GATEWAY DESIGN OUTLINE

PREPARED FOR

IAS Project : AI-on-the-Edge Platform

IIIT-Hyderabad

PREPARED BY

Group1-Team 3:

Amit Kumar (201550848)

Rajesh Dansena (20163005)

Document : V3.0

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]

1. Gateway Registration and Bootstrap : Gateway details would be in the deployment package provided by the user in a configuration file. Based on this Gateway details, deployment services will copy the required content to bootstrap the gateway and run the bootstrap service to startup the Gateway services.

   Input: Gateway deployment package and bootstrap services
   Output: Gateway bootstrap Status
   Invocation Mechanism: through ssh or http protocol to the Gateway machine.
   Expected behaviour: Gateway should be installed with all dependencies and service and util managers. Also, AI model should be deployed in case of on-premises model deployment.

2. Sensor connection and Data Processing : Sensors keep on sending the data to Data processing unit of the Gateway which do the sampling of the data stream from sensor and send it to the Message Queue Manager(RabbitMQ) or send it to on-premises deployed model for prediction and processing.

   Input: Input data stream from the sensors.
   Output: Data readiness in the Message Queue(RabbitMQ) or Data streaming to the model.
   Invocation Mechanism: http protocol, opencv for streaming of sensor video/picture data.

Expected behaviour: Sensors should give un-interrupted live streaming data for processing it and sending it to Model directly or through Message queue.

3. Inference : User inference request comes to Gateway Action and Notification Service Managers. This services keep on i.e. inferencing based on the action defined by the user. Once user condition satisfies, it triggers the notification to the user.

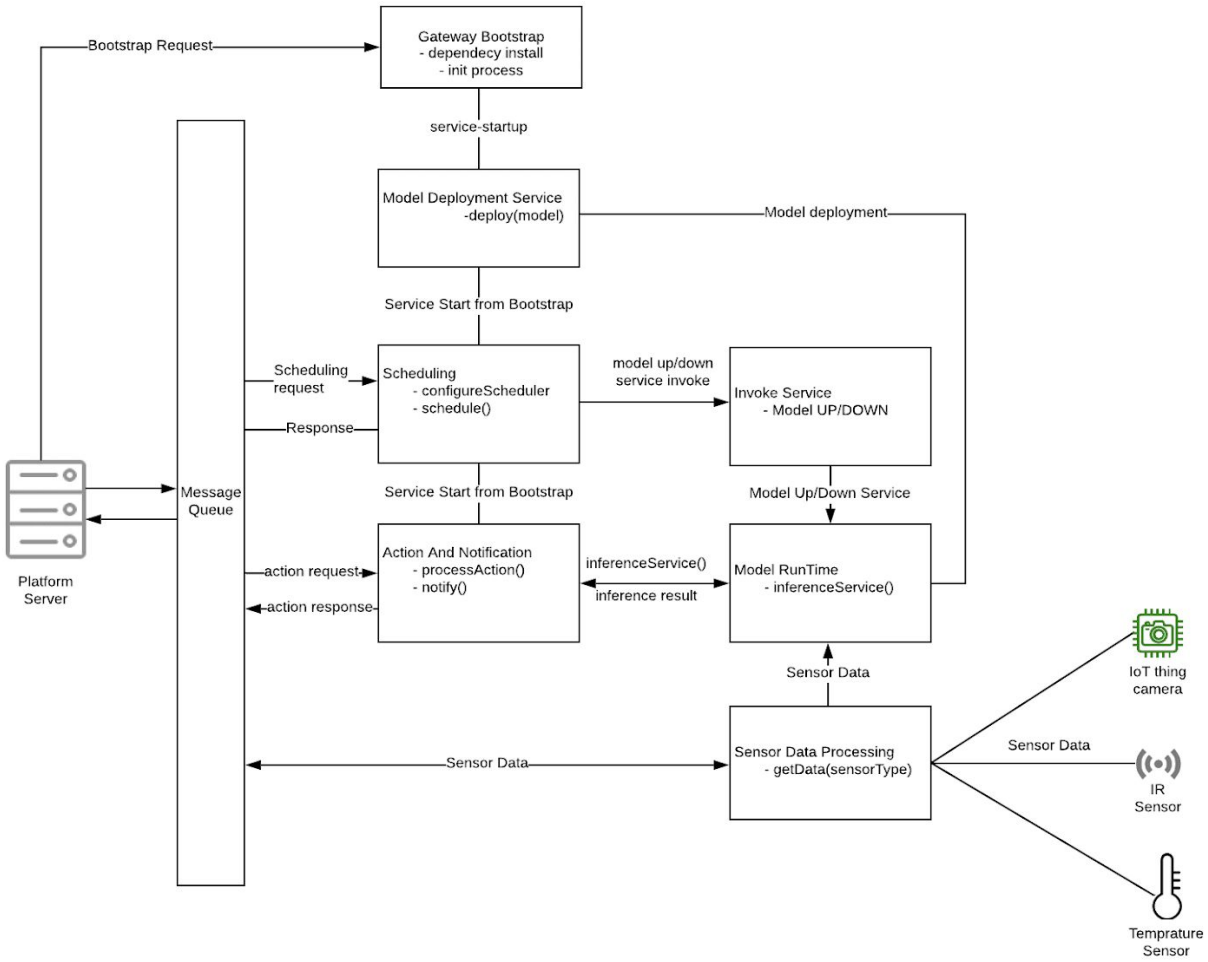
   Input: Image or Some other User data for prediction.
   Output: Status of the User request for inference.
   Invocation Mechanism: http services.
   Expected behaviour: Notification service should be doing the inferencing and do the prediction. If it satisfies user's condition, it does notify/alert to the user specified systems.

4. Scheduling : Scheduling information will be retrieved from configuration file provided by the user. Based on this scheduling details, Scheduler keeps model services up and down.

   Input: Scheduling details from config files..
   Output: Model up and down based on the scheduling detail.
   Invocation Mechanism: direct system call..
   Expected behaviour: Model(s) would be available based on the scheduling times only.

3. Solution Design Considerations

3.1 Design big picture



3.2 Environment to be used[Gateway Device Specification]

- 32/64-bit OS (Linux or Android OS powered Gateway)
- Minimum RAM requirement : 1GB
- Processor : Intel Atom® E3826, 1.46GHz(check NISE 50-IoT Gateway)
- SSH support to OS for transferring files and models.

3.3 Technologies to be used

- Tensorflow Lite on the edge
- Scheduler lib of python
- Bash Shell scripts can be used to make installation/configuration automated for deployment of model.
- In house Python based Notification and Action System
- OpenCV for Image and Video based Data Streaming from IoT Sensors/Cameras.
- RabbitMQ for queueing data stream to the Model.
- IoT sensor/Camera device drivers for preprocessing of the data stream.

3.4 Overall system flow & interactions

- Gateway Bootstrap program runs at the start of the platform and brings up the gateway.
- Gateway Bootstrap checks if all the dependencies are installed on the Gateway or not. If dependencies are not installed it calls INIT service to install dependencies. If it is already present it invokes service manager to bring up the model.
- INIT service will call the deployment service to install the model and its corresponding dependencies.
- Based on the request received by user scheduler will decide when to bring up or down a model.
- Scheduler will in turn invoke service lifecycle manager to bring up or down model service and inference service.
- Action and notification service will listen for request from server and based on the request process user input.
- Once user response in processed inference service will share the output . Based on the output predefined action will invoked (Provided by the user)
- Sensor data and processing will continuously receive data from different sensors attached to the gateway. This data is provided as input to either inference service or is sent to the server for further processing.

3.4 Gateway Registration

- When new user registers on our platform , gateway registration will be done based on input provided by user.
- User will provide type of sensors he needs to attached to gateway. These sensor type will be validated against out platform compatible sensor .(Based on config file)

- Each gateway will be registered against one user and entry about the same will me maintained on repository . This will be used by authorization service of the server.

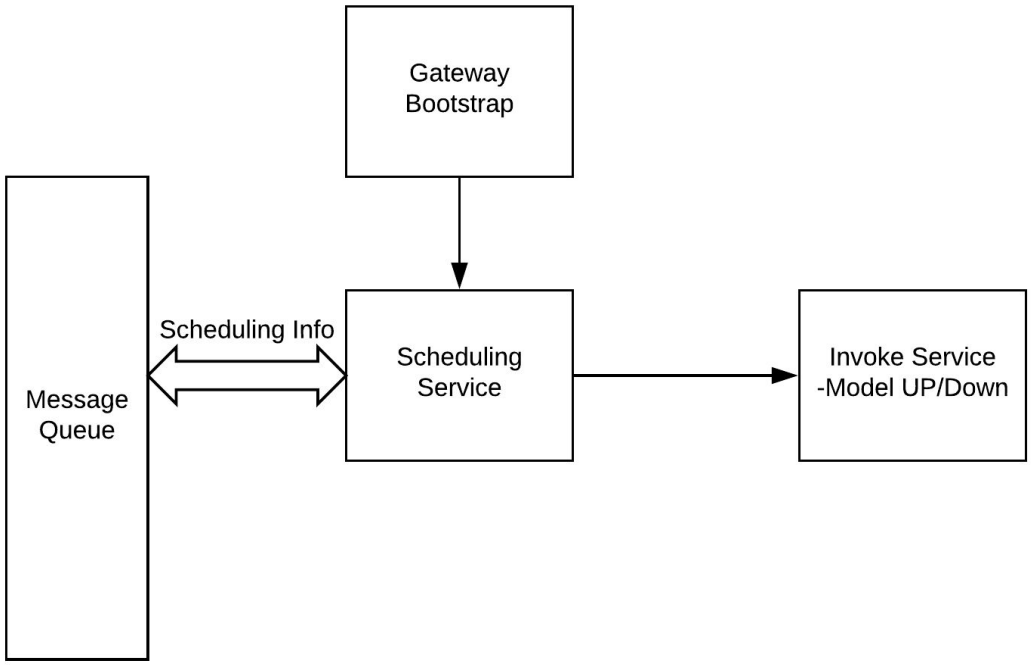
3.5 Model Deployment Service

Main feature of this module are

- Receives request from Gateway bootstrap.
- Gets latest model from central repository
- Deploys the model and its dependencies on the gateway.
- Informs scheduler about model deployment.

3.6 Scheduling service

Block diagram



Main functionality of this module are

- Receive request form bootstrap during start of the gateway to invoke scheduler

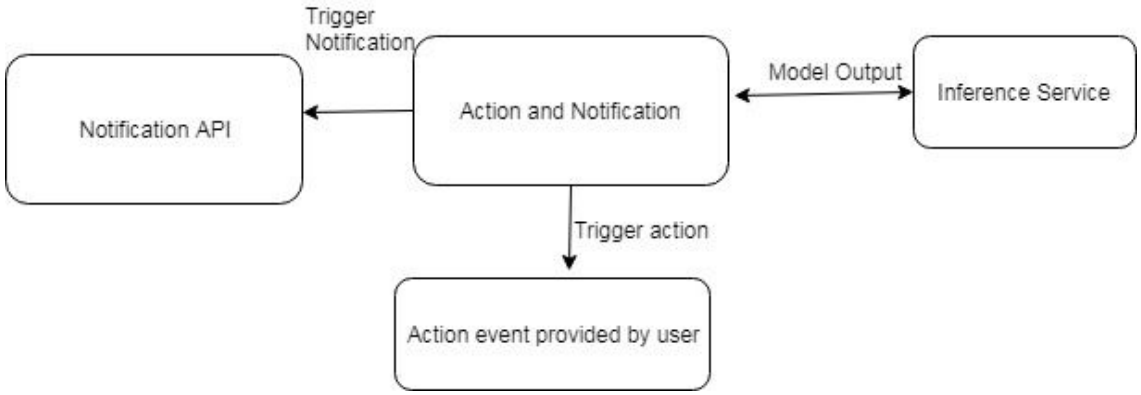
- Receives scheduling info from central server via message queue.
- Based on the scheduling info invokes service manager to bring up or down the model at provided time.

3.7 Inference Service

Main functionality of this module are

- This service is invoked during model run time.
- It processes the output based on input and sends the result to Action and Notification service

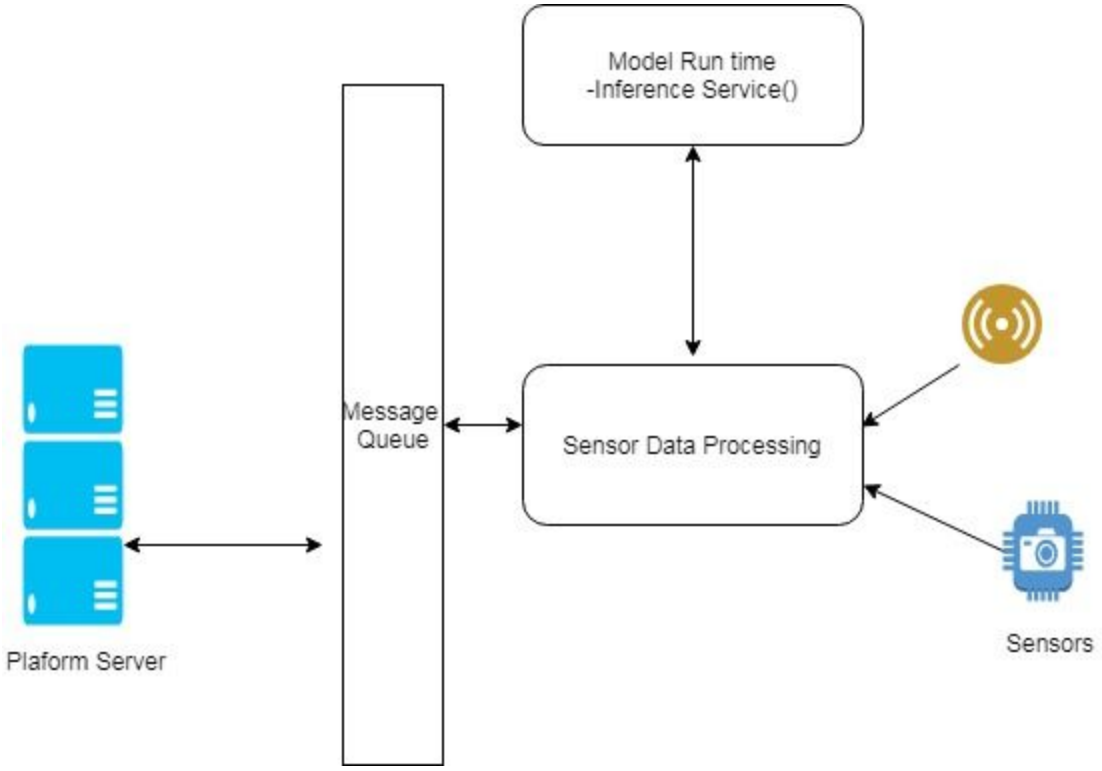
3.8 Action and Notification



Main functionality of this module are

- Receives the notification from inference service
- Invokes the action defined by user based on the notification.
- Also processes action request from server.

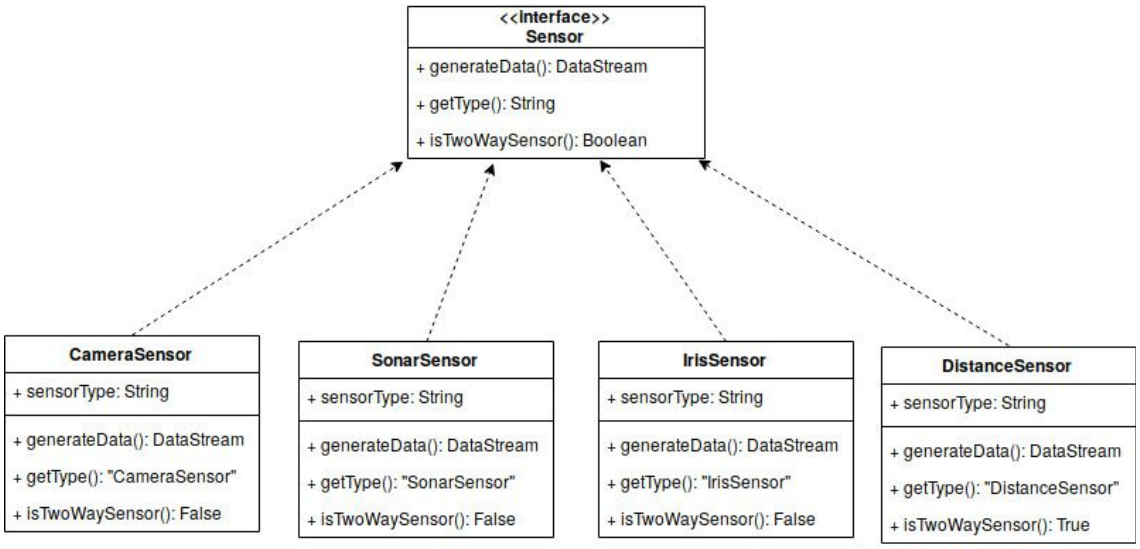
3.9 Sensor Data processing



Main functionality of this module are

- Gateway will have drivers installed for different sensors.
- It will process sensor data via RabbitMQ
- Sensor data will be provided as input to inference service.
- It will also be sent to central server via message queue.

3.10 Sensor Interface



3.11 Approach for communication & connectivity

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

3.12 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 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.

3.13 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 - json format
4. XML file - xml format
5. Properties file - properties format

4. Subsystems and APIs

4.1 Gateway Bootstrap

Sub-System Detail: 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.

API:

- **installDependecy()** : This API is inoked by the cloud server and bootstrap installs all the dependency into the Gateway.
- **initProcess()** : Initializes all the sub-system across this Gateway Services. This helps to recover from a previous crash of the Gateway services. It sends the Service start request to all the Sub-systems in this Gateway and starts the services.

4.2 Deployment Service

Sub-System Detail: 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.

API:

- **deploy(Model)** : Takes input as Model data structure which contains Model related information and Model URL from where Model is downloaded and deployed into the Gateway.

- **undeploy(Model)** : Takes input as Model data structure and undeploy any existing Model running/deployed on the Gateway. This API has been given as future scope to the platform to clear up the memory and Model running on the Gateway.

4.3 Scheduling Service

Sub-System Detail: 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.

API:

- **configureScheduler(CongurationDetail)** : configures the scheduler before hand. This service takes configuration details like Model and system details where the Model is deployed and details about the Invocation Service in the Gateway. Invocation Service is the System which is responsible to invoke the start and stop service. Scheduling Service does call the invocation service based on scheduled time.
- **schedule(SchedulingDetail)** : This service schedules the configured service based on the Scheduling details provided by the User or other system.

4.4 Action and Notification

Sub-System Detail: 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.

API:

- **processAction()** : This API is invoked on inference request. Here, inferencing happens based on user's inference input and on correct inference, action gets triggered from here.
- **notify()** : This API is to notify the user. Here user can be notified on email, sms or any configured way. This could be future scope of the platform and can be extended further.

4.5 Invoke Service

Sub-System Detail: Invoke Service invokes the system services of different sub-systems like service stra-up and stop, service recovery etc.

API:

- **serviceStart(ServiceType)** : This is to start a Service based on provided Service Type.
- **serviceStop(ServiceType)** : This is to stop a Service based on the provided Service Type.
- **pingService(ServiceType)** : This API is to check a service whether it is in UP or DOWN state.

4.6 Model RunTime

Sub-System Detail: This is where the Model runs and does expose the Model inference gRPC services for user inference request.

API:

- Note that this will have dynamic service APIs based on the User Model file and it's configuration details.

4.7 Sensor Data Processing

Sub-System Detail: 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.

API:

- **getData(SensorType)** : This API is provides the processed data. It takes Sensor Type as input and gives that sensor's data stream.

5. Storage/Memory 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.
- Scheduler Data Structures: runqueue (push_cpu,migration_thread,migration_queue)
- User Defined Data Structures : User defined Classes, Interface, Tree.

Storage Structure:

1. Sensor Information
    - Sensor Type
    - Sensor Location
    - Sensor ID
    - Gateway ID in which sensor is deployed

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
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

Gateway Config file

```
<?xml version="1.0" encoding="UTF-8"?>

<gateways>
    <gateway>
        <gatewayId>1</gatewayId>
        <gatewayName>Submarine1</gatewayName>
        <gatewayIP>10.42.0.1</gatewayIP>
        <gatewayLocation>Vizag</gatewayLocation>
        <sensors>
            <sensorID>1</sensorID>
            <sensorID>2</sensorID>
        </sensors>
```

```
    </gateway>
    <gateway>
        <gatewayId>2</gatewayId>
        <gatewayName>Submarine2</gatewayName>
        <gatewayIP>10.42.0.2</gatewayIP>
        <gatewayLocation>Vizag</gatewayLocation>
        <sensors>
            <sensorID>1</sensorID>
            <sensorID>2</sensorID>
            <sensorID>3</sensorID>
        </sensors>
    </gateway>
</gateways>
```

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>
```

6. Interactions & Interfaces

- User will not directly interact with the gateway.
- During initial registration user will provide details of the sensors to be attached to the gateway.
- Gateway will have different type of sensors attached to it . It will process sensor data and send it to Inference service on gateway or to central server.
- User can choose weather to run their model on central server or on gateway.
- Central server will interact with gateway via a message queue.

7. Persistence

- Using Serialization and pickle library in python we can make periodic backup of our trained model and make it persistent.
- We also used tensor serving library for versioning of model and make it persistent.
- We maintain persistent log file for continuous even triggering.
- In case system crashes, these services brings back the system in the last state.

-