RustyCoder: A competitive programming platform based on microservice architecture.

*Varun Amrutiya (2017H1120244P), Pranjal Priyadarshi (2017H1120242P), Siddhant Jhamb (2017H1120243P)*

## Abstract

The main aim of this project is to understand the software architecture involved in microservices based software development. To better understand the design principles we are developing an online competitive programming platform. The goal here is to understand the software requirements and to design a microservices based architecture to build the project.

## Objectives

The main objectives of this project are mentioned below,

* To build an online programming platform where users can write programs and run them them using a web browser.
* The platform should have support of multiple programming languages.
* Platform should have a large set of questions from which user can choose freely.
* Platform should be able to process a large number of program requests and should be able to compute results in reasonable time.
* Platform should be built using microservices based architecture, which will allow it to scale linearly with user traffic.

## Technologies Used

* Flask web server (Python)
* RabbitMQ messaging Queue
* Celery distributed task management
* Bulma CSS framework
* Ace code editor
* Database ( MongoDB )
* JMeter (stress testing)
* Flower (Celery Monitoring)

## Work distribution

The main modules of this project can

|  |  |
| --- | --- |
| **Module Name** | **Programmed by** |
| User Interface | Varun |
| Authentication Service | Varun |
| Broker Service | Siddhant |
| Datastore Service | Pranjal |
| Language Support | Pranjal |

Detailed work distribution can be given as,

|  |  |  |  |
| --- | --- | --- | --- |
| **Module Name** | **Sub-Module** | **Status** | **Programmed By** |
| User Interface | Login Page | Completed | Varun |
| Register Page | Completed |
| Dashboard Page | Completed |
| Theming (Bulma CSS) | Completed |
| Coupling with broker service | Completed |
| Coupling with Auth service | Completed |
| Admin Page | Completed |
| Problems page | Completed |
| Authentication Service | Register User | Completed |
| Generating Tokens | Completed |
| Validating Tokens | Completed | Siddhant |
| Broker Service | Run Job | Completed |
| Submit Job | Completed |
| Run Job Status | Completed | Siddhant |
| Submit Job Status | Completed |
| Integrating with Datastore service | Completed |
| Authentication Integration | Completed |
| Multi-Language Support Integration | Completed |
| Celery Integration | Completed |
| Flower Monitor Integration & API stress testing | Completed |
| Datastore Service | User history management | Completed | Pranjal |
| Api endpoint | Completed |
| Code management | Completed |
| Database Integration | Completed |
| Authentication Integration | Completed |
| Language Support | Python Support | Completed |
| C Lang. Support | Completed |
| C++ Lang. Support | Completed |
| Java Support | Completed |
| Perl Support | Completed |
| Ruby Support | Completed |

## 

## API Description

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Module: UI Service** | | | | **By Varun** | | | |
| **Method** | **Path** | **Description** | | | | | |
| GET | / | Returns home page. | | | | | |
| GET | /register | Returns the register page. | | | | | |
| GET | /login | Returns the login page. | | | | | |
| GET | /dashboard/<string:pid> | Returns the coding dashboard for the particular problem. (authentication required) | | | | | |
| GET | /problems | Returns a page consisting of all the problems user can choose to solve. (authentication required) | | | | | |
| GET | /admin | Returns the administrator homepage. (authentication required) | | | | | |
| GET | /adminproblems | Returns the list of problems available for administrator. (authentication required) | | | | | |
| POST | /api/login | API for logging in to website, further contact the auth service | | | | | |
| POST | /api/register | API for registering user, further contacts the auth service | | | | | |
| POST | /api/addProblem | API for adding problem to the database via datastore service (only for admin). (authentication required) | | | | | |
| POST | /api/queryUserStatus | API for querying all the user submission history for a particular problem. (authentication required) | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Module: Auth Service** | | | | **By Varun** | | | |
| **Method** | **Path** | **Description** | | | | | |
| POST | /api/register | This API endpoint is used to register the user in the auth database. | | | | | |
| POST | /api/login | This API endpoint checks the user credentials and returns the status along with token. Which would be further required for authentication purpose. | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Module: DataStore Service** | | | | **By Pranjal** | | | |
| **Method** | **Path** | **Description** | | | | | |
| GET | /fetchProblems | This API endpoint is used fetch all the problems from the mongodb based datastore and display on the UI. | | | | | |
| POST | /addProblem | This API endpoint is used to load problems into the database.This endpoint is mainly used by the Admin to load problems into the datastore through the admin dashboard | | | | | |
| POST | /problemsSearchedByUsers | This API endpoint is used by the user to search for problems when the user wants to find a particular problem. | | | | | |
| POST | /queryUserStatus | This API endpoint is used to retrieve all the previous submissions of the user for a particular problem. | | | | | |
| GET | /fetchUserStatus | Returns status of all the users | | | | | |
| POST | /saveStatus | Saves the status of a particular user according to the taskid of the problem | | | | | |
| GET | /deleteallProblems | Deletes all the problems present in the user database.This API is for the admin for testing purposes. | | | | | |
| GET | /deleteUserHistory | Deletes all the user history present in the user database.This API is for the admin for testing purposes. | | | | | |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Module: Broker Service** | | | | **By Siddhant** | | | |
| **Method** | **Path** | **Description** | | | | | |
| POST | /run  Method Type : POST  Sample Payload :  {  "code": "YSA9IGlucHV0KCkNCmIgPSBpbnB1dCgpDQpwcmludChhICsgYik=",  "lang": "Python",  "stdin": "Mg0KMw=="  } | This is the end point responsible for triggering the ‘run’ functionality, wherein a user can chose the programming language, and provide the source code along with some expected input. As a result, a celery task is formed, and the task identifier of the corresponding task is returned. | | | | | |
| GET | /runStatus/<task\_id> | This endpoints lets the user to query the status of the already constructed celery task. The task identifier of the task to query is provided in the URL, and the parameters returned include actual output and task status.  Sample Payload returned :  {  "message": "Test Case passed",  "output": "5\n",  "status": "SUCCESS"  } | | | | | |
| POST | /submit  Sample Payload  {  "code": "YSA9IGlucHV0KCkNCmIgPSBpbnB1dCgpDQpwcmludChhICsgYik=",  "lang": ".py",  "problemId": "1",  "testcases": {  "testcase1": ["Mg0KMw==","NQ=="],  "testcase2": ["Mg0KMw==","NA=="]  }  } | This API endpoint constructs and returns the task identifier of the task created to handle submit request. | | | | | |
| GET | /submitStatus/<task\_id> | This endpoints lets the user to query the status of the already constructed celery task. The task identifier of the task to query is provided in the URL, and the parameters returned include actual output and task status.  Sample Payload returned :  {  "jobId": "147de639-809c-433d-b2ec-da58c0f016cb",  "message": "Completed",  "overallStatus": "Fail",  "problemId": "1",  "result": [  "Passed",  "Failed"  ],  "status": "SUCCESS",  "testCaseStatus": "1/2",  "timestamp": "2018-04-27T03:31:00.673469",  "userId": "1",  "username": "abc"  } | | | | | |

## Unit Tests

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Module: WebApp Service** | | | **Tested by : Varun** | | | |
| **Name** | **Status** | **Description** | | | | |
| HTML rendering | Pass | Test all web pages are rendered based on their templates via Flask’s Jinja templating engine. | | | | |
| Static Library handling | Pass | Check all required css and js libraries are loaded correctly on their respective web pages. | | | | |
| Authentication redirection | Pass | All unauthenticated users would be redirected to the login page and all existing authentication tokens would be flushed from browser storage. | | | | |
| Logout | Pass | Deletes all the tokens from the browser session storage and redirects to the login page. | | | | |
| Problems page | Pass | Check whether all the problems are loaded correctly from the datastore service and rendered on the webpage | | | | |
| Dashboard Page | Pass | Check whether the code editor is loaded correctly and the theming would auto change with the changing programming languages. | | | | |
| Admin Page | Pass | Check the functionalities where the admin can input new problems along with their test cases and would be saved in the database via the datastore service. | | | | |
| Run Job | Pass | Take the code from the code editor window along with the standard input provided by the user and submits it to broker service and poll the status of the job every 3 sec for 10 times, and display the result on the web page when job status of success is received. | | | | |
| Submit Job | Pass | Takes the code and required test cases and is submitted to the broker service and then poll the status of the job every 3 sec for 10 times and when the job status is received as success, shows the test case statistics on the webpage. | | | | |
| User Submissions | Pass | Retrieves the last statistics for that particular problem and shows them on the page. | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Module: Auth Service** | | | **Tested by : Varun** | | | |
| **Name** | **Status** | **Description** | | | | |
| Register User | Pass | Stores the user information in the database only if none of the parameters are missing and user is not already registered. | | | | |
| Login | Pass | Checks if the password hash is matching and generates token for that user, which can be used further for authentication. | | | | |
| Database schema | Pass | Checked if the database schema is migrated properly to the database defined. (using ORM libraries) | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Module: Broker Service** | | | **Tested by : Siddhant** | | | |
| **Name** | **Status** | **Description** | | | | |
| Run Job | Pass | The user should be able to provide the source code and required input to the /run API endpoint and must get the task identifier of the celery task created underneath, so that he can query its status later. | | | | |
| Submit Job | Pass | The user should be able to provide the source code and the required set of inputs along with their corresponding expected outputs in the form of multiple test cases to the /submit API endpoint and must get the task identifier of the celery task created underneath, so that he can query its status later. | | | | |
| Run Job Status | Pass | The user should be able to successfully query the status of the already created ‘run’ task by providing the task identifier. | | | | |
| Submit Job Status | Pass | The user should be able to successfully query the status of the already created ‘submit’ task by providing the task identifier. | | | | |
| Integration with Datastore | Pass | While processing the ‘/submitStatus/<task\_id>’ query and having populated the status of the submit job including parameters like the number of test cases passed, the overall status of the job, the user Id and user Name of the requestor, etc., the user must be able to save this in the database by calling the ‘/saveStatus’ endpoint of the database server. So, this would save a mapping of the user Id, problem Id and test case result in the database, for it to be rendered on the UI. | | | | |
| Authenticated User | Pass | Each exposed API endpoint must be able to verify the authenticity of the user by validating the token. If the token gets validated, the user must be allowed in. | | | | |
| Unauthenticated User | Pass | If an unauthenticated user tries to call the API, a 401 error must be returned. | | | | |
| Multi Language Support Integration | Pass | For both ‘Run’ and ‘Submit’ cases, the user must be able to provide source code in any of the six supported programming languages, including C, C++, Ruby, Perl, Java and Python. Along with the source code in the input payload, the user must provided the language name as well. | | | | |
| Celery Integration | Pass | Celery, the task queue, must be setup along with RabbitMQ as the message broker on the broker server. The RabbitMQ management console must be accessible at localhost:15672, and producer must be able to submit tasks and query their status. | | | | |
| Flower Integration | Pass | Flower, the dashboard which lets us access the list of workers, tasks, queues and their corresponding status must be setup on the broker server at port 5555. | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Module: Datastore Service and Language Support** | | | **Tested by : Pranjal** | | | |
| **Name** | **Status** | **Description** | | | | |
| Fetch Problems | Pass | The purpose of this endpoint is mainly to get all the problems stored in the datastore.The admin should get all the problems that are stored in the datastore. | | | | |
| Add Problems | Pass | The purpose of this endpoint is to help the admin to add any new problem which he wants to add to the datastore. | | | | |
| Problems Searched By Users | Pass | Once the intended user of the system clicks a particular problem the problem id of the system is sent to this endpoint which is then searched for in the database. If that particular problem is present in the database then the entire problem is fetched and sent to the user. | | | | |
| Query User Status | Pass | After running the code submitted by the user is executed in the backend, the system generates the status of the problem whether it has successfully completed compilation/execution or not. The entire json of the status of the problem submitted by user is stored in the datastore. This is the API endpoint from which we can get all the relevant information regarding the problems. | | | | |
| Fetch User Status | Pass | For each user all the submission made by the user can be queried using this endpoint. | | | | |
| Save Status | Pass | This API endpoint is used to save the entire status json into the datastore for all the problems submitted by the user. | | | | |
| Delete all Problems | Pass | This API endpoint is provided for the Admin to interact with the system smoothly. Once this API endpoint is invoked the entire document containing problems is removed from the datastore | | | | |
| Delete User History | Pass | This API endpoint is provided for the Admin to interact with the system smoothly. Once this API endpoint is invoked the entire document containing user history is removed from the datastore | | | | |
| run\_code\_python | Pass | This function forms the heart of the entire system. This function takes in the source code and stores the file appended with the task id of the system, so that the file is unique for each user. It then uses the Popen facility of the python language to spawn a subprocess to run the Python code which is given as input to the function.On successful execution of the function the output/ error which occured while executing the code is returned. | | | | |
| run\_code\_c | Pass | This function forms the heart of the entire system. This function takes in the source code and stores the file appended with the task id of the system, so that the file is unique for each user. It then uses the Popen facility of the python language to spawn a subprocess to run the C code which is given as input to the function.On successful execution of the function the output/ error which occured while executing the code is returned. | | | | |
| run\_code\_ruby | Pass | This function forms the heart of the entire system. This function takes in the source code and stores the file appended with the task id of the system, so that the file is unique for each user. It then uses the Popen facility of the python language to spawn a subprocess to run the Ruby code which is given as input to the function.On successful execution of the function the output/ error which occured while executing the code is returned. | | | | |
| run\_code\_java | Pass | This function forms the heart of the entire system. This function takes in the source code and stores the file appended with the task id of the system, so that the file is unique for each user. It then uses the Popen facility of the python language to spawn a subprocess to run the Java code which is given as input to the function.On successful execution of the function the output/ error which occured while executing the code is returned. | | | | |
| run\_code\_perl | Pass | This function forms the heart of the entire system. This function takes in the source code and stores the file appended with the task id of the system, so that the file is unique for each user. It then uses the Popen facility of the python language to spawn a subprocess to run the perl code which is given as input to the function.On successful execution of the function the output/ error which occured while executing the code is returned. | | | | |
| run\_code\_cpp | Pass | This function forms the heart of the entire system. This function takes in the source code and stores the file appended with the task id of the system, so that the file is unique for each user. It then uses the Popen facility of the python language to spawn a subprocess to run the C++ code which is given as input to the function.On successful execution of the function the output/ error which occured while executing the code is returned. | | | | |
| find\_Error | Pass | This function is used to detect any kind of runtime error which occurs while the code is being executed. Example, when a divide by zero error occurs while execution of anycode, the OS traps the code by returning an error code. This function catches the error code and returns the message pertaining to that particular error code. | | | | |
| create\_file | Pass | This function creates a source code file and appends the file with required extension depending on the language which is given as input by the user.If the code is of python type then the file is appended is .py extension. | | | | |
| find\_extention | Pass | This function finds the correct extension for the particular type of language given as input by the user. | | | | |

## Performance Test

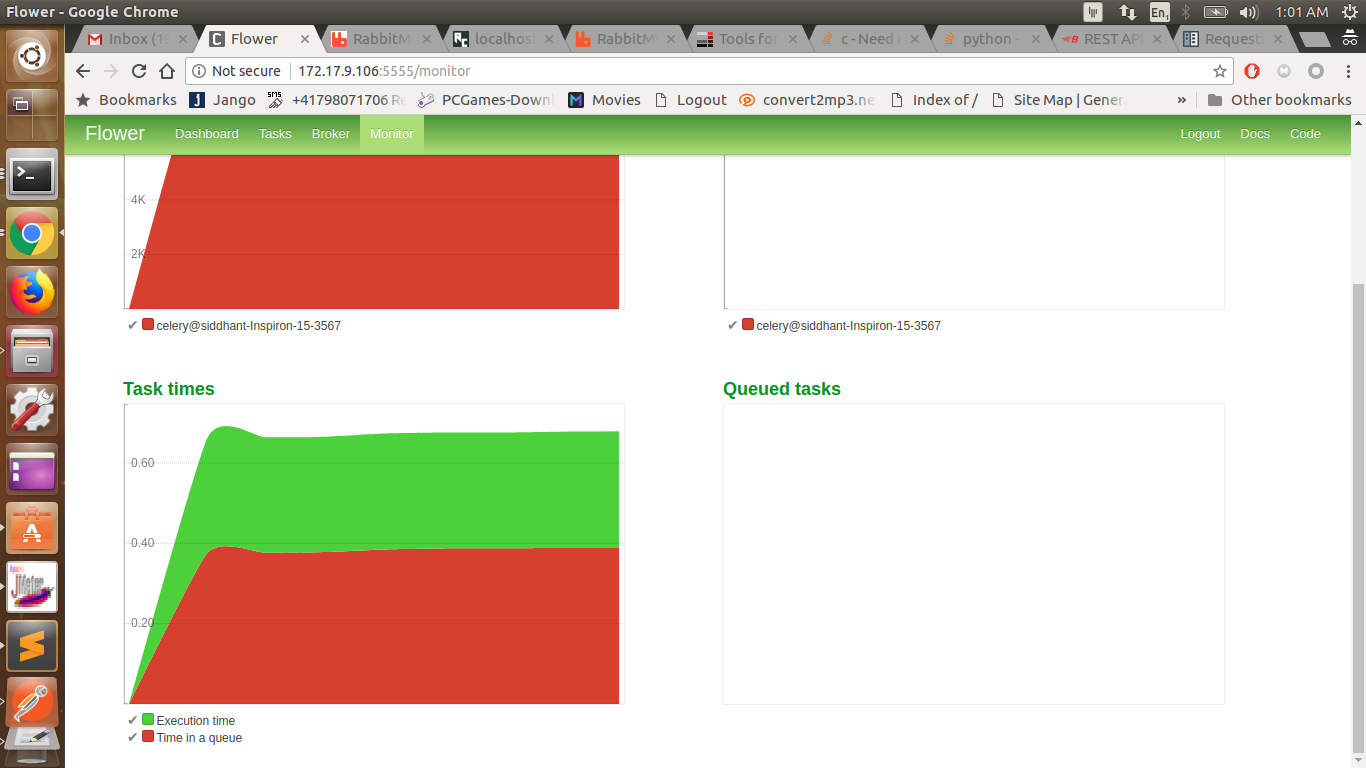
To test the performance of our workers who were mainly tasked to compile the code and execute them we used JMeter a widely used API stress testing tools.

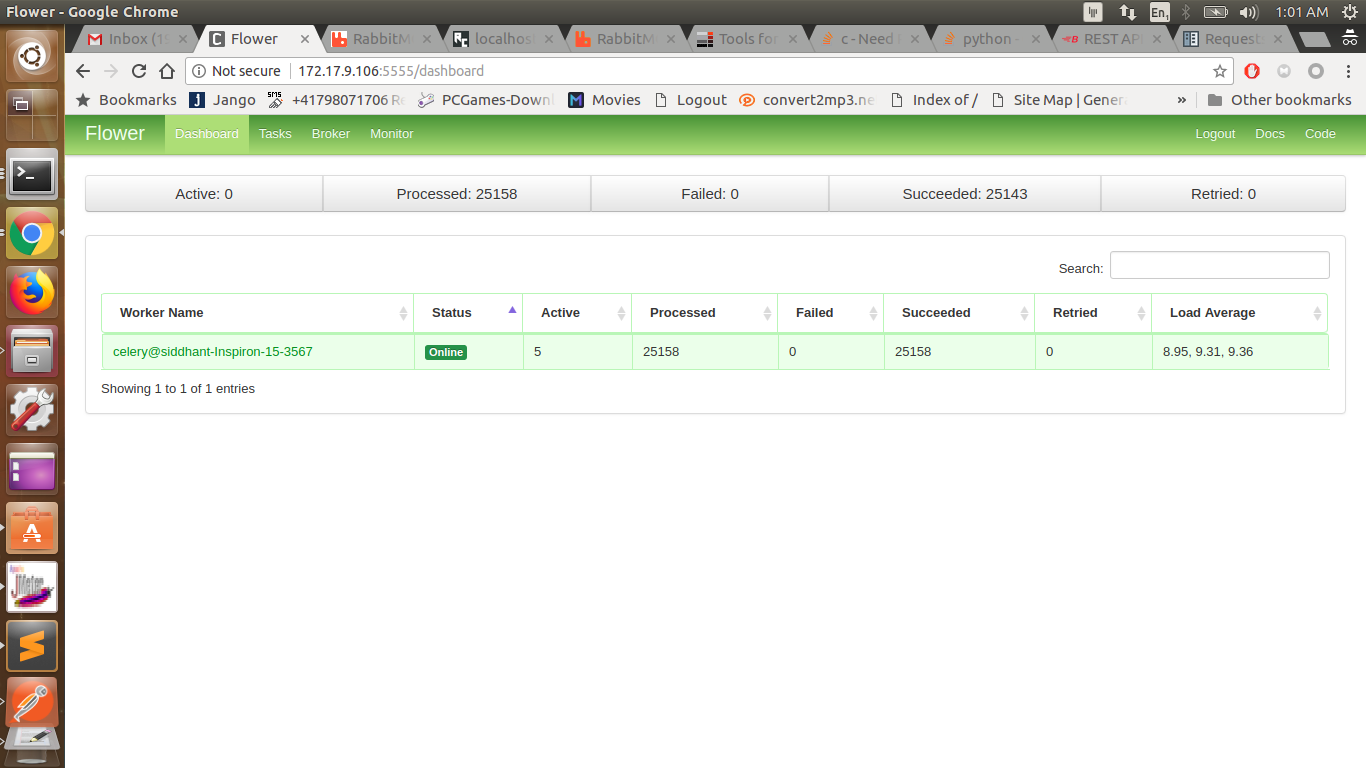
Our workers were pre-configured to autoscale themselves from 3 replica to maximum to 10 replicas based upon the messaging queue size. To stress test the workers to their max performance we configured the JMeter to hit our service broker API with 500 concurrent connections each with 500 run jobs in C programming language.

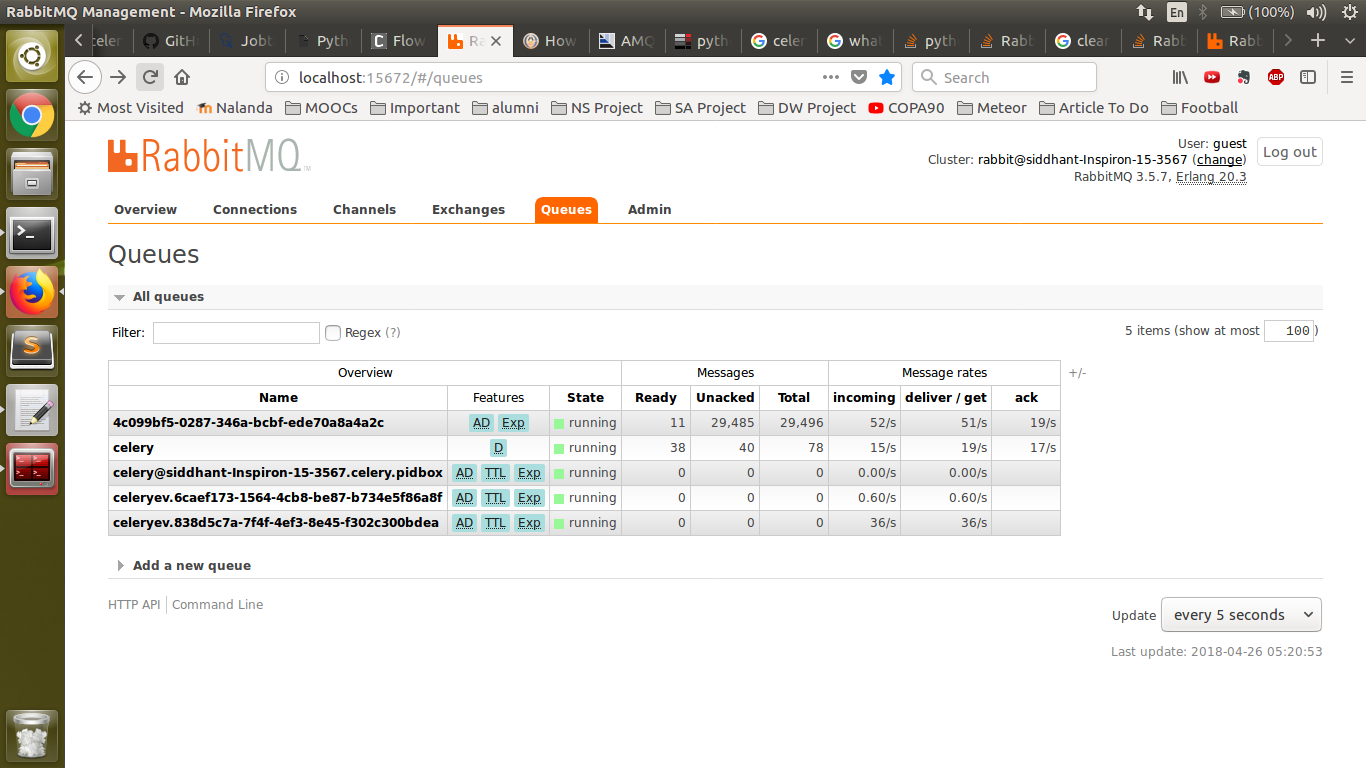
To monitor performance of the workers we installed ‘Flower’ a widely used Celery task monitoring tool along with RabbitMQ’s default dashboard to see the queue statistics.

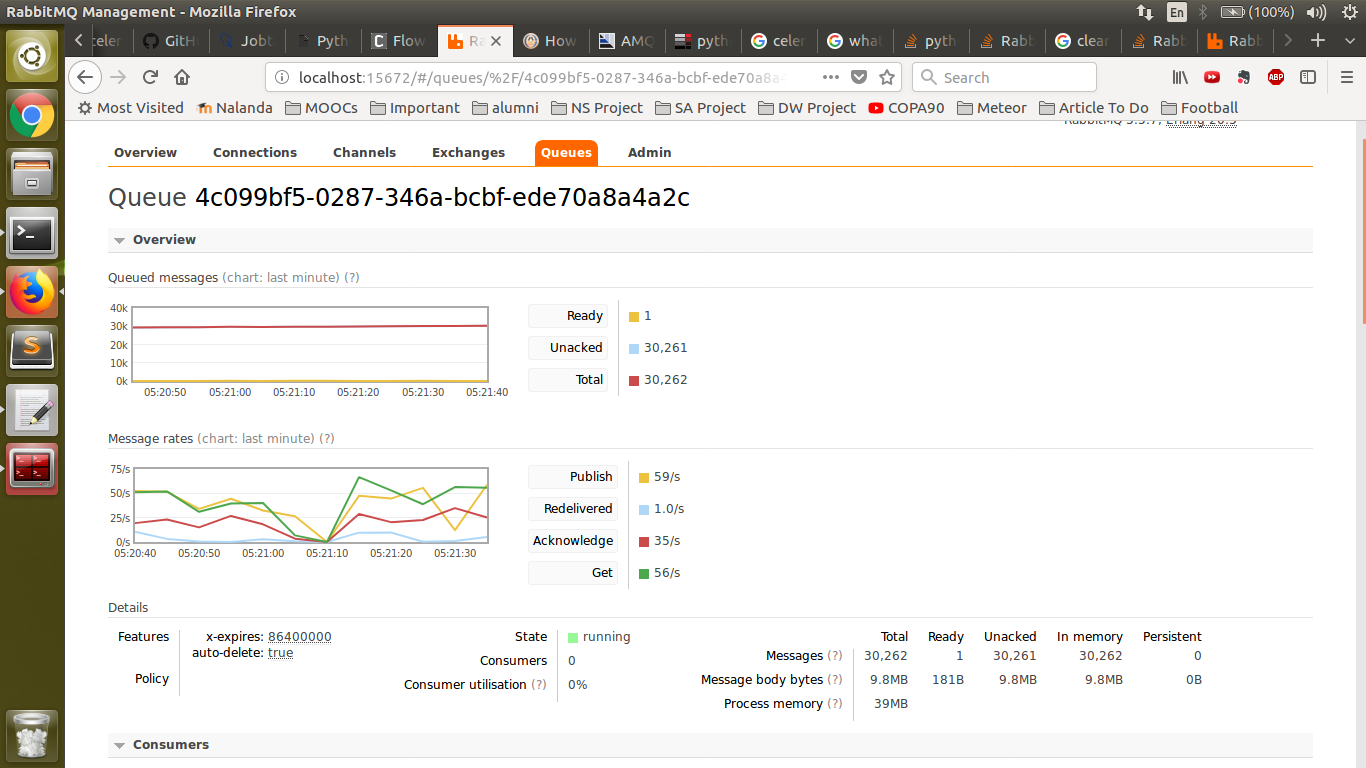
We observed to following benchmark statistics,

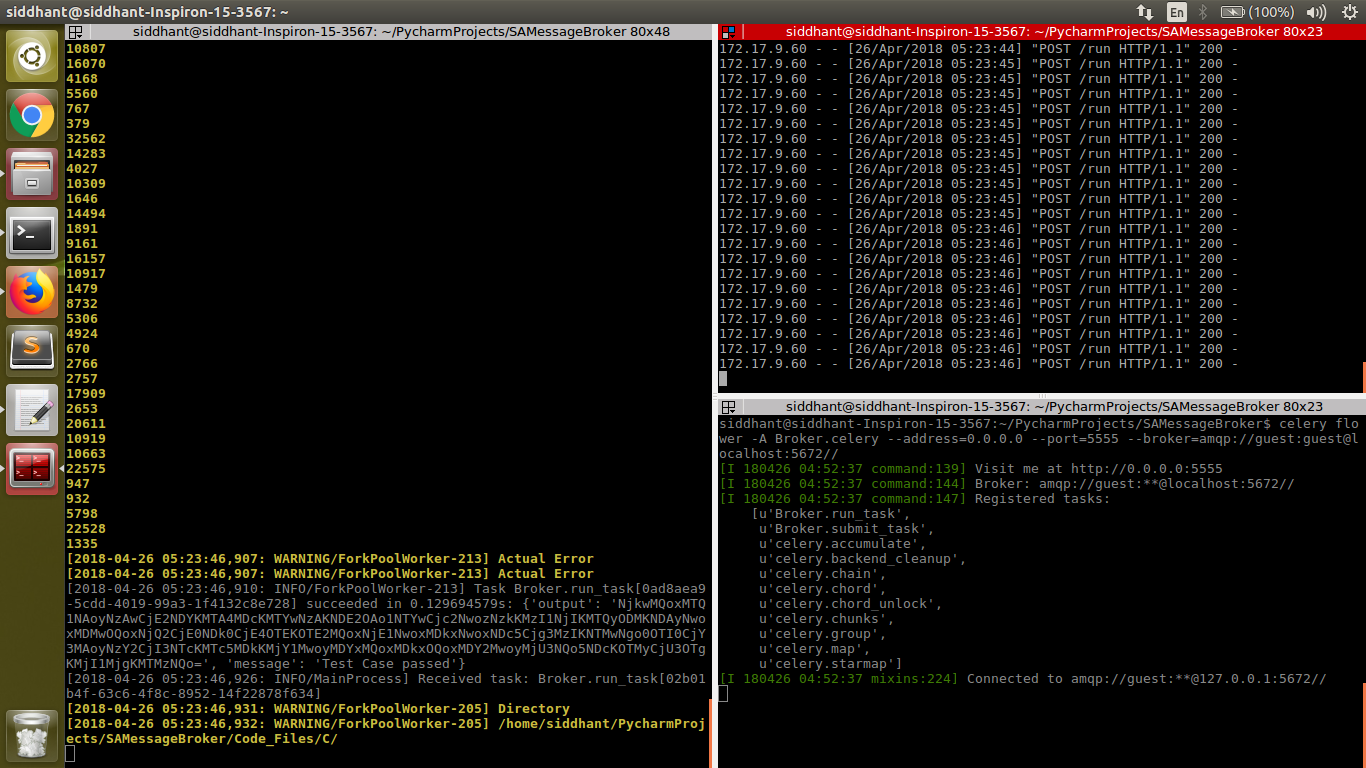
Avg Execution Time: 0.22 sec  
Avg Queue Time: 0.9 sec

****

****

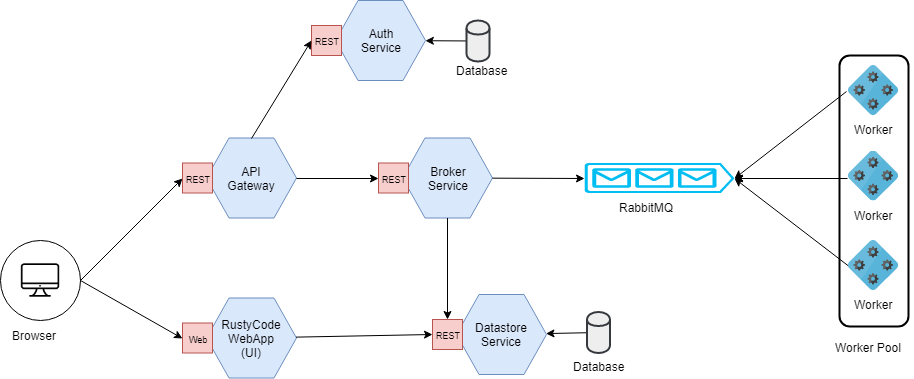
****

****

****

## 

## Architectural Diagram

****

## Screenshots

