PROJECT DESIGN OUTLINE

PREPARED FOR

IAS Project : IIIT-Hyderabad

PREPARED BY

Team 1:
Bhavin Kotak
Priyendu Mori
Rushit Jasani
Team 2:
Darshan Kansagara
Vatsal Soni
Dhawal Jain
Team 3:
Amit Kumar
Rajesh Dansena

Document : V2.0

1. Introduction to the Project

Definition: AI on edge platform is a distributed platform that provides build, development and deployment functionalities. Most data becomes useless just seconds after it is generated, so having the lowest latency possible between the data and the decision is critical. With this platform, we bring AI capabilities to edge gateway.

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]

[List the cases- simulate what the other modules/users can do with your module]

[At least 10 test scenarios. Give name, and a brief 3-5 line description. Inputs. Outputs. Invocation mechanism. Expected behavior. Other external considerations (say, when to start a container or how many to start and such- as needed by the test). This shd be based on the use cases listed in the Team's reqt document]

- IOT devices and gateway:
    1. IOT device down
        a. Trying to get data from IOT device but no response from the device.
        b. Trying to send commands to device but can't create connection
    2. Gateway connection issue
        a. Gateway not connected to device
        b. Gateway can't connect to message queue
- AI model related issues:
    1. Requested model not found: The model required does not exist in repository.
    2. Model not responding: Model is deployed in the container but not responding to inference requests.
    3. Scheduler: Scheduler didn't start container as expected
    4. Action handler: Action handler invoked an action upon a particular inference but action didn't take place.
- Security Service:
    1. Authorization:
        a. Invalid user.
        b. Invalid password.

   c. Valid user and valid password.

2. Authentication:
   a. Request for valid service.
   b. Request for invalid service.
   c. Valid token exist.
   d. Invalid token for service.

- Load Balancer:
  1. Requested Service is not loaded.

     Invocation method: Request a service that has not been loaded.

     Expected output: Service is loaded successfully.

  2. Requested Service is not available.

     Invocation method: Request a service which is not available in the platform.

     Expected output: Appropriate error message generated.

  3. All machines are fully loaded.

     Invocation method: Request service when all server are fully loaded.

     Expected output: Wait for servers to response.

  4. Requested Service is already loaded.

     Invocation method: Request a service which is loaded.

     Expected output: New instance of the service is created and requests are processed and response is returned.

  5. No model is up and running.

     Invocation method: Request a model first time.

     Expected output: Model is loaded in docker and instance of service is executed.

- Data Services:
  1. Requested data is available.

     Invocation method: Request for the available data.

     Expected output: The request will be served properly.

  2. Requested data does not exist.

     Invocation method: Request for data that does not exist.

Expected output: The request can't be completed as data is not available.

3. Requested table is not available.

   Invocation method: Request for data in non-existing table.

   Expected output: The requested data can't be returned.

4. Unauthorized data access.

   Invocation method: Data is requested by user who is not having access to that data.

   Expected output: The data will not be returned.

5. Database connection errors.

   Invocation method: The data is requested from the database but connection can't be established.

   Expected output: The data will not be returned.

6. SQL injection.

   Invocation method: request for non authorized data by appending trivial condition.

   Expected output: the data corresponding to valid name should be appear

7. Requested data updation.

   Invocation method: Request for data updation.

   Expected output: data will be updated in the database successfully.

- **Health Check:**

1. Running service gets killed unexpectedly

   Invocation method: Kill a service manually using kill command.

   Expected output: The killed service should come up at any server that is the least loaded.

2. Running server gets killed unexpectedly

Invocation method: Power off a server.

Expected output: Another machine comes up with all the services running that were on the killed server.

3. AI model gets killed in middle of a schedule.

   Invocation method: Kill the running AI model using kill command.

   Expected output: The same AI model comes up on any server that is least loaded.

4. Exclusive service or AI model gets killed

   Invocation method: Kill an exclusive service or AI model using kill command.

   Expected output: The service or AI model will come up in an exclusive environment.

2.2 Test cases- [used to test overall project]

Authentication and Authorisation :

1. Check Username and password

   1. Input : username and password
   2. Output: Success or failure
   3. Description : It will verify username and password from DB and return result.

2. Check User access permission

   1. Input : Username
   2. Output : User's token along with list of services accessible to user
   3. Description : It check user access permission from repository and return list of services accessible to user.

Deployment Service

1. Check model is deployed/ Check Model endpoint
   1. Input : Given Model API endpoint

2. Output : Got response if model is deployed and running or URL not found if model is not deployed
3. Description : It will check whether model is deployed or not by accessing its end-point. If API is accessible we got response. But if model is not up and running we got 404 page not found error.

Scheduling Service :

1. Check model is up between start and end time
    1. Input : Start time , Endtime, Model API endpoint.
    2. Output : Check Model is schedule and running up between start and end time.

    3. 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 whether model data is packaged properly.

1. Input: Model information,
2. Output: Package model
3. 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 whether package is received by AI run machine.

1. Input: Packaged data (Zip file)
2. Output: Send it to AI run machine.
3. Description: It will send Packaged file to AI run machine where actual model will run.

Notification & Action Service :

1. Check Action performed or not

    1. Input : Model API endpoint action code
    2. Output : Check console that action performed and output seen on console or not and notification receive to user registered email/mobile or not.
    3. 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

    1. Input : Model API endpoint and prediction output.
    2. Output : Whether user receive notification on register mobile or email or not.
    3. Description : After model got result and it should notify user about it. Checking whether user receive any notification about it or not.

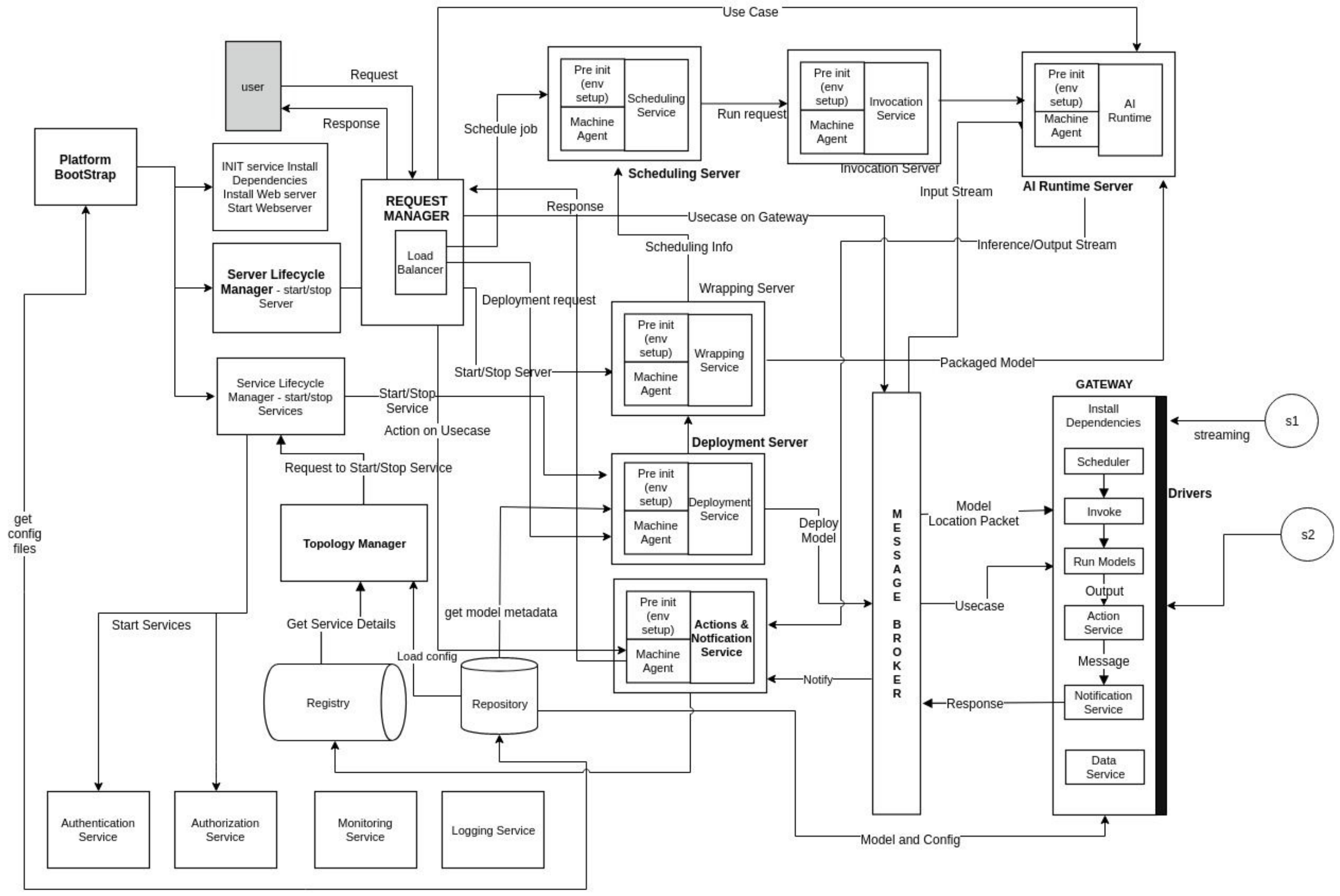
Logging and Monitoring

1. Check log file Content is updated or not

    1. Input: Log file path , ServiceName, Logmsg
    2. Output : Log File content change or not
    3. Description : Check whether logmsg given by service is written on log file or not.

3. Solution Design Considerations

3.1 Design big picture



3.2 Environment to be used

- The platform should be installable on any 64-bit linux machine/distributed machines.
- The internal messaging will be done through RabbitMQ and thus the RabbitMQ Server will be installed and will be a part of the environment.
- Docker is used for containerization of AI model and runtime of AI application is Tensorflow serving.

3.3 Technologies to be used

- Python and Flask framework should be primarily used for the development of the platform.
- Bash Shell scripts can be used to make installation/configuration automated.
- RabbitMQ must be used for MQ implementation, MySQL and MongoDb as Database, NFS as common File System.

- Docker for containerization and running tensorflow serving models
- Elastic search, logstash and kibana for centralized logging and monitoring

3.4 Overall system flow & interactions

- User will login to system using the URL provide.
- Application will authenticate the user (Based on username and password), get an authentication-token back from Authentication service, then all the further user requests will contain authentication-token in them. And based on the authentication-token and service request the user will be checked for authorization. If the user is authorized the request is sent to the Load Balancer. API gateway will also have a MQ on which it will receive the output from Load Balancer.
- Load Balancer will have two components - Routing and Load Balancing. The Load Balancing component will gather machine as well service level stats from Monitoring Module. And based on the machine level stats it will communicate with the Server LifeCycle Manager to provision a new machine or kill a machine. And based on service level stats it communicates with Service LifeCycle Manager to invoke a new instance of the service. The Routing component based on the service request will push the request message into the respective service queue.
- Service LifeCycle manager will receive request from Load Balancer to start a new service instance along with the machine domain name on which the service instance should be started.
- Server lifecycle manager will analyse the machine load (from machine level stats from monitoring module) and will assign a machine to be used to invoke another instance of the service to Load Balancer.
- Each machine on the network will have a Machine Agent running on it.The task of the Machine Agent will be to gather service level as well as machine level stats(ram and cpu usage) and send it to Monitoring Module.Machine Agent can also start or kill a service instance.
- There will be Service Listener running in a Docker Container. This Service Listener will receive the request from its request queue, parse the json request received and then based on the request response will be sent to the user.
- The server will communicate with AI and IoT subsystems based on the type of request received from the user.
- 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.

- Notification and action service will perform user's action and return back response to the user.

Types of requests and sub system interactions

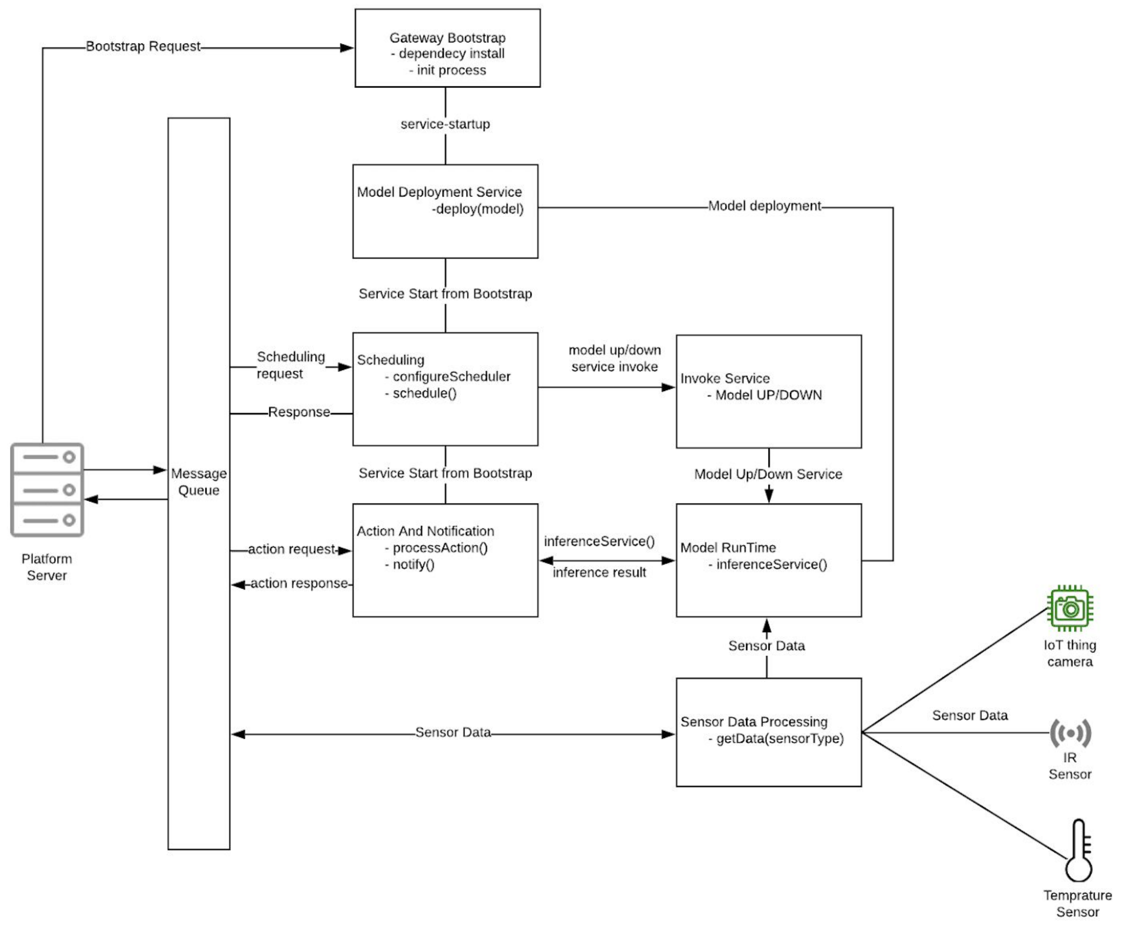
1. Upload a trained model - Model will saved in the repository and details will be updated in the DB. Server will handle the request and send response to the user (Success/failure/status).
2. Deploy it on cloud and generate API endpoint to access the same - Server will interact with the AI subsystem which will fetch the details from registry and models from repository. The Docker image will be run and the API endpoint of the service will be exposed to the user.
3. Run already uploaded model on cloud - The model could be running on the cloud or on the Edge. User can send a request to run an already uploaded model on Edge or on the cloud based on the requirement. Server will interact with the Iot Subsystem if the model needs to run on the Edge. MQ will be used to communicate with server and Edge Gateway.

   If the request is to run the model on the cloud, AI subsystem will take the inputs and schedule it accordingly (Run once/ schedule as per request)

3.5 IOT system design

Block Diagram

This sub system will provide below given components:



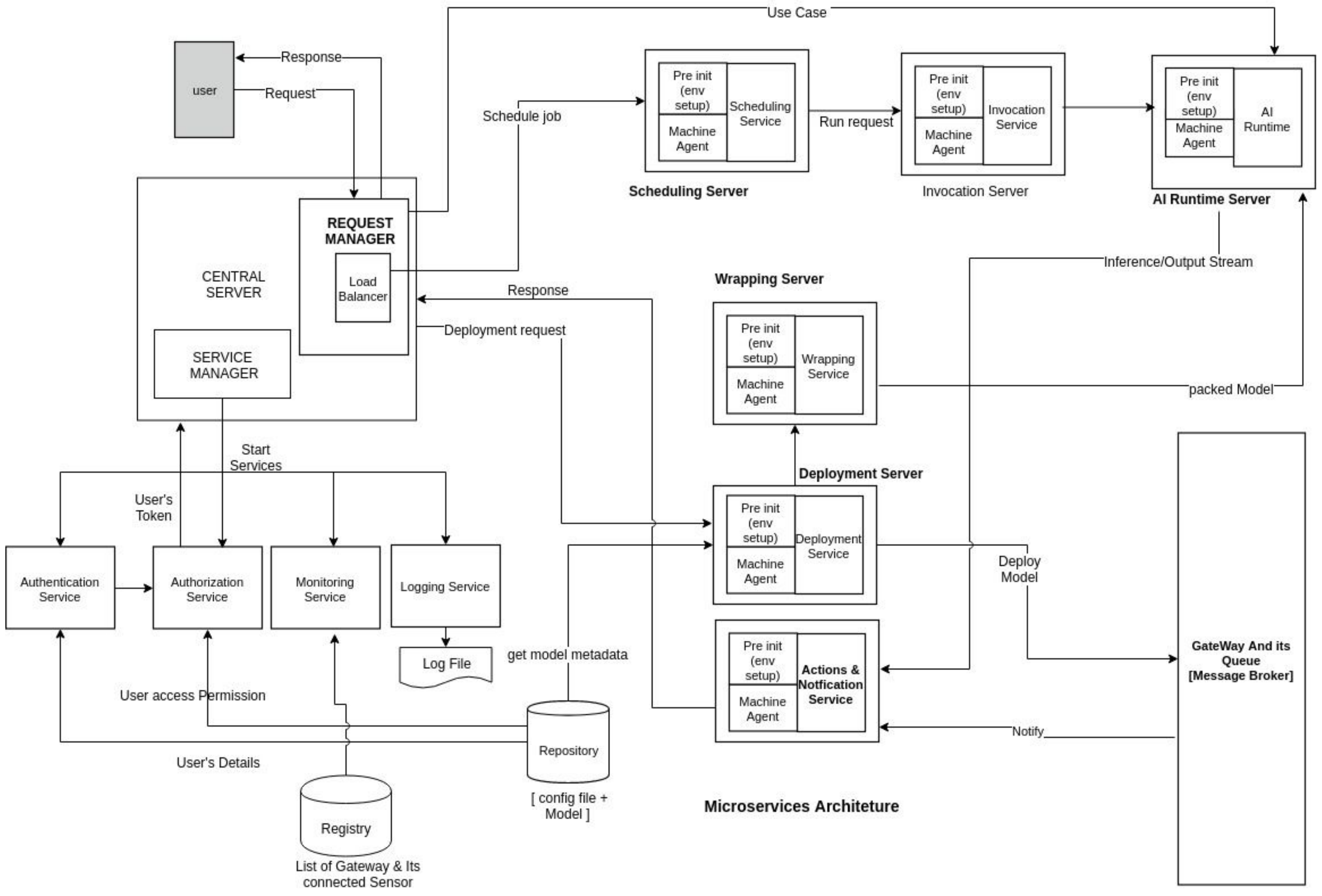
This sub system will provide below given components:

- Gateway Bootstrap: This sub-system acts as agent to make the Gateway ready for Model deployment and other services as Notification and Action, Inference, Sensor Data processing and streaming etc. It has following APIs.
- Deployment Service: This Sub-System has all the deployment related services like Model deployment and Inference Service startup. Once bootstrap installs all the dependencies, This service worker deploys the Machine Learning Model on Gateway and keep ready for inference.
- Scheduling Service: Scheduling Service is responsible to schedule the Model deployment and Model GRPC service startup. There can be many scheduling type possibilities. Based on the User's scheduling input it does the scheduling of the Model on Gateway. Scheduling service is helpful to save the memory and processing usage of the system.
- Action and Notification: This Sub-System is responsible for doing some action or notifying user based on the User's given action condition. User gives the action file while deployment itself as part of the zipped packaged file. The action file is plugged in here and used to act and notify accordingly.

- Invoke Service: Invoke Service invokes the system services of different sub-systems like service stra-up and stop, service recovery etc.
- Model RunTime: This is where the Model runs and does expose the Model inference gRPC services for user inference request.
- Sensor Data Processing: This sub-system connects with sensors through their drivers and gets the raw data of streams. It processes the raw data into proper model input form and starts streaming it to the on-premise Model or to the Message Queue which is consumed by the Model deployed on the server.

3.6 Services design

Block diagram



This sub system will provide below given components:

- Authentication service : Authentication service checks for the valid user using the authentication mechanism used in platform and send the authentication token back to the user for future use.

- Authorization service : Authorization service checks whether the authenticated user is authorised for a requested service or not.If the user is authorised then only the request is carried forward.

- Scheduling service : It will provide variety of scheduling. It will schedule model for particular time interval, between start and end time, repeat and many more .

- Deployment Service :It will package model and auto gen script to deploy model in any gateway or server instance.

- Notification & Action Services : It will notify user when user's response is ready. Also device sensor will notify server regarding specific event and server will take appropriate action.

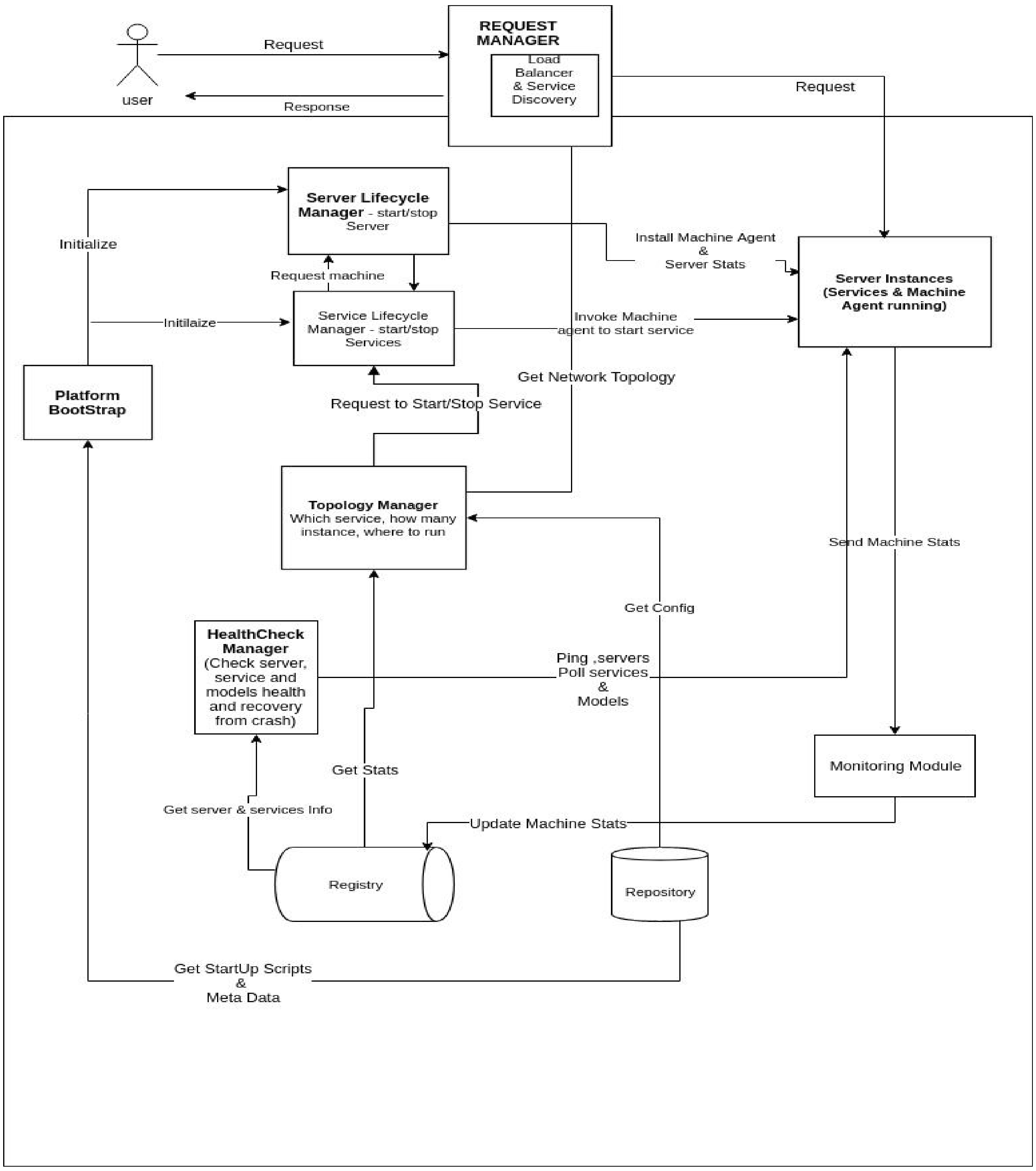
- Monitoring : It will continuously monitored all service to check health of each service and create new instance of any failed/downed service immediate as action of fault tolerance.

- Logging : Provide an easy to use, general purpose logging system that keeps track of all events associated with a particular service. It all provides logs for the whole platform as well.

- NFS : It contains user's saved model, It will common interface between all services for storage purpose.

3.7 Server system design

Block Diagram


Request
REQUEST MANAGER
Load Balancer & Service Discovery
user
Response
Request
Server Lifecycle Manager - start/stop Server
Initialize
Install Machine Agent & Server Stats
Server Instances (Services & Machine Agent running)
Request machine
Service Lifecycle Manager - start/stop Services
Initilaize
Invoke Machine agent to start service
Get Network Topology
Platform BootStrap
Request to Start/Stop Service
Topology Manager Which service, how many instance, where to run
Send Machine Stats
Get Config
HealthCheck Manager (Check server, service and models health and recovery from crash)
Ping ,servers Poll services & Models
Monitoring Module
Get Stats
Get server & services Info
Update Machine Stats
Registry
Repository
Get StartUp Scripts & Meta Data

a. Security (Authentication and Authorization)

    i. This service takes care of authentication of user and the service it is authorized to use.
    ii. For maintaining data security and preventing system from external attacker it is used.

b. Notification services
    i. It notify user about specific event.
    ii. It gives alert or notify user when response of user's request is ready.
c. Load Balancing
    i. Load balancer is used to maintain load among the multiple server instances. It accepts input from user and based on the load of each server instance it will send user's request to one of the server instance.
    ii. For scaling purpose, load balancer is used.
d. Service Manager
    i. It task is to maintain proper interaction between all the services.
    ii. It also takes care of interaction with AI module and Gateway module.
e. Logging and Monitoring
    i. Logging and monitoring service is used to monitor all the activities which are happening in server.
    ii. It interacts with the every other module and dump all the logs in log file.
    iii. Functionality involves like creating log file, retrieve recent logs, Delete/modify logs.

f. Machine Agent

   This module performs below given functionalities:

    i. Each server runs a machine agent and service lifecycle manager sends commands to Machine Agent
    ii. Machine agent does the task of starting or stopping service on receiving commands from SLC manager
    iii. It copies the zip file of the service from NFS, unzips it, installs dependencies and starts the service.
    iv. On receiving stop request, it stops the given service and sends response to the Service LC which unregisters it from Service Discovery.

g. Health Check

   This module performs below given functionalities:

    i. Health check calls /health endpoint of every registered service to check if the service is running. If found not running, it communicates with load balancer to get a machine with least load and run the service on that machine.
    ii. Health check pings all the server IPs registered already to check if any machine is down or out of the network. If found not reachable, another server comes up with all the service running which were previously running on the unreachable server.

iii. Health check calls the prediction URL of all the AI models that are running and if any of the AI model is found not running during its schedule that the AI model comes up on the least loaded machine.

iv. Health check also takes care that if an exclusive service or exclusive AI model is down, it runs it again in an exclusive server only.

h. Performance Optimization

  i. It performs caching and other techniques to optimise response time.

3.8 Approach for communication & connectivity

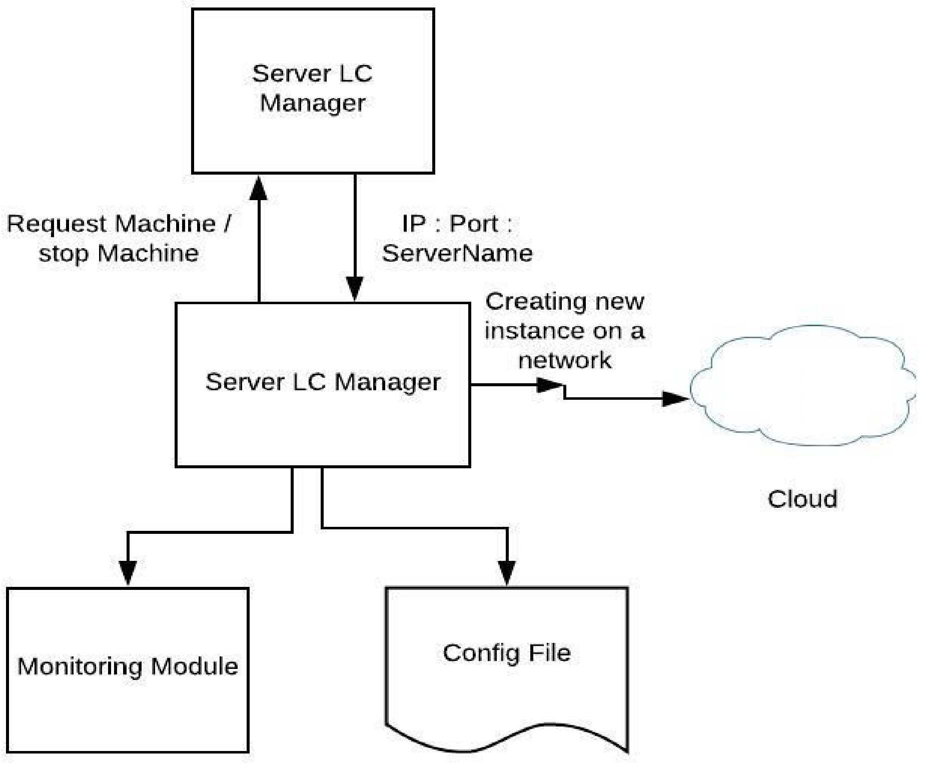
- Communication between various components will be primarily done by RabbitMQ and RESTful web services.

3.9 Registry & repository

- Registry 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.)
- Repository/NFS 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.

3.10 Server & service lifecycle

Server Lifecycle Manager:



Main functionalities of this module are as follows :

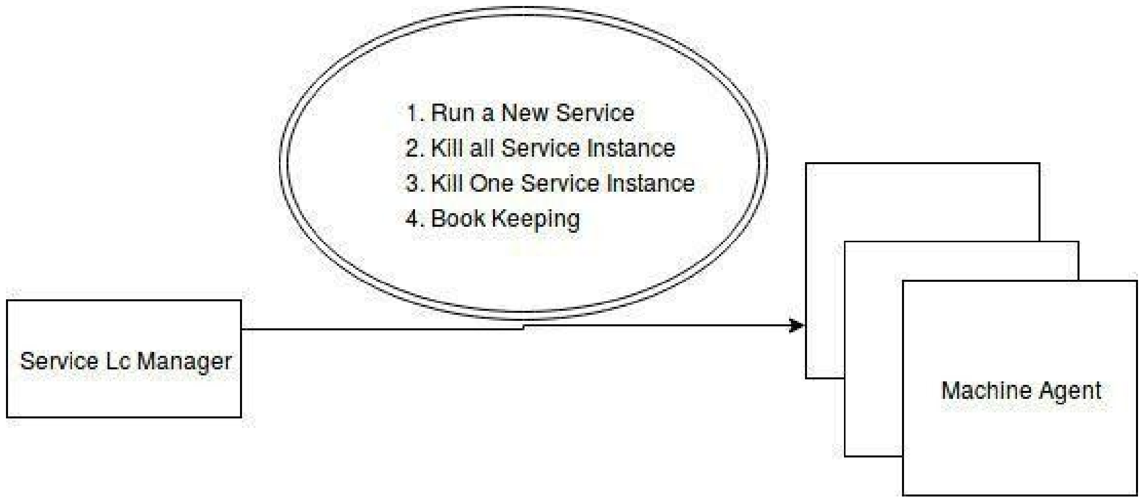
1 . StartMachine ()

- ➢ Get stats from Monitoring Module
- ➢ Get ip and hostname from config file
- ➢ Login to machine through Ssh using ip address and hostname
- ➢ Install Docker (if required)
- ➢ Initialization and BookKeeping
- ➢ Run Agent
- ➢ Make Agent send its stats to Monitoring Module

2. StopMachine ()

- ➢ Shutdown the machine
- ➢ Then notify the Load Balancer.

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.

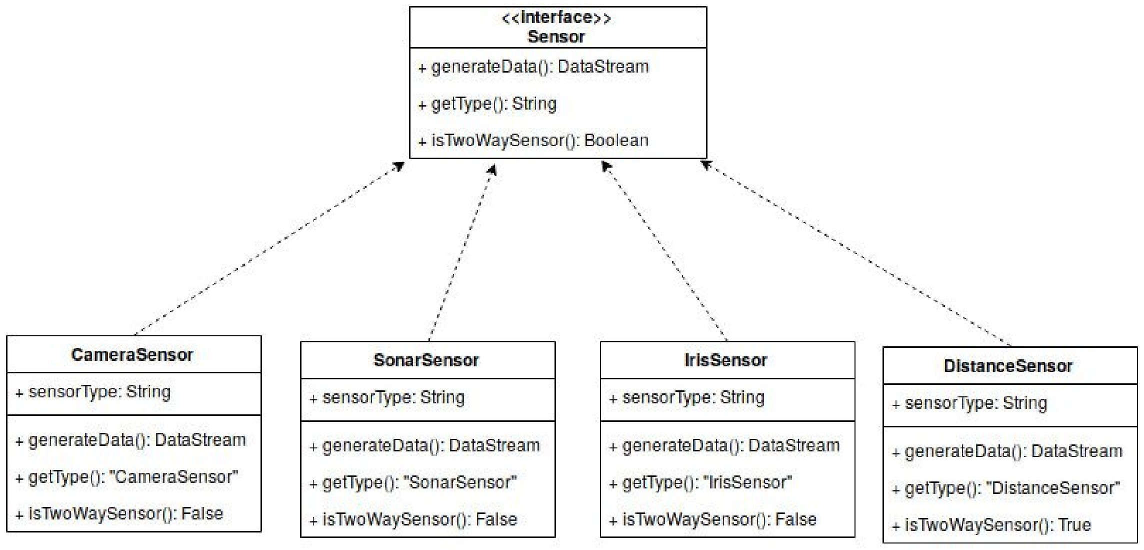
2. Kill all Service instances running on single machine input (ipaddress, hostname): Get info about services running on that machine from registry. Then kill those service containers.

3. Kill a single Service instance input (ipaddress, hostname, serviceid): Kill that service instance from docker container using input.

3.11 HA & Load Balancing

Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a server farm or server pool. To cost‑effectively scale to meet these high volumes, modern computing best practice generally requires adding more servers. A load balancer acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures about application high availability and ensures that no one server is overworked, which could degrade performance. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it.

3.13 Sensor Interface


<<interface>>
Sensor
+ generateData(): DataStream
+ getType(): String
+ isTwoWaySensor(): Boolean
CameraSensor
+ sensorType: String
+ generateData(): DataStream
+ getType(): "CameraSensor"
+ isTwoWaySensor(): False
SonarSensor
+ sensorType: String
+ generateData(): DataStream
+ getType(): "SonarSensor"
+ isTwoWaySensor(): False
IrisSensor
+ sensorType: String
+ generateData(): DataStream
+ getType(): "IrisSensor"
+ isTwoWaySensor(): False
DistanceSensor
+ sensorType: String
+ generateData(): DataStream
+ getType(): "DistanceSensor"
+ isTwoWaySensor(): True

3.14 Wire and file formats (Interactions between modules)

Wire 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 - XML/json format

4. User’s view of system

4 .1 Executable

The application builder can download the zip file to setup the platform. The zip file will contain all the dependencies (Rabbit MQ, NFS, MySQL, Tensorflow, Tensorflow Serving, Python etc...) needed for the platform to run.

The application builder must install this zip to setup the platform on his premise.

README contained in the zip will list down all the installation steps needed.

4.2 Setup

After all dependencies are installed, they must be setup for being able to use them :-

- ➢ Python and pip must be installed to run tensorflow.
- ➢ Tensorflow must be setup in a directory where user has the permission access. Must configure the port for tensorflow serving.
- ➢ Rabbit MQ is to be installed on a dedicated machine which should act as the Message Queue broker. The ip and port of the machine where this broker listens to must be saved to be used by all other machines in the network.
- ➢ Database like MySQL must be installed along with an user. The endpoint to where and how to access the database must be defined/stored in a configuration file.
- ➢ API Gateway must be setup to serve any HTTP requests. API gateway must be configured to have a single endpoint for the whole platform. All available services would be given a sub-domain of this endpoint.
- ➢ For Repository and registry we must have a Network FIle System, which all machines can access. Setup of this common file system should be done.

4.3 Configure

Configuration of the platform according to dependencies setup in the above phase must be done here.

- ➢ Configuration of how to connect to LDAP (ip:port) for the configuration of security service.
- ➢ Configuration of how Data service will connect to the DB.

- Similarly how email service will connect to the email server.
- Configuration of common services like File service - which directory (NFS or local) the file service will use as its working directory; rabbitMq - configuration of ip and port where the MQ broker is running.
- Configuration of machines which will be used for scaling up and down of the services' instances. They should have access to a common directory having executables of all the services and a bootstrap script which will setup the machine to be used in a distributed fashion.

4.4 Develop

Application developer can now develop , deploy and run his own model using the base services provided by the platform.

4.5 Package

All files for services should be bundled up.Then along with this ,we also include all the dependencies and config files needed for the code to run in the zipped folder.

4.6 Deploy

The tensorflow model file should be uploaded at the deployment portal which will return URI which will be the end-point to use the application.

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, User customized Objects.
- User Defined Data Structures : User defined Classes, Interface, Tree.

Storage Structures

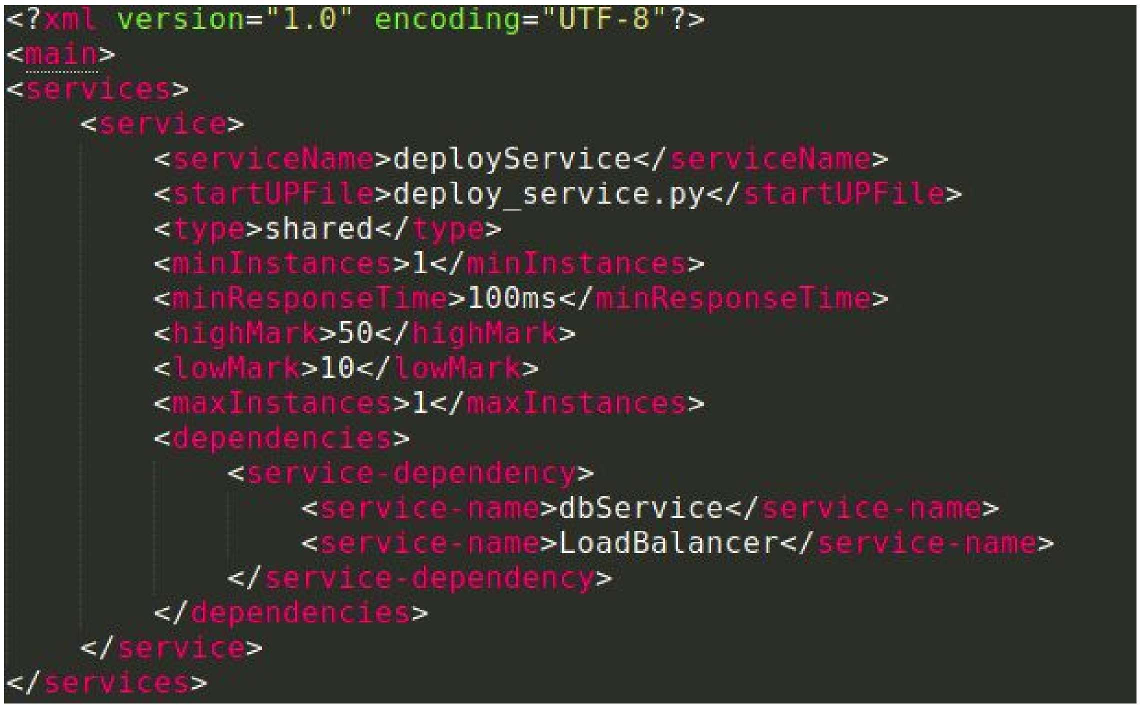
1. Sensor Information
    - Sensor Type
    - Sensor Location
    - Sensor ID
    - Gateway ID in which sensor is deployed
    - Streaming rate
2. Gateway Information
    - Gateway ID
    - Gateway Name
    - Gateway IP
    - Gateway PORT
    - Sensor Lists
3. Server Information
    - Server ID
    - Server Instance Name
    - Server IP
    - Server PORT
    - Services information
    - CPU Utilization
    - RAM utilization
4. Scheduling Information
    - Schedule Type
    - Schedule Start Time
    - Schedule Stop Time
    - Model to schedule
5. User Information
    - User name
    - User Id
    - User Password Salt
    - Registered Gateways
    - Models uploaded
6. Network Topology
    - Service Location
    - Number of services running
    - Server Instances
    - Server-Service mapping

7. Service Information

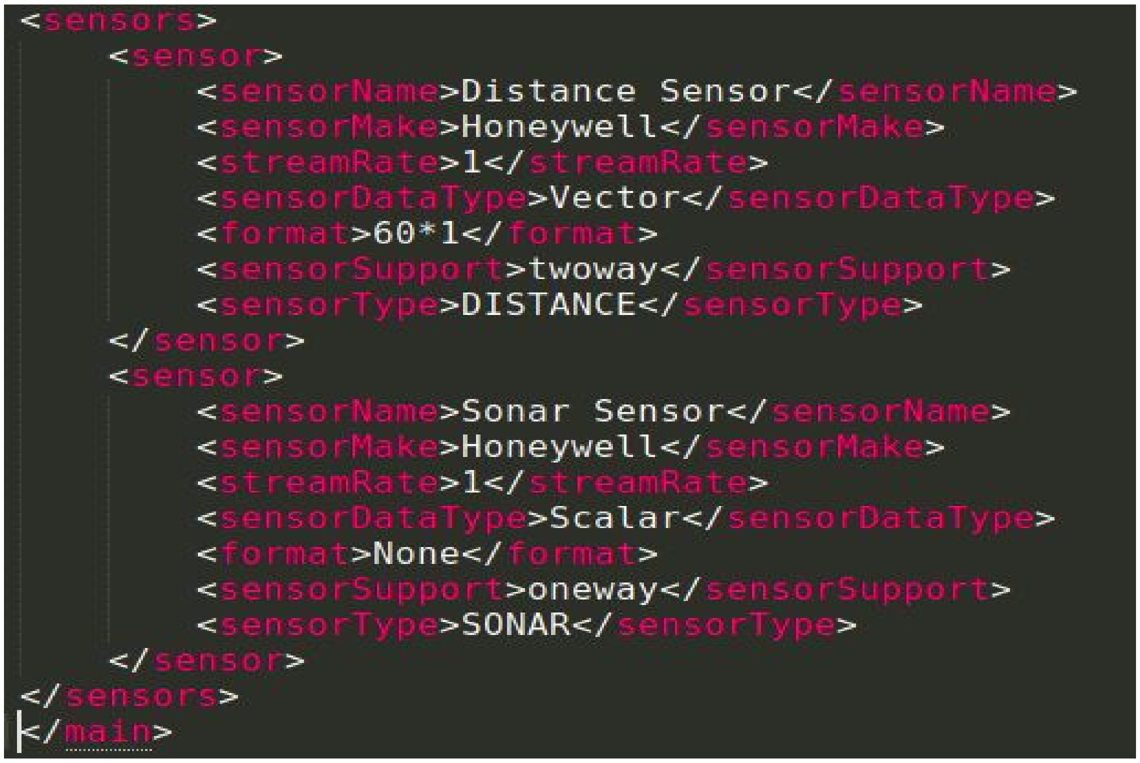
- Service ID
- Service Name
- Instance URL (IP & PORT)
- Dependencies list
- Threshold values to start or stop service
- Minimum no. of instances

Storage structure in the form of XML

Platform config

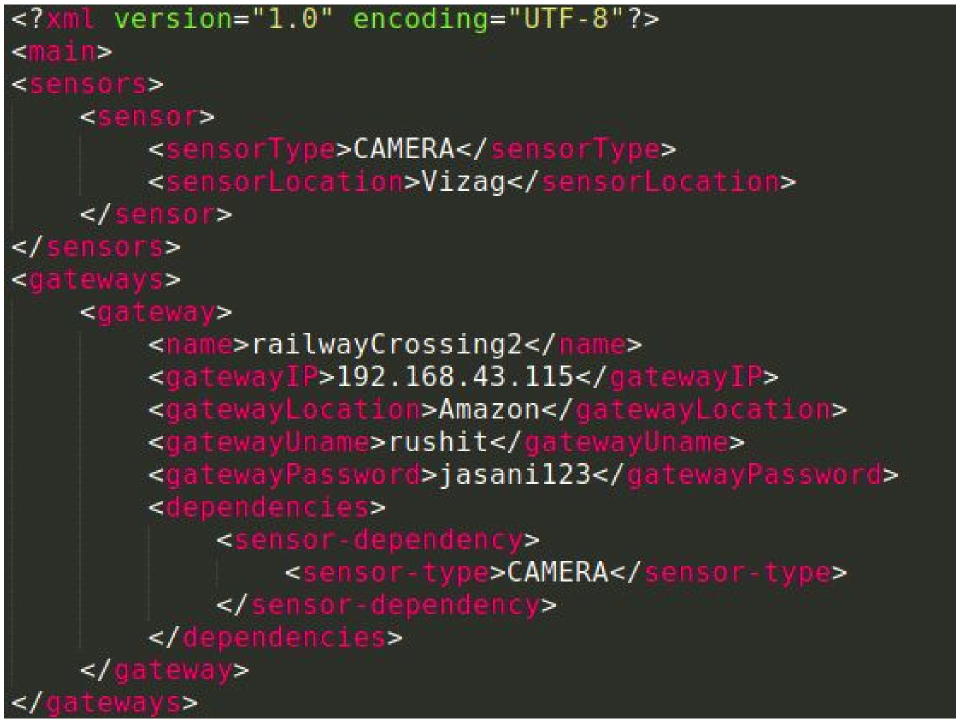


```
<?xml version="1.0" encoding="UTF-8"?>
<main>
<services>
    <service>
        <serviceName>deployService</serviceName>
        <startUPFile>deploy_service.py</startUPFile>
        <type>shared</type>
        <minInstances>1</minInstances>
        <minResponseTime>100ms</minResponseTime>
        <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>
```



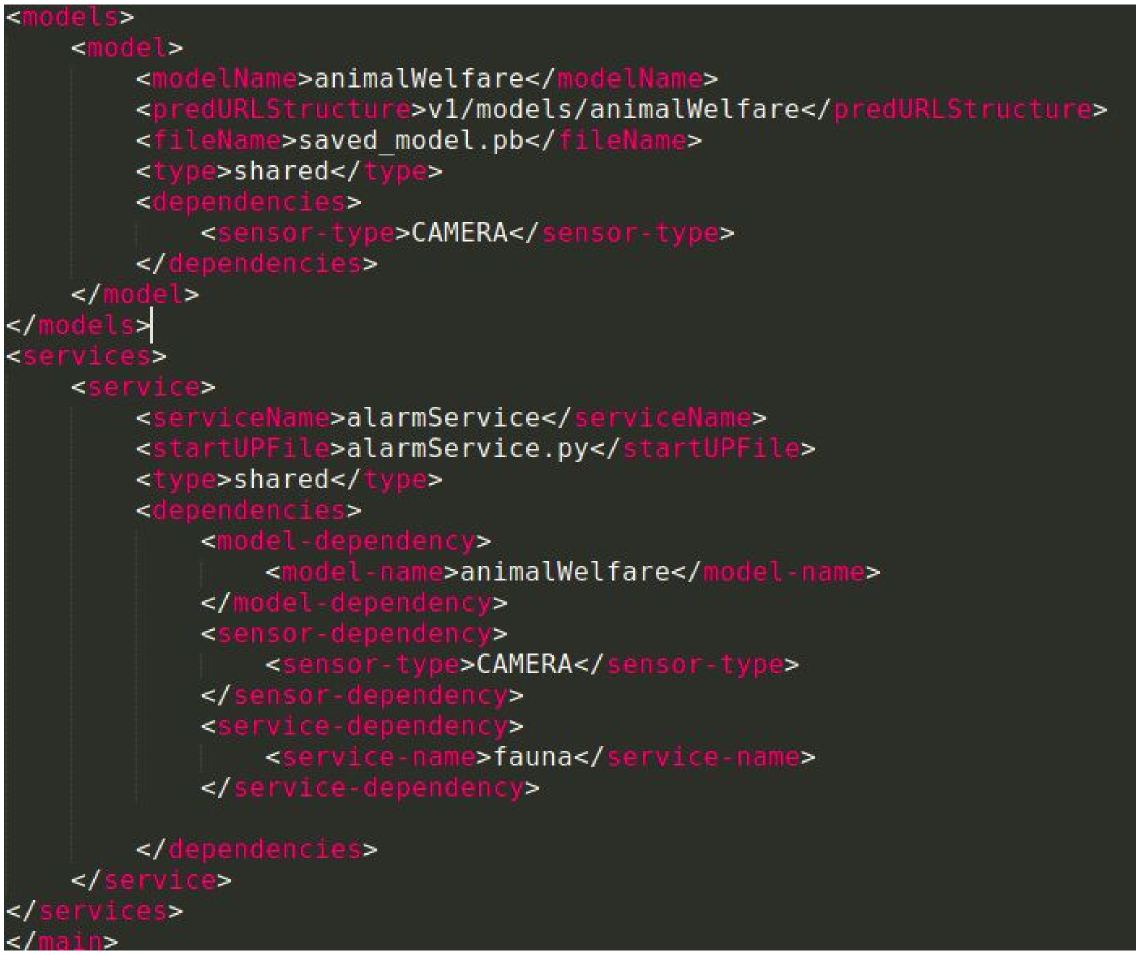
```xml
<sensors>
    <sensor>
        <sensorName>Distance Sensor</sensorName>
        <sensorMake>Honeywell</sensorMake>
        <streamRate>1</streamRate>
        <sensorDataType>Vector</sensorDataType>
        <format>60*1</format>
        <sensorSupport>twoway</sensorSupport>
        <sensorType>DISTANCE</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 config



```xml
<?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>
```

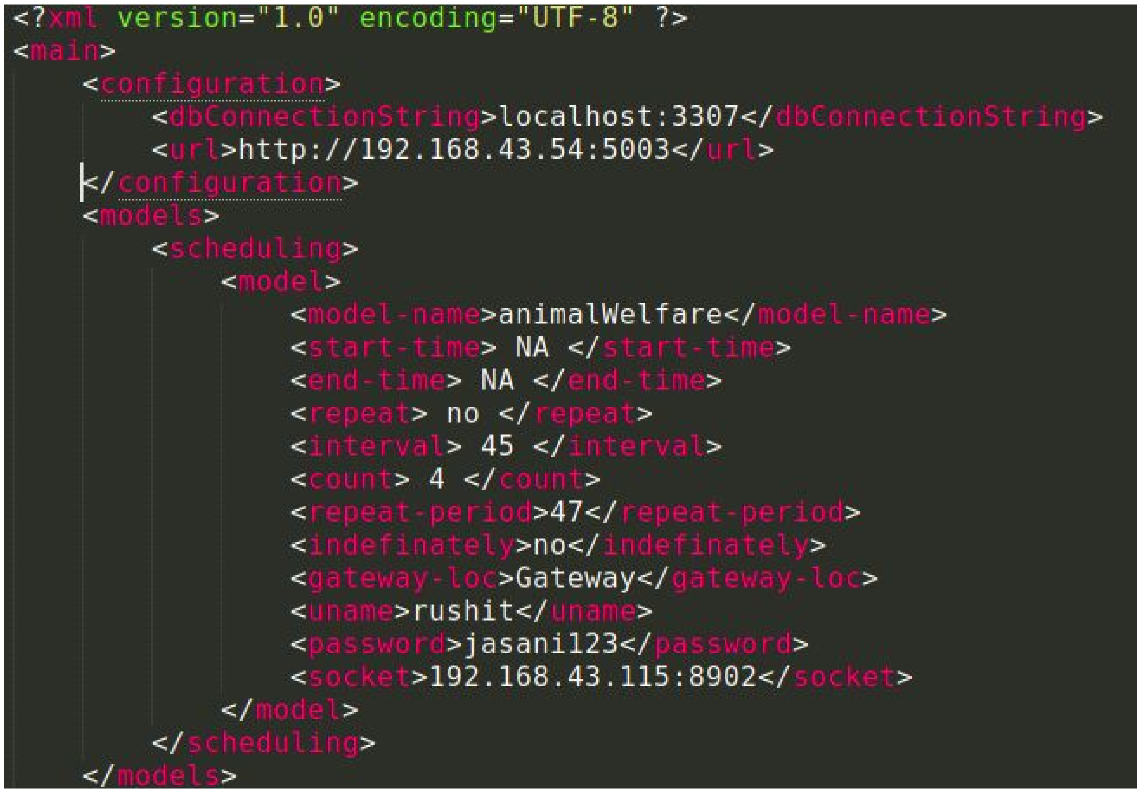
Model config :



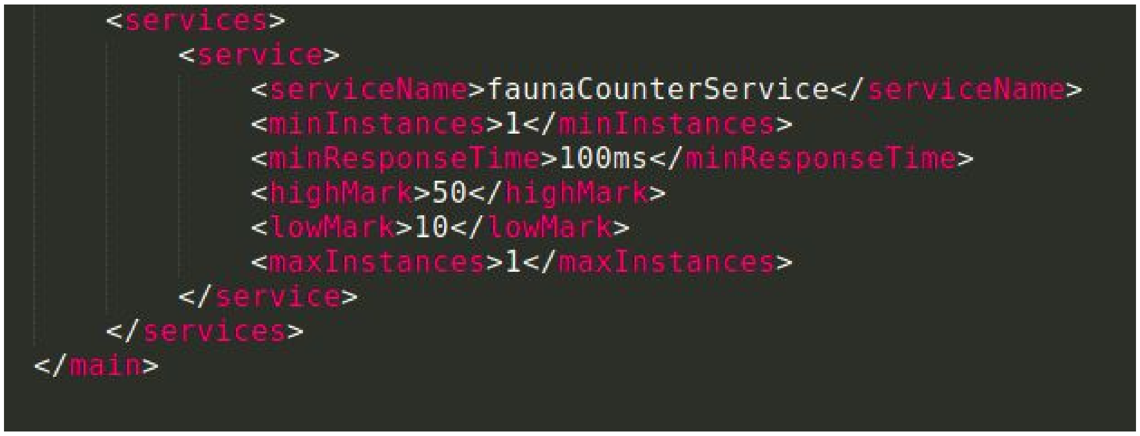
```xml
<models>
    <model>
        <modelName>animalWelfare</modelName>
        <predURLStructure>v1/models/animalWelfare</predURLStructure>
        <fileName>saved_model.pb</fileName>
        <type>shared</type>
        <dependencies>
            <sensor-type>CAMERA</sensor-type>
        </dependencies>
    </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 config



```xml
<?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>
```



```xml
    <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

- User of the platform will interact with Web User Interface/Mobile App interface. User shall be able to see a summarized details of the system and user specific details in a user dashboard which is going to be landing page after the success full login of the user.

- 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.
- User shall be able to add new IoT devices with a User interface which asks for the details of the IoT device in an html form like IoT device type, capabilities, network details, location etc.
- 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 IoT it has to deal with, input and output format/type etc.

7. Persistence

- Using Serialization and pickle library in python we can make periodic backup of our trained model and make it persistent.
- All the data is stored in centralized repository which is a Network File System. User related and model related information is stored in persistent database (MySQL DB)
- We also used tensor serving library for versioning of model and make it persistent.

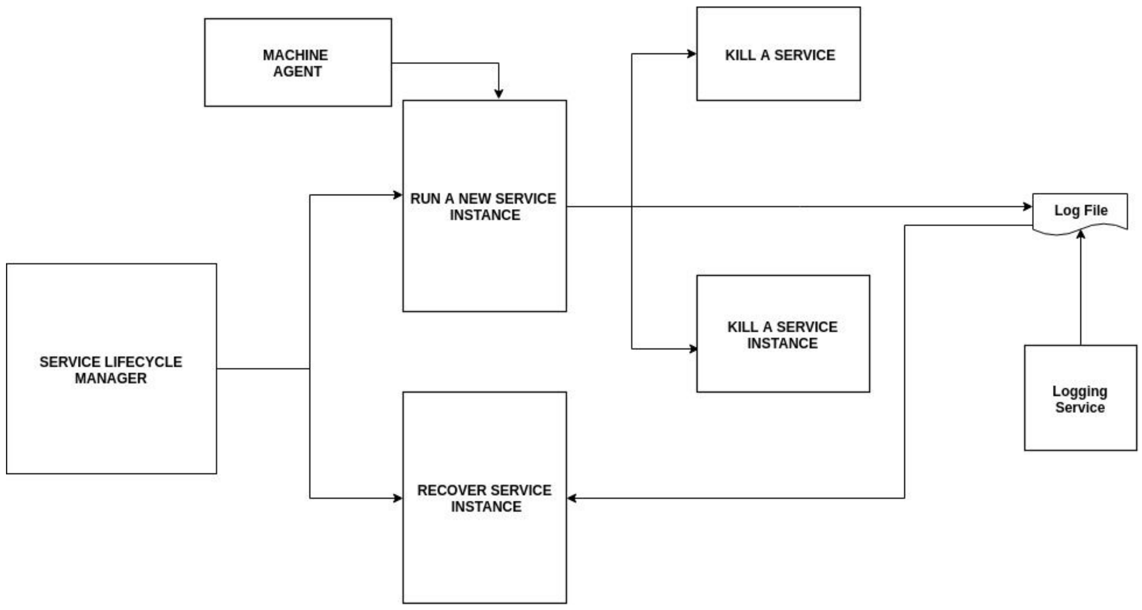
8. Three Modules [Divided for teams]

➢ Server Module
➢ Service Module
➢ IOT Module

Low Level Design

1. Lifecycle of the module

Service Module Lifecycle



2. List the sub modules

The complete AI on the edge platform has been divided into three subsystems/modules as below:

- **Server Module**

  This module handles server provisioning and service life cycle handling. It has load balancer and service discovery that helps in routing request and discover micro services running on the platform. It also hosts request manager.

  Platform bootstrap is again a part of server module.

- **Service Module**

  Service module consists of all the utility services like Logging, Monitoring, Notification to the user and other subsystems, User authentication and Security services of the platform, Caching of the data as and when required for performance optimization.

  Deployment , scheduling , invocation and actions are key services of this module.

- **IOT Gateway Module**

  IOT Gateway Module consists of all the IoT devices like camera, temperature sensor or any other type of sensors. It also consist a lightweight IoT computation gateway which can have the Machine Learning Model deployed on the edge for edge inference. If the Model is deployed on the cloud, gateway will be able to stream the IoT data in real time.

3. Brief overview of each sub module

1. **Server Module**
    a. **Component of Server**
        i. Request Manager
        ii. Scalability and Load Balancing
        iii. Service Discovery
        iv. Server Lifecycle Manager
        v. Service Lifecycle Manager
        vi. Topology Manager
        vii. Machine Agent

2. **Service Module**
    a. Authentication service
    b. Authorization Service
    c. Centralized logging Service
    d. Monitoring service
    e. Deployment Service
    f. Scheduling Service
    g. Invocation service
    h. Actions and Notifications service

3. **Gateway Module**
    a. **Component**
        i. Gateway Message Broker
        ii. Sensors management
        iii. Commands and Notifications service

4. Interactions between sub modules

Interaction between and Server and Service Lifecycle manager:

1. Service lifecycle manager interacts closely with server lifecycle manager to provisioning servers with all dependencies.
2. Both the systems are microservice based and invokes using REST api services.

Interaction between Server and platform service :

1. When Scheduling or deployment request came, It find available socket of scheduler from service discovery and then send scheduling request to that service socket.
2. Load balancer will distribute load among different platform services.
3. Server monitoring will continuously doing health check and when any service goes down it will create new instance of it.

Interaction with IOT sub module:

1. IOT sub module has a Gateway manager installed as to make it compatible with the platform.
2. All the interactions happens using message broker and services using AMQP.
3. IOT module streams the data at a particular rate specified by used and puts it in respective sensor queues and exposes the MQ host to the services which run the AI model

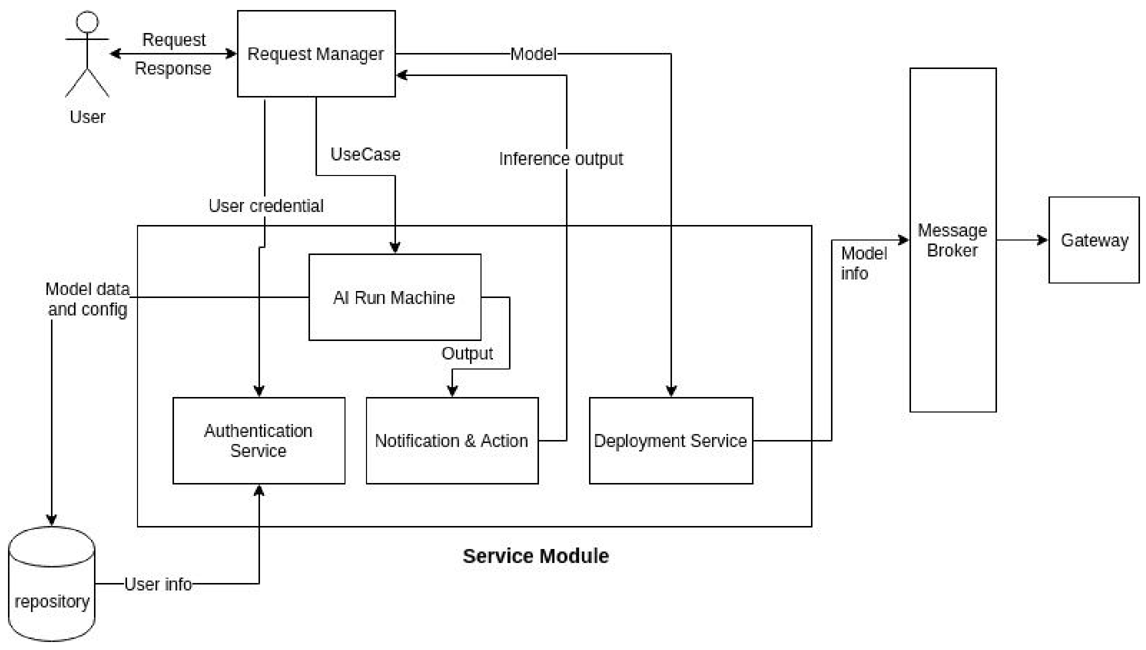
5. Interactions between other modules

The 3 modules manage the requests, schedule and deploy AI models on cloud or on Gateway as specified by the user.

The interactions between each service or subsystem is mostly through REST or AMQP.

There is a common repository hosted as NFS in a server on which each service has access to from which it can read config details and write the required data.

Service component interaction with other module



6. Other design considerations

- **Fault-tolerance** - The software is resistant to and able to recover from component(like container) failure.
- **Security** - The software is able to withstand and resist hostile acts and influences.
- **Performance** - The software performs its tasks within a time-frame that is acceptable for the user, and does not require too much memory.
- **Scalability** - The software adapts well to increasing data or number of users.
- **Centralized logging** - All the logs generated are available in a central store which can be visualized using a dashboard and with filter options available to the user/admin.