Communication Module with Bootstrap and Sensor Module

IAS Team Document

PREPARED FOR

IAS Project : Platform for Smart city like Application.

IIIT-Hyderabad

PREPARED BY

Group3-Team 5:

Ankush Mitra (2019202009)

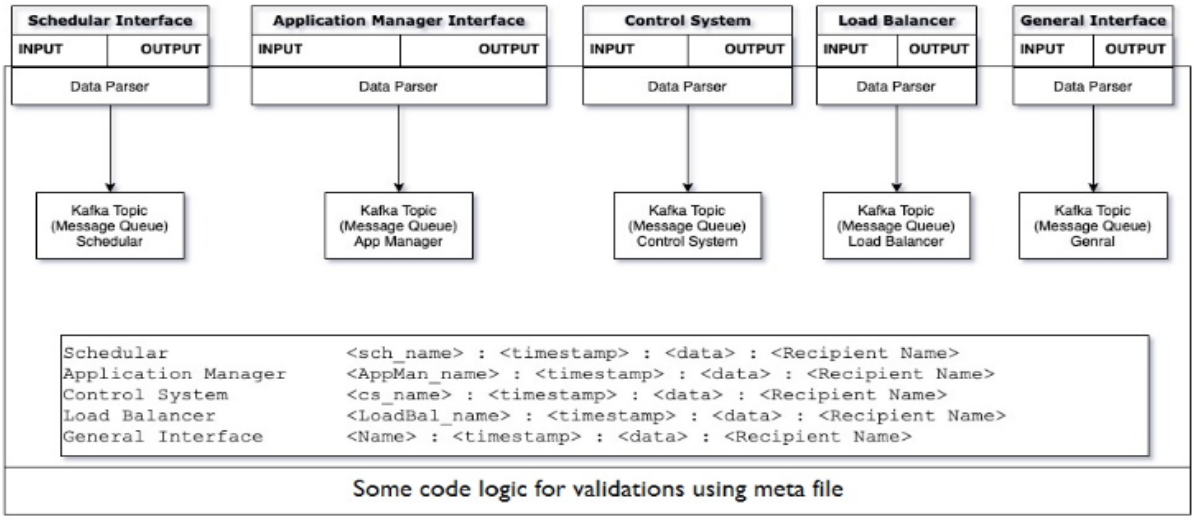
Adiyta Todi

1. Introduction

- **Definition**:
  The overall project is about making a platform where any one can build, run an IOT application(like smart city application) .The platform should be self sufficient to run any kind of complex smart city application. The platform contains module like sever life cycle , service life cycle , topology manager, load balancer, deployer etc. So the overall project is about making a distributed IOT platform (like Kaa) to run any complex smart city like application.

  In this project our team is building communication platform .The best thing about communication platforms is that it is the core of the system. We are planning to build the communication platform using Kafka. We will setup some standard Communication Standard to communicate with the various modules of the system. Basically we want to make Information Model for Communication system with all input and output APIs.

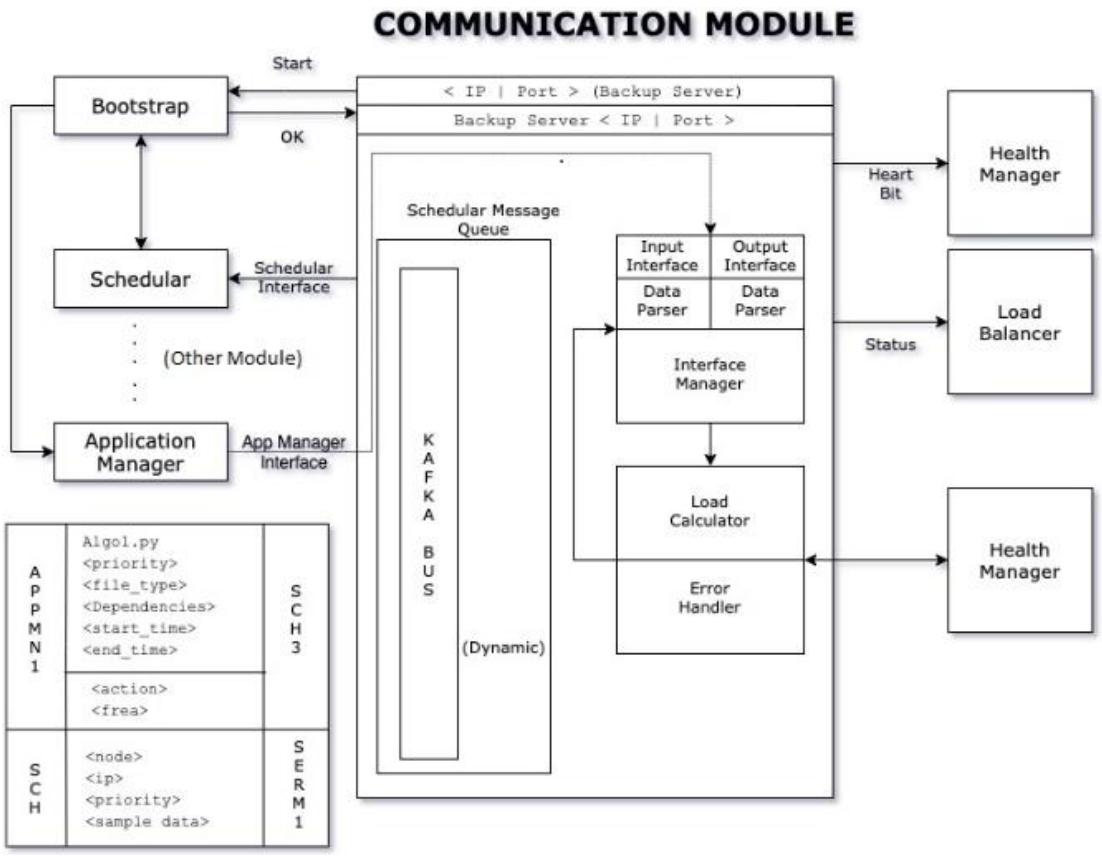
- **Diagram**:



- **Scope**:
  1. Our platform provides a set of independent services that the app developer can use.
  2. For app development with his own custom code updates. The platform provides various Independent Service.
  3. Platform can be used for development of IOT applications from scratch.
  4. Used can build smart hostel, smart city kind of applications on it.
  5. Platform also provides useful APIs to integrate independent applications on it.

2. Solution Design:

1. **Design big Picture**:



2. **Environment**:
    - 32/64-bit OS
    - Minimum RAM requirement : 2GB
    - Minimum Processor requirement : Intel Atom® E3826, 1.46GHz,Intel celeron 1.46 GHz
    - SSH support to OS for transferring files and models.
    - Kafka(32/64 bit)
3. **Technologies**:
    - Kafka
    - Bash Shell scripts can be used to make installation/configuration automated for
    - Python script for bootstrap.
    - Python virtual environment.
    - Sensor simulator.
    - Kafka for queueing data stream to the Model.
    - Configuration , meta-data file to setup the standard.

4. **Overall system flow**:

- First of all, We will get the **config file** from the platform repository and start the **bootstrap program (init.py).**

- It will then start the **kafka** and the application repository according to the config file and also start the **mongoDB** database.

- Then it will start the **Service lifecycle manager** which will manage the services on the platform and **Server lifecycle manager** which manages all the runtime servers info.

- After that all the modules of the system get up and according to the dependencies.

- Then the **Application Manager** will fetch the data from the message queue, It will make a configuration file from the extracted data and send it to the scheduler.

- **Scheduler** will create an instance which will contain **User_id, Algo_id, Start_time, end_time, sensor_types, geolocation, action.**

- Then Scheduler will map this algorithm id with the specific algorithm from the algorithm repository and It will schedule it accordingly.

- This algorithm repository can be updated by the system developer.

- Now, the scheduler will give this instance to the Service lifecycle manager and Service lifecycle manager contact the server lifecycle manager. Server lifecycle manager will select the runtime server on which this service will run.

- **Server lifecycle manager** will check which node is having less load which is updated in the registry by monitoring service and After applying the load balancing algorithm, it will give the response back to the service lifecycle manager. The response will contain the assigned Server Ip and Server port.

- State of each node is saved, so if any node fails then according to the last saved state it is resumed.

- Now the Service lifecycle manager will create a service(containing the **user_id, application_id, server_ip and server_port**) and give this to the deployment manager.

- This service information will be updated in the **registry**.

- **Deployment manage**r will bind the sensors to the algorithm instance. When the sensors are binded according to the geolocation provided by the user and then the algorithm instance can now access the stream of data related to that sensor and then run it on the server chosen by the server runtime manager.

- When the service is deployed, the deployment manager ping service lifecycle manager with the ack that “Service Deployed successfully”.

- All the communication is done by kafka and the sensor will produce the data on the sensor topic.

- Then finally that algorithm is run on that node and pushes its output on the message queue.

- **Action server** will perform the necessary action(Email, sms, etc).

- The **Topology Manager** manages the overall topology of the system. It is connected to the registry and it continuously checks whether a service is running or not with the help of the health checker module.

- If the service is down then It is re-executed by the service lifecycle manager from the state which was saved in the repository.

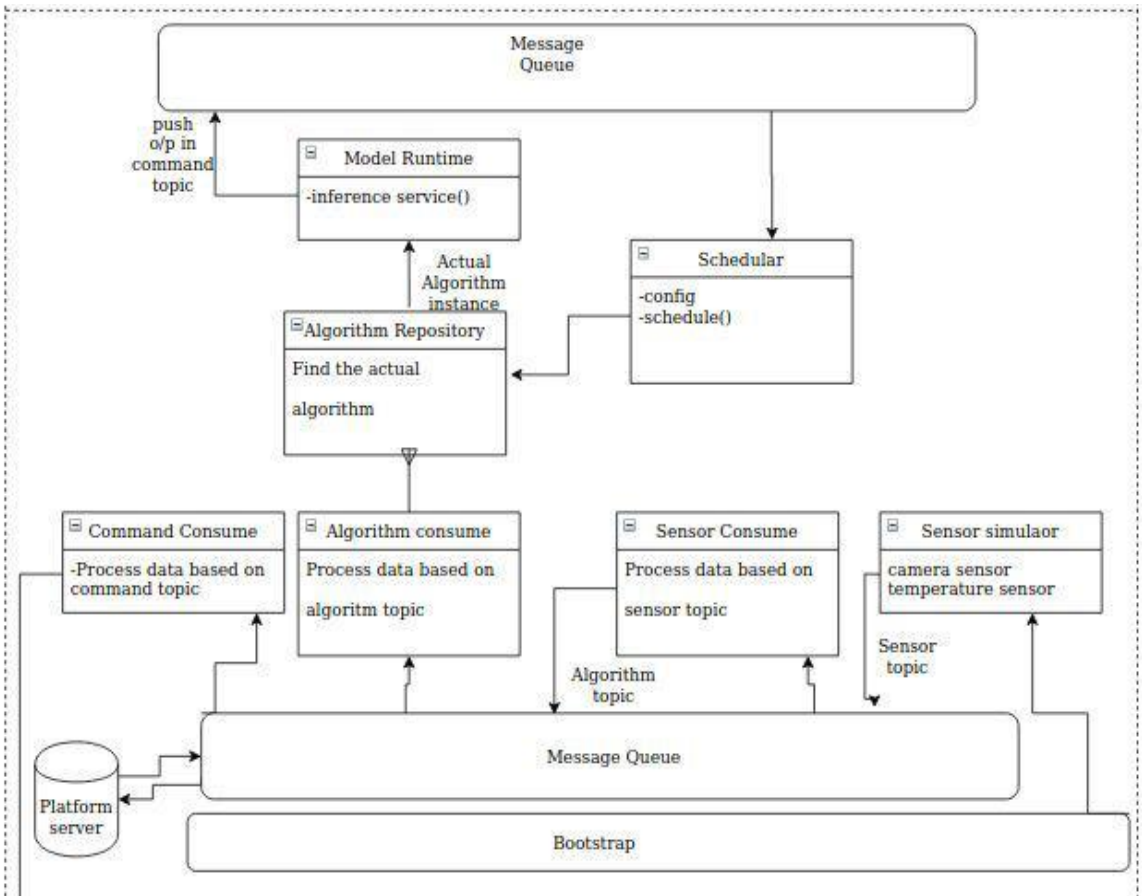
- **Monitoring module** will monitor all the services in the system. It will ping each and every service in the system and update its status in the registry. It will have a submodule heartbeat manager which will tell us whether it is alive or not.

- **Logging module** will maintain all logs of the system and dump it in the repository.

5. **Message Queue**:

- Message queue is the heart of this module
- There will be 3 kind of topic on message queue – sensor ,algorithm, command
- Message queue is the main communication channel between the whole system
- The structure of the message queue is defined in the config and meta data file
- Message queue is connected to almost all subsystem as shown in big picture

- Message Queue Interaction diagram:



6. **Integration**:

- We need certain kind of integration to run this module -
- Integration with the scheduler module is required to run the actual algorithm.
- Integration with the algorithm repository module is also required.

- Integration with deployment service is required as deployment service will query the sensor for appropriate sensor .

- Integration with Service life cycle module is very important as it controls all service.

- Integration with Server life cycle is also required as it check where to run a service.

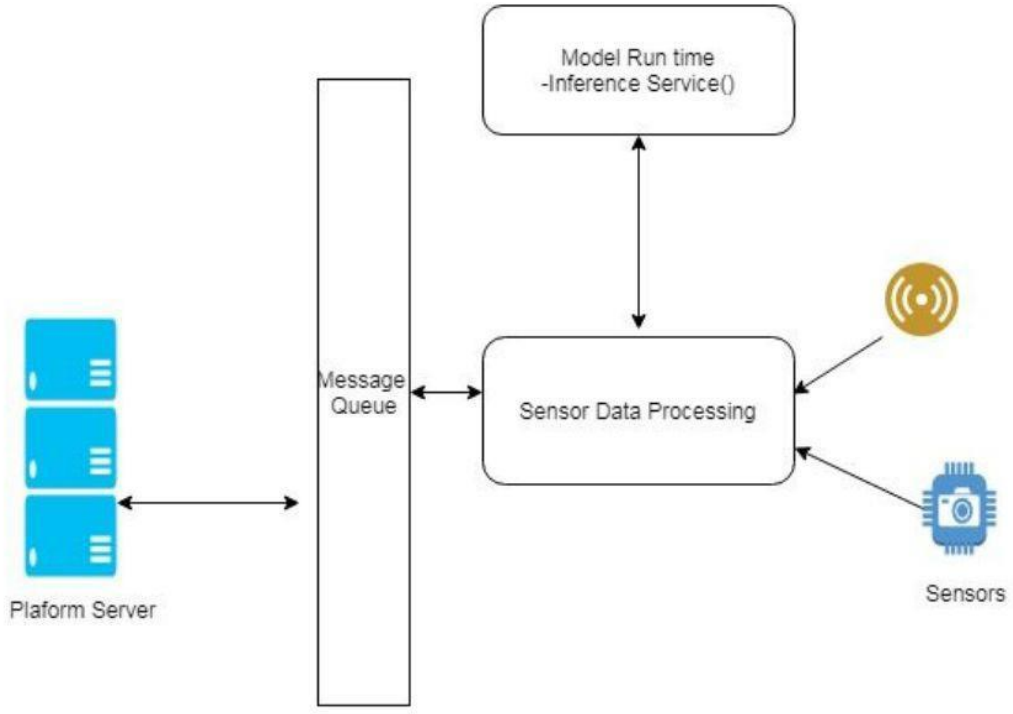
- Integration with Request manager is also required as it controls the Runtime server.

- Integration with Action server is also required as it takes the data from the Run time server.

- Integration with sensor module is also required as sensor will push the data in the sensor topic.

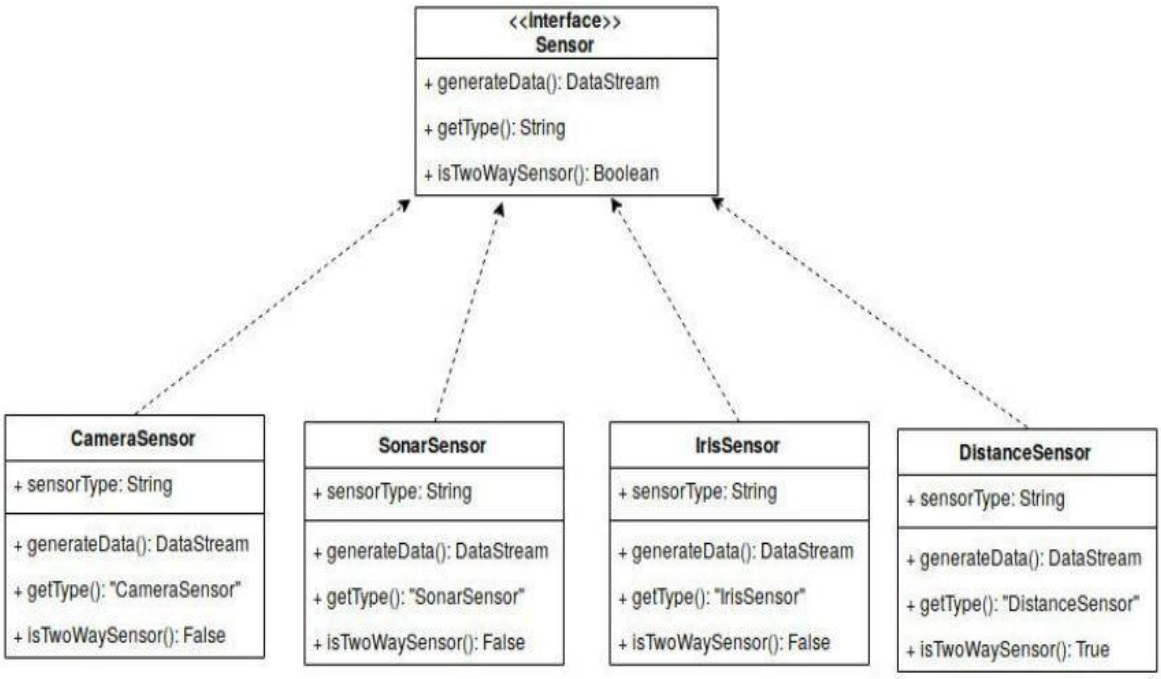
7. **Sensor data processing**:

- It will fetch the data from sensor topic(message queue) and do some processing
- Processing involve analog to digital conversion based on sensor type
- Match frequency of the sensor
- Diagram to show how sensor data processing is integrated with other subparts.



3. Subsystem and APIs:

1. **Sensor Interface**:



2. **Bootstrap**:

   - Bootstrap will contain init.py and config.yml and meta-data file to load the module based on the config . It will read the dependencies from the config file and up the required module in the sequence. It will also do some basic setup like installing nfs,docker etc.

3. **Storage Structure**:

    - Sensor Information:
        - Sensor Type
        - Sensor Location
        - Sensor ID
        - Gateway ID in which sensor is deployed
        - Algorithm to which it can be binded.

    - Gateway Information:
        - Gateway ID
        - Gateway Name
        - Gateway IP
        - Gateway PORT
        - Sensor Lists
    - Algorithm Information:
        - Algorithm ID
        - Algorithm Name
        - Sensor Lists can be binded

4. **Sensor topic Consumer**:

    - Responsible for fetching data from message queue on sensor topic.
    - Process the data(analog to digital conversion,frequency match)
    - Push data on message queue on algorithm topic.

5. **Algorithm topic Consumer**:
    - Responsible for fetching the data from the message queue on algorithm topic.
    - Find the algorithm from the algorithm repo based on scheduler output.
    - Call the actual algorithm.

6. **Algorithm repository**:

    - Contain all algorithm in a directory
    - Provide model runtime for each algorithm

7. **Command topic consumer**:

    - Responsible for fetching the data from the message queue on command topic.
    - Fetch the actual command and give output to external system.

4. Test cases:

- Test Case for Sensor Module:
  - Given sensor type and location will it able to filter out all valid sensor id.
  - During sensor registration will it successfully able to parse the data coming form Application manger and store the write information to registry.
  - Will it able to properly simulates the real time sensor.
  - How to control the sensor .

- Test Case for Communication Module:
  - If two module call it for their communication then if it is able to take data from one and send it to other.
  - How data parsing is handled in input as well as output interface.
  - How to manage sensor data of different type.
  - How to handle dynamic topic creation.
  - How to handle load and do partition on topic at runtime.

- Test Case for Bootstrap:
  - Is it able to understand the config file and load the system in sequence.
  - Is it able to install all basic software to make the system runnable.
  - Is it able to login to remote server to deploy the whole platform.

5. Configuration and metadata:

1. Configuration file:

- Config.yml (bootstrap):

{
Version:
Services:
Scheduler:
Image:
Ip:
Port:
Volume:
Dependencies:
App mang .
App man:
Image:
Ip:
Port:
Dependencies:
Service Life Cycle
Service Life Cycle:
Image:

}

2. Metadata:

- Sensor Registration:

{
Type:
Location :
Location type:
IP:
Port:
Range:
Frequency:
Sign-up time:
}

- Sensor manger:

```
{
    Type:
    Topic Name:
    Stream type:
    Id: [list of sensor id that satisfy the range
    and other requirement]
}
```

- Com Module to Deploy module:

```
{
    Topic type:
    Topic Name: (Dynamic)
    Data: {
        Type:
        Id:
        }
}
```

- Fixed Rule:

```
{
    Rule type:
    Rule:
    {
        Strat time:
        End time:
        Ip:
        Port:
        Priority:
    }
}
```

- Bootstrap start sh:

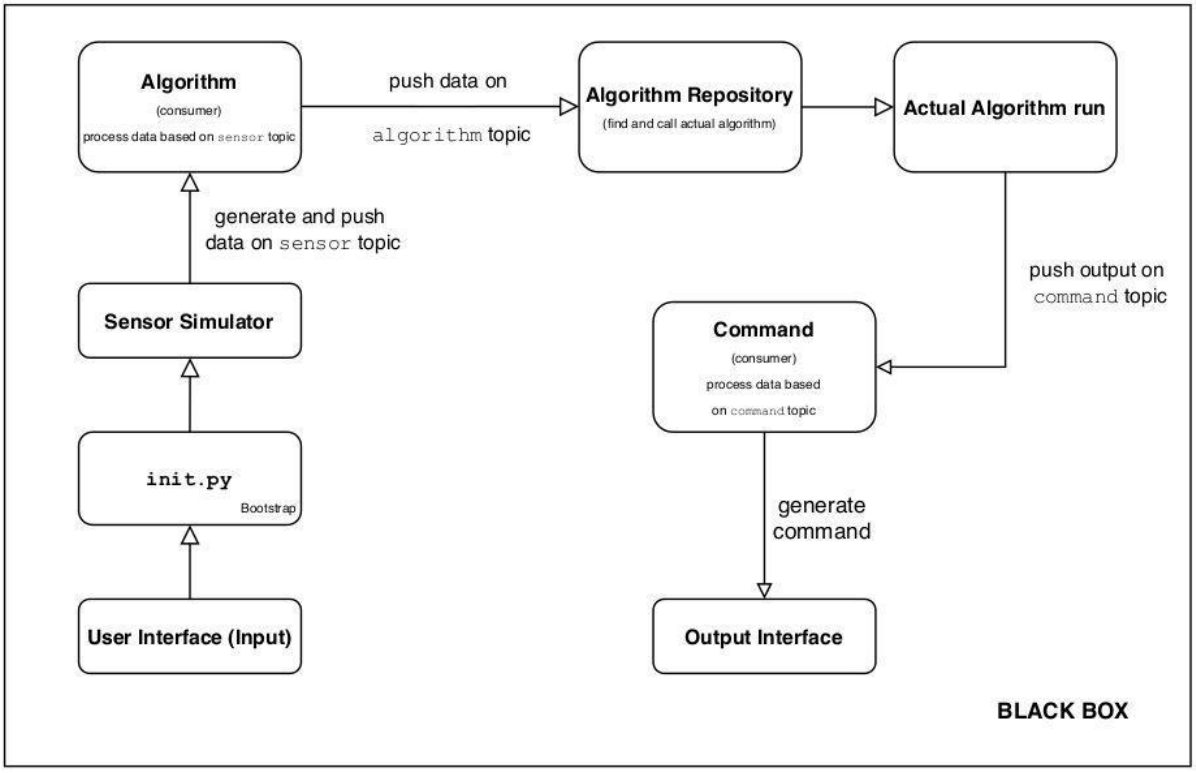
  -install requirement.txt
  -install basic package
    -nfs
    -docker
    -putty
  -deploy server lifecycle
  -remote login
  -run init.py based on config file

6. Persistence:

- Using the pickle library in python we can periodically back up the necessary data.
- For restarting the failed service form the point of failure we can use the logging service.
- We can use backup server for fault tolerance.
- We can use distribute the load by portioning the topic on kafka to help the system to handle huge stream .

7. Conclusion:

- Overall system diagram:



In this module we have build the base communication platform between user and rest of the system. In integration phase we will also some more interface for intercommunication between different module. In integration phase we will also add API form other module within it (like scheduler API, Load balancer API for choosing right algorithm from algorithm repo).