Platform Application UI, Scheduler, Load Balancer

Group 1, Team 2

May 14, 2021

Abhishek Gorisaria, Divyansh Shrivastava, Aman Nautiyal

Mentor and Professor : Prof. Ramesh Loganathan

TA: Pratik Tiwari, Shubham Agarwal, Jay Krishna

Contents

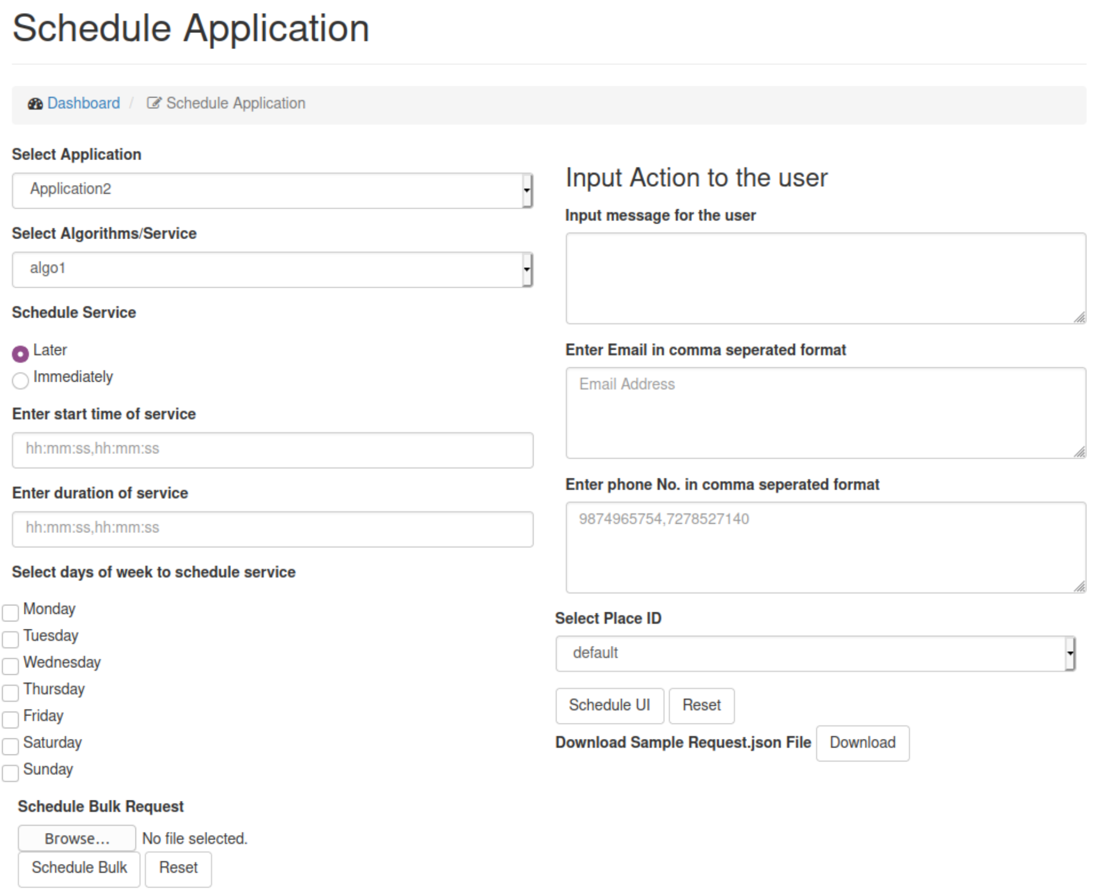
1 Platform and Application UI

1.1 Introduction

GUI provides user and developer/admin both with easy to use interface to develop and/or run algorithm/service of different class of application.

1.2 System Design of UI

Entire UI is made using bootstrap CSS framework. There are multiple options in UI:



Schedule Application

Dashboard / Schedule Application

Select Application: Application2

Select Algorithms/Service: algo1

Schedule Service

[✓ Later]
Immediately

Enter start time of service: hh:mm:ss,hh:mm:ss

Enter duration of service: hh:mm:ss,hh:mm:ss

Select days of week to schedule service

- Monday
- Tuesday
- Wednesday
- Thursday
- Friday
- Saturday
- Sunday

Schedule Bulk Request

Browse... No file selected.

Schedule Bulk | Reset

Input Action to the user

Input message for the user

Enter Email in comma seperated format: Email Address

Enter phone No. in comma seperated format: 9874965754,7278527140

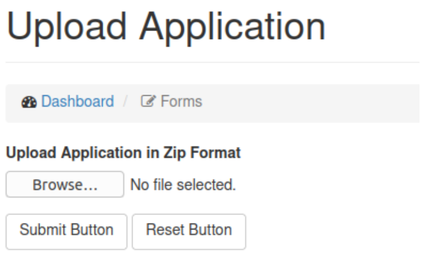
Select Place ID: default

Schedule UI | Reset

Download Sample Request.json File | Download

Figure 1: UI for scheduling request

- UI provides multiple ways to schedule the request:
  - **Bulk Option**: Through this end-user can upload multiple request by uploading *.json file.
  - **Field Option**: Through this option user needs to fill the Text fields given, and the schedule the request.



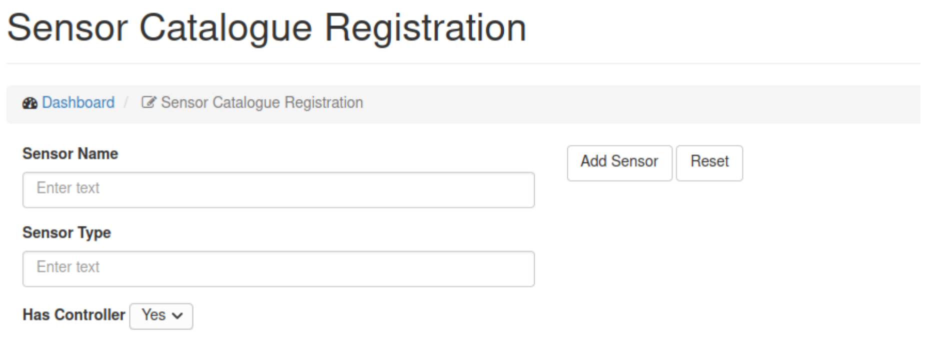
Upload Application

Dashboard / Forms

Upload Application in Zip Format

Figure 2: UI for uploading application

- UI provides way to upload the application and config.json inside *.zip format, which is being checked for format validity, if so, it uploads the application.zip into Application Repository.



Sensor Catalogue Registration

Dashboard / Sensor Catalogue Registration

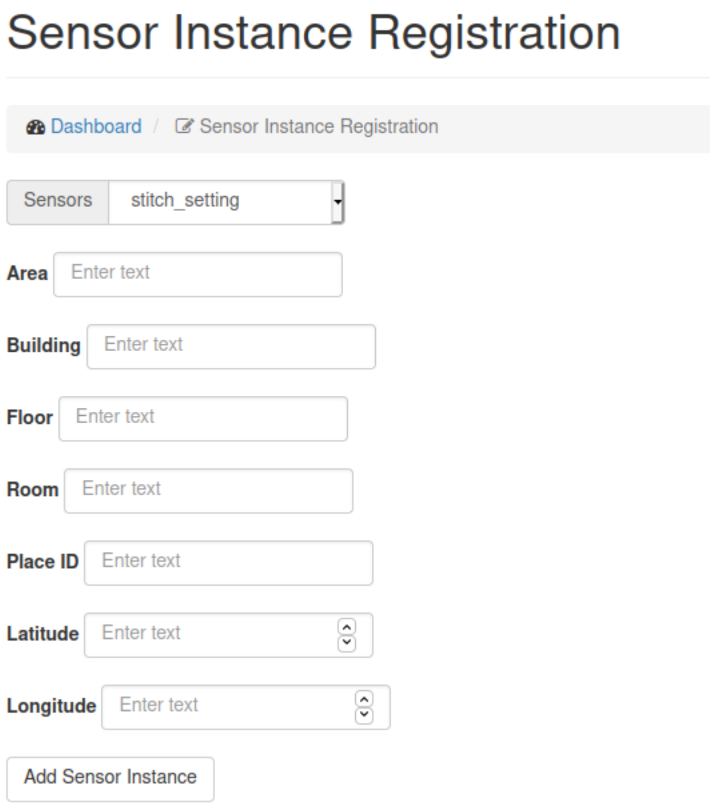
Sensor Name: Enter text

Sensor Type: Enter text

Has Controller: Yes

Figure 3: UI for registering sensors catalogue

- UI provides way to register sensors in catalogue, which developer can use to develop his/her algorithm.


Sensor Instance Registration

Dashboard / Sensor Instance Registration

Sensors: stitch_setting

Area: Enter text

Building: Enter text

Floor: Enter text

Room: Enter text

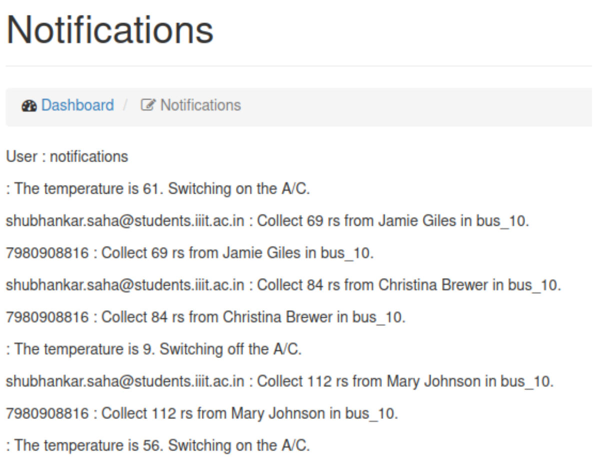
Place ID: Enter text

Latitude: Enter text

Longitude: Enter text

Figure 4: UI for registering sensors instance

- UI provides way to register the instance of sensor, like the place, room, building, floor where sensor resides. (Figure 4)



User : notifications

: The temperature is 61. Switching on the A/C.

shubhankar.saha@students.iiit.ac.in : Collect 69 rs from Jamie Giles in bus_10.

7980908816 : Collect 69 rs from Jamie Giles in bus_10.

shubhankar.saha@students.iiit.ac.in : Collect 84 rs from Christina Brewer in bus_10.

7980908816 : Collect 84 rs from Christina Brewer in bus_10.

: The temperature is 9. Switching off the A/C.

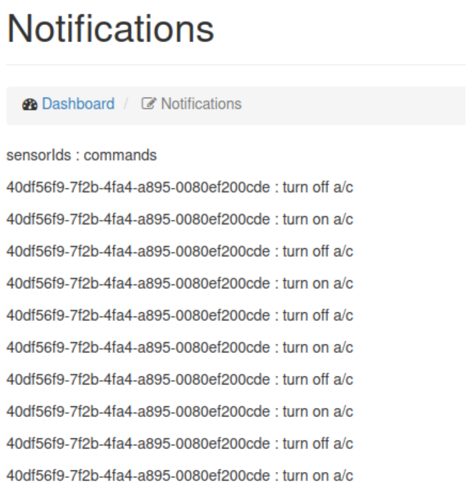
shubhankar.saha@students.iiit.ac.in : Collect 112 rs from Mary Johnson in bus_10.

7980908816 : Collect 112 rs from Mary Johnson in bus_10.

: The temperature is 56. Switching on the A/C.

Figure 5: UI for seeing notification of user

- UI provides way to see notifications sent to user.



sensorIds : commands

40df56f9-7f2b-4fa4-a895-0080ef200cde : turn off a/c

40df56f9-7f2b-4fa4-a895-0080ef200cde : turn on a/c

40df56f9-7f2b-4fa4-a895-0080ef200cde : turn off a/c

40df56f9-7f2b-4fa4-a895-0080ef200cde : turn on a/c

40df56f9-7f2b-4fa4-a895-0080ef200cde : turn off a/c

40df56f9-7f2b-4fa4-a895-0080ef200cde : turn on a/c

40df56f9-7f2b-4fa4-a895-0080ef200cde : turn off a/c

40df56f9-7f2b-4fa4-a895-0080ef200cde : turn on a/c

40df56f9-7f2b-4fa4-a895-0080ef200cde : turn off a/c

40df56f9-7f2b-4fa4-a895-0080ef200cde : turn on a/c

Figure 6: UI for seeing notification of controller

- UI provides way to see controls sent to sensor to update the parameter of sensor.

2 Scheduler

2.1 Introduction

- Scheduler Service is responsible for scheduling various jobs that are requested by the client.
- Scheduler Service can schedule jobs with fixed starting times as well as those jobs which are to be executed at some fixed interval or on some specific days.
- There is a 1-second time resolution with which this scheduler service can distinguish between the scheduled time of two requests.
- Scheduler also uses a MongoDB collection for storing the information of all the requests that it has handled.
- Scheduler is also responsible for assigning a unique service ID to all the incoming requests with which it will be identified in the entire platform.
- Scheduler also maintains the state of all the services (`SCHEDULED, RUNNING, TERMINATED, ABORTED`) in the persistent database.

2.2 Control and Data Flow

- The scheduler service gets it's input from the UI or direct request json file upload after the the request is validated. The request obtained is in JSON format.
- Scheduler checks if the request is to be deployed immediately or at some later stage. If it needs to be deployed immediately, it goes to the last step.
- Scheduler uses a internal min-heap based priority queue which is used to hold the scheduled jobs on the basis of their scheduled deployment time.
- Scheduler can add jobs to this priority queue based on the request's details. In some fixed time intervals, Scheduler will also back up this queue in a persistent database.
- Whenever the scheduled time arrives, the scheduler generates a request that invokes the Service life cycle manager to forward the request to deployer for deployment.
- If the request happens to be periodic in nature, the scheduler again pushes a new entry into the priority queue that corresponds to the next deployment time which is basically current time + time period specified.

2.3 System Design

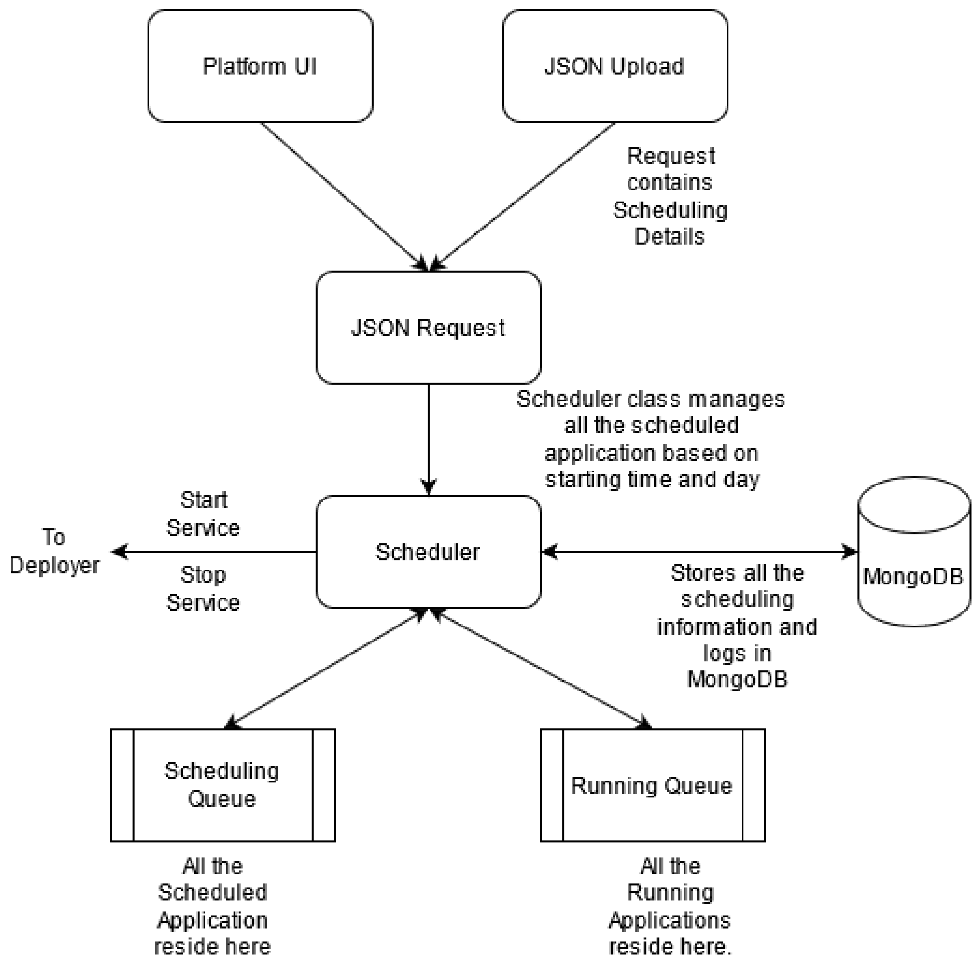

Platform UI
JSON Upload
Request contains Scheduling Details
JSON Request
Scheduler class manages all the scheduled application based on starting time and day
Start Service
Stop Service
To Deployer
Scheduler
MongoDB
Stores all the scheduling information and logs in MongoDB
Scheduling Queue
Running Queue
All the Scheduled Application reside here
All the Running Applications reside here.

Figure 7: System Design of Scheduler

2.4 Data Structure Used

The two main data structures used in this module are :

1. Dictionary : This is used to store the JSON objects in python.
2. Min-Heap Priority Queue : This is used to store the Scheduled jobs on the basis of their scheduled time of deployment.

2.5 Technologies Used

1. **Flask** : Flask is a micro framework in python that can be used for web development as well as writing REST APIs. We will be using flask extensively for writing REST APIs for various functionalities like adding a request for scheduling.
2. **MongoDB** : MongoDB is a NoSQL Database which will be used to store the scheduling queue in a persistent storage. MongoDB is used to store the information of all the present, past and future services that are to be deployed on the platform. It also stores the state of all the requests.

2.6 Input and Output APIs

2.6.1 Input APIs

There are two Input APIs for the scheduler. The `schedule_request` API is used by the user to request the running of some algorithm. The `aborted_service` API is used by the deployer to indicate that the service that was running as stopped abruptly and needs to be restarted.

1. `/schedule_request` :
   This API is used to register a request for running some algorithm on our platform. It accepts simple POST requests with payload in JSON format. An example of a sample request is shown below :

```
{
    "myapp" : {
        "user_id" : "ias",
        "application_name" : "Application2",
        "algorithms" : {
            "algo1" : {
                "isScheduled" : false,
                "schedule" : {
                    "time" : {
                        "startTimes" : ["00:45:00", "00:45:15"],
                        "durations" : ["01:00:20",  "02:00:15"]
                    },
                    "days" : ["Monday", "Thursday", "Friday", "Saturday", "Sunday"]
                },
                "action" : {
                    "user_display" : "Results",
                    "sensor_manager" : [{"sensor_id1" : "command1"}],
                    "notify_users" : ["123456789", "abc@gmail.com"]
                },
                "place_id" : "bus_10"
            }
        }
    }
}
```

2. `/aborted_service/<service_id>` :
   This API is used to change the state of some running algorithm/service from `RUNNING` to `ABORTED`. It accepts GET Requests with aborted service being the part of the route itself as mentioned above.

2.6.2 Output APIs

Scheduler needs to hit these two APIs for starting and stopping the deployment of the requested algorithms. Both of these APIs are defined in the Deployer.

1. `/startdeployment` :
   This API is used to start the deployment of the requested algorithm at the appropriate time. It requires the scheduler to send a json based POST request. The structure of the json request is shown below :

```
{
    "user_id" : "abc",
    "app_id" : "xyz",
    "service_name_to_run" : "algo1",
    "service_id" : 3,
    "place_id" : "Himalaya 105",
    "action" : {} //Action Information
}
```

2. `/stopdeployment` :
   This API is used to stop the deployment of the requested algorithm at the appropriate time. It requires the scheduler to send a json based POST request. The structure of the json request is similar to what is shown for the startdeployment API.

2.7 Interaction with other Modules

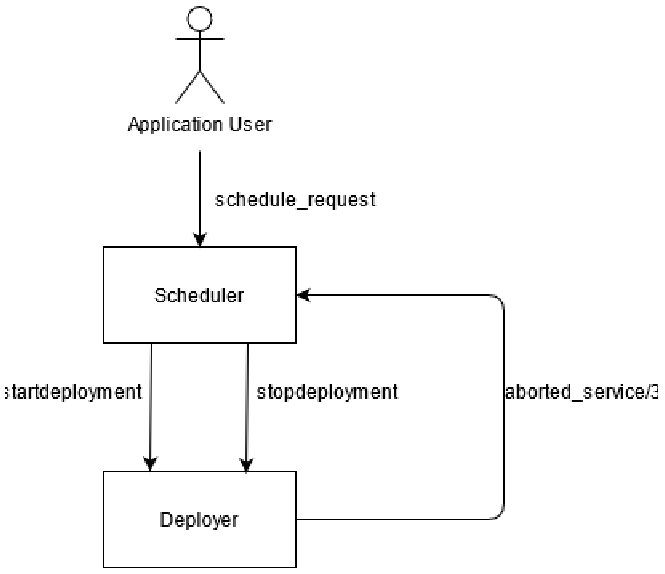


Figure 8: Interaction of Scheduler with other Modules

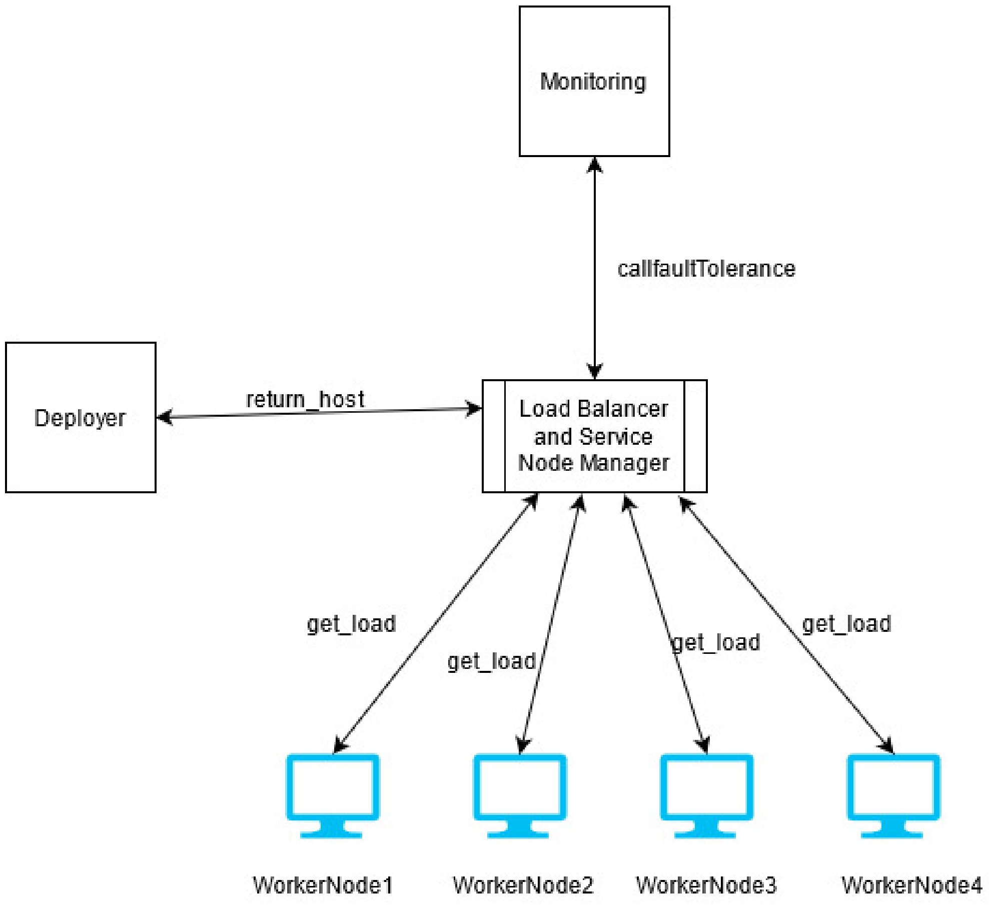


Figure 9: Load Balancer and Service Node Manager

3 Load Balancer and Service Node Manager

3.1 Introduction

The load balancer is a platform service which facilitates the deployment of application services on hosts such that the workload is balanced. The function of load balancer within the platform is to provide the deployment module with the appropriate host machine on which the application service is to be deployed.

In its function as the Service Node manager, this module polls the Service nodes to see whether they are running, and calls the Monitoring module for the fault tolerance.

3.2 System Design

The flask machines have a continually running flask api in the background. When the api is hit, the service nodes return the current cpu and memory usage of the node. The load balancer hits all service nodes for info about their resource usage. Consequently, it returns the node with lowest resource usage (priority given to memory usage) to the deployer module. During the same process of polling for resource usage, the module can recognize which nodes are down. Subsequently, it calls the monitoring module for the nodes that are down to ensure fault tolerance.

3.3 API's Provided

1. **return_host** : load balancer returns a host ip on which the new algorithm can be deployed based on lowest resource usage. Doesn't return a host if memory usage is full on all hosts or if no hosts are connected.
2. **get_load** : service nodes return the resource usage statistics on the node.

3.4 Technology used

1. **MongoDB** : MongoDB is a NoSQL Database which is used for persistent storage.
2. **Flask** : Flask is a micro framework in python which is used for providing the REST API's.
3. **Pymongo** : Pymongo is the python driver to access the Mongodb server.
4. **Docker** : Dockers are used to run modules in structures called containers, which are separate environments of their own.

3.5 Implementation details

The service nodes are simulated through seperate containers which are connected to the same bridge network as the rest of the modules in the network. Ip resolution takes place automatically by the use of container names. The container names are stored in the MongoDB database when bootstrap initializes the platform. The database acts as the persistent storage for the service nodes that are connected to the platform. Flask is used for providing the API in case of load balancer and service nodes. The Load balancer gets resource usage from service nodes, sorts them in ascending order based on use and provides node with lowest resource usage to the deployer. The service nodes use the cgroup file system of Docker containers to get runtime resource usage statistics.