PROJECT DESIGN OUTLINE

PREPARED FOR

IAS Project : IIIT-Hyderabad

PREPARED BY

TEAM - Two

[Vatsal Soni - 2018201005]

[Darshan Kansagara - 2018201033]

[Dhawal Jain - 2018201065]

1. Introduction to the Project

Definition: Service component is microservice based architecture of platform that coordinate with all other subsystem and different services. It includes user authentication, model packaging, deployment, and its relevant monitoring logging. It provide runtime for model deployment and invoke user's action and notification. It will also manages scheduling of model on gateway or server with different scheduling policy.

Scope: Our platform provides a set of independent services that the app developer can use for app development with his own custom code updations. The platform provides various independent services like Security service (Authentication and Authorization), Build and Deployment capabilities , Logging and monitoring, Notification and Actions Service, Auto Scaling, Prediction and inference of AI model, Resource management, Scheduling service and repository to store data.

2. Test Cases

2.1 Test cases- [used to test the team's module]

Authentication and Authorisation :

1. **Check Username and password**
   a. Input : username and password
   b. Output: Success or failure
   c. Description : It will verify username and password from DB and return result.

2. **Check User access permission**
   a. Input : Username
   b. Output : User's token along with list of services accessible to user
   c. Description : It check user access permission from repository and return list of services accessible to user.

Scheduling Service :

1. **Check model is up between start and end time**
   a. Input : Start time , Endtime, Model API endpoint.
   b. Output : Check Model is schedule and running up between start and end time.
   c. Description :It will check whether model is up and running between given time. It will ping model in after every time interval.

Wrapping Service:

1. **Check weather model data is packaged properly.**
   a. Input: Model information,
   b. Output: Package model
   c. Description: It will check model information in registry and then get the model data and config file from repository. It will create script file and package these files in one zip.

2. **Check weather package is received by AI run machine.**
   a. Input: Packaged data (Zip file)
   b. Output: Send it to AI run machine
   c. Description: It will send Packaged file to AI run machine where actual model will run.

Notification & Action Service :

1. **Check Action performed or not**
   a. Input : Model API endpoint action code
   b. Output : Check console that action performed and output seen on console or not and notification receive to user registered email/mobile or not.
   c. Description : It will check whether user's action code is executed or not after Model had correctly predicted output.
2. **Check Notification receive or not**
   a. Input : Model API endpoint and prediction output.
   b. Output : Whether user receive notification on register mobile or email or not.
   c. Description : After model got result and it should notify user about it. Checking whether user receive any notification about it or not.

Logging and Monitoring

1. **Check log file Content is updated or not**
   a. Input: Log file path , ServiceName, Logmsg
   b. Output : Log File content change or not
   c. Description : Check whether logmsg given by service is written on log file or not.

2.2 Overall project test cases (relevant to the module)

- **Check New Deployment services instance created or not when high Load**
  a. Input : User Model with High CPU load on deployment service
  b. Output : new deployment service should run and up
  c. Description : When High load in CPU load balancer of Server should create new instance of Deployment service and it should running and up to handle other request.

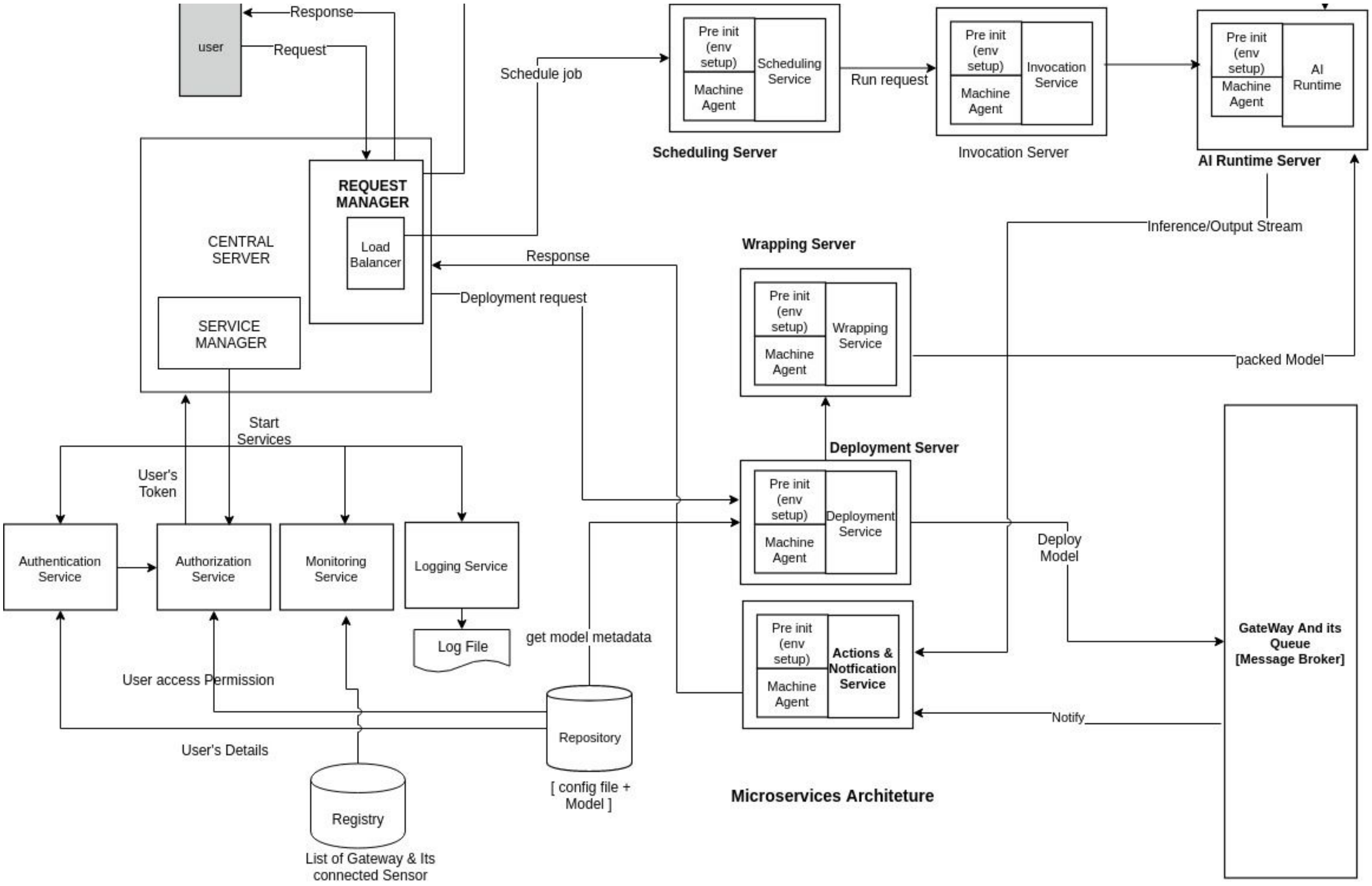
- **Check Model is schedule and running on gateway or not**
  a. Input : Model with start and endtime , Gateway socket
  b. Output : Model must schedule and running up on given gateway.
  c. Description : Given user model should schedule and running up on given gateway and must given endpoint of it.

- **Check User should receive response or not**
    a. Input : Notification and response of request Id
    b. Output : User should gets response
    c. Description : When central server receive response from notification service it should be redirect to user and provide response of it.

- **When service goes down, New service instance should up and running.**
    a. Input: When abruptly service goes down
    b. Output : new service instance should up
    c. Description : When any platform or user service goes down new service instance should up and running by monitoring service.

3. Solution design considerations

3.1 Design big picture



3.2 Environment to be used

- 64-bit OS (Linux)
- Minimum RAM requirement : 4GB
- Processor : intel pentium i3
- Dockerised image used for model transfer.
- SSH support to OS for transferring files and models.

3.3 Technologies to be used

- Microservice Based Architecture - Python Flask based
- Tensorflow serving runtime
- Docker
- Scheduler lib of python
- Bash Shell scripts can be used to make installation/configuration automated for deployment of model.
- NFS : Network File System
- Elastic search, kibana for centralized logging and monitoring

3.4 OVERALL SYSTEM FLOW AND INTERACTION

- As Service Manager starts common services such as authentication, Logging and monitoring will starts and running up.
- When User request comes, Machine agent of central server will create new instance of microservice and Load balancer will distribute load among them.
- Deployment Service takes model as input and config file and pass it to wrapper service.
- Wrapper service will package model along with action, model and script file into zip and passed it to scheduler.
- When User will give config file in which start and end time of deployment of model is mentioned and based on that scheduler will invoke service to deploy model.
- User also able to start/stop its user's services and able to see respective UI page.
- Also We have provide scheduling of user services. Fault tolerance provide to user service when its goes down, its new instance up.
- AI run time provide support to deploy and run model to do inferencing.
- Finally AI model do inferencing and match given use case provided by user and send notification to the Notification service.
- Notification and action service will perform user's action and return back response to the user.

3.5 Approach for communication & connectivity

- We used Rest API endpoint for inter-service communication.
- RabbitMQ used for queueing and interaction between various component.
- We used client-server model for communication between various services.

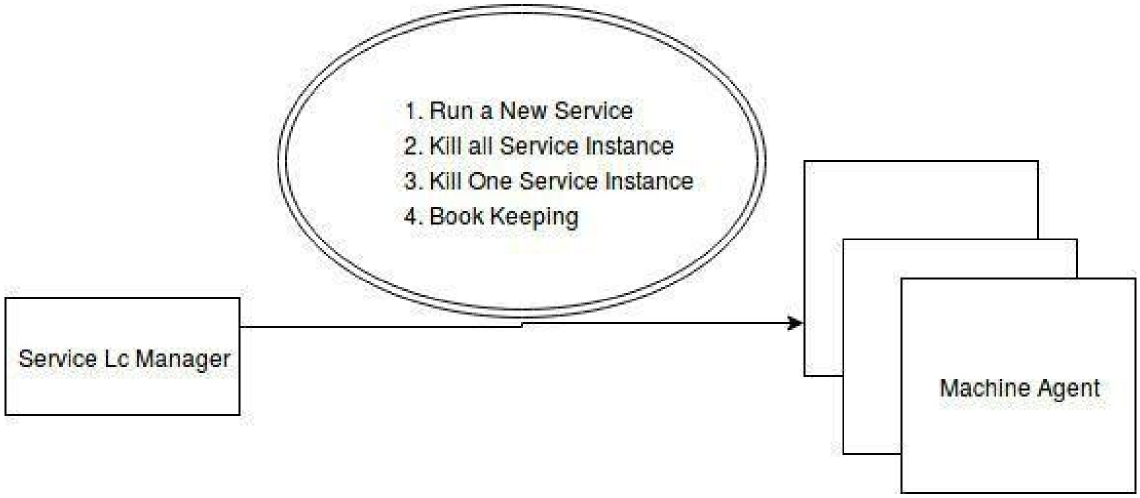
- NFS used for common platform for data storage

3.6 Registry & repository

- **Registry**
  - It will contain the stats of every service (what instances are running on what machines, binary files location of each service in the common file system etc.)
  - It contains list of gateway and and its connected sensors(sensor id, geolocation, Type).

- **Repository:**
  - It will contain one directory per user which will in-turn contain one directory per service and will contain all the version-revisions of that service.
  - It also contains config file and saved model.

3.7 Service lifecycle Manager

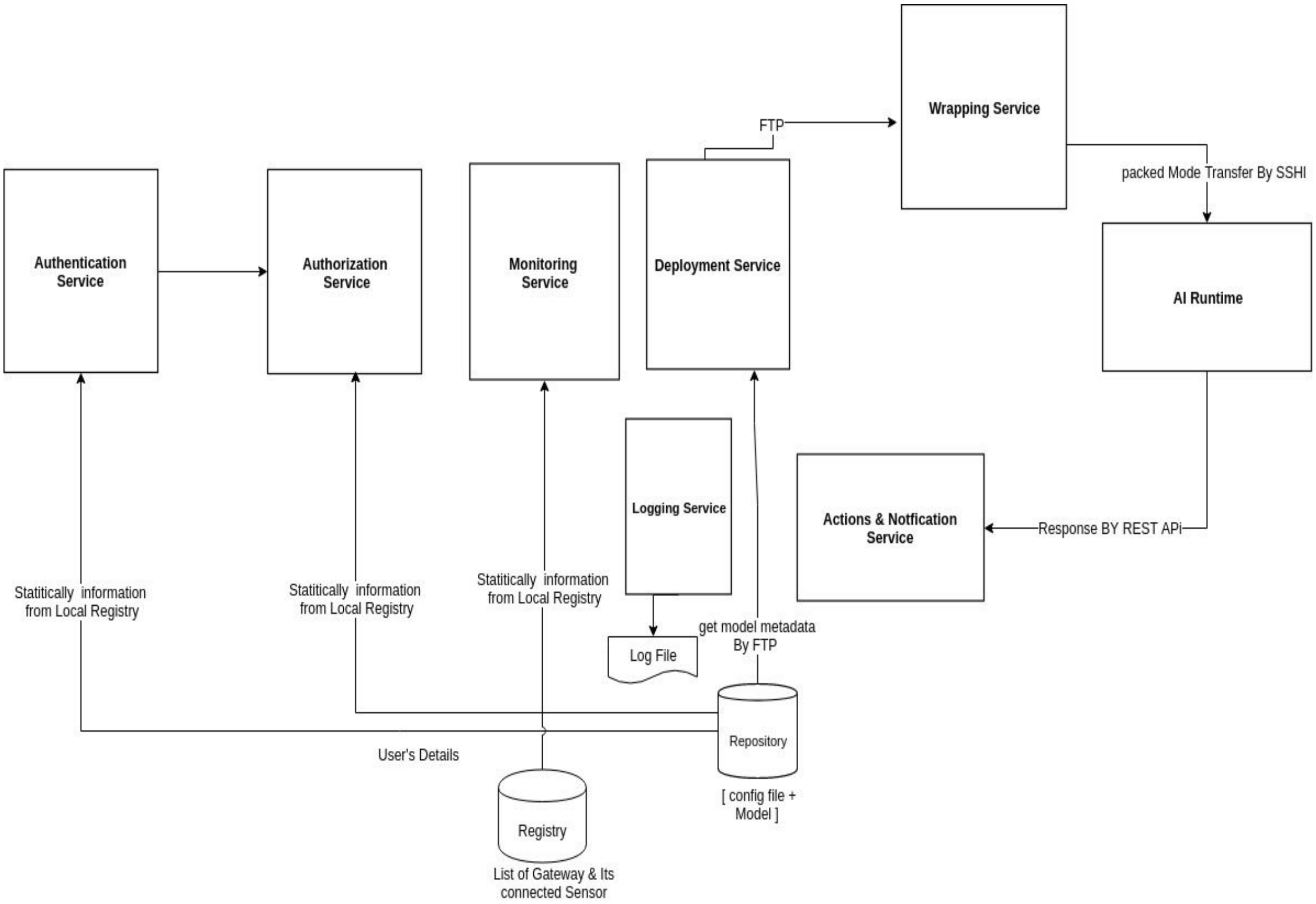


Main functionalities of this module are as follows :

1. **Run a Service instance:** Gather stats from monitoring module and based on the stats, start a service in a docker container on that machine.
2. **Kill all Service instances running on single machine input:(ipaddress, hostname)** Get info about services running on that machine from registry. Then kill those service containers.
3. **Kill a single Service instance input:(ipaddress, hostname, serviceid** Kill that service instance from docker container using input.

3.8 Interactions between modules

- Service life cycle is responsible for invoking services. Each service is itself a micro service which runs on different-different server instances. Through rest api one service will interact with another service. Also AI model transfer will be done by network file system. Through ssh, invocation service service run AI model on AI run machine. Run model service will interact with input stream device through RabbitMQ.


Wrapping Service
FTP
packed Mode Transfer By SSHI
Authentication Service
Authorization Service
Monitoring Service
Deployment Service
AI Runtime
Logging Service
Actions & Notfication Service
Response BY REST APi
Statitically information from Local Registry
Statitically information from Local Registry
Statitically information from Local Registry
get model metadata By FTP
Log File
Repository
[config file + Model]
User's Details
Registry
List of Gateway & Its connected Sensor

3.9 Wire and file formats

a. REST/GRPC to communicate between TF Serving APIs and server
b. AMQP to communicate between IoT Sub system and Server / AI system
c. REST APIs carrying JSON data to communicate between client and server

d. Each message sent to a MQ will have an authentication token in the msg header. Message body will contain the svc name, function name and its parameters along with their data types. We will also have an identifier(correlation id) for every message and will be included as key of that message in the MQ.

File formats:

Multiple files with different file formats will be used.

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

4. User's view of system

4.1 Application As per User :

- We had taken two Application scenario 1)IRIS and SONAR model use case 2)Object detection use case.
- User can able to deploy its whole application on our platform , our platform provides abstraction to user for scheduling , deployment, start and stop its user services.
- User will see its model status(running, abnormal, no) and service status. It will also show display button to see UI for user's services.

4.2 File Used :

- User will upload its application config files such as app.config
- User will provide runtime.config file for model scheduling and deployment.
- User will also provide service config file when he/she want's to start its services.

4.3 Setup :

- User machine have system that support NFS, python, Shell, RabbitMQ support.
- User have to change configuration of all config file related to its application. Such as app.config, deploy.config etc...
- User have to provide list of gateways and sensor used by its application. So all these gateways and sensor will be in working status.
- All required machine to run user's application which be ready , dependency will be installed by machine agent to start any application services.

4.4 start/stop services Mechanism :

- After application user had deploys its application , app user will able to see its list of services and its start and stop button, so at any point of time user will able to start and stop its services.

4.5 Accessing/Invoking services Mechanism :

- User will able to get display button when user's services started, So user will click on it to get UI interface of that services.

4.6 Web Service :

- Any kind of user will have access to platform and its services through a flask web app.

- It can upload a model through it and create a config file corresponding to that and has a facility to upload it along with the corresponding models.
- User can also see the status of all running models on its dashboard and the status of all the gateway on the same web page.

5. Key Data structures

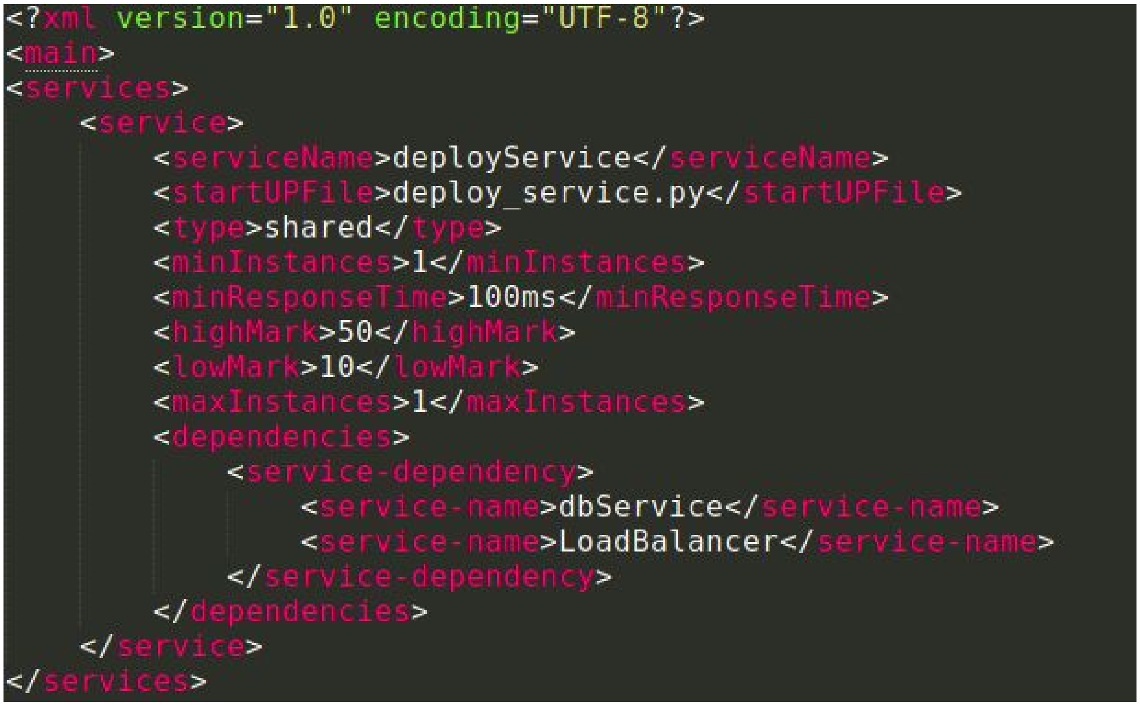
Below are the list of data structures we have used in the application development :

- Python Library Data Structures :List, Dictionary, Set.
- Special Data Structure:
    - 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.
- **APIs**
    - Authentication Service:
        - /user_authenticate : It will taking input as username and password and authenticate user from DB and return success or failure result.
    - Deployment Service
        - /deployService : It will handle deployment request and package model with start and stop script and setup runtime environment.
    - Scheduling Service
        - /ScheduleService : it will schedule model with different scheduling policies and start and stop model at specified time.
    - Notification & Action Service
        - /notification : It will notify user based when inference is done and found result for user. And sent email to user about the same.
    - Logging Service
        - /loggingService : It will make logs of all server events with respect to serviceName.
    - Monitoring Service
        - /monitoring/model_status : It will show model status like model is down, up , running, etc..
        - /monitoring/service_status : It will showing status of different gateways and its health information
    - Inferencing Service

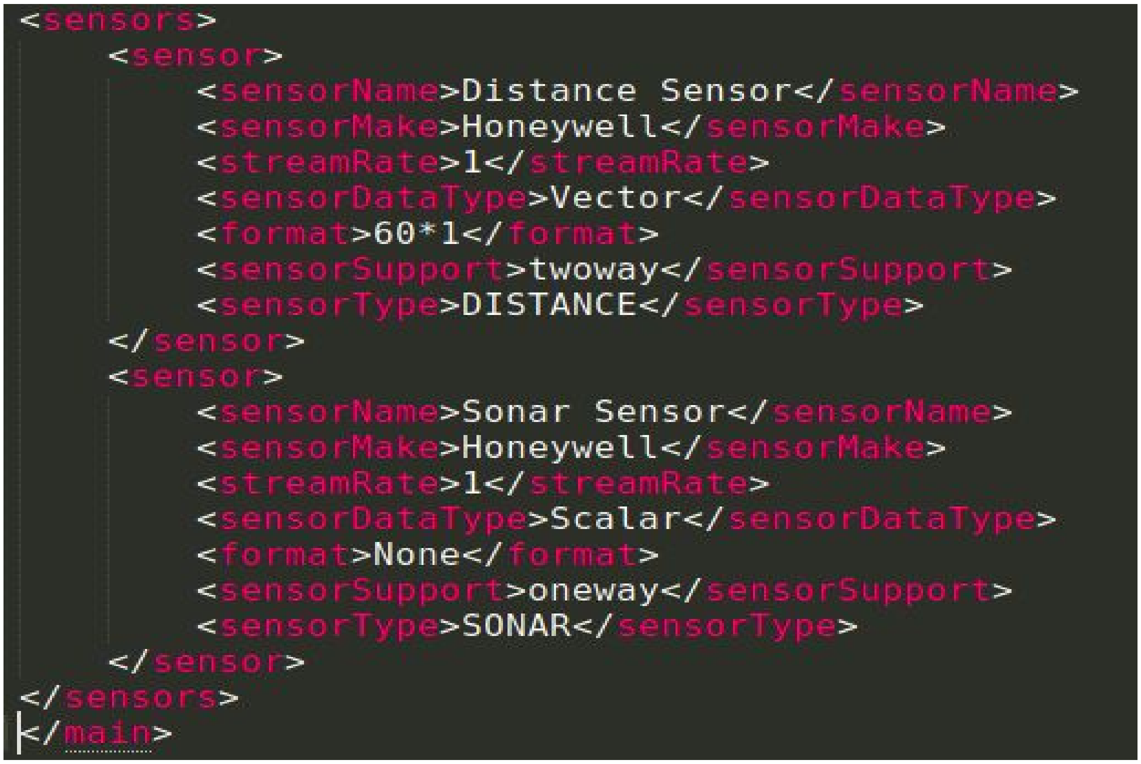
- /inferenceSerivce : It will start inferencing and make prediction on model and store predection output.

Storage structure in the form of XML

Platform

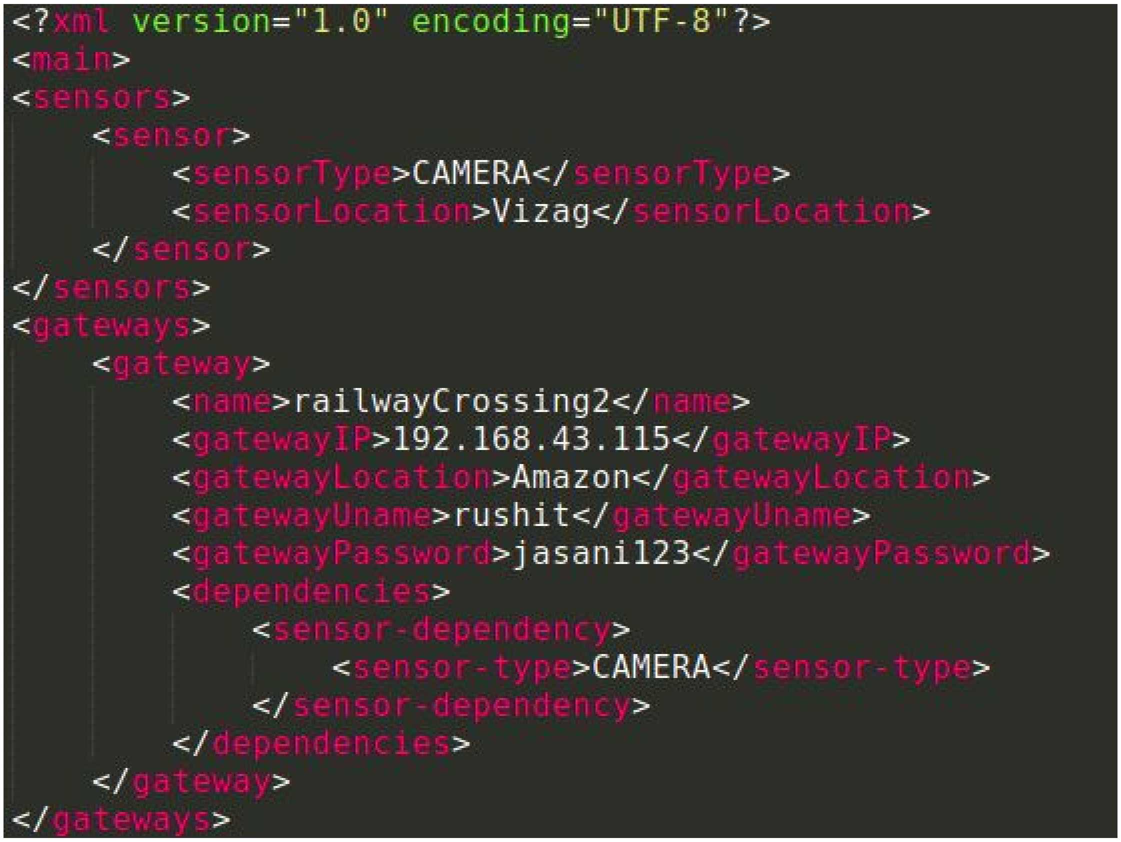


```
<?xml version="1.0" encoding="UTF-8"?>
<main>
<services>
    <service>
        <serviceName>deployService</serviceName>
        <startUPFile>deploy_service.py</startUPFile>
        <type>shared</type>
        <minInstances>1</minInstances>
        <minResponseTime>100ms</minResponseTime>
        <highMark>50</highMark>
        <lowMark>10</lowMark>
        <maxInstances>1</maxInstances>
        <dependencies>
            <service-dependency>
                <service-name>dbService</service-name>
                <service-name>LoadBalancer</service-name>
            </service-dependency>
        </dependencies>
    </service>
</services>
```

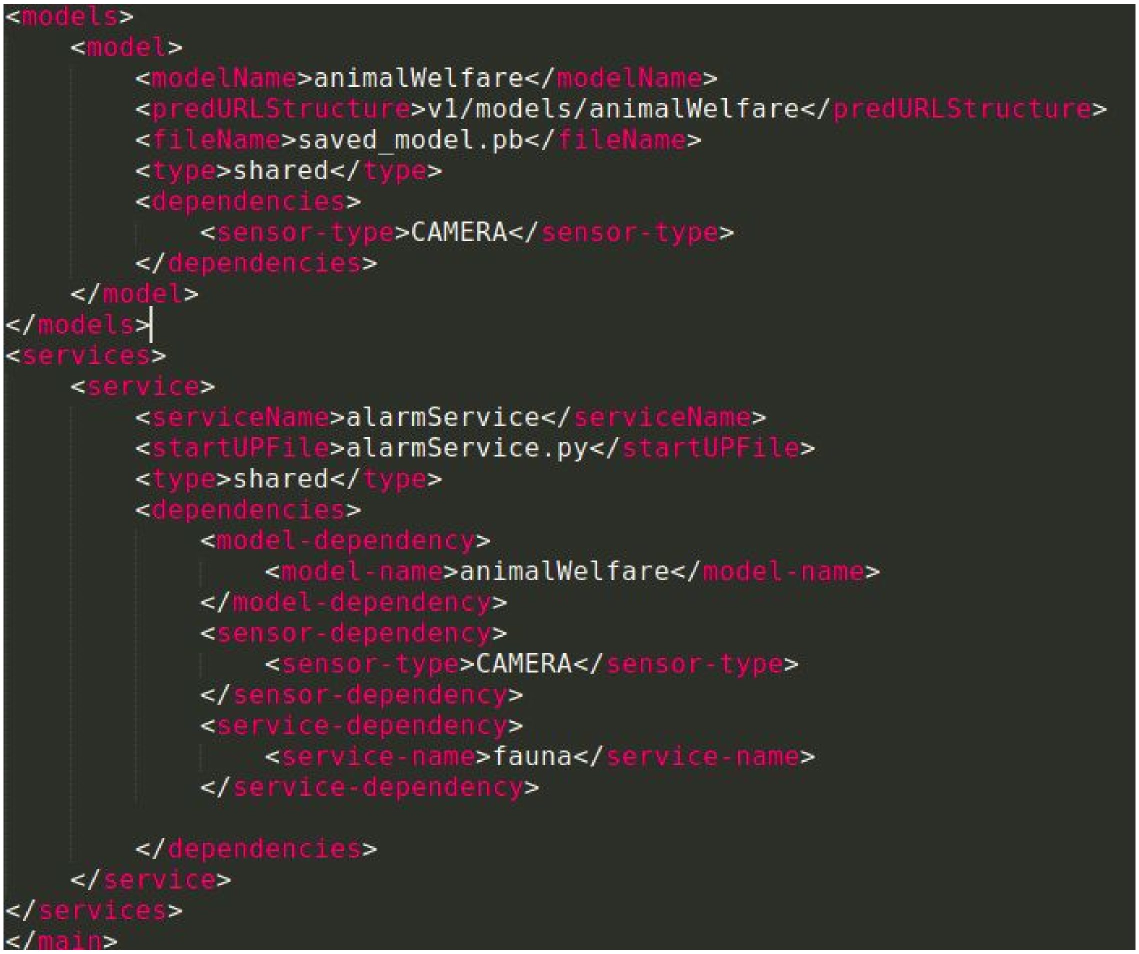


```
<sensors>
    <sensor>
        <sensorName>Distance Sensor</sensorName>
        <sensorMake>Honeywell</sensorMake>
        <streamRate>1</streamRate>
        <sensorDataType>Vector</sensorDataType>
        <format>60*1</format>
        <sensorSupport>twoway</sensorSupport>
        <sensorType>DISTANCE</sensorType>
    </sensor>
    <sensor>
        <sensorName>Sonar Sensor</sensorName>
        <sensorMake>Honeywell</sensorMake>
        <streamRate>1</streamRate>
        <sensorDataType>Scalar</sensorDataType>
        <format>None</format>
        <sensorSupport>oneway</sensorSupport>
        <sensorType>SONAR</sensorType>
    </sensor>
</sensors>
</main>
```

Deploy



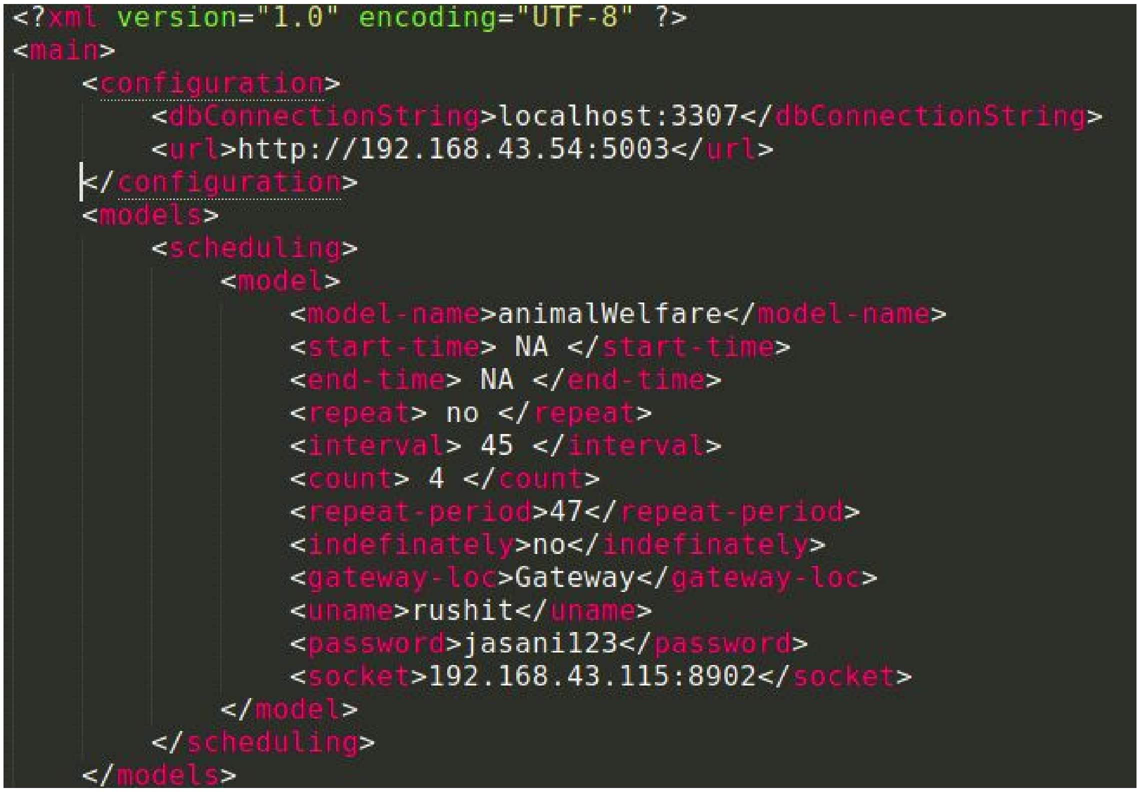
```
<?xml version="1.0" encoding="UTF-8"?>
<main>
<sensors>
    <sensor>
        <sensorType>CAMERA</sensorType>
        <sensorLocation>Vizag</sensorLocation>
    </sensor>
</sensors>
<gateways>
    <gateway>
        <name>railwayCrossing2</name>
        <gatewayIP>192.168.43.115</gatewayIP>
        <gatewayLocation>Amazon</gatewayLocation>
        <gatewayUname>rushit</gatewayUname>
        <gatewayPassword>jasani123</gatewayPassword>
        <dependencies>
            <sensor-dependency>
                <sensor-type>CAMERA</sensor-type>
            </sensor-dependency>
        </dependencies>
    </gateway>
</gateways>
```



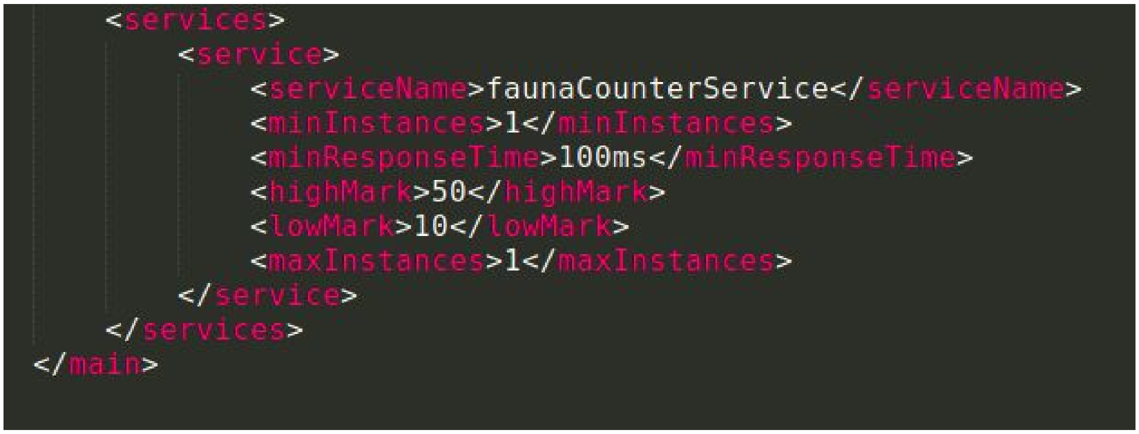
```
<models>
    <model>
        <modelName>animalWelfare</modelName>
        <predURLStructure>v1/models/animalWelfare</predURLStructure>
        <fileName>saved_model.pb</fileName>
        <type>shared</type>
        <dependencies>
            <sensor-type>CAMERA</sensor-type>
        </dependencies>
    </model>
</models>
<services>
    <service>
        <serviceName>alarmService</serviceName>
        <startUPFile>alarmService.py</startUPFile>
        <type>shared</type>
        <dependencies>
            <model-dependency>
                <model-name>animalWelfare</model-name>
            </model-dependency>
            <sensor-dependency>
                <sensor-type>CAMERA</sensor-type>
            </sensor-dependency>
            <service-dependency>
                <service-name>fauna</service-name>
            </service-dependency>

        </dependencies>
    </service>
</services>
</main>
```

Runtime



```
<?xml version="1.0" encoding="UTF-8" ?>
<main>
    <configuration>
        <dbConnectionString>localhost:3307</dbConnectionString>
        <url>http://192.168.43.54:5003</url>
    </configuration>
    <models>
        <scheduling>
            <model>
                <model-name>animalWelfare</model-name>
                <start-time> NA </start-time>
                <end-time> NA </end-time>
                <repeat> no </repeat>
                <interval> 45 </interval>
                <count> 4 </count>
                <repeat-period>47</repeat-period>
                <indefinately>no</indefinately>
                <gateway-loc>Gateway</gateway-loc>
                <uname>rushit</uname>
                <password>jasani123</password>
                <socket>192.168.43.115:8902</socket>
            </model>
        </scheduling>
    </models>
```



```
     <services>
         <service>
             <serviceName>faunaCounterService</serviceName>
             <minInstances>1</minInstances>
             <minResponseTime>100ms</minResponseTime>
             <highMark>50</highMark>
             <lowMark>10</lowMark>
             <maxInstances>1</maxInstances>
         </service>
     </services>
 </main>
```

6. Interactions & Interfaces

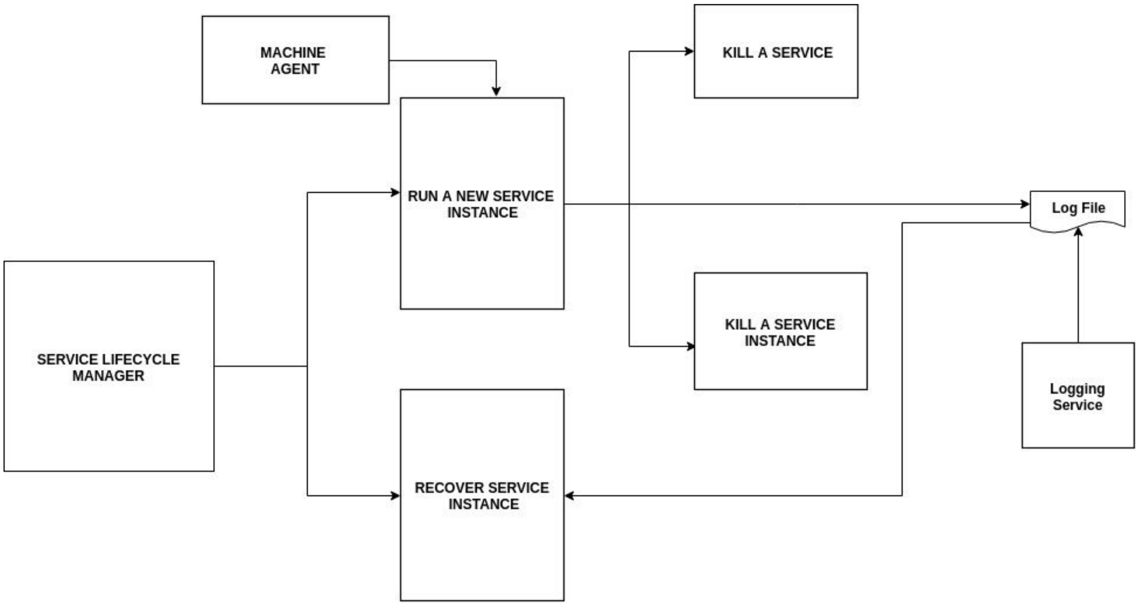
- The first interaction of the both the user(one uploading model and other giving use case) with the system is with web service(we have used flask) with the help of which it can send its request for model deploy or can send a use case for the model that are already deployed.
- User can also see all the deployed and to be deployed Models in the list on the dashboard or say UI interface.
- All the file transfer is done using secure shell.
- An html page where user shall be able to submit a Machine Learning Model for the deployment. User shall be able to give the details like, On premises or On Cloud deployed type, type of AI/IoT it has to deal with, input and output format/type etc.

7. Persistence

- We maintain persistent log file for continuous event triggering.
- We used python pickle library for storage and tensorflow model saver for saving model into repository.
- In case one microservice is down ,even than rest of the system will remain entact.
- In case system crashes, these services brings back the system in the last state with the help of logging.

Low Level Design

1. Lifecycle of the Service module



2. List the sub modules

2.1 Logging

2.2 Monitoring

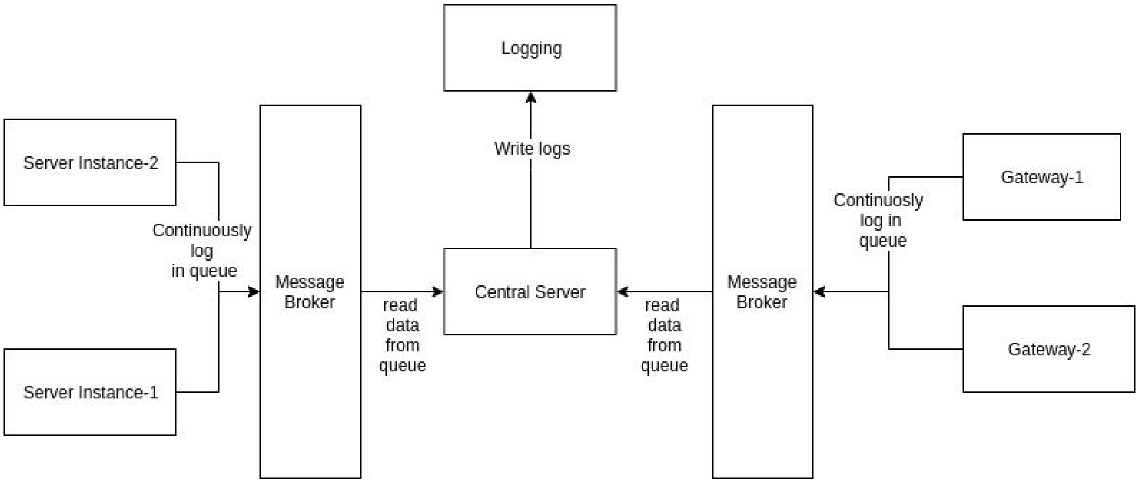
2.3 Deployment

2.4 Notifications and Actions

2.5 Security

3. Brief overview of each sub module

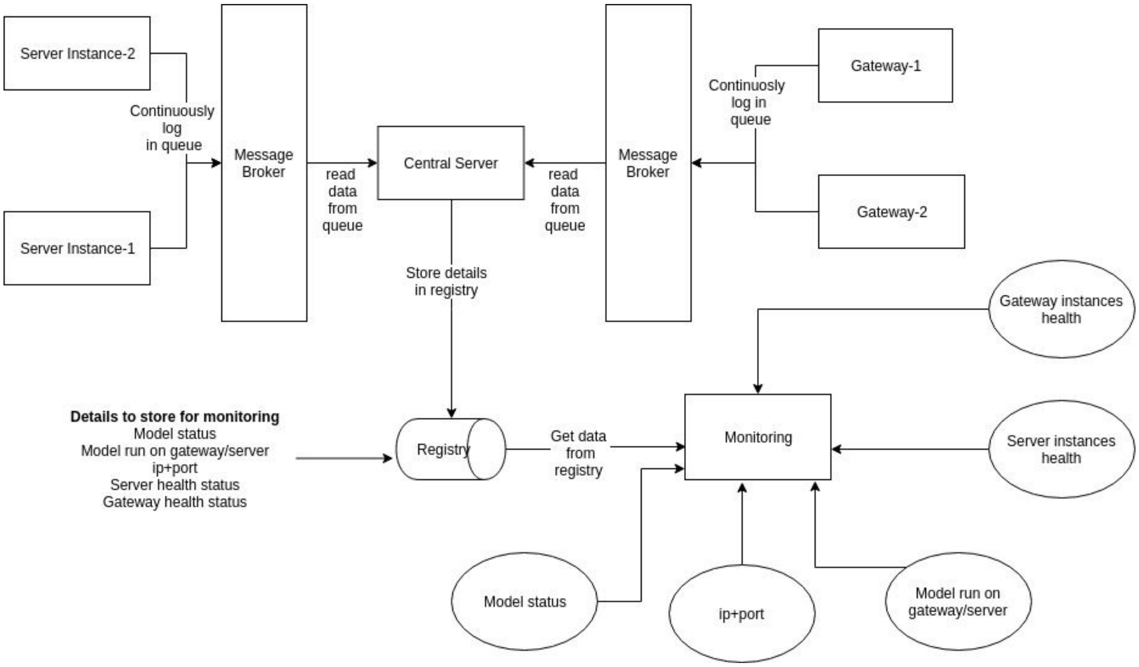
3.1 Logging service



Main functionalities of logging are as follows:

- Logging all the activities to manage resources.
- To find root cause of crash of our application platform by writing error and debugging logs.
- Search and download log.
- It will access the log file and pinpoint the last event that was triggered and take appropriate action.

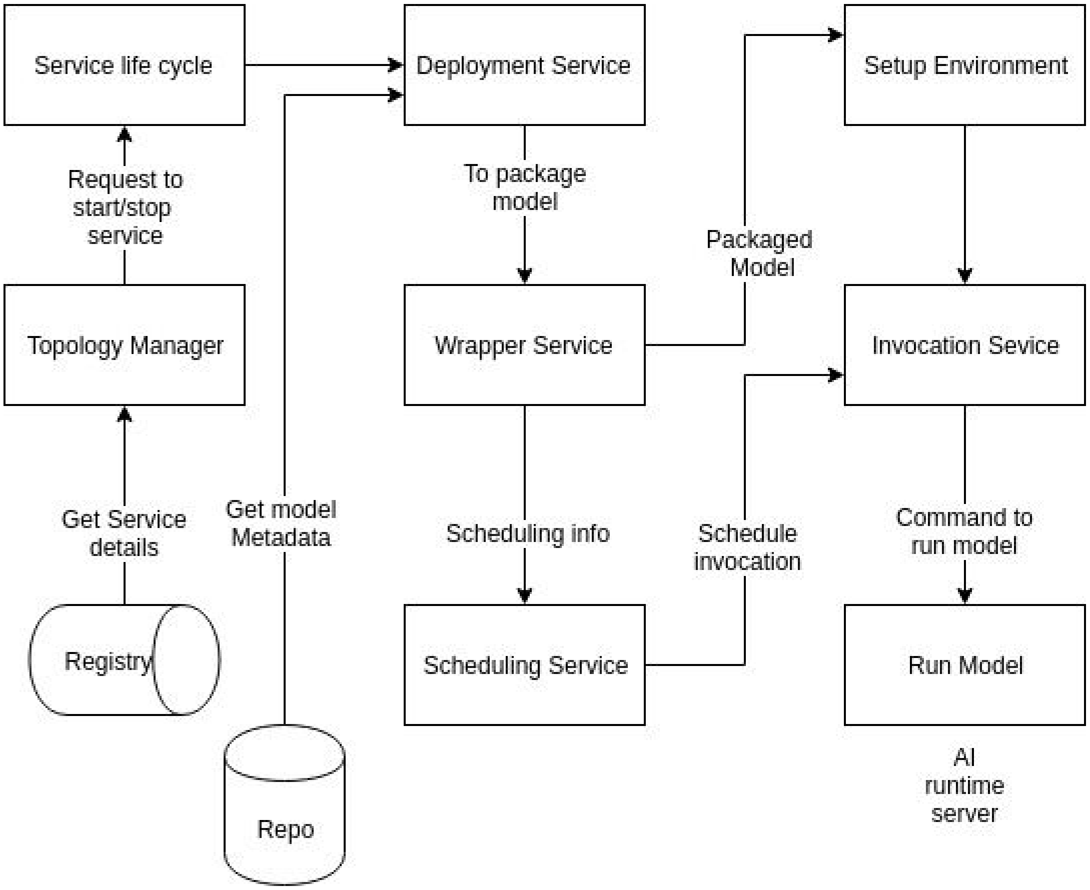
3.2 Monitoring service



Main functionalities of monitoring are as follows:

- It monitors the health of all the server instances and gateways.
- It helps in load balancing task because it keeps track of all the server instances.
- It continuously updates the registry to store dynamic information which helps topological manager.

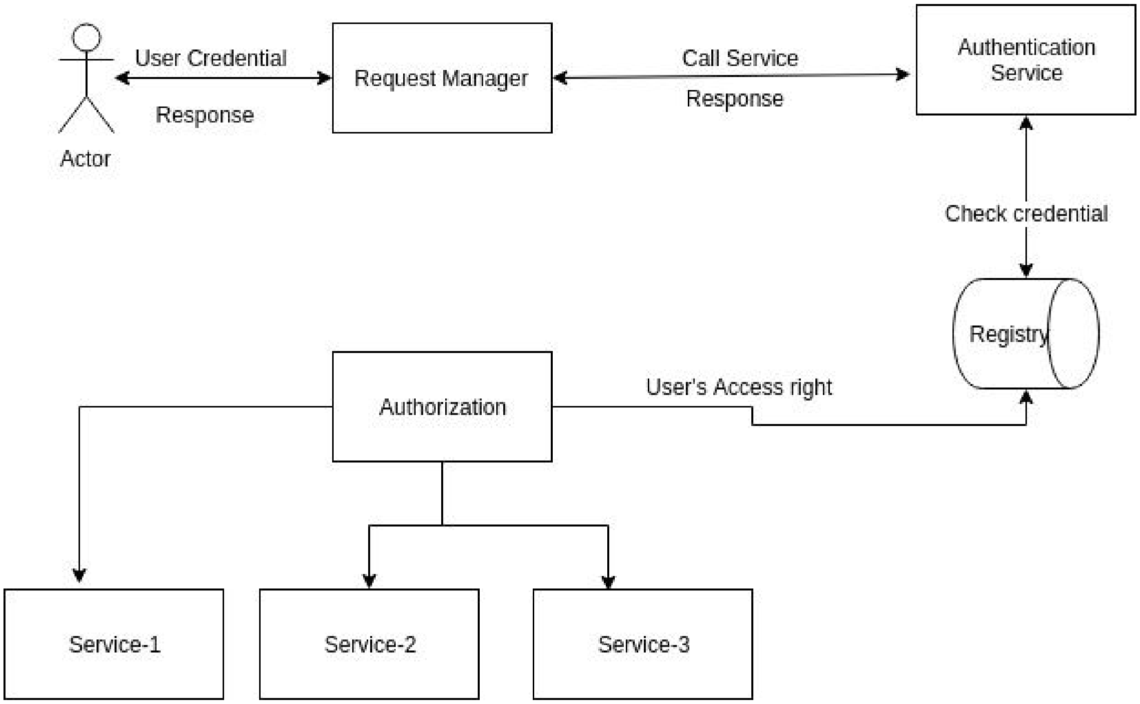
3.3 Deployment service



Main functionalities of deployment services are as follows:

- It helps in deploying AI models in gateways and server instances.
- Inferencing can be done using deployment service only.
- Deployment service is responsible for scheduling of AI models.
- It is responsible for setting up runtime environment in server instances/gateways.

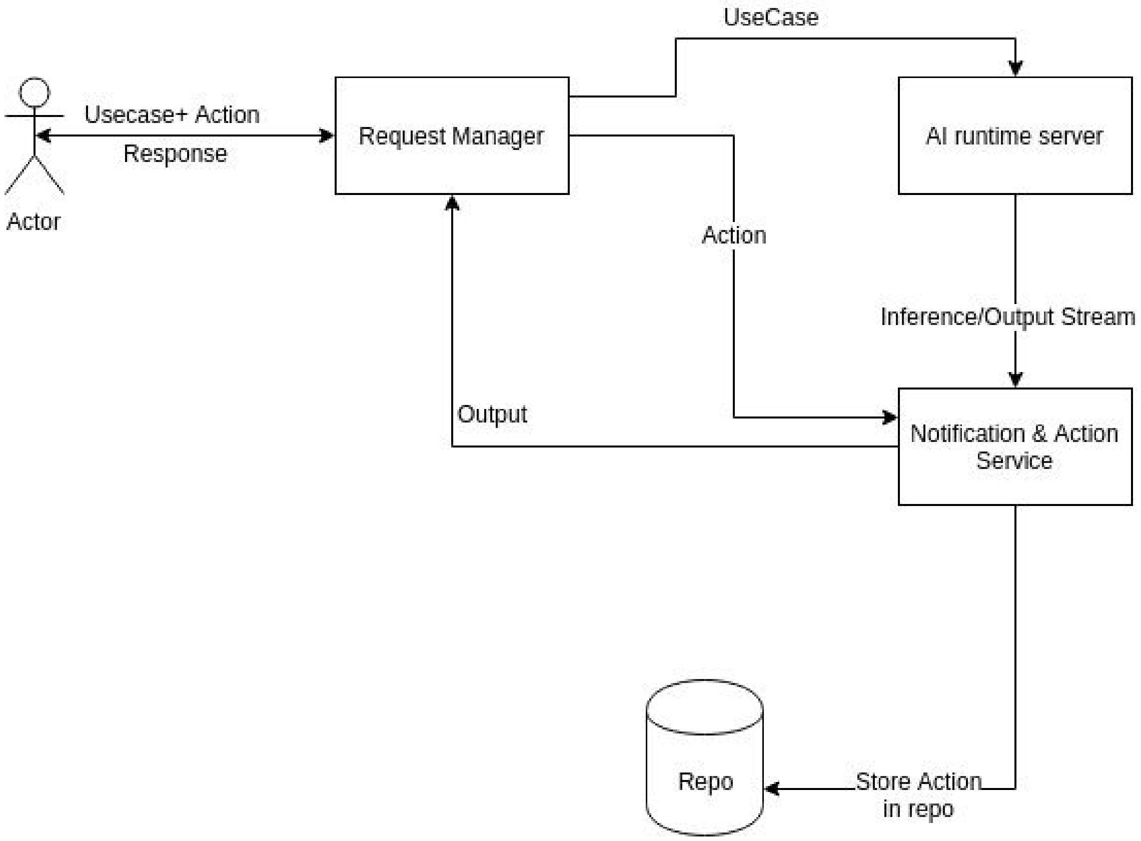
3.4 Security (Authentication, Authorization and Encryption)



Main functionalities of Authentication and Authorization are as follows:

- This service takes care of authentication of user and the service it is authorized to use.
- For maintaining data security and preventing system from external attacker it is used.
- It also maintains the integrity of data.
- Data is transferred in encrypted manner to prevent middleman attack.

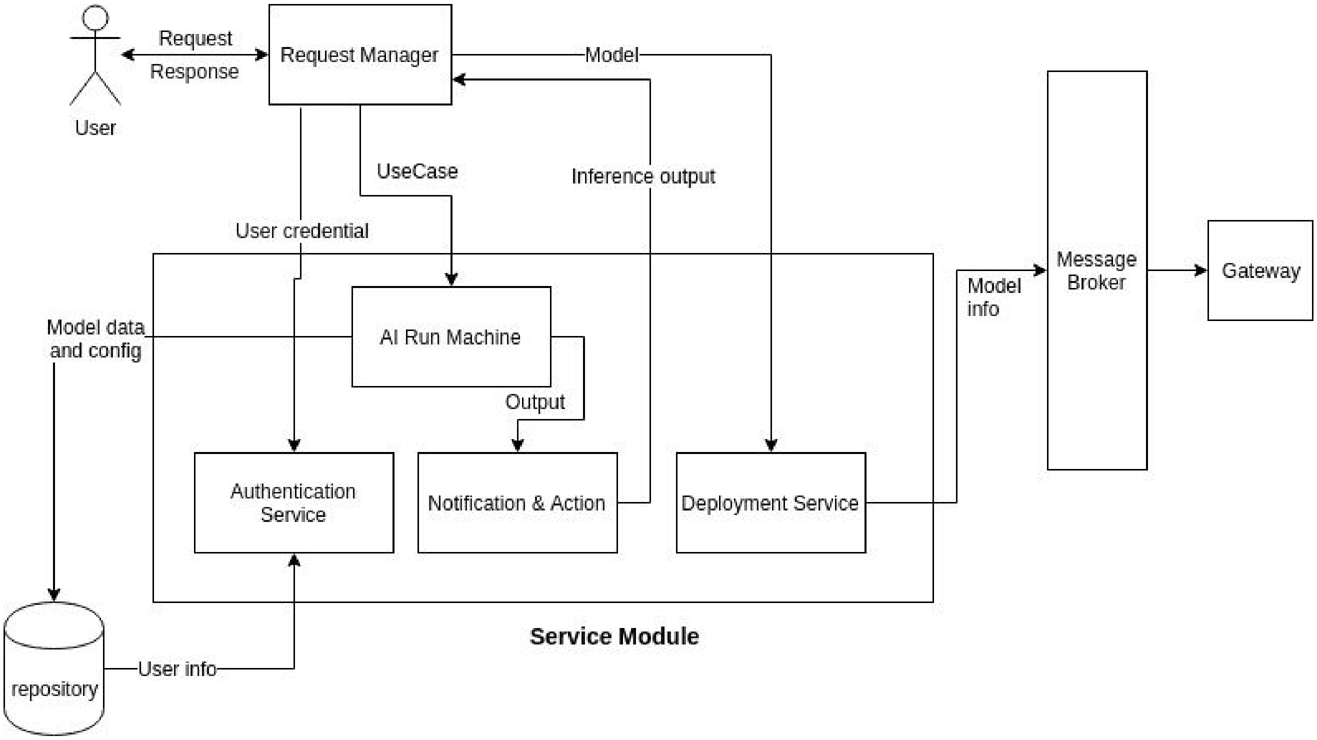
3.5 Notification & Actions



Main functionalities of Notification & Actions are as follows:

- It notify user about specific event.
- Notification service send response along with status code to users.
- It gives alert or notify user when response of user's request is ready.
- Action service is responsible for taking appropriate action which user has specified as input.

4. Interactions between other modules


User
Request
Response
Request Manager
Model
UseCase
Inference output
User credential
Model data and config
AI Run Machine
Output
Authentication Service
Notification & Action
Deployment Service
Service Module
Model info
Message Broker
Gateway
repository
User info