"Python and JS development experience improvements" and "Improve local data ingestion system"

moz://a

Project Name: Treeherder

Mentor Name: Armen Zambrano Gasparnian

Index of Content

Personal Details

Name: Shubhank Saxena

Email Address: saxena.shubhank.19@gmail.com

Riot Nic: shubhank(@shubhank:mozilla.org)

Other contact methods:

- **Hangouts:** saxena.shubhank.19@gmail.com
- **Twitter:** https://twitter.com/19_saxena

Country of residence: India

Timezone: Asia/Kolkata(+0530)

Primary Language: English

Project Details

Treeherder is a reporting dashboard for check-ins to Mozilla projects. It allows users to see the results of automatic builds and their respective tests. Treeherder also provides a rich set of APIs that can be used by other projects interested in this information. Treeherder is the successor to Tinderbox Pushlog and for tracking performance data, it uses Perfherder.

This project would primarily comprise of the following enhancements that are stated below -

1. **Use poetry as the recommended approach than pip**

   Currently, the project uses pip installation method to read all the files from `dev.txt` and `common.txt` files and then we also need to set up a virtual environment for the same. This has some disadvantages.

   - The virtual environment version needs to be set up independent of the machine. It may or may not be in the exact required version in our case.
   - While for this pip installation method, we need to keep hashes for each and every file to maintain the integrity of the files, whose maintenance is a tough task.

   Poetry helps to create a .toml file, which one needs to install to work on the project locally. The file consists of just packages, their sources(required only for non-pip repositories) and versions. Poetry installs all the packages and automatically creates a virtual environment of its own on the basis of the Python version which is mentioned in the .toml.
   This type of installation makes sure that we have uniformity across all the developer platforms, as it minimises the local machine dependencies.

2. **Use tox to replace runchecks.sh (deprecation of runchecks.sh**)

   Currently `runchecks.sh` works fine on Unix based architecture systems, but bash scripts don't work on Windows. So to unify the working of functions of runchecks.sh, we may use Tox.
   Tox is generic virtual environment management and test command-line tool that is used for:

   - checking our package installs correctly with different Python versions and interpreters
   - running our tests in each of the environments, configuring our test tool of choice
   - acting as a frontend to Continuous Integration servers, greatly reducing boilerplate and merging CI and shell-based testing.

   We want all of the commands that are mentioned in `runchecks.sh` and create an alias in our `tox.ini` file.

```
[testenv:runchecks]
envlist = python 3.7
commands =
   "Mention all commands of runchecks.sh"
```

3. **Running all pre-commit checks should pass**

   Currently, the pre-commit-config.yml file contains hook commands that are to be installed and work locally on staged files. This works on running the pre-commit hook which we install when we use `pre-commit install`. Now in order to run the intended commands, we can import `pre-commit run --all` script as an alias in out tox.ini file. Then, we need to add that particular alias file to `travis.yml`.

This will make sure we run all the commands intended in our `tox.ini` file.

#Outcome of first three tasks -

The first three tasks that I have mentioned are solving towards creating a unified solution to achieve one simple goal, that is "**Make sure that our installation and testing systems are uniform across all the OS/Platforms and should not require any special setup**". Also, the tests that we run on Travis should be easily replicable even on the local machines This is the process of workflow that I aim to achieve

- Unify all requirements installation of all the required packages using **poetry.**
- Create various testing environments under the `tox.ini`

```
[tox]
isolated_build = true
envlist = py37,poetry
[testenv]
deps = poetry
commands =
 poetry install -v
[testenv:python]
 poetry install -v
 pip install codecov
 !integration: poetry run pytest cov=
Deps = poetry
[testenv:pre-commit]
deps = pre-commit
commands= pre-commit run --all-files --show-diff-on-failure
```

(Not actual code snippet, just a representation showing different aliases/testing module scripts that can be used)

- Integrate pre-commit run --all-files which can replace all sort of linting tests which are currently mentioned in `runchecks.sh` .
- Replacing various manual commands in `.travis.yml` with our defined `tox` testing aliases. This will help us to run all the tests on a local machine in a similar way of Travis.

4. **Add check for valid Markdown syntax**

Currently, there are no checks on the markdown files that we make changes to. This may lead to serving broken markdown syntax files. To reduce this, we may add a pre-commit hook for the same. We can use Markdownlint for achieving this. Markdownlint has various pre-defined alias rules, and we can implement the ones that are desired to us. Once we have this installed, we can import it as a local repo of the project in our `pre-commit-config.yml` and run a custom predefined alias list (`.markdownlint.json`)accordingly. Suggested aliases to be implemented -

- MD004 ul-style - Unordered list style
- MD005 list-indent - Inconsistent indentation for list items at the same level
- MD009 no-trailing-spaces - Trailing spaces
- MD012 no-multiple-blanks - Multiple consecutive blank lines
- MD014 commands-show-output - Dollar signs used before commands without showing output
- MD028 no-blanks-blockquote - Blank line inside blockquote

Other aliases can be easily implemented with whatever is required.

5. Upgrade the Python version to 3.8

As this project is quite diverse, we need to constantly be upgraded with the latest versions of python. This particular task requires us to migrate to `Python 3.8`, and also, the corresponding dependency files' versions need to be set to work in accordance with Python 3.8.

6. Add prototype2 and treeherder-taskcluster-staging to whatsdeployed

Currently, whatsdeployed only shows Prototype, Stage and Production state and differences. We want to add prototype2 and treeherder-taskcluster-staging too to it. For this, we have to do the following tasks-

- Link the whatsdeployed backend to required states' SHA values of operating environments so that it can access the current state of the state environment we intend to link.
- Insert the received from the SHA values into the database and creating a new table and linking the key to existing DB configuration
- Make changes to UI to make changes to the **main display table** and also to **compare various forks** in the **comparator table**.

The backend of the whatsdeployed is written in `Flask==1.1.1` and the frontend is written in Javascript. The database used in default is `SQLAlchemy`

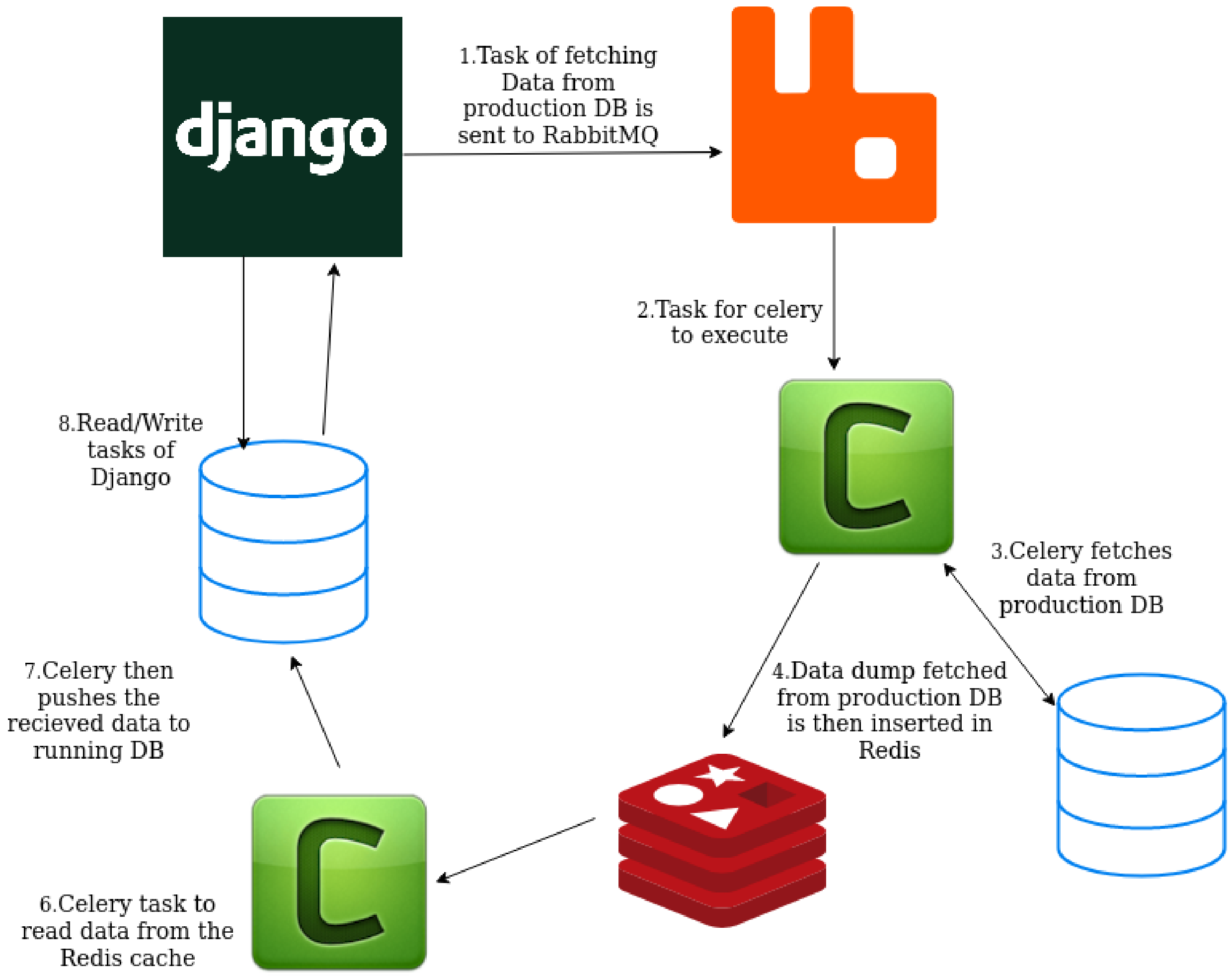
7. Initialize Docker with some pushes

Currently, whenever we start a local instance of Treeherder using Docker, it initialises with a blank MySQL DB. So, we need some initial pushes (preferably 10) to be fetched.
This can be achieved by including the pulse ingestion script `./manage.py ingest pushes -p <required-components>` in the `initialize.sh`, which is always run whenever we fire `docker-compose up`, to initialise push data to the local database.

8. Fetching data from production SQL database and initialising it inside Docker

The aim of this task is that we create a periodic task that connects to the production database and extracts a number of pushes and associated data, dump it and store it somewhere, We want then the initialised database to be fed in this dumped data and then start the instance
For this, we will create a periodic task using Celery for Docker and will create a fetch task using **Celery** and **RabbitMQ**. This task will be executed as soon as the Docker container is going to be initialised. This will fetch the data from the production database, create a dump, store it in **Redis** and then another **Celery** task can be used to ingest it into our local database. This task will be executed in various steps

- Create a user associated to test and write a script that will read data and dump it locally
- then create a script that will take the dumped data and ingest it into the local MySql container
- Celery tasks shall be used for both the processes.



django
1.Task of fetching Data from production DB is sent to RabbitMQ
2.Task for celery to execute
3.Celery fetches data from production DB
4.Data dump fetched from production DB is then inserted in Redis
6.Celery task to read data from the Redis cache
7.Celery then pushes the recieved data to running DB
8.Read/Write tasks of Django

(Basic workflow of how we can fetch data from production database to local one)

9. **Remove deprecation warnings for manual ingestion**

Currently, ingestion script `treeherder/etl/management/commads/ingest.py` has its connection setup using asyncio for handling all the asynchronous calling.

Currently the session creation outside of asyncio task (module-level code or something executed in low-level asyncio callbacks like `Protocol.connection_made()`) Currently, we use `aiohttp==3.6.2`, which does not have this exception handling. Moreover, in order to preserve the modularity of code (12-factor app), we can use (already setup) `Celery` to perform asynchronous handling efficiently.

```
@retryable_task(name='log-parser', max_retries=10)
def parse_logs(job_id, job_log_ids, priority):
   newrelic.agent.add_custom_parameter("job_id", str(job_id))

   job = Job.objects.get(id=job_id)
   job_logs = JobLog.objects.filter(id__in=job_log_ids,
                                    job=job)

   if len(job_log_ids) != len(job_logs):
        logger.warning("Failed to load all expected job ids: %s", ",
".join(job_log_ids))

   parser_tasks = {
       "errorsummary_json": store_failure_lines,
       "buildbot_text": parse_unstructured_log,
       "builds-4h": parse_unstructured_log
   }
```

(example scheduler code snippet of async task implemented by celery)

10. Merge ingest_push into ingest management command

Currently, we have two commands governing ingestion of pushes `treeherder/etl/management/commads/ingest.py` is used to ingest PR, Git-Push(es), Push from Celery, whereas `treeherder/etl/management/commads/ingest_pushes.py` is used to ingest multiple pushes from Celery. So it's better to combine every sort of functionality into `ingest.py` so that we can have just one command for handling ingestions of all types.

This can be done by adding Celery pulse handling commands in class Command(BaseCommand): and adding new asyncio commands from ingest_pushes to ingest.py to handle

11. Calling initialize_data.sh w/o a MySql container fails

`Initialize_data.sh` is used for initialising some data in our local MySQL container. But this particular script ideally should not work when we don't use `docker-compose up` (as it creates a SQL server) We can throw an error checking the binding of the port assigned to MySQL.

```
if (lsof -Pi :{mysql_port} -sTCP:LISTEN -t >/dev/null)
   run script
else
   don't
```

12. Separate Node installation paths for Docker and host

When we run yarn install outside of Docker a series of packages get installed under node_modules. These packages are OS dependent. Meaning that depending if you're on Windows or Mac there might be some differences in the overlapping, thus causing issues.

For this, we can separate installation packages for Docker and let default hosts unchanged. This can be done by making changes in dockerfile. We can shift node_modules up by one directory and set default in dockerfile to look for them in a location defined.

```
RUN mkdir /opt/node_app && chown node:node /opt/node_app
WORKDIR /opt/node_app
COPY package.json package-lock.json* ./
RUN yarn install --no-optional && yarn cache clean --force
ENV PATH /opt/node_app/node_modules/.bin:$PATH
```

Schedule of Deliverables

May 4th - June 1st Community Bonding Period	Discuss with the mentor the exact procedure and workflow. I have mentioned the path and way for the solutions in the proposal but they are flexible and can change as per the common view with the mentor. Along with this, I will learn the necessary skills which will help me to make the solutions easier and accurate.
June 1st - June 7th	Addition of prototype2 and treeherder-taskcluster-staging to whatsdeployed
June 8th - June 13th	Replace runchecks.sh with Tox.ini and Addition of valid markdown checker
June 15th - June 25th	configuring pre-commit --all check on Travis CI and Upgrade the python version to 3.8
June 26th - July 1st(Phase 1 evaluation June 29th - July 3rd)	Remove deprecation warnings for manual ingestion
July 3rd - July 31st (Phase 2 evaluation July 27th -31st)	Fetching data from production SQL database and initialising it inside Docker.
August 1st - August 7th	Separate Node installation paths for Docker and host
August 10th - August 20th	Initialization of Docker with pushes and fixing initialize.sh
August 20th - August 30th (August 24 - 31, 2020 Final Evaluation)	Complete Poetry installation method to completely replace pip installation

Apart from this, I will be maintaining a 3-part blog, writing about my work and the entire workflow and evaluations.

Availability Schedule

During the entire GSoC period, I am available to work for **35-40 hours per week**. I don't have any pre-planned vacations or plans during the break and will be available full time.
Due to **COVID19 shutdown** in my country, our college schedule is likely to be shifted ahead by a **minimum of 2 weeks**, thus will be consuming my summer break's first few days. But that **won't be an issue** to work schedule. I will make sure that compensation of that time is done during the following weekends of the weeks indulged in my college timings (if required).
Also, I have **kept a buffer** of few days between the tasks in my timeline, which will give me time to complete pending/delayed tasks (if any)

Future Development

For me, the open-source contribution is not just restricted to Google Summer of Code. I would love to keep working on the developments and enhancements. Here are some of the ideas that I would love to implement after the entire GSoC timeline -

- **GitHub integration of Bugzilla**

  Currently, the bugs and tasks are mentioned on Bugzilla. This is a good method for the organisation to maintain its database and

keep a track of everything. But the issue with Bugzilla is that issues or bugs are disjoint from the exact codebase(bugs are raised here, code is on Github)
Also, GitHub these days is a better (and user-friendly way) to mention issues and bugs. It helps in exact code highlight, better workflow, review on the changes, etc. This way, it will be convenient for every new contributor and existing users. To bridge down this difference, I intend to create a script which will act as an event listener and will push the issues on Github if they are raised on Bugzilla and vice versa. In this way, we can have the best of both the worlds.
This task will first be tried only on bugs of Treeherder, and then can be implemented across the entire Bugzilla.

Approach -

- Every bug on Bugzilla has a meta which points to the project that it belongs to (example, treeherder, firefox, etc). This meta can be then tagged to the respective repositories of the projects on Github.
- An event listener script can be deployed along the backend of Bugzilla, which will push the new bug raised on Bugzilla to the issues page of the tagged repo by using GitHub API.
- For issues being raised on Github, we can use custom Webhooks (which is there inbuilt in Github for every repository), which can be used to post content of an issue raised to a defined URL (post URL to Bugzilla backend in our case).
- This system will create perfect sync between both Bugzilla and Github repositories.

- **Test Improvements**
  Currently there are a lot of code segments for the backend of Treeherder, whose coverage is quite less. Our aim is to keep them above 60 % (min threshold). Will work with unit-testing library pytest and will improve the code-coverage

Contributions to Treeherder

- 5860 - Added Python code coverage on codecov
- 5982 - Initialised some default values for local development
- 5994 - Added minimum UI coverage to 40% to merge a PR
- 6058 - Added pre-commit checks for isort and flake8
- 6067 - Used poetry to generate and serve docs
- 6108 - Add prettier to pre-commit hooks
- 6113 - Docs:Polished pre-commit section
- 6122 - Combine pulse listener for tasks and pushes
- 6182 - Fixed isort behavior in context to pre-commit hook
- 6194 - Creation of tox.ini file to replace some Travis commands (and can be used locally)

Academic Experience

I am a pre-final year engineering undergraduate(Bachelor of Technology in Electronics and Communication Engineering) at Thapar Institute of Engineering and Technology, Patiala.

I wouldn't say electronics engineering was my passion when I chose to pursue it. I have always loved to tinker with old electronic items in my house but I never thought I would pursue it as a career. I have always loved STEM, especially in computers and mathematics. Though I program a lot, I try my level best to stay focussed on my coursework and I hope to pursue a career that has an apt combination of the two.

A link to my complete academic resume is here.

Personal Projects

- **Amaurotic:** A braille equivalent of an electronic book reader(Kindle), for visually impaired people. (RasPi | Python | Electro-Mechanic)
- **Blindspot:** A wearable device which is the equivalent replacement of mobile phones for visually impaired people. Key features- payments, text reading, navigation.(RasPi | Python | Electro-Mechanic)
- **Felix:** A Google Lens like wearable AR device which assists Autistic kids with their therapy and constant learning.(RasPi | Python | Electro-Mechanic)

Open Source Projects

- **Movie Spoiler Blocker**(JS | CSS): Link
- **Freecodecamp Python CLI Tool** (Bash | Python): I am helping **Freecodecamp Foundation** to create a CLI tool for teaching Python. Link

Work/Internship Experience

- Research Intern under **Department of Cognitive Sciences, Indian Institute of Information and Technology, Allahabad**
    - During my entire summer internship, I researched and developed a device used to imitate human nose and to perform the preliminary non-physical examination on the quality of fruits on the basis of their natural enzyme secretion (ripe, under-ripped and over-ripped)

Why Me

I think this project is apt for me. It involves a lot of constant learning about new stacks and technology. I've loved contributing to (Treeherder) in the last few months, which has helped me to be familiar with the backend parts of the codebase. I am consistent with my efforts and never leave any task in between, no matter how hard or new it is for me. I make sure that I keep researching the issue at hand and then work to make it happen. I think this skill of mine proves to be very helpful in order to achieve feature addition and bug solving for this particular project. This project will help me to equip essential real-world software skills as well as soft skills which I believe, will help me a lot in the longer run.

Why Mozilla?

Mozilla is one of the most prominent advocates of open-source development. I really appreciate Mozilla’s commitment to the community and the work it has been doing to empower the people worldwide to shape the modern Internet - an Internet that is open and accessible to all. I truly align with Mozilla's belief of promoting acts of human collaboration across an open platform that contributes to individual growth and the collective future. I came to know about the concept of open-source software through Mozilla. So, for me, there is a special connection with Mozilla. Also, after contributing to Treeherder, I genuinely liked the coding standards, especially the backend and testing part. I want to work in such an environment. I feel that it will be a good learning experience for me.