Sample Platform

CCExtractor's Sample and Continuous Integration Platform

March 21, 2021

1 Abstract

Sample Platform is CCExtractor's platform to manage tests, upload samples, run regression tests and much more. The main purpose of the platform is to test each and every pull request of the CCExtractor repository with the samples uploaded (which currently has 180GB of samples) and report status. This will ensure the pull request being merged is safe and secure. Sample Platform has been developed on a view that not just CCExtractor but even other organizations can use it as a part of their development process.

2 Motivation

Currently, the platform runs a test using KVM instance, so it is capable of running only one test at an instant which is a major drawback. With a lot of cloud offerings available, the process of running tests can be parallelized thus making the system more efficient and powerful.
With the **migration of sample platform from KVM to Google Cloud platform**, long run time issue can also be resolved which happens when there are a lot of commits or pull requests opened.

3 Objectives

During GSOC 2021 I'll be working on migrating sample platform to Google Cloud Platform along with adding additional features. Below is a tentative list of modularized tasks I'll be aiming to achieve:

- **Migrating current platform to google cloud**
  Before migrating the current platform to *n1-standard* instance, the samples should be transferred to the bucket. By mounting the bucket to the current platform using *Google Fuse* and transferring the samples to the bucket, will make the migration easier.
  After the samples have been migrated, the platform itself will be configured on the newly launched *n1-standard* instance.

- **Rewriting the 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 when ever triggered. **launch_gcp_instances()** uses Google Cloud Compute Python API to create instances.

- **Configuring launched VM instances**

  Each and every VM instance should have read-only access to the sample bucket for a test to run. The sample bucket consists of regression test samples, testsuite and runCI file, so it is necessary that each instance gets to access the bucket. Google Fuse can be used to mount a bucket, but first the dependency needs to be fetched. This is where the *startup script* comes into the picture. The *startup script* will take care of the necessary dependencies, mount bucket to the VM instance using Google Fuse and execute the runCI file.

- **Trigger Webhook from Github Actions**

  Running tests for each pull request is an overkill, especially if the edits are related to docs and help screens. Triggering Webhooks from Github Actions would launch test only if the actions were succeeded and since the actions have been configured to be launched only if the edits are related to the code, this would also ensure us the tests will not be triggered if the edits are related to docs and help. Conditional launch can be done by using *webhook-action*, an action for sending webhook to any endpoint.

- **Rewriting Unit tests for mod_ci module**

  After rewriting mod_ci module, unit tests for the module needs to be updated. This will ensure the test coverage of the platform doesn't go down.

- **Make necessary changes in mod_sample**

  The mod_sample deals with the uploading samples, downloading samples and generate media_info for the same. For the module to work after migrating the samples to bucket would be mount the bucket to the platform and give the platform read-write access for the same. This will allow in uploading and downloading samples from the bucket.

Additional features can be added accordingly after discussing with mentors and contributors. The platform will be ready with the above stated objectives by the end of GSOC.

4 Proposed Deliverables

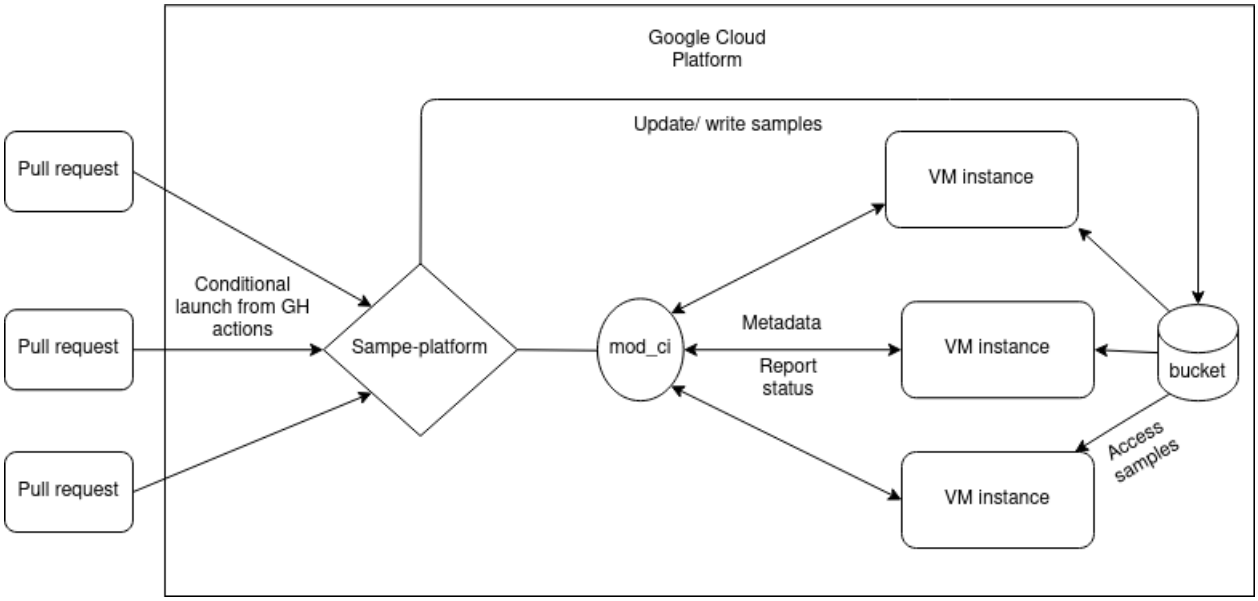
1. Complete migration of platform to Google Cloud
   (a) Migrating current platform to *n1-standard* instance
   (b) Rewriting mod_ci module
   (c) Configuring launched VM instances
2. Trigger Webhook from Github Actions
3. Rewrite Unit tests for mod_ci module
4. Make necessary changes in mod_sample
5. Monthly blogs on development status

5 Implementation and Development

Migration of platform to Google Cloud

The very first step would be to create a bucket to store all the regression test samples, tester and runCI file. Samples can be migrated from the current storage to bucket by mounting the bucket to the current platform using *Google Fuse*. This would help the VM instances to mount the bucket and access samples, tester and runCI file. After the samples have been migrated, the next step would be to launch a *n1-standard* instance and perform a clean installation of platform either using Ansible or by using the default Installation script.

To give a clearer insight below is a diagram depicting how the proposed platform would work:



Rewriting the 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_processor()** method and write a new method to launch google cloud instances. This would require **GoogleAPIClient** module, so the very first step would be to import it and build a compute service object.

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

The next step would be to choose an *image family* and setup the *config*. The *image family* would depend on the TestPlatform.Type. Machine type is chosen to be *n1-standard* and the reason *n1-standard* is chosen is explained in the further sections.
Below is a block of code which creates a VM instance:

```
def create_instance(compute, project, zone, instance_name):
    image_response = compute.images().getFromFamily(
        project='ubuntu-os-cloud', family='ubuntu-minimal-2004-lts').execute()
    source_disk_image = image_response['selfLink']
    machine_type = "zones/us-central1-f/machineTypes/n1-standard-1"
    config = {
        'name' : instance_name,
        'machineType' : machine_type,
        'disks': [{
            'boot' : True,
            'autoDelete' : True,
            'initializeParams' : {
                'sourceImage' : source_disk_image
            }
        }],
        'networkInterfaces': [{
            'network': 'global/networks/default',
            'accessConfigs': [
                {'type': 'ONE_TO_ONE_NAT', 'name': 'External NAT'}
            ]
        }],
    }
```

```
    return compute.instances().insert(project=project, zone=zone, body = config).execute()
```

Setting up the **networkInterfaces** in the *config* is mandatory. In the above block of code, the method which launches the VM is **execute()**.

We have launched the VM but there should be an **acknowledgement** if the VM has successfully launched or failed. This can be dealt by adding a new method **wait_for_operation()**. We wait until the VM initializes and check for its status. Adding the status reported by wait_for_operation() method to the Test Progress will be very much helpful for contributors and users.

```
def wait_for_operation(compute, project, zone, operation):
    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'])
            return result
```

Configuring launched VM instances

We have successfully launched a VM instance, but for a VM to mount the bucket and run tests, additional dependencies are required. To install these dependencies a **startup script** needs to be passed as metadata from mod_ci module.

```
'metadata': {
            'items': [{
                # Startup script is automatically executed by the
                # instance upon startup.
                'key': 'startup-script',
                'value': startup_script
            }]
        }
```

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 startup script.

```
'metadata': {
            'items': [{
                # Startup script is automatically executed by the
                # instance upon startup.Notice the key used for windows
                'key': 'windows-startup-script-ps1',
                'value': startup_script
            }]
        }
```

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
```

XML files, **Google Credentials**, **reportURL** are passed as meta-data to VM instances. Since XML files are retrieved at run-time, we no longer need to store the XML files. reportURL is required since it contains the token and url to which the progress of a test needs to be reported.

```
'metadata': {
            'items': [{
                # Startup script is automatically executed by the
                # instance upon startup.Notice the key used for windows
                'key': 'windows-startup-script-ps1',
                '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()
                }
            ]
        }
```

Resolving concurrency issues

We have the XML files, google credentials, reportURL. The startup script will mount the bucket and execute the runCI file. But if two tests which are running simultaneously triggered via two different pull requests produce same result and try uploading the same result file to the **TempFiles** directory on the platform, will create a concurrency issue and thus leading the corrupt file being uploaded back to the server. The quickest and easiest way to resolve this issue is by changing the temp files directory to *TempFiles/token/* in the method **upload_type_request()**

```
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)
```

Conditional launch from Github Actions

Triggering Webhooks using Github Actions would eliminate the overkill of running tests for each pull request, especially if the pull request is related to docs or help screen. Since the actions have been configured to be triggered only when the pull request changes the main code base, triggering a webhook at the end of actions should work.

Conditional launch can be done by using webhook-action, an action for sending webhook to any endpoint. Below are the changes that are need to be added in GitHub Actions, for a webhook to be launched from a workflow.

```yaml
# build_linux.yml
jobs:
webhook:
  needs: [build_shell, build_autoconf, cmake, cmake_ocr_hardsubx]
  # execute webhook only if each of the linux builds succeed
  # Change build jobs in the above needs field for windows (build_non_ocr_release...)
  runs-on: ubuntu-latest
  steps:
  - name: Webhook
     uses: joelwmale/webhook-action@master
     with:
      url: ${{ secrets.WEBHOOK_URL }}
      headers: '{"repository": "CCExtractor/ccextractor"}'
      body: '{"platform": "linux", "status": "passed", pr_no": ${{github.event.number}}}'
      # Change platform to 'windows' for build_windows.yml
```

In the body field sent using webhook-action , **pull request number** is also passed as a parameter. By default, the workflows *build_linux.yml* and *build_windows.yml* run parallel, so for us to check if both linux and windows builds were successful for a particular pull request, we need to know the pull request number. Passing pull request number as a parameter would also help us deal with multiple pull requests simultaneously.

We have successfully triggered a webhook at the end of Github actions, but there are some changes that are need to done at the endpoint side, especially in **start_ci()** method of mod_ci module, so the successful build status of both linux and windows can be confirmed.

```python
dict = {}
# a global dictionary
def start_ci():
    ...
    if request.headers['Content-type'] == 'application/json':
        payload = request.get_json()
        pullRequest = payload["pr_no"]
        # get pullRequest number
        if pullRequest in dict and dict[pullRequest]!= payload["platform"]:
        # check if pr already exists in the dict
        # pr already in dict implies that previously status has been reported for the same pr
```

```
        # check if the current status reported isn't of same platform previously reported
            dict.pop(pullRequest)
            # remove the pr from the dict since we no longer need it.
            launch_gcp_instances()
        else:
            dict[pullRequest] = payload["platform"] # Insert pr number into dict


    ...
```

Make necessary changes in mod_sample

For the current upload and download methods to work, it is necessary to mount the bucket on the platform and give read-write access to the same. This will allow the users to upload, download and update samples. Mounting the bucket can be done using **Google Fuse**. Serving files from **server_file_download()** will cost the same as downloading files directly from bucket.(Ref.)

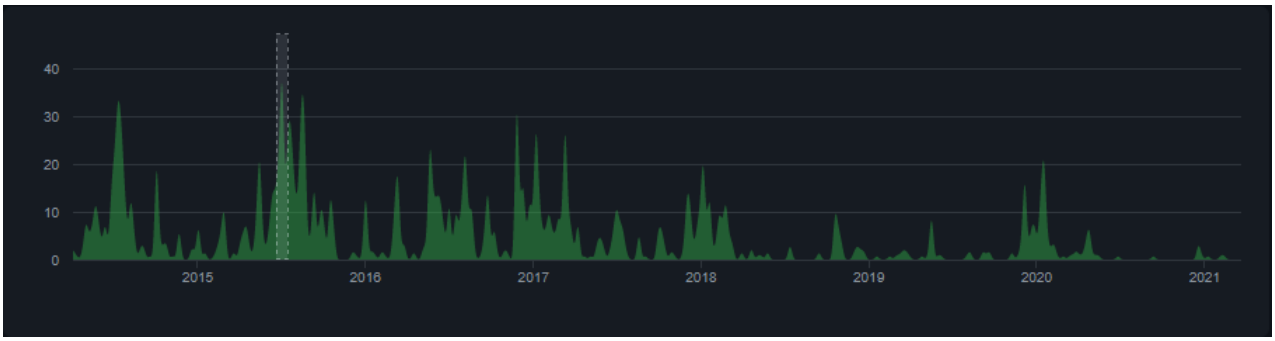
```
mkdir /path/to/mount/point
gcsfuse my-bucket /path/to/mount/point
config.get('/path/to/mount/point', ''), 'samples', sample.sha)
```

Rewrite Unit-tests for mod_ci

With the mod_ci module rewritten it is necessary to update the unit tests so the test coverage of the platform doesn't go down. Mocking newly written methods, will help us deal with GCP instances.

```
    # Unittest for launch_gcp_instances()
     @mock.patch('run.log')
     @mock.patch('mod_ci.controllers.MaintenanceMode') #mock compute service object if needed
     def test_kvm_processor_empty_kvm_name(self, mock_log):
         """
         Test that gcp dosn't launch with empty instance name and config.
         """
         from mod_ci.controllers import kvm_processor #import launch_gcp_instances()
         resp = kvm_processor(mock.ANY, mock.ANY, "", mock.ANY, mock.ANY, mock.ANY)
         #replace above method with launch_gcp_instances()
         self.assertIsNone(resp)
         mock_log.info.assert_called_once()
         mock_log.critical.assert_called_once()
```

6 Monthly Cost Prediction



Above figure shows the number of commits merged to CCExtractor master repository. But for multiple commits merged at once to the master branch, a test is executed only once, which is for the final commit. So the above data isn't so relevant.

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

$$max(no.of\_tests) = max(no.of\_pullrequests\_in\_a\_month) + max(no.of\_commits\_merged\_to\_master)$$

$$max(no.of\_commits\_merged\_to\_master) = max(no.of\_pullrequests\_in\_a\_month)$$

$$max(no.of\_tests) = 2 * max(no.of\_pullrequests\_in\_a\_month)$$

Maximum number of commits merged in a month is equal to the maximum number of pull requests in a month, since in the very extent there are chances that **every pull request will be merged to the master branch**. Maximum number of pull requests in a month were found to be **50** in the month of Jan 2020. So the number of tests executed in a month wont exceed **100**. Since there are two platforms i.e. Linux and Windows, the total number of tests executed will be atmost **200**.

Google Cloud Service	Service Type	Price	Usage	Max Aggregate Price	Min Aggregate Price
Data Storage	Standard storage (Bucket)	$0.02 per GB	190 GB (samples + tester)	190*0.02 = $3.8	$3.8
Network egress	Standard storage	$0.12 per GB	0.5GB (passing metadata such as XML files)	$0.06	$0.01
Network egress (downloading samples)	Standard storage	$0.12 per GB	20GB (say per month contributors download 20 GB of samples)	$2.4	$0.6
Class A Operations (write into bucket)	Standard storage	$0.05 per 10,000 operations	200 writes (upload samples)	$0.001	$0.001
Class B Operations (read from bucket)	Standard storage	$0.004 per 10,000 operations	200*(200+10) reads (200 tests, 100 for linux and 100 for windows, each test has access to around 200 samples)	$0.0168	$0.000004
VM instances	n1-standard-1 (memory 3.75GB , 1 vCPU)	$0.04749975 per hour	200 instances launched at max in a month and each test takes around 30 minutes	(30/60)*200*0.04749975 = $4.749975	(30/60)*20*0.047949975 = $0.47949975
Platform (Server)	n1-standard-1 (memory 3.75GB , 1 vCPU)	$24.27 per month (running 720 hours per month)	1	$24.27	$24.27
Total				**$35.30 per month**	**$29.16 per month**
Price deviations (safe assumption)				(+) $20	(+) $10
Total Cost				**$55.30 per month**	**$39.16 per month**

NOTE:

1. *Network Ingress is free, so uploading samples would cost zero in terms of network.*
2. *Bucket and VM instances should be launched in the same zone; this would eliminate the network egress to fetch samples for VM instances.*
3. *Network egress to download samples is based on the assumption that per month contributors at max. download around 20 GB of samples.*
4. *Serving files from the **serve_file_download()** will cost the same as retrieving objects directly from the bucket, since the bucket is always mounted to the platform using Google Fuse.*
5. *Google Fuse service is free of charge, but the storage, metadata and network it generates to and from cloud storage is charged like any other Cloud Storage. So fetching sample from the bucket mount point will cost same as fetching samples directly from the bucket.*

7 Brief Timeline

- Phase 0 - [Till 17 May]
  [Pre-GSOC period]
- Phase 1 - [17 May - 7 June]
  [Community Bonding]
- Phase 2 - [7 June - 12 July]
  [Coding Period 1]
- Phase 3 - [12 July - 16 July]
  [Phase 1 Evaluations]
- Phase 4 - [16 July - 16 August]
  [Coding Period 2]
- Phase 5 - [16 August - 23 August]
  [Submission and Final Evaluations]
- Phase 6 - [23 August - 30 August]
  [Mentor Evaluation]

8 Detailed Project Timeline

Phase 0 [Pre-GSOC period]

- 1 week (13 April - 21 April)
  I have previously worked on AWS cloud services but not on Google cloud Platform. I will utilize this week to explore the Google Cloud Platform services.(Getting familiar with Compute Engine, Cloud Storage, Google Fuse and exciting services provided by the Google Cloud)
- 1 week (22 April - 2 May)
  I will be having my End Semester exams in this week, so it is hard to contribute during this period. Reference: Institute's Annual Calendar
- 2 weeks (3 May - 17 May)
  These 2 weeks are utilized to get a clearer insight of Google Cloud APIs. There are vast number of libraries to build cloud service objects. Becoming familiar with all those would help me choose an optimal one.

Phase 1 [Community Bonding]

- 2.5 weeks (17 May - 7 June)

  Since I've been contributing to Sample Platform for the past 3 months, it'd be easier for me to get into the community. In the first week of this period, main focus would be to get even more clear insight on Google Cloud services, make a blueprint for the same and resolve my doubts with mentor(s).

  From second week, I will be working on finalizing the blueprint/design for the platform after constantly taking input from the mentor(s). In the same week, a bucket will be initialized and the samples will be migrated. Migrating samples will be done by mounting the bucket to the current platform using *Google Fuse*.

Phase 2 [Coding Period 1]

- 1 week (7 June - 14 June)

  The first week in this period will be dedicated to migrate the platform to a Google Cloud *n1-standard-1* instance and mount the bucket to the new proposed platform. This week will be used to finish micro tasks and also work on the initial phase of rewriting mod_ci module.

- 3 weeks (14 June - 12 July)

  After the migration of server to Google cloud instance, the next step would be to rewrite mod_ci module and configure launched VM instances.

  This period will play a crucial part in the project. Rewriting mod_ci module, configuring both Linux and Windows VM instances, writing **startup script**, passing **metadata**, fetching metadata, modifying the runCI file will be the set of tasks completed in this period.

Progress will be reported to the mentor(s) each and every week and a blog will be maintained so mentor(s) can easily track my progress.

Phase 3 [GSOC Phase 1 Evaluations]

This period will be used to write a report on work done in the Coding Period. All concerning documentation will be done along with the update of my progress on the blog. Below are the deliverables that are promised to be done before the first phase of evaluation.

Deliverables

- Initializing bucket
- Migrating server to n1-standard-1 instance
- Rewriting mod_ci module
- Configuring launched VM instances
- Weekly progress on a blog

Phase 4 [Coding Period 2]

- 1 week (17 July - 24 July)

  This week will be utilized to complete pending work from the Coding period 1, if any and will start working on the next promised deliverables.

- 3 weeks (24 July - 16 August)

  My next semester classes will be starting from 26 July. Since it is just the starting of the semester, there will be very less course work, so I will be able to achieve progress as intended. If not I will work on weekends to complete any due work. Reference: Institute's Annual Calendar

  In this period I will be working on modifying Github actions to launch VMs, rewrite unit tests for mod_ci module, make necessary changes in mod_sample. Documentation will be included for each every modularized task, so it easy for contributors to setup the new project. If time permits I will be working on passive tasks such as generating correctness score for variant files, fixing bugs if any in the new proposed platform.

Progress will be reported to the mentor(s) each and every week and a blog will be maintained so mentor(s) can easily track my progress.

Phase 5 [Final Evaluations]

This period will be used to write a report on work done in the Coding Period 2. All concerning documentation will be done along with the update of my progress on the blog. Below are the deliverables that are promised to be done before the final evaluation.

Deliverables

- Conditional launch from Github Actions
- Rewrite unit tests for mod_ci module
- Necessary changes in mod_sample
- Weekly progress on a blog

By the end of this phase, all the promised deliverables in the proposal will be provided.

9 Additional Information Regarding Timeline

- The timeline provided above is tentative, changes can be done after discussing with the mentor(s).
- Since I have no other commitments during summer, I will be able dedicate 34-36 hours per week. During the final phase of the project, my next semester begins, so I will be working on weekends if weeks days aren't enough to complete proposed deliverables.
- Detailed plan will be mentioned to the mentor(s) at the beginning of every week and my progress will also be reported at https://kvshravan.github.io.

10 Contributions

I started using Sample Platform in January and since then I have been contributing to the platform. Adding new features to the platform such as Multiple Correct Output Feature, got me highly familiar with the platform and below is the list of contributions (most of them are gsoc proposal tasks) I have made to this community.

Sample Platform

- Pull requests merged
  - [FIX] Incorrect number of queued tests
  - [IMPROVEMENT] Remove button to the blocked users functionality
  - [FIX] MediaInfo not showing up on the platform
  - [FIX] Show log file even when build partially succeded
  - [FIX] Diff not showing in browsers.
  - [IMPROVEMENT] Remove duplicates for the file hashes in the SelectField
  - [FIX] Mock Module error in Github Actions
  - [FEATURE] Multiple correct output Feature
  - [FIX] Fixed installation error
- Pull requests closed (Unmerged)
  - [FIX] Fixed the number of queued tests in controllers.py
  - [FIX] Mock module error
  - [FIX] Fixed installation error

CCExtractor

- Pull requests merged
  - [FIX] Documentation fix
  - [FIX] Updated Github actions and reduced steps required to upload artifacts.
- Pull requests closed (Unmerged)
  - [FIX] Updated Github actions and reduced steps required to upload artifacts in Windows.

11 Why Sample Platform?

I have worked on Sample Platform for the past 3 months and since then I have been constantly learning new skills from the community. I was an Open Source Enthusiast but never contributed to any organization until I found CCExtractor.

Every one here in the organization is very welcoming. Special thanks to Willem Van Iseghem for guiding me through every problem I've faced while solving issues. Setting up the platform for new contributors is very tedious, thanks to Shivam and Willem for helping me configure it. There's so much to learn from the community and above all I wish to become a permanent member of this community.

Moreover I'm **NOT** submitting proposals to any other organization other than CCExtrator since I've had such an exciting journey in this community and has constantly motivated me for the best.

12 About Me

The Student

- **Name** : Kandadi Venkata Shravan
- **Email** : kandadi.shravan@iiitg.ac.in
- **Github** : kvshravan
- **Slack** : Venkata Shravan
- **Timezone** : IST (UTC+5:30)
- **Phone** : (+91) 8179422963

The Institute

- **University** : Indian Institute of Information Technology Guwahati
- **Major** : Computer Science and Engineering
- **Year** : Pre-final year