PROJECT DESIGN OUTLINE

PREPARED FOR

IAS Project : IIIT-Hyderabad

PREPARED BY

TEAM - One

[Bhavin Kotak - 2018201071]

[Priyendu Mori - 2018201103]

[Rushit Jasani - 2018201034]

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]

Request Manager

1. Wrong API endpoint.

    Input: Enter wrong API endpoint

    Expected output: Appropriate error message

2. Unauthorized user accessing our API.

    Invocation method : Request to api from unauthorized user

    Expected output : Request not granted.

3. Authorized user accessing end-point

    Invocation method: Request to api from authorized user

    Expected output: Request granted

4. Authenticated user but unauthorized service.

    Invocation method: Request to api from authorized user for a service which they have no permissions.

    Expected output: Request not granted.

5. API Gateway down

    Invocation method: Request to api while api gateway is down.

    Expected output: Proper error message.

6. Multiple requests at the API

    Invocation method: Request to api from 2-3 user simultaneously.

    Expected output: All Request handled successfully.

7. Stress testing.

    Invocation method: Request to api from large number(1000-2000) of users.

    Expected output: All request handled successfully.

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/busy.

    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 server is up and running.

    Invocation method: Request a service first time.

    Expected output: Server is loaded and instance of service is executed.

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.

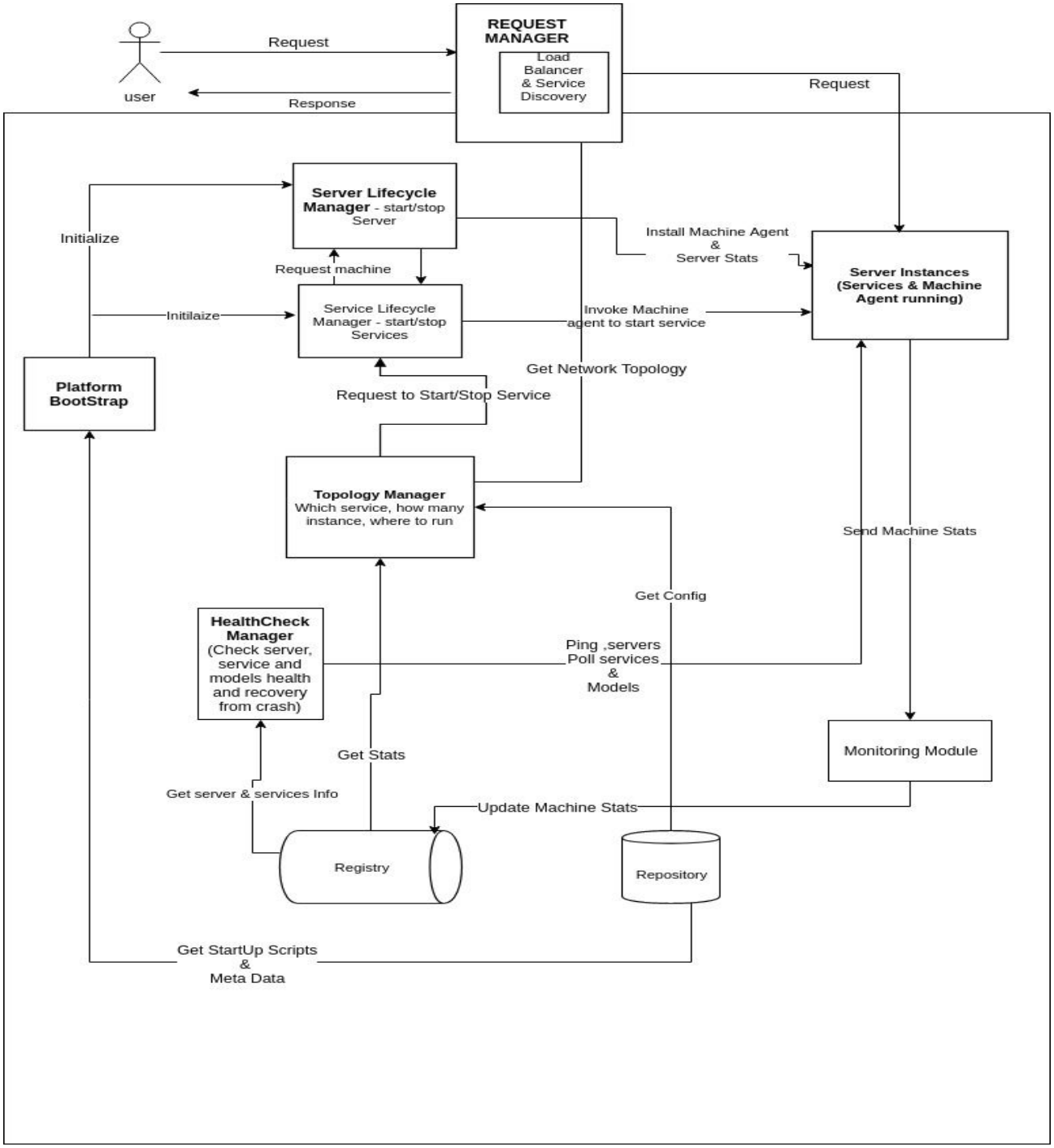
4. Exclusive service or AI model gets killed

    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.

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 flask restful APIs 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

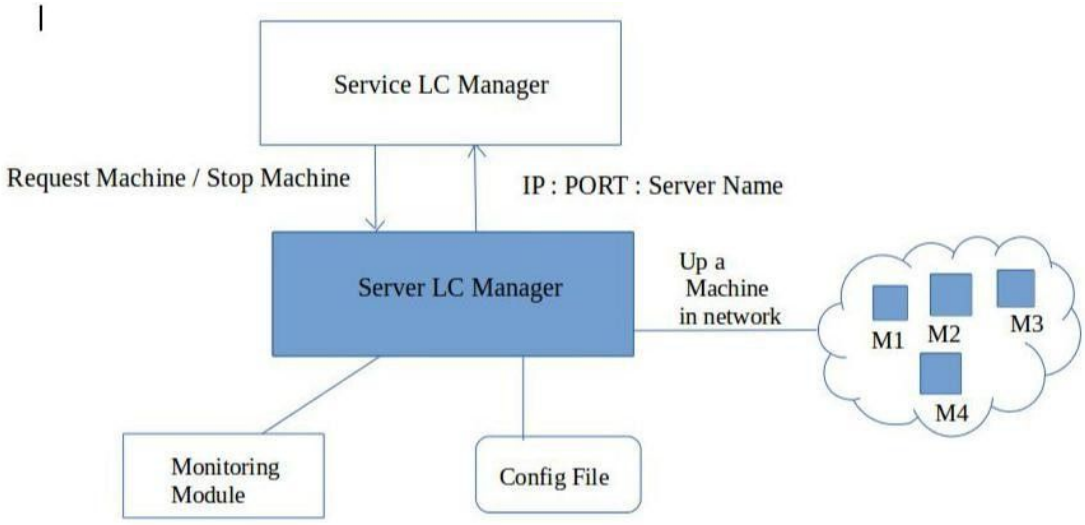
- Python and Flask framework should be primarily used for the development of the platform.
- Bash Shell scripts can be used to make installation/configuration automated.
- RabbitMQ must be used for MQ implementation, MySQL Database, NFS as common File System.
- Docker for containerization and running tensorflow serving models.
- Elastic search, logstash and kibana for centralized logging and monitoring

3.4 Overall system flow & interactions

- Bootstrap program runs at the start of the platform and performs following action:
    - Get startup scripts from repository.
    - Start Machine Agent & server stats in all the machine. Mount NFS in all the the servers.
    - Initialize server life cycle manager.
    - Initialize service life cycle manager.
    - Read which services are to be started at boot up.
    - Run these services using server and service lifecycle manager which in turn sends request to Machine Agent.
    - Start Service Discovery, Load Balancer and Health Check
    - Start WebServer/Request Manager.
    - Read information about gateways, sensors and store them in in-memory data structures to use them in other services.
    - Initialize topology manager.
- Request manager waits for the request.
- On arrival of request, load balancer and request manager decide where to route it and takes action accordingly.
- Gets responses from services and displays appropriate messages through request manager.

- Topology manager uses stats from registry and data from monitoring module to decide which services are overloaded and which servers are overloaded and sends appropriate message to server and service lifecycle manager to start instances of services which exceeds high mark.

3.5 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

3.6 Topology Manager

Functionalities:

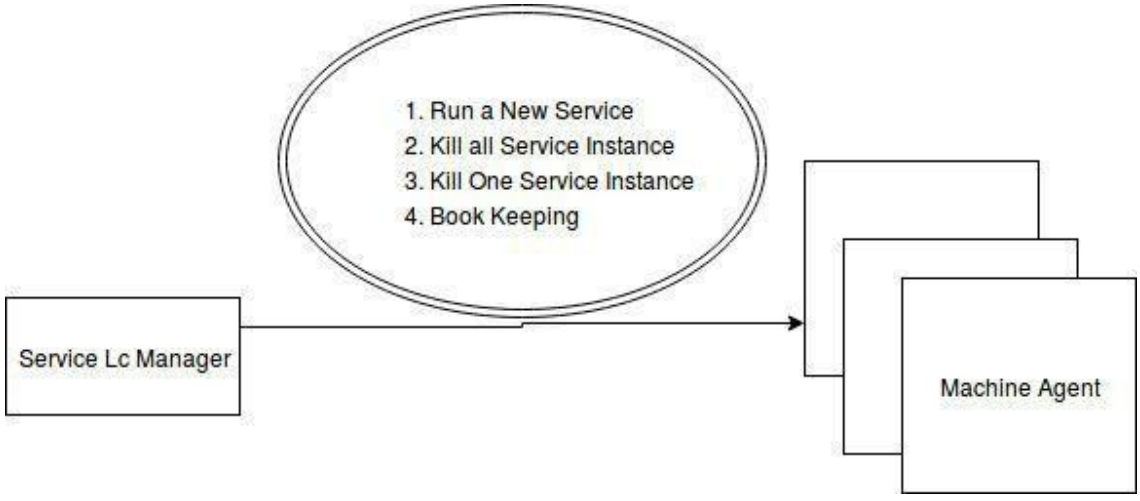
1. Contains run-time information of services
2. Load details regarding services and threshold info from Repository (Config)
3. Get information Machine stats info from Registry
4. Decides how many services to run
5. Where to run services
6. What services need to run
7. Request Service LC Manager to start/stop services
8. Update load balancer about Service instances for request routing

3.7 Platform BootStrap

This process does following things when the entire platform comes up:

1. Get startup scripts from repository.
2. Initialize server life cycle manager.
3. Initialize service life cycle manager.
4. Start Machine Agent in all the available servers
5. Read which services are to be started at boot up.
6. Run these services using server and service life cycle manager.
7. Start WebServer/Request Manager.
8. Read information about gateways, sensors and store them in in-memory data structures to use them in other services.
9. Initialize topology manager.

3.8 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)**

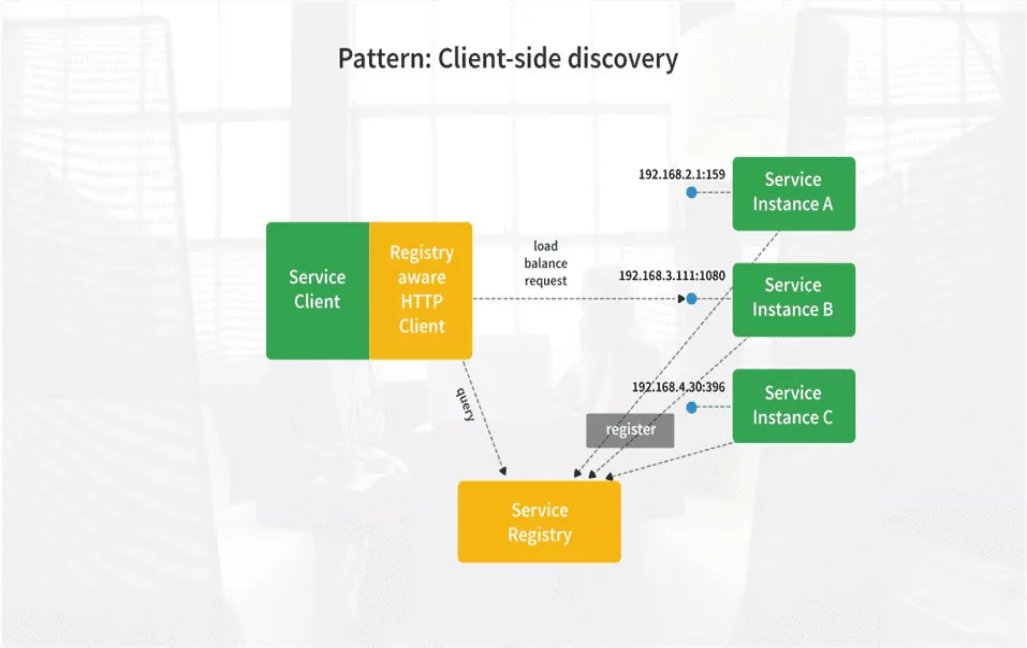
   Kill that service instance from docker container using input.

3.9 Request Manager

This module performs below given functionalities:

1. Get Request from user
2. Resolve service to invoke
3. Load balancer discovers the instance of the service to invoke out of multiple services
4. Receive response from the service invoked
5. Send response to the user.

3.10 Service Discovery Manager



This module performs below given task:

1. Each microservice will register itself with Service Discovery manager.
2. Services when killed/stop will be unregistered.
3. SDM has the running instances of all services
4. Client that wants to invoke any service, will first contact SDM for service URL.

3.11 Machine Agent

This module performs below given functionalities:

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

3.12 Health Check

This module performs below given functionalities:

1. 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.
2. 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.
3. 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.
4. 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.

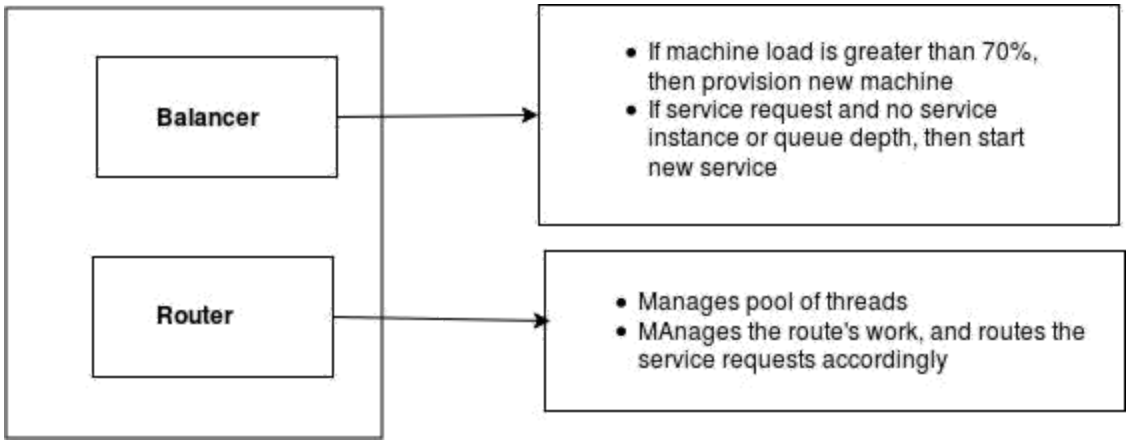
3.13 Approach for communication & connectivity

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

3.14 Registry & repository

- Registry will contain the stats of every service (what instances are running on what machines, binary files of each service in the common Network file system etc.)
- Repository will contain one directory per user which will in-turn contain one directory per models, services and will contain all the version-revisions of that service and models

3.15 Load Balancing



Main functionalities of this module are as follows :

1. Routing – Place the message in respective Service MQ
2. Monitoring :
    - Gather stats from Monitoring Module.
    - And using Rules (Memory Threshold,CPU Threshold,etc) send request to Service LifeCycle Manager or Server LifeCycle Manager.
    - The request for (Service LC Manager) can be :
        - Kill a Service instance.
        - Kill all Services on a particular machine.
        - Start a new Service instance.
    - The request for (Server LifeCycle Manager) can be:
        - Start Machine
        - Stop Machine

3.16 Interactions between modules

- All the modules running interact with each other by invoking REST APIs. Service first contacts the Service Discovery to get the instance url of the service it needs to interact with. SD responds with the instance url which can be invoked and communication can be done.
- Some services use RabbitMQ ,i.e, one service puts message on queue and other service picks up that message and acts accordingly. It's is used primarily for centralized logging and streaming sensor data.

3.17 Wire and file formats

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 and between services.
d. Each service when invoked using REST API will have an authentication token in the msg header.

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

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 Structure:

```xml
<?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>
```

1. Sensor Information
    - Sensor Type
    - Sensor Location
    - Sensor ID
    - Streaming rate
    - Capability (TWO - WAY/ ONE WAY)
2. Gateway Information
    - Gateway ID
    - Gateway Name
    - Gateway IP

    - Gateway PORT
    - Location
    - 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
6. Network Topology
    - Service Location
    - Number of services running
    - Server Instances
    - Server-Service mapping
    - Threshold values to start or stop services

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

- The system will be persistent with the use of NFS, databases and services that maintain the state of the platform.
- In case system crashes, these services brings back the system in the last state.
- In case any system or application service or running model crashes then it will be bring back by healthCheck service.