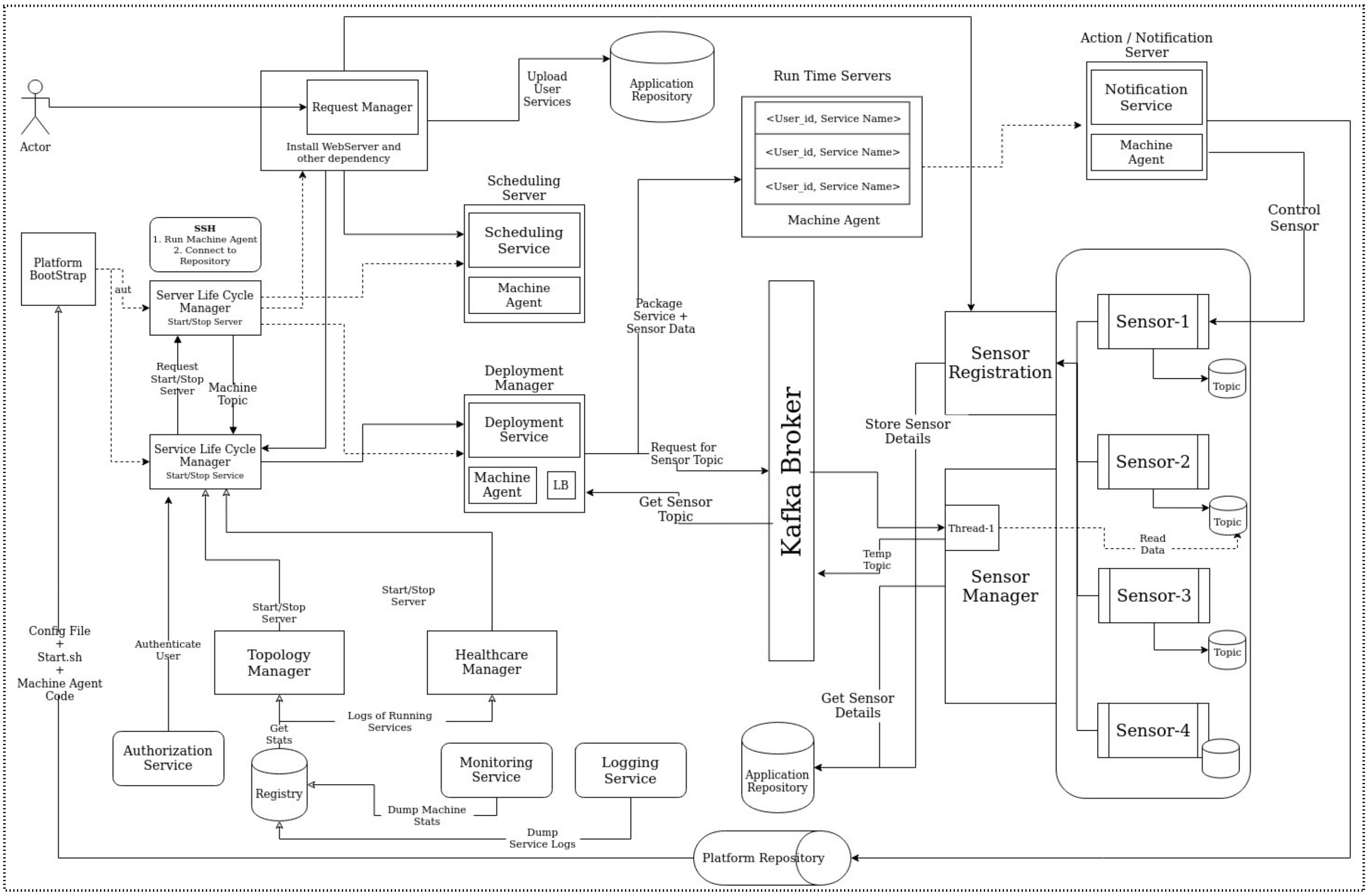
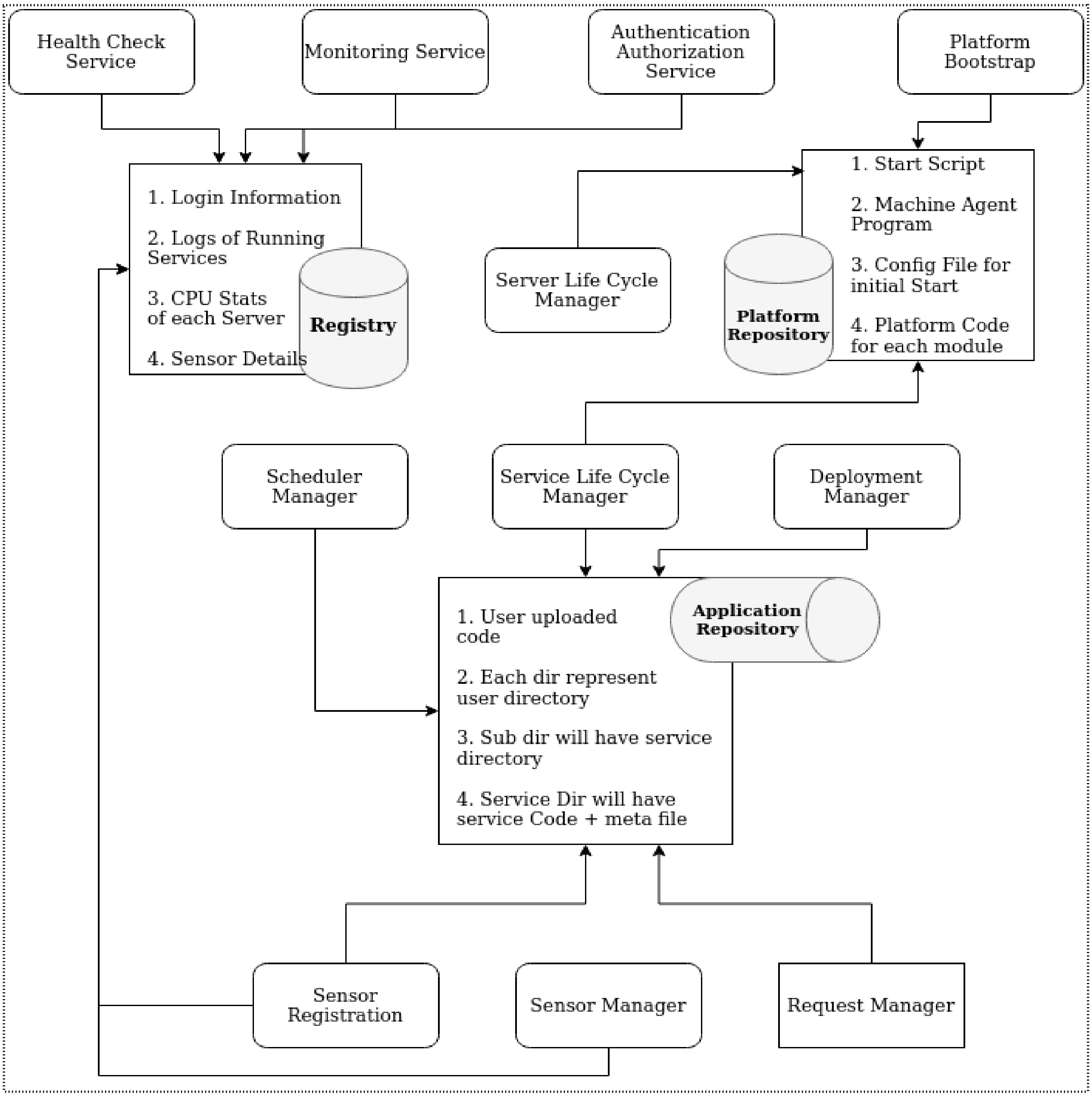
Internals of Application Servers
Project Design
Group-1

Big Picture:

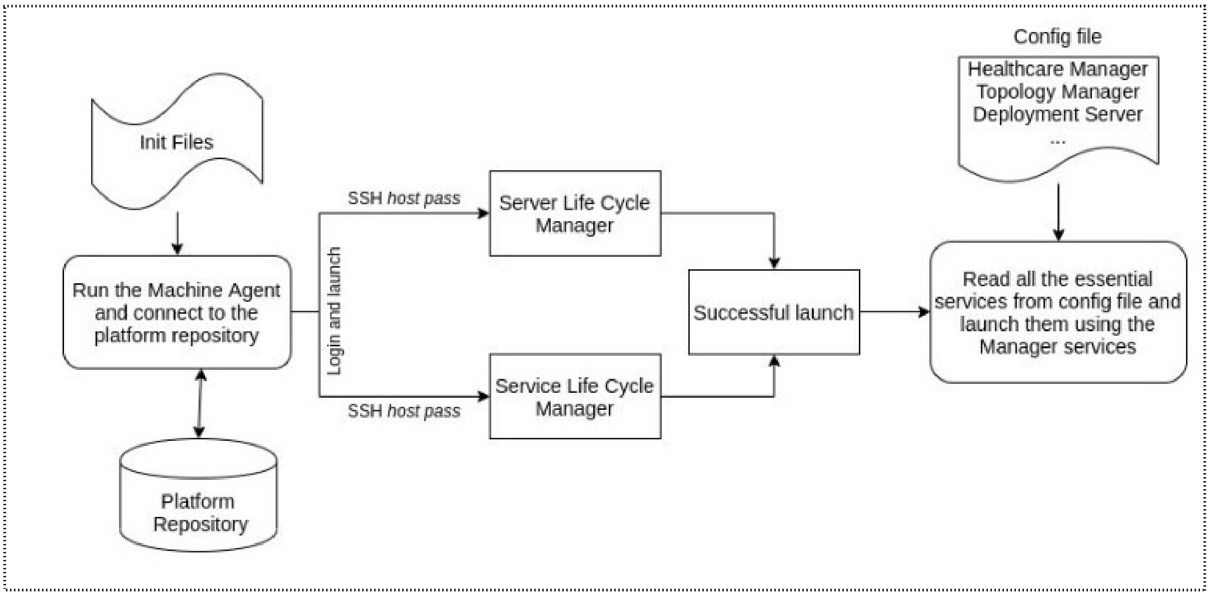


Registry / Repository:


Health Check Service
Monitoring Service
Authentication Authorization Service
Platform Bootstrap
1. Login Information
2. Logs of Running Services
3. CPU Stats of each Server
4. Sensor Details
Registry
Server Life Cycle Manager
Platform Repository
1. Start Script
2. Machine Agent Program
3. Config File for initial Start
4. Platform Code for each module
Scheduler Manager
Service Life Cycle Manager
Deployment Manager
1. User uploaded code
2. Each dir represent user directory
3. Sub dir will have service directory
4. Service Dir will have service Code + meta file
Application Repository
Sensor Registration
Sensor Manager
Request Manager

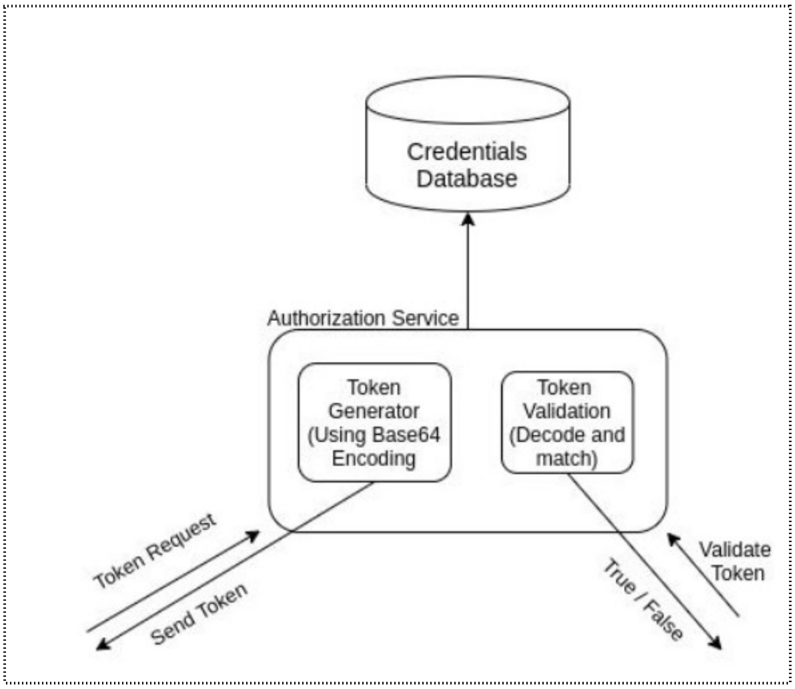
1. Platform Bootstrap

- The bootstrap manager performs all the initialization work. It is the first service launched.
- Bootstrap Manager first runs the machine agent and then connects with the repository.
- Using SSH, it runs the Service Lifecycle Manager and Server Lifecycle Manager services.
- All other required services for the platform to work are stored in the config file as <servicename, params>.
- These services are then launched one by one by the Bootstrap Manager.


Config file
Healthcare Manager
Topology Manager
Deployment Server
...
Init Files
SSH *host pass*
Server Life Cycle Manager
Login and launch
Run the Machine Agent and connect to the platform repository
Successful launch
Read all the essential services from config file and launch them using the Manager services
SSH *host pass*
Service Life Cycle Manager
Platform Repository

2. Authorization Service

- There are 2 tasks for Authorization service.
  1. Generate Token
  2. Validate Token

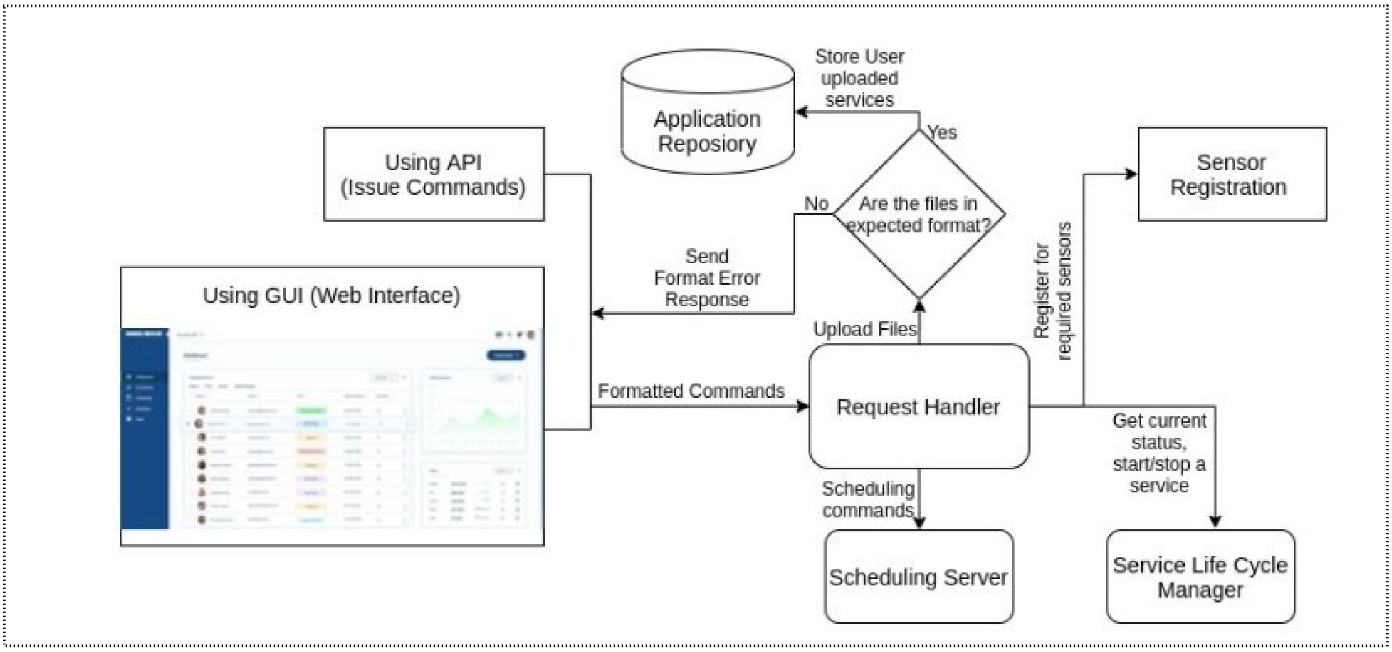


4. Request Manager

- This service accepts user requests, processes it and returns a response to the query.
- The request handler runs on Apache server, expects input commands in a particular format and generates response. Hence,it can be operated using both command-line/GUI as long as they comply with the format.

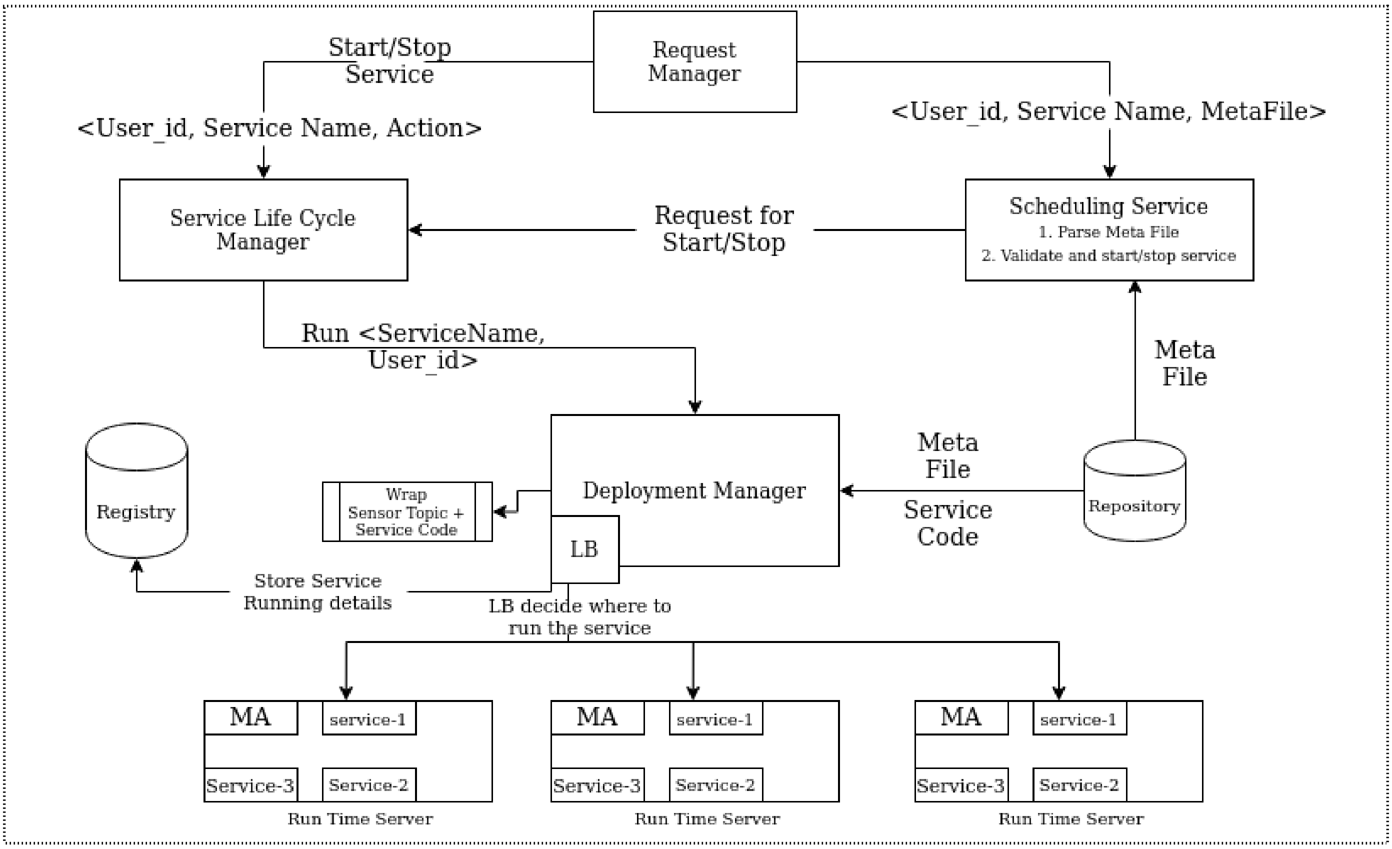
Commands:

○ getFormatForUpload ○ uploadFile ○ listServices ○ startService ○ stopService ○ scheduleService ○ getInfoAboutServie



Application Reposiory
Store User uploaded services
Yes
Using API (Issue Commands)
No
Are the files in expected format?
Sensor Registration
Send Format Error Response
Register for required sensors
Using GUI (Web Interface)
Upload Files
Formatted Commands
Request Handler
Get current status, start/stop a service
Scheduling commands
Scheduling Server
Service Life Cycle Manager

Scheduler and Deployment Manager:

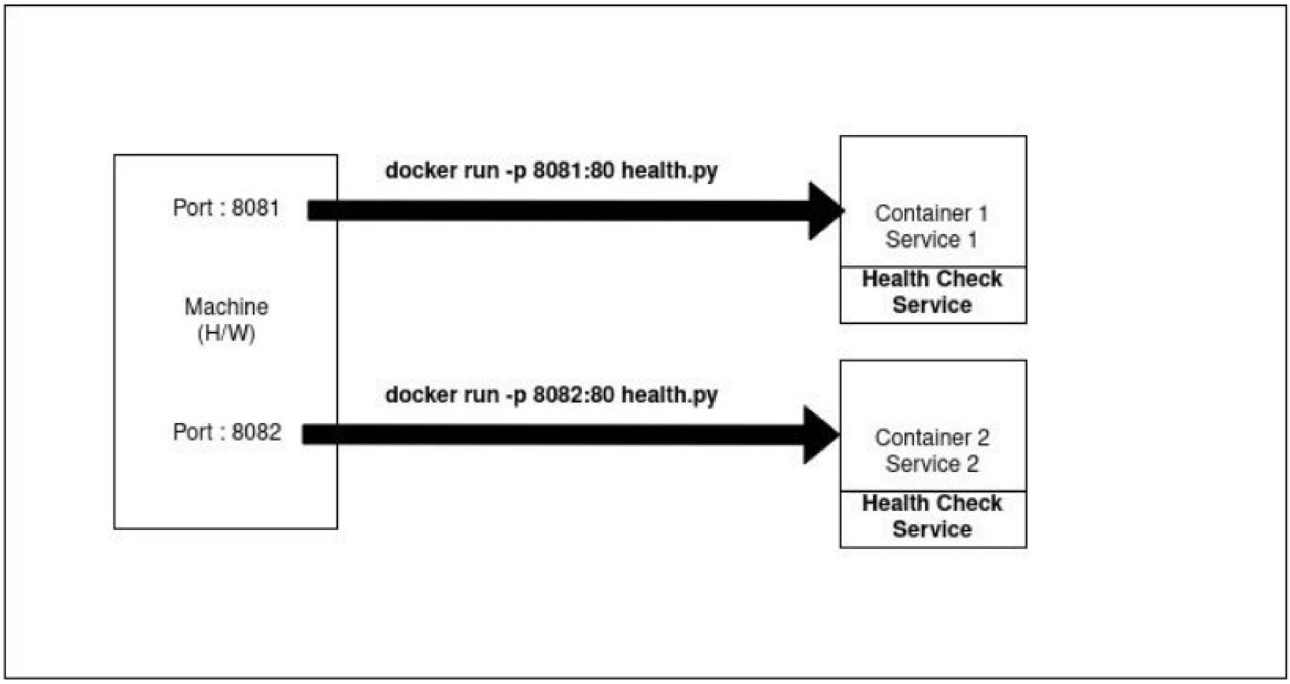


Health Check Service

1. This service is responsible for sending the health status of the service along which it is deployed.
2. It is invoked by a GET request and sends a success code (200 OK) if the service is up and running.
3. In case the service is down, this endpoint will be unreachable resulting in a response code starting with 4XX or 5XX.
4. The topology manager continuously polls this service in order to get the system diagnostic and takes appropriate actions based on the service's response as summarized in the table below.

Requirements to deploy health check:

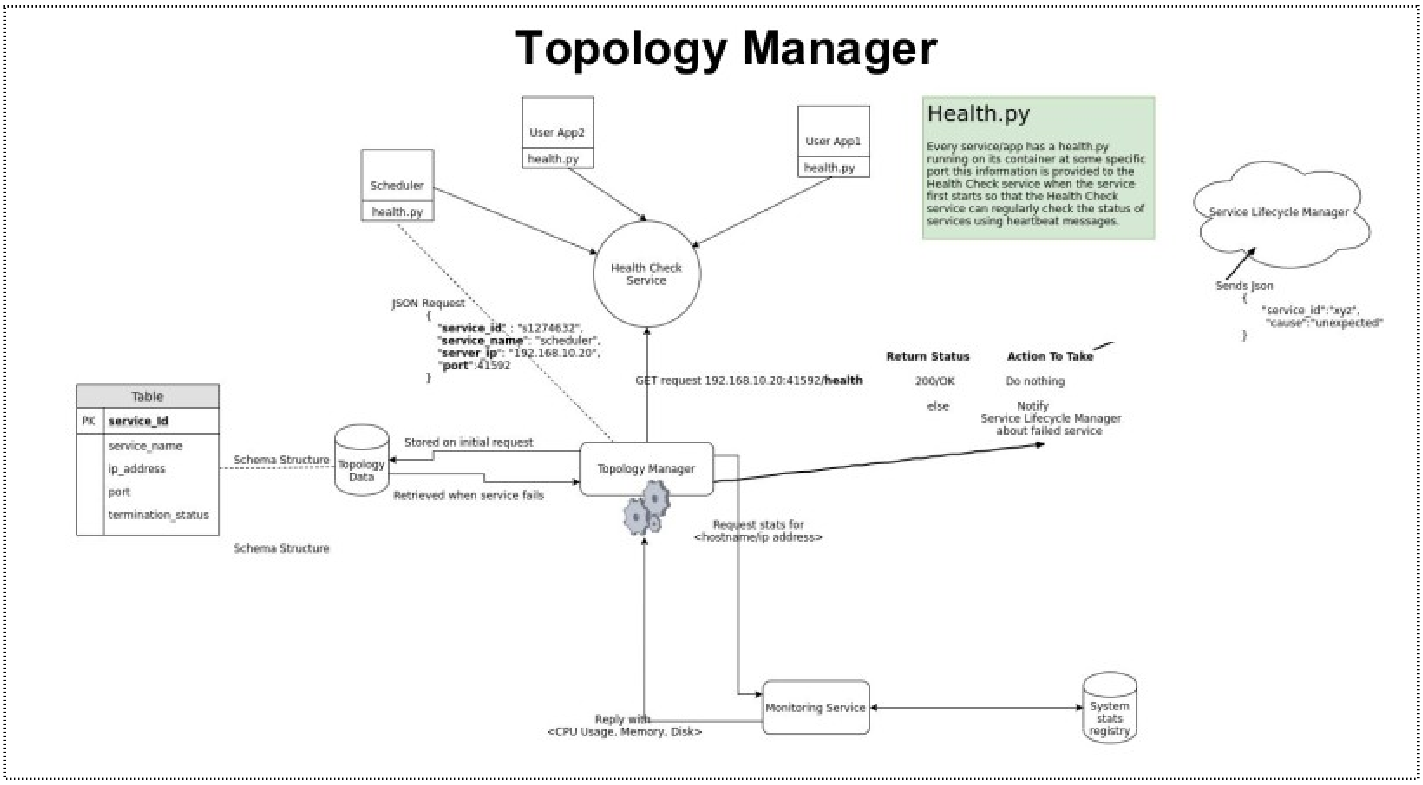
1. Every container runs a ,**health**.py ,script which contains an endpoint ,**http://ip:port/health**
   ip = hostip and port = hostToContainerForwardedPort



An example of how Health Check Service is deployed within each container with port forwarding (**COMPLETE DATA FLOW DIAGRAM BELOW**)

Invoking Health Check Service (Data Flow between Modules)

Response	**Action taken by Topology Manager**
200/OK True	Do nothing
Other	Invoke Service Lifecycle Manager to re-deploy the service.


Topology Manager
User App2
health.py
Scheduler
health.py
User App1
health.py
Health.py
Every service/app has a health.py running on its container at some specific port this information is provided to the Health Check service when the service first starts so that the Health Check service can regularly check the status of services using heartbeat messages.
Service Lifecycle Manager
Sends Json
{
"service_id":"xyz",
"cause":"unexpected"
}
Health Check Service
JSON Request
{
"service_id" : "s1274632",
"service_name": "scheduler",
"server_ip": "192.168.10.20",
"port":41592
}
GET request 192.168.10.20:41592/health
Return Status
Action To Take
200/OK
Do nothing
else
Notify Service Lifecycle Manager about failed service
Table
PK
service_Id
service_name
ip_address
port
termination_status
Schema Structure
Topology Data
Stored on initial request
Retrieved when service fails
Topology Manager
Schema Structure
Request stats for <hostname/ip address>
Reply with <CPU Usage, Memory, Disk>
Monitoring Service
System stats registry

1. Topology Manager has information about the entire platform’s network topology.

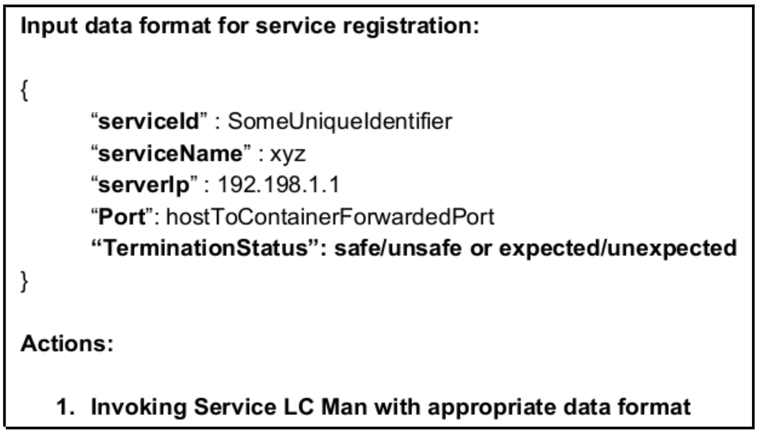
2. Each service(user/platform service) that is deployed on the platform registers itself with the topology manager.
3. After which, Topology Manager gets information about the exact server and port at which the service(user/platform) is deployed. This information is later used by the manager to uniquely identify the service if it needs to be re-deployed.

4. Topology Manager polls the network graph to obtain system diagnostics about each module/service which was previously registered with it by hitting the Health Check service endpoint, as described in the above section.

5. If the health check returns a success code (200 OK true) this means that the service is up and running and nothing needs to be done.

6. In case of an alternate response code, Topology Manager invokes Service Life Cycle Manager to re-deploy the affected service.

7. In addition to this, the topology Manager uses statistics from the Monitoring Module to decide which servers are overloaded and sends a message to server or service lifecycle manager to start instances of servers which exceed high mark.



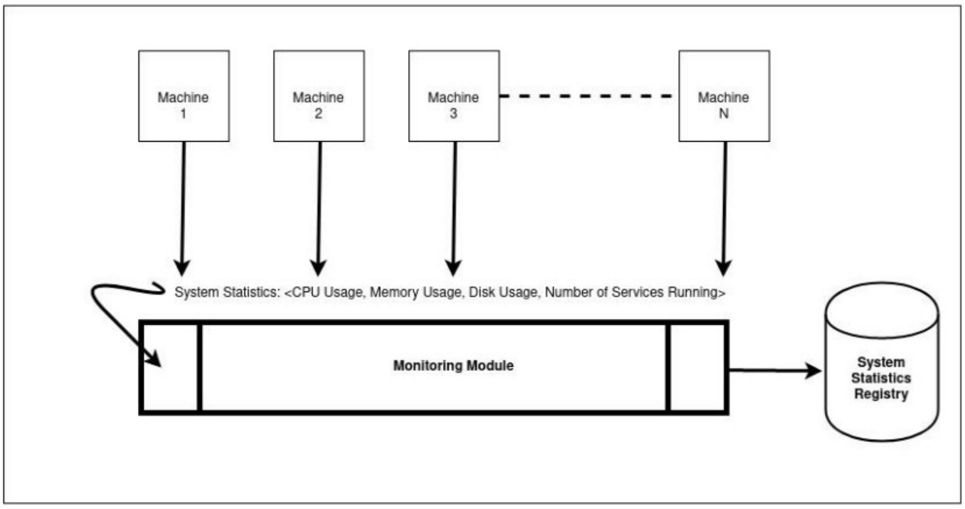
Input data format for service registration:

```
{
        “serviceId” : SomeUniqueIdentifier
        “serviceName” : xyz
        “serverIp” : 192.198.1.1
        “Port”: hostToContainerForwardedPort
        “TerminationStatus”: safe/unsafe or expected/unexpected
}
```

Actions:

1. **Invoking Service LC Man with appropriate data format**

Monitoring Service


Machine 1
Machine 2
Machine 3
Machine N
System Statistics: <CPU Usage, Memory Usage, Disk Usage, Number of Services Running>
Monitoring Module
System Statistics Registry

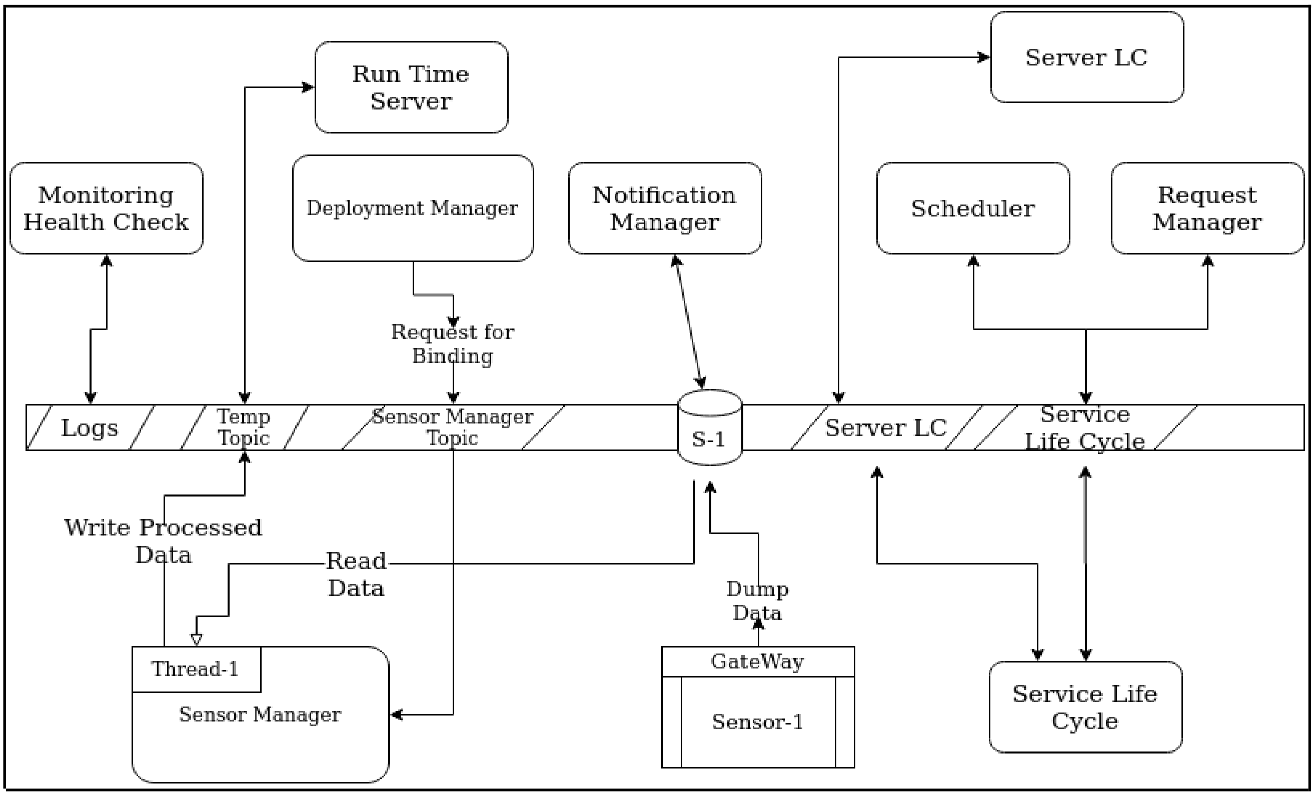
1. The Monitoring Service is responsible for collecting system statistics from all the machines across the platform.

2. This module then converts the data into a pre-fined form and writes it into a registry.

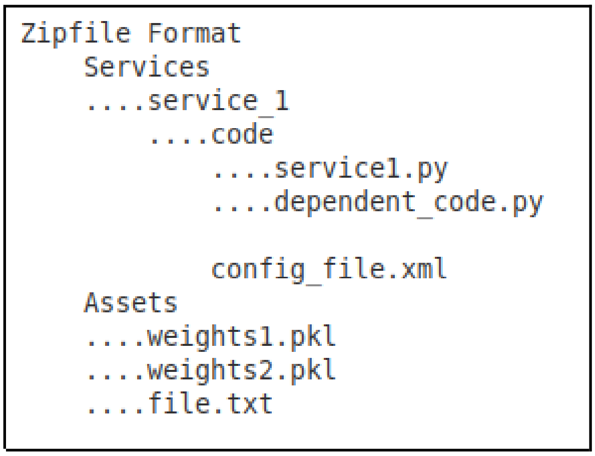
3. Each entry in the registry is of the form : <CPU Usage, Memory Usage, Disk Usage,Number of Services>.

4. This data is then read by the Topology Manager to decide which servers are overloaded and exceeding high mark so that their instances can be deployed again.

Communication Model:

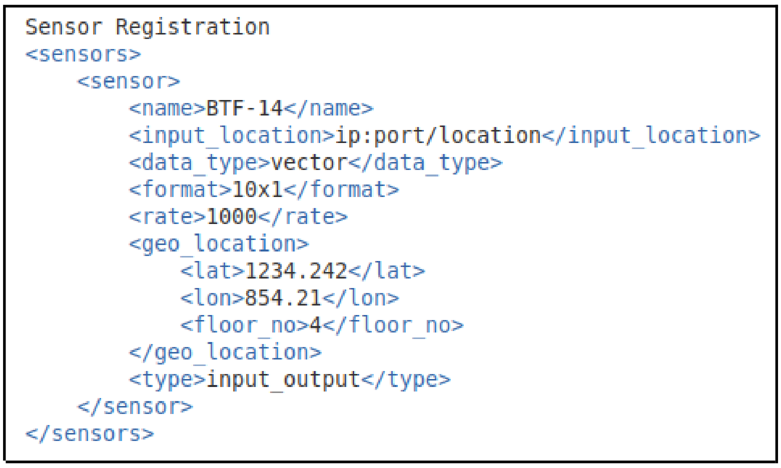


Meta Data Flow:

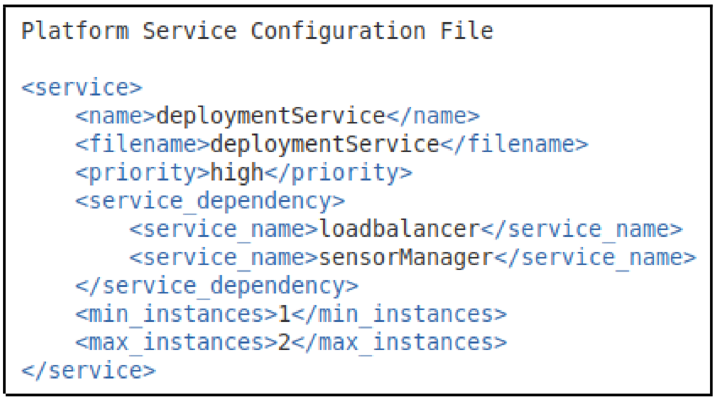


```
Zipfile Format
    Services
    ....service_1
        ....code
            ....service1.py
            ....dependent_code.py

            config_file.xml
    Assets
    ....weights1.pkl
    ....weights2.pkl
    ....file.txt
```

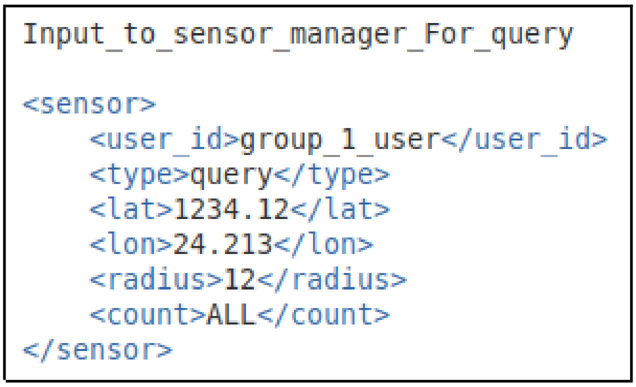


```
Sensor Registration
<sensors>
    <sensor>
        <name>BTF-14</name>
        <input_location>ip:port/location</input_location>
        <data_type>vector</data_type>
        <format>10x1</format>
        <rate>1000</rate>
        <geo_location>
            <lat>1234.242</lat>
            <lon>854.21</lon>
            <floor_no>4</floor_no>
        </geo_location>
        <type>input_output</type>
    </sensor>
</sensors>
```



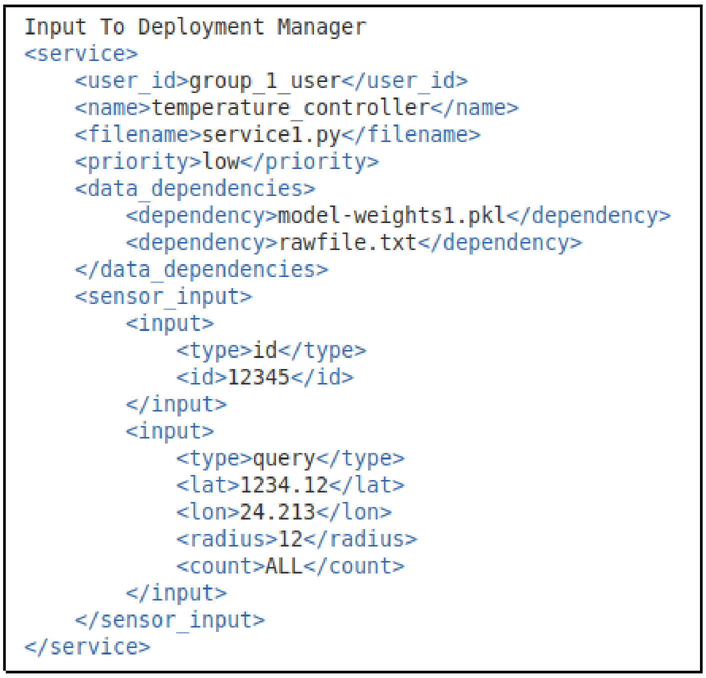
```
Platform Service Configuration File

<service>
    <name>deploymentService</name>
    <filename>deploymentService</filename>
    <priority>high</priority>
    <service_dependency>
        <service_name>loadbalancer</service_name>
        <service_name>sensorManager</service_name>
    </service_dependency>
    <min_instances>1</min_instances>
    <max_instances>2</max_instances>
</service>
```

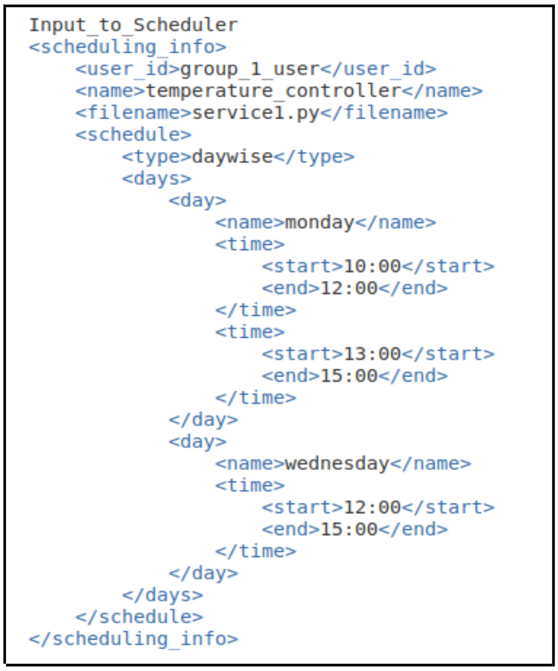


```
Input_to_sensor_manager_For_query

<sensor>
    <user_id>group_1_user</user_id>
    <type>query</type>
    <lat>1234.12</lat>
    <lon>24.213</lon>
    <radius>12</radius>
    <count>ALL</count>
</sensor>
```



```
Input To Deployment Manager
<service>
    <user_id>group_1_user</user_id>
    <name>temperature_controller</name>
    <filename>service1.py</filename>
    <priority>low</priority>
    <data_dependencies>
        <dependency>model-weights1.pkl</dependency>
        <dependency>rawfile.txt</dependency>
    </data_dependencies>
    <sensor_input>
        <input>
            <type>id</type>
            <id>12345</id>
        </input>
        <input>
            <type>query</type>
            <lat>1234.12</lat>
            <lon>24.213</lon>
            <radius>12</radius>
            <count>ALL</count>
        </input>
    </sensor_input>
</service>
```



```
Input_to_Scheduler
<scheduling_info>
    <user_id>group_1_user</user_id>
    <name>temperature_controller</name>
    <filename>service1.py</filename>
    <schedule>
        <type>daywise</type>
        <days>
            <day>
                <name>monday</name>
                <time>
                    <start>10:00</start>
                    <end>12:00</end>
                </time>
                <time>
                    <start>13:00</start>
                    <end>15:00</end>
                </time>
            </day>
            <day>
                <name>wednesday</name>
                <time>
                    <start>12:00</start>
                    <end>15:00</end>
                </time>
            </day>
        </days>
    </schedule>
</scheduling_info>
```

User Service Configuration File

```
<service>
    <name>temperature_controller</name>
    <filename>service1.py</filename>
    <priority>low</priority>
    <data_dependencies>
        <dependency>model-weights1.pkl</dependency>
        <dependency>rawfile.txt</dependency>
    </data_dependencies>
    <sensor_input>
        <input>
            <type>id</type>
            <id>12345</id>
        </input>
        <input>
            <type>query</type>
            <lat>1234.12</lat>
            <lon>24.213</lon>
            <radius>12</radius>
            <count>ALL</count>
        </input>
    </sensor_input>
    <scheduling_info>
        <schedule>
            <type>daywise</type>
            <days>
                <day>
                    <name>monday</name>
                    <time>
                        <start>10:00</start>
                        <end>12:00</end>
                    </time>
                    <time>
                        <start>13:00</start>
                        <end>15:00</end>
                    </time>
                </day>
                <day>
                    <name>wednesday</name>
                    <time>
                        <start>12:00</start>
                        <end>15:00</end>
                    </time>
                </day>
            </days>
        </schedule>
        <schedule>
            <type>period</type>
            <start_date>12-12-2020</start_date>
            <time>
                <start>12:00</start>
                <end>15:00</end>
            </time>
        </schedule>
    </scheduling_info>
<service>
```