Team 3
Module Design

PREPARED FOR

IAS PROJECT - GROUP 4

IIIT-Hyderabad

PREPARED BY

TEAM 3

Dhruv Sachdev (2020201060)

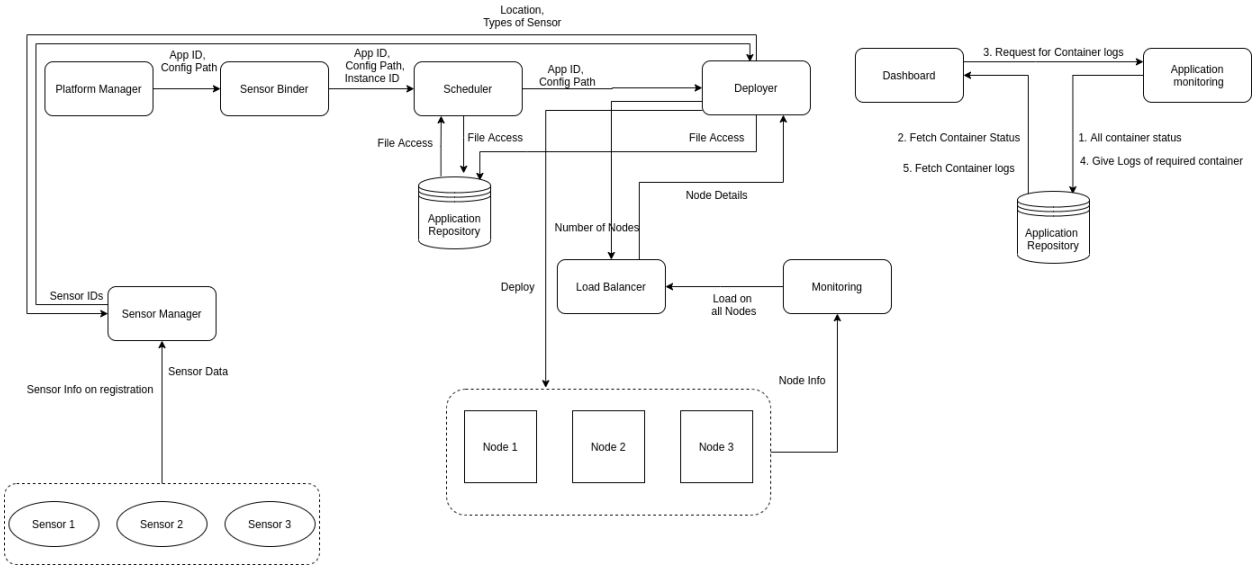
Nisarg Sheth (2020201049)

Pujan Ghelani (2020201083)

List Sub modules

1. Platform Manager
2. Sensor Binder
3. Scheduler
4. Deployer
5. Load Balancer
6. Application level monitoring
7. Service Lifecycle

Block Diagram



Link for above Diagram:
https://drive.google.com/file/d/1fV1IFgRyLO8SQyL-tTXlv7z_vwWNVbCn/view?usp=sharing

Interaction among sub modules

1. Sensor Binder
    a. Platform Manager
    b. Scheduler

2. Scheduler
    a. Sensor Binder
    b. Deployer
    c. App Repo

3. Deployer
    a. Scheduler

    b. Load Balancer
    c. App Repo
    d. Service Lifecycle

4. Load balancer
    a. Deployer
    b. Monitoring

5. Monitoring
    a. Deployer
    b. Load Balancer
    c. Nodes

6. Service Lifecycle
    a. Deployer
    b. App Repo
    c. Nodes

Brief Overview of each submodules

- **Interaction between submodules and Functionality of submodules**

1. **Sensor Binder:**
    - Sensor binder receives a request from platform manager in the form of deployConfig.json file.
    - We use the filters given in the config file which are used to decide which sensors should we bind to the application.
    - The module looks into the database where the details of sensors are stored to find the ones matching to our criteria.
    - It throws an error if it does not find the sensors to bind required by the application.
    - Else, it generates an instance id and forwards the deploy config file to scheduler which handles from there on.

2. **Scheduler:**
    - Scheduler receives the application id and the location of the configuration file in the application repository from the platform manager.

- Scheduler will fetch the configuration file and it would have to verify if the configuration in the file are in the correct format and according to the constraints of the platform(if any).

- Scheduler will have to store the scheduling details in a list and set some form of cron job which would grab the necessary files and pass it to the deployer at the appropriate time/interval.

- It will also send a request to stop the application to the deployment manager either at the scheduled end time or an interrupt received from the platform manager.

- It will do so by maintaining two priority queues, one queue will be used to handle the start time of the scheduled applications and the other queue will be used for handling the stop times.

3. **Deployer:**

    - Deployment Manager receives the application id and the location of the configuration files in the application repository from the scheduler.

    - It will have to locate and identify the sensors which are associated with the application that is to be deployed. It would have to communicate with sensor manager to get the information about the sensors.

    - Deployment manager will have a sub system called a load balancer which will receive the number of nodes required for the application from the deployer. It will try to find the required number of nodes with least load and will create nodes if there aren't enough nodes available. It would return the details of the nodes bound with the application to the deployer.

4. **Load Balancer:**

    - Load Balancer keeps receiving information about the load(RAM usage, CPU Usage) on computing instances from the monitoring module so that it can make decisions to balance the load.

    - Deployer will setup the environment and packages on the nodes provided by the load balancer and then start the application program on those nodes.

5. **Application level Monitoring**
    - It will monitor various application statuses and If it is one of the following 137,143,133 status codes, it shows that the application was stopped/exited abruptly by any reason like shutting down of the machine or any external

interrupt. In that case, our module will detect such cases and send the restart request and restart the application on the machine suggested by load balancer.

- Dashboard will ask this module about container logs detail and It will provide required container logs to dashboard.

6. **Server lifecycle:**

   - Each server will have a kafka topic on which it will send its status every 30 seconds.
   - The Server lifecycle will consume from each server's topic and if it does not receive anything for 80 seconds, it assumes the server is down and will restart the server.
   - If any applications were deployed on the server app monitoring module will detect it and after the server is restarted it will re-run all the applications that were running at the time of crash.
   - If any platform services were deployed on the server service lifecycle manager will detect it and after the server is restarted it will re-run all the services that were running at the time of crash.
   - Service Lifecycle will fetch a list of all containers running in the platform from MongoDB.
   - Every service of the platform is running in a docker container. It will only consider service containers from a list of containers and will check if the container is running fine using container logs. If it finds out that service has crashed, it will restart a service and update it in MongoDB collection (service_status) with a new container id.

Config Files of our Submodules

1. **deployConfig.json**

```
{
  "application_name":"app1",
  "instance_id" : "13",
```

```
  "script_names" : ["script1.py"],
  "algorithm_name":"algo1",
  "sensor_info":[
     {
        "sensor_type":"soil-moisture-sensor",
        "filter_sensors":[
           {
              "location":"Indore"
           }
        ]
     }
  ],
  "scheduling_info":{
     "request_type": "start",
     "start_time": "",
     "end_time": "",
     "day": "",
     "interval": "30",
     "repeat": "no",
     "job_id": "123 exists only if request type is stop"
  },
  "environment":{
     "lang":"python",
     "dependencies": [["requests", ""], ["kafka-python", ""]]
  }
}
```

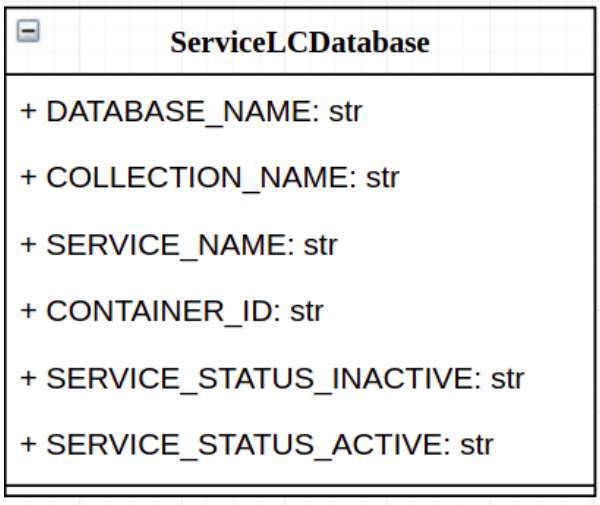
2. **serverDetails.json**

Final_lifecycles_server_service > service_life_cycle > {} server_details.json > ...

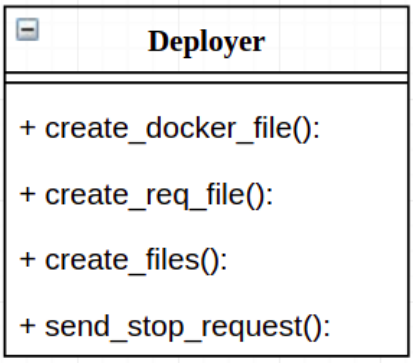
```json
{
    "1": {
        "node_name": "IAS-Node-1-new",
        "node_ip": "52.188.83.232"
    },

    "2": {
            "node_name": "IAS-Node-2-new",
            "node_ip": "20.62.200.216"
    },
    "3": {
            "node_name": "IAS-Node-3-new",
            "node_ip": "20.84.81.153"
    }
}

```

Classes In our submodules

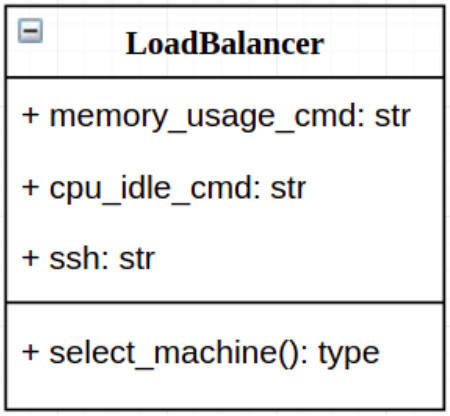
1. **ServiceLCDatabase**



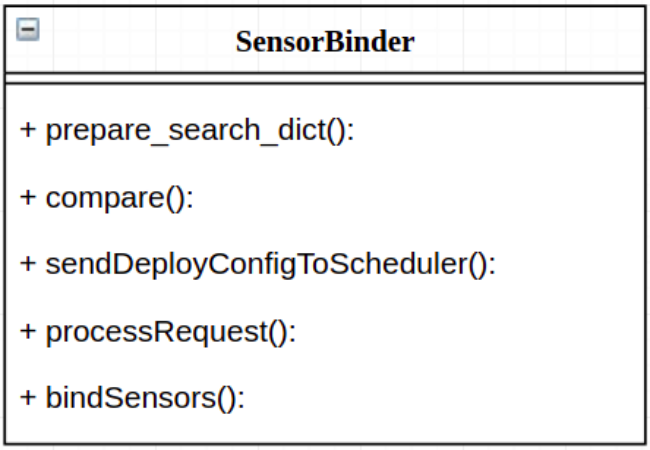
2. **Deployer**



3. **LoadBalancer**



4. **SensorBinder**

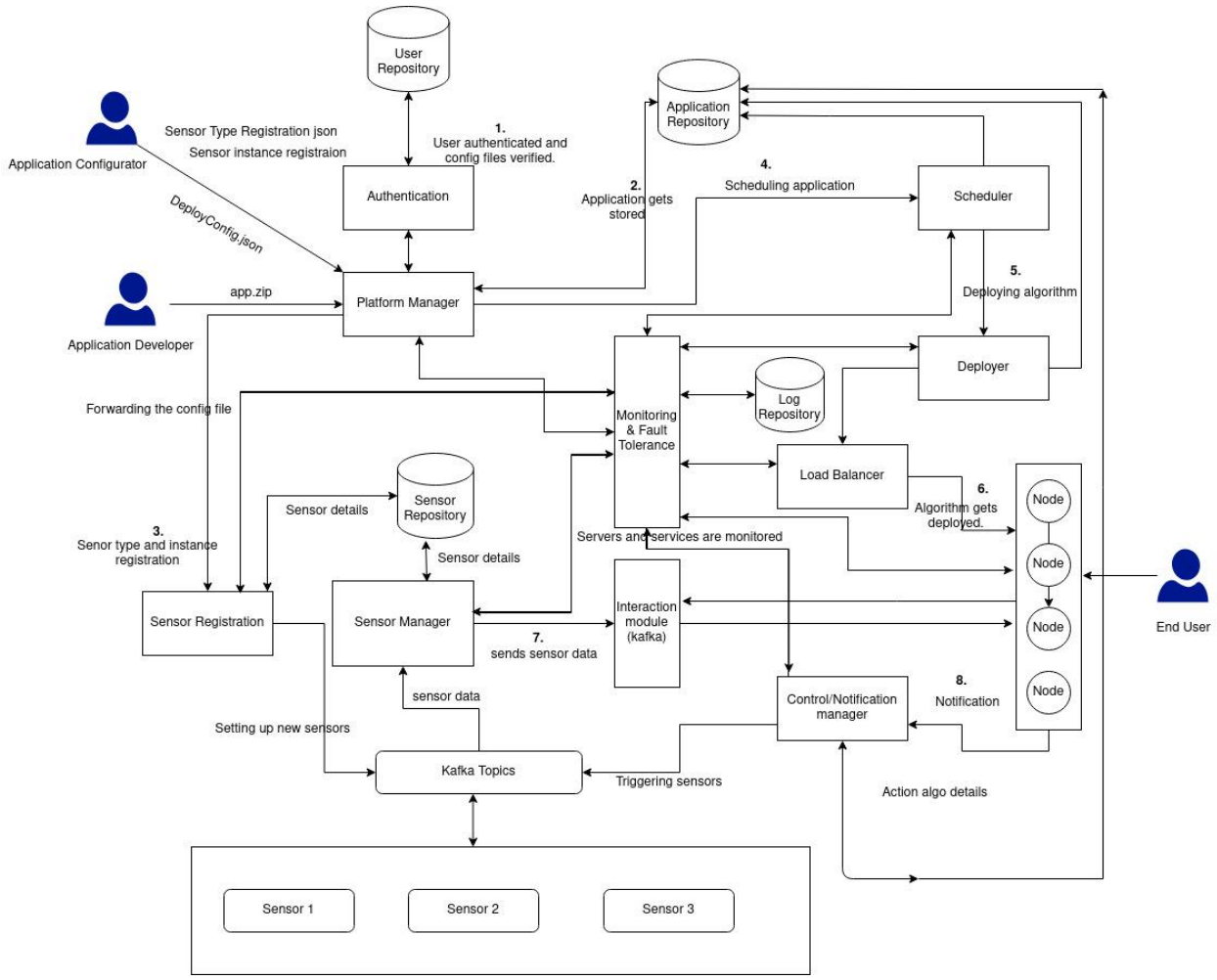


Interaction between sub modules

- All the communication happens between modules are using kafka.
- Below we are displaying kafka topic name by which communication is done and which files or data we are sharing from that topic.

platform_manager	pm_to_sensor_binder	sensor_binder	Send deployConfig.json contents
sensor_binder	sensor_binder_to_scheduler	scheduler	Send deployConfig.json contents
scheduler	scheduler_to_deployer	deployer	Send deployConfig.json contents

Interaction Between all other modules using Diagram



Interactions between other modules

1. Platform manager and Sensor Registration- The config files received from from the admin/configurator is forwarded via api from platform manager to the sensor registration to validate and register the sensor types and instance.
2. Sensor Manager and Node(with an application instance running) , the application forwards a request via api of a sensor data it requires , so the sensor manager gets the sensor data , binds it and forwards the sensor topics to the interaction module and it sends the data to the application.
3. Control/Notification manager and Node(with an application instance running) , the output generated by the application is given to the notification manager to notify the end users or trigger actions on the sensors.

Dashboard Screenshots