCEXtractor

I started contributing to CCEXtractor in late December 2021 and continued exploring and learning.

Issues Involved:

- Bugs Related To Updated Dependencies
 - <https://github.com/CCEXtractor/sample-platform/issues/606>
- Proposal to Implement Dark Theme for the Platform
 - <https://github.com/CCEXtractor/sample-platform/issues/273>
- Bugs Related to foundation up-gradation
 - <https://github.com/CCEXtractor/sample-platform/issues/620>
- Question-Related to setting up the local environment for platform installation.
 - <https://github.com/CCEXtractor/sample-platform/issues/508>

All the issues opened by me are listed [here](#).

PRs created:

Merged:

- Bug Fixes Related to Installation
 - <https://github.com/CCEXtractor/sample-platform/pull/612>
- Implemented Dark Theme with Toggling
 - <https://github.com/CCEXtractor/sample-platform/pull/613>
- Fixed Foundation Upgrade Issues
 - <https://github.com/CCEXtractor/sample-platform/pull/621>

Others Closed:

- <https://github.com/CCEXtractor/sample-platform/pull/608>
- <https://github.com/CCEXtractor/sample-platform/pull/624>
- <https://github.com/CCEXtractor/sample-platform/pull/627>

All the PRs created by me are listed [here](#).

Why are you the best person to execute this proposal?

I have fair experience building apps using Python and HTTP APIs and have a good knowledge of Git, GitHub, and Google Cloud Platform, the required tech-stack for the project. I also know the importance of writing clean and scalable code. I have worked on several projects. Therefore, I do have the skillset and experience to execute this project. Also, I have an interest in testing frameworks as well.

Prior Experience

I have been doing Web Development since the last year. I started with Frontend frameworks after getting acquainted with the basics of web development.

I am comfortable with frameworks like VueJS, NuxtJS, ReactJS, and NextJS.

I worked with Techlious Network as a ReactJS Developer Intern. During this Internship, I added React JS to my skill set and got experience with AWS services.

I have also made an ASGI web application named Commutify, based on Django, DRF, Django WebSockets.etc., a real-time communication platform.

I am also comfortable with NodeJS and worked with Firebase as well.

I also contributed to developing our Institute's Software Development Group's Website. I also completed Hacktoberfest 2021.

And from the last six months, I have contributed to open-source organizations.

During this period, I learned about the CCExtractor through one of my seniors. And since the last 2 to 3 months, I have been contributing to CCExtractor projects and specifically to the sample platform.

Post GSoC plans

There are very few chances that some things might go unimplemented because 12 weeks is plenty of time, but still, If it happens, I'll complete them post-GSoC, during the extended timeline period .

I'll keep contributing to CCExtractor to the best of my ability and participate in the community's discussions.

Regardless of GSoC, I would love to engage in discussions with the CCExtractor community to get exposure to new technologies and Ideas. I would love to be of any help even after the GSoC period.