IAS
Project Design

Prepared by:
Team 1(Group 3)
Harshit Pandey , Shobhit gautam

Introduction to the Project:

Definition:

Scheduling component is a microservice based architecture of a platform that coordinates with all other subsystems and different services. It will also manage scheduling of models on gateway or server with different scheduling policy.

App manager manages all incoming requests from user through various apps, also ensuring user is valid and is sending data in described platform format. Further sending it to scheduler and service life cycle manager.

Problem Space:

This platform will provide a set of independent services which will take input from the application manager module and deliver it to the topological balancer. This component of the project will map the correct algorithm with the required service.

Test Cases:

1. **Check User Access:**
   To ensure the user accesses the correct set of information to which he is authorised to use.
   Input:Username
   Output:Returns list of services accessible by the user.

2. **Binding with algorithm:**
   Scheduler will take input from the Application Manager and will map it to the correct algorithm.
   Input: Start Time,End Time.
   Output:To ensure if the model is running between the start and end point.

3. **Packaging Information:**
   To ensure the Information is properly managed and send to topological manager for load balancing.

4. **Ensure Package is received correctly:**
   It will make sure our Topological manager has received the package correctly for load balancing.

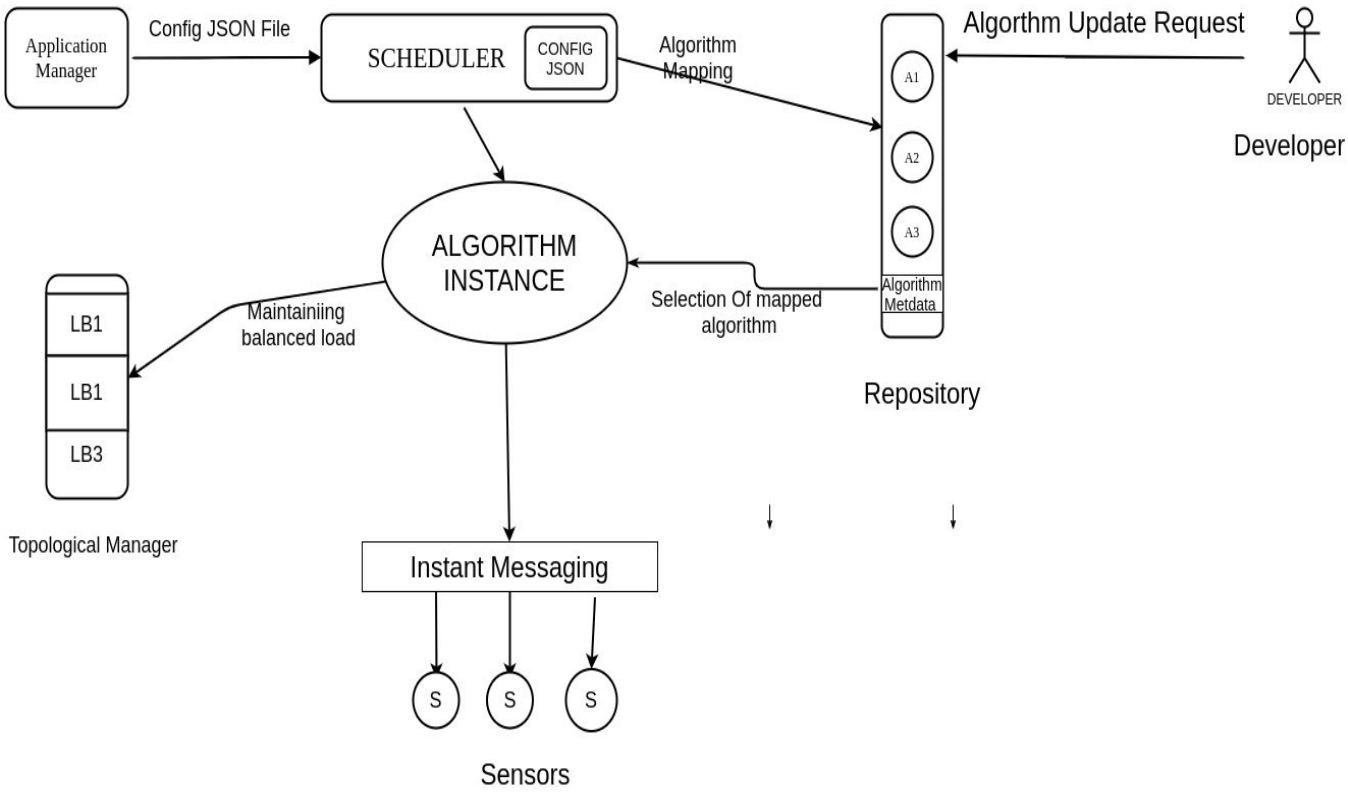
5. **Ensure Action is performed:**
   It will make sure that Action is performed correctly or not as desired by the user .

6. **Sensor Binding:** Scheduler has the task to bind the sensor with correct application.

7. **Notifying User:**
   To make sure the user gets notified when the service requested is processed correctly.

Solution design considerations:

3.1 Design big picture:



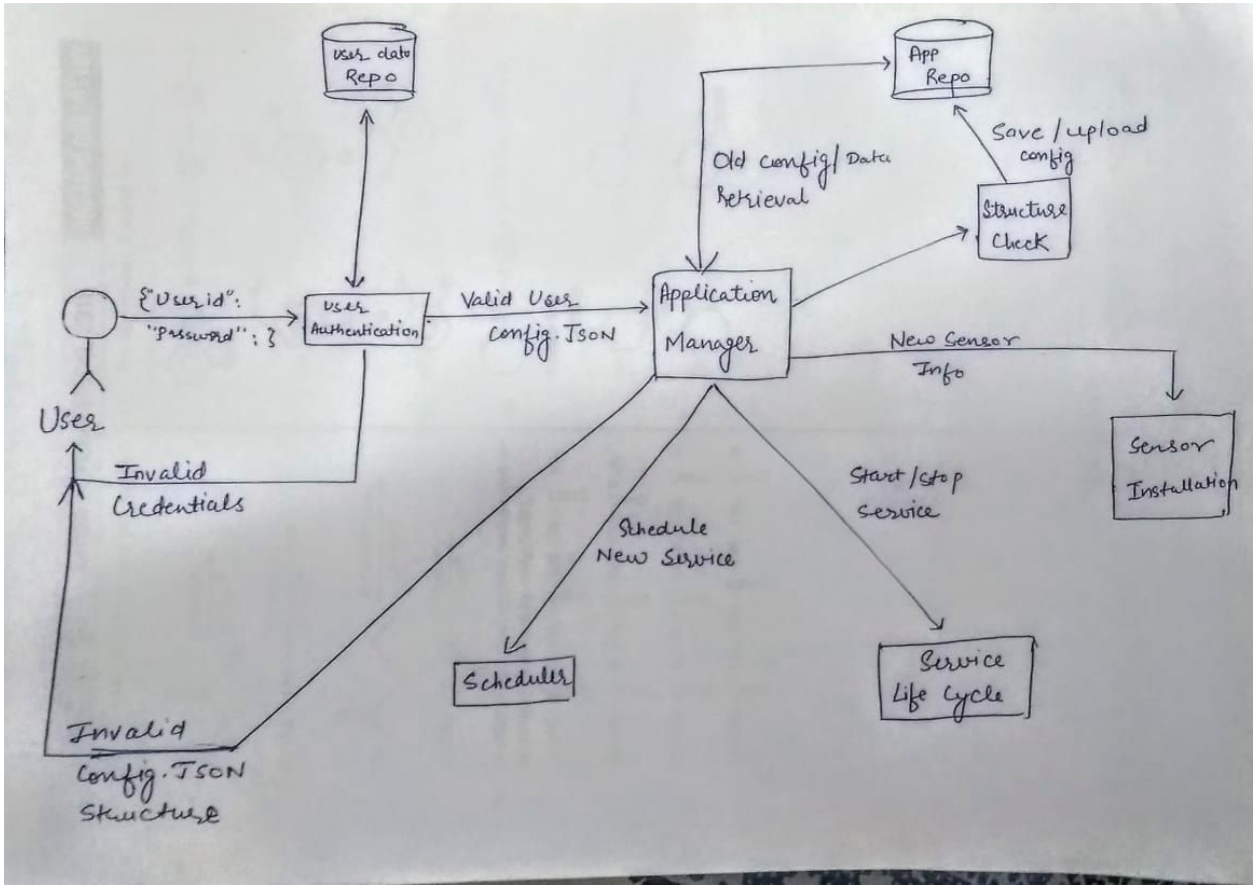
3.2 Technologies to be used:

1. Python framework should be primarily used for the development of the platform.

2. Bash Shell scripts can be used to make installation/configuration automated for deployment of models.
3. Kafka, for communication between different modules
4. Python script for bootstrap.

3.3 System Flow:

1. It starts with user authorization and declaring a set of services deliverable to the user.
2. Then this module will take input from the application manager.
3. Start and end points will be extracted from the configuration file and metadata.
4. Mapping of service with the correct algorithm will be done through meta data received.
5. Requests for service of that particular instance will be sent to the Topological Manager component.
6. The Topological Manager will perform Load balancing and check for distributing load evenly across the processing unit.
7. Notification of the performed service and response will be sent back.



Lifecycle of the Application manager module

1. It will be started by the initial (init) module or monitoring module (like health manager). Init module will start all the major components of the system like the scheduler module, load balancing module and application manager. Init will assign unique ports to each module system to start.

2. Monitoring will be done by the health manager module. It will basically keep sending heartbeat (ping message) to each module in the system to keep track of these modules.

3. Here the user will send the application's zip file to the application manager. This zip will hold algorithms files needed to execute. Along with algorithm files it will send metadata files also, describing priorities, background language, dependencies etc.

4. This metadata xml file’s data will be validated by application manager in order to ensure the client's xml file format is the same as their own defined format or not.

5. Now after validating, it will place all extracted files from zip folder to remote server with the help of nfs file system protocol. This protocol will ensure sharing of files between each module in the system.

6. Depending on the dependencies mentioned in the metadata file, that is, which files to execute before this file , it will start sending details in the form of xml file to the scheduler module .

7. Application manager will send xml file to scheduler module. This xml file will include necessary details of the file to execute. Like start and end time, name of any other file it might need during execution etc. which will help the scheduler to schedule that file to execute. Now the scheduler module will communicate with the load balancer module to assign an idle server to execute that file.

8. Now after assigning a server to execute the file, application manager will see what all executing bodies needed to run this file. It will check if such ide's are available at the assigned server or not. Like to run java code, jdk is available on that server or not. Basically at this step application managers will set certain environment variables in order to ensure successful execution of that file.

9. After setting the environment , the application manager will ping the scheduler to run that particular file on the assigned server.

4. Interactions between sub modules

Interactions will be done via kafka between any modules in the system because kafka can maintain brokers (memory space to store chat) as per 2 modules communicating . These brokers are managed by zookeeper . The ability to handle streaming data extends Kafka's capabilities beyond operating as a messaging system to a streaming data platform.

To top it all off, Apache Kafka provides fairly low latency when using it as a microservices communication bus, even though it incurs an additional network hop for all requests

5. Interactions between other modules

Application manager majorly interacts with scheduler and load balancer.Also it is the one whose work is to get the zip of the client application and extract it.It extracts the zip , checks the semantics of unzipped data(according to defined semantics) and then stores it in some remote server so as load balancer and scheduler can access it.\

It also sets the environment at the server according to the dependencies it gets to know from the zip of the application and make it ready for application to be hosted.It directly don't communicates with load balancer but data it sends to scheduler is sent to load balancer and load balancer accordingly balances the loads and tells where to load the application to scheduler.

3.4 Environment to be used

1. 64-bit OS (Linux)
2. Minimum RAM requirement : 4GB
3. Processor : intel pentium i3

3.5 Registry & Repository:

Registory:

- It will contain user access for each type of user, what all services he is subscribed to.
- It contains a list of gateway and its connected sensors(sensor id, geolocation, Type).

Repository:

- Set of all algorithms which will be mapped according to the service required.
- It will contain configuration files.

3.6 Approach for communication & connectivity:

- Client-server model for communication between various services.
- Kafka is used for communication between different major components.
- Xml file as meta data is passed to process the required request.

3.7 Wire and File Formats:

File Format:

1. Bash scripts - sh format
2. Python files - py format
3. Metadata - json format
4. XML - file format

4.1 Key Data Structures:

- Python Library Data Structures :List, Dictionary, Set.
- HashMap For model and its location storage and its fast retrieval.
- Queue:For message passing.
- JSON: For storing config file in json format.
- XML:For config and the Packaging structure.

5.1 Life cycle of the module

1. A queue is maintained to process a request.

2. A request will be pushed considering its metadata values(ex: priority , time delay etc).

3. Scheduling will be done according to start and end points mentioned in the meta data.

4. Kill the processed instance to make room for the other request to be processed.

6.1 Interaction and Interfaces:

- Interaction for the information to be processed is done using meta data and configuration files.

- An interface where currently processed and to be processed requests will be displayed, and their start and end point will be mentioned.
- Normal User shall be able to see all the deployed and to be deployed Models in the list. In the same way, an admin/special user will be able to see the up and running list of IoT devices and it's information(like IoT device Type, location of the device, capabilities etc.) in the system.