Deployment Manager

Internals Of Application Server

Team Report

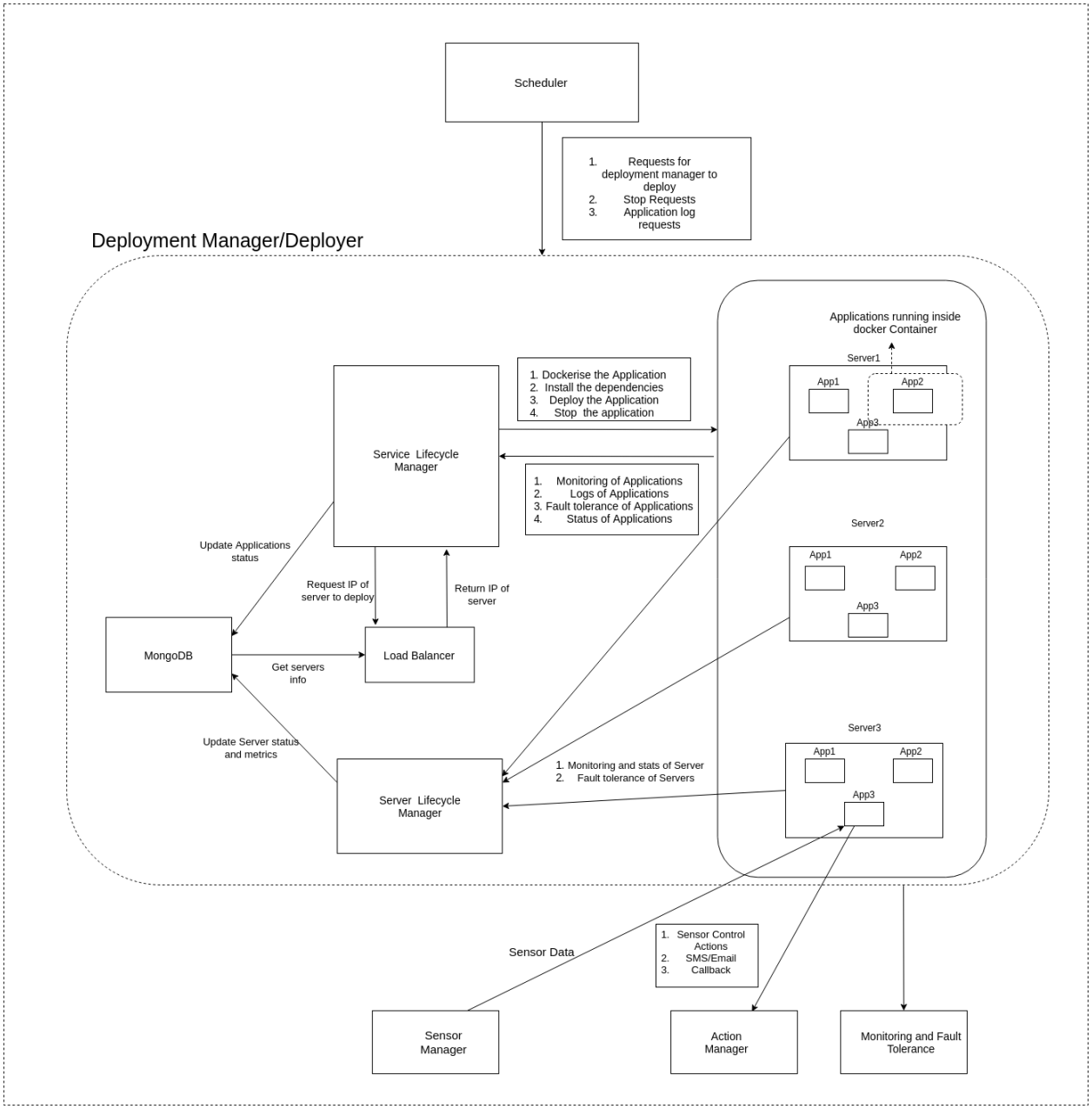
Group 6 Team 2

Submitted By :
AbhiRam DV (2020201030)
Janmejay Singh(2020201089)
Vinaya Sai Revanth Bachu(2020201090)

Index

1. **Introduction to the project** -

The Deployment Manager will receive scheduling requests from Scheduler. The deployment manager will then dockerise the application and deploy it in the appropriate server chosen by the load balancer. Deployment manager will monitor the applications, fetch their logs, and do fault tolerance on applications. Deployment manager will also monitor the servers on which user applications are running and provide fault tolerance on them.



Block Diagram of Deployment manager

Services Provided

1. Load Balancing
2. Dockerizing Applications and deploying on server
3. Monitoring of the applications
4. Logs of the applications
5. Fault tolerance of the applications
6. Monitoring of the servers
7. Fault tolerance of the servers

2. **Use Cases** -

- **Dockerizing Applications** - The application manager will accept and authenticate requests.
- **Load Balancing**- Web GUI Dashboard will provide an interface to the users based on their access rights.
- **Deployment** - The uploaded config files will be validated if they are in specified format.
- **Monitoring Status of Applications -** Users should be able to monitor the status of the applications running on the platform.
- **Stop the applications :-** The user should be able to terminate a running application on the platform
- **Logs of the Applications-** The user will specify schedules for algorithms.
- **Fault Tolerance of Applications :-** If an application is terminated abnormally, it must be restarted automatically
- **Monitoring of the Severs :-** If one of the
- **Fault tolerance of the Servers :-** If a deployment server goes down, it should be automatically brought up or start a new server for deployments.

3. **Test Cases** -

- **Check Application Deployment -** Check if the application has been deployed properly
  Input :- Deploy request from scheduler
  Output :- SSH into the deployment node and look for docker container status
- **Stop the Application-** Check if the application which is running has been stopped properly
  Input :- Stop request from scheduler
  Output :- SSH into the node and look for docker container status code. Also check if the deployer is not re starting the container as part of fault tolerance module
- **Monitoring status of Application :-** Monitor all the applications running in all the deployment nodes.

Input :- Check if the ServiceLCM is accumulating status of applications running inside docker correctly

Output :- Look at the mongo DB collection for the appropriate updation of the containers.

- **Get logs of Application -** Check if the scheduler is able to properly get the logs of the applications.

  Input :- Get logs request from scheduler.

  Output :- check if the application logs are proper and are matching with the real application logs.

- **Fault Tolerance of Application :-** Check if the application is being restarted in case of any down time of the servers

  Input :- manually Terminate one of the servers using aws dashboard.

  Output :- ServerLCM should be able to restart the server and the docker containers which are abruptly stopped must be restarted.

  Input :- manually attach to a container after sshing into the node and give a forced keyboard interrupt to terminate.

  Output :- the container should automatically restart.

  Input :- Stop the container using docker command api (using command “Docker Stop Container”).

  Output :- the container must be automatically restarted and the program should run.

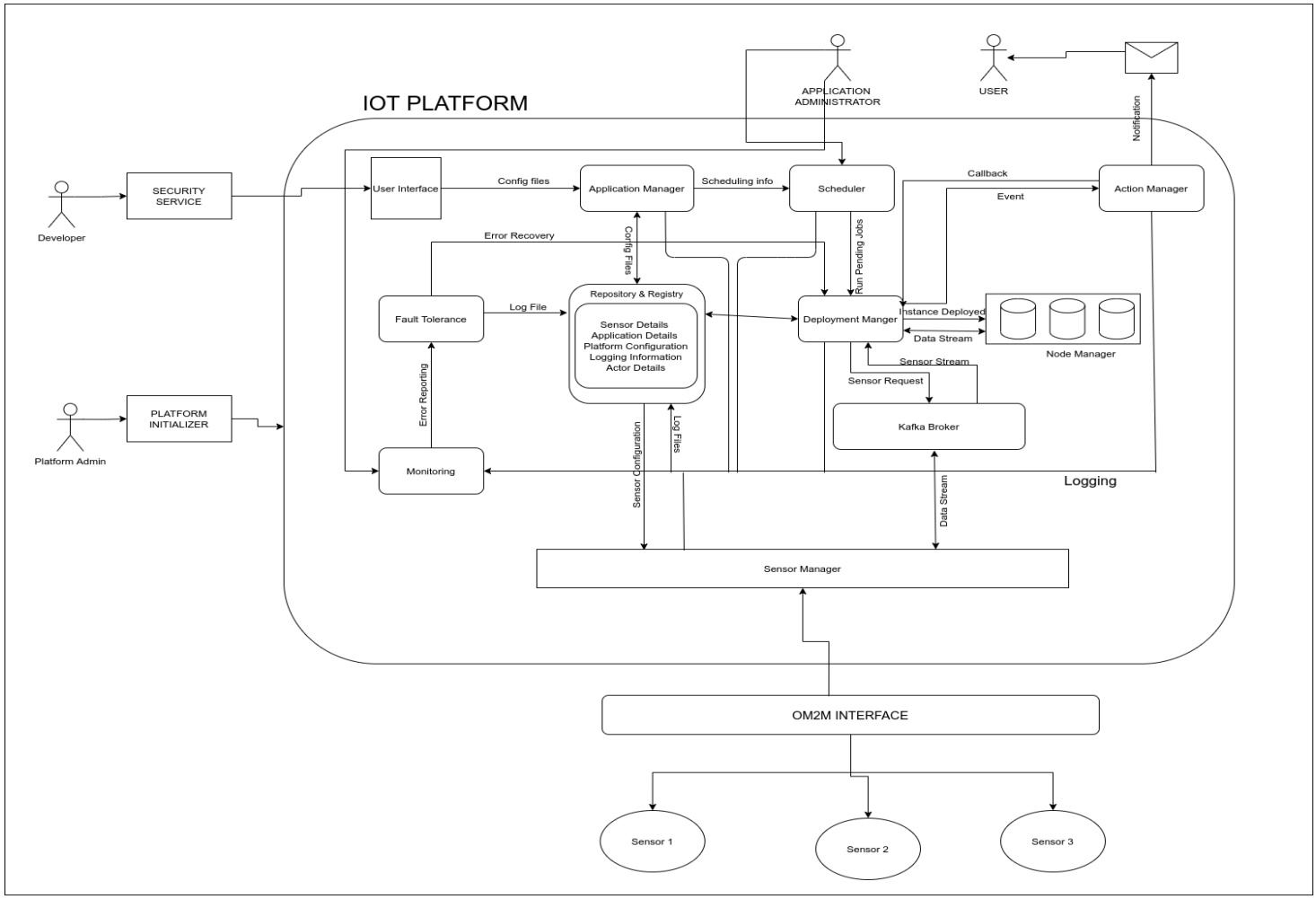
- **Monitoring of the Servers :-**

  Input :- Stop the deployment servers manually using AWS console manager.

  Output :- check if the down time has been correctly recognized and handled by the server LCM.

4. **Solution Design Considerations** -

4.1 Design Big picture



4.2 Environments to be used

- 64-bit OS(Linux)
- Minimum RAM requirements: 4GB
- Processor: Intel i3 5th Generation

4.3 Technologies to be used

Flask

Why: Tool provided by it will help to expose Api endpoints for the platform to use and it gives developers a good base framework of request/response management to work with. Each node is using flask to expose endpoints to the deployer to accommodate deploy requests and exposes other functionality like getting logs of containers and giving the status of all the applications running inside the node.

Kafka
Why: it will be used for managing communications among various modules. It is being used to accommodate requests from scheduler to deployer.

MongoDB
Why: For use as repository and Database

Python
Why: For deploying and developing platform

Schedule

Why: for updating the status to monitoring module periodically.

4.4 Overall System flow and Interaction

We will initiate the bootstrap or platform initializer program to initialize our platform. The bootstrap module will make sure that our platform will be up and running.

It will then start the additional external services like Kafka and the database for the repository.

After all the components are running, we will initiate the web interface.

The application manager will start the web interface and accept the request to authenticate it and then will ask for the zipped files.

After the zipped files are uploaded the application manager will extract the zipped files and validate them.

The scheduler will read the application configuration file and create a schedule instance. The scheduled instance will have an algoid , start_time , end_time ,job_type and then it will execute accordingly.

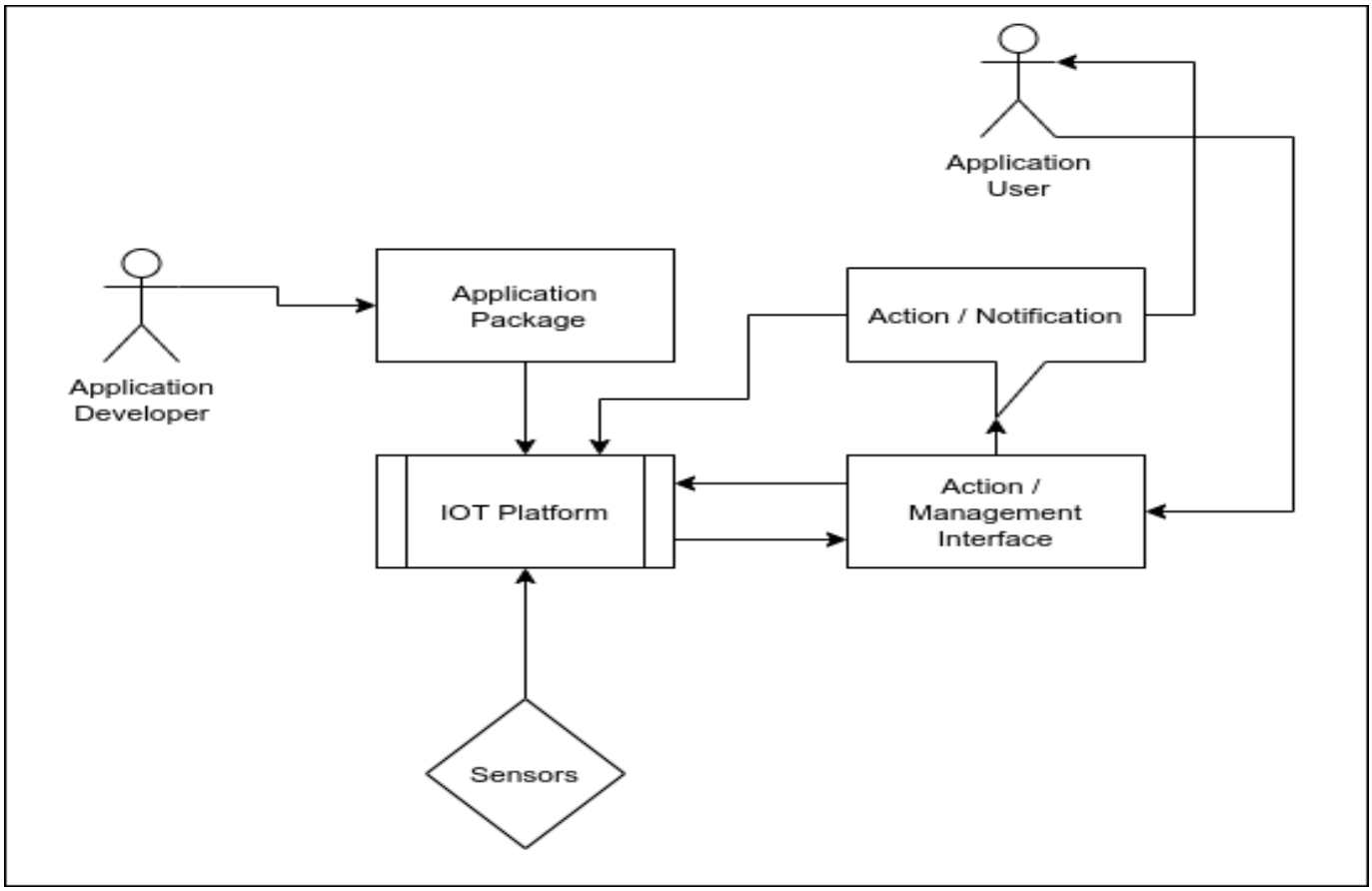
Once the scheduler triggers the deployment module at the time a particular algorithm must be run, the deployer will deploy the application, gives some status, and deploys the application.

The Deployment manager monitors the applications and servers, fetches the logs of the applications, provides fault tolerance to applications and servers

The algorithm running can also trigger some actions based on certain events and these actions are taken care of by the action Module.

The monitoring modules monitor all the platform components for their status and if some components go down then restores them to their running state.

4.5 IOT System Design:



The application developer will develop an algorithm or application which based on user requirements performs some operations on the data provided by the sensors and will send the result back to the users in required response format I.e. either a notification or some action on the sensor itself.

4.6 Approach for communication & connectivity:

We are using Kafka to enable communication between different components of the platform. Initially the different platform components will be registered with the Kafka broker and will be sending status/messages/acknowledgements in predefined format.

4.7 Registry & repository:

The Repository will be used to store different files related to different modules and will be separated based on the modules so that each module can access its repository and use them as required.

- **Application Repository** : To store application Configuration and executables.
- **Credentials Repository** : To store credentials of different actors interacting with the platform.
- **Scheduling Repository :** To store scheduling details of the applications.

4.8 Approach for Communication and connectivity(Deployment manager):

The Deployment manager takes the request for the service to be deployed using kafka topic, the deployer also accesses all the file and api dependencies of the application using a shared directory.

4.9 Interaction with other Components (Internal design overview)

★ **Platform Initializer and Others**
Platform Initializer passes configuration requirements to all the platform services and deploys them on servers after fetching them from the platform repository.

★ **Application manager and Scheduler**
The Application manager will send application service name that needs to be scheduled which will be forwarded to the deployment manager according to the schedule given by the user.

★ **Scheduler and Deployment Manager**
Scheduler passes the application and scheduling details to the deployment manager which uses it to deploy applications on servers.

★ **Application Manager to Sensor Manager**
Application Manager sends the details of sensors registered by the platform administrator to sensor manager.

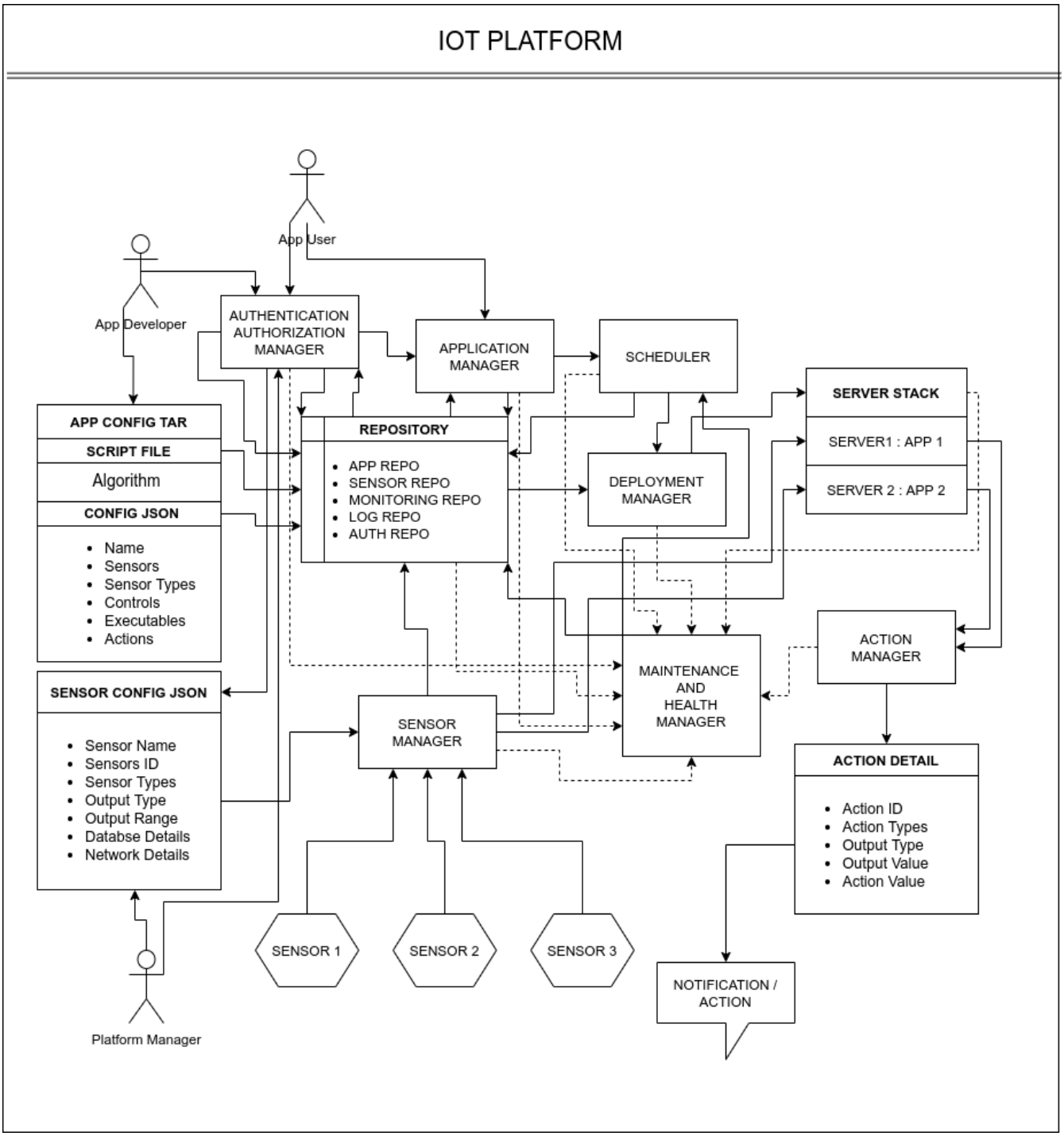
★ **Monitoring and Fault Tolerance and Others**
Monitoring module receives status updates, heartbeat pings and logs from all the modules and fault manager uses this information to reschedule failed services, applications and services.

4.10 Wire and file formats

- The main package files are provided in the zipped manner.
- The zipped files consist of the application, application configuration file, sensor configuration file.

- The application file is a python script.
- The application configuration file is json.
- The sensor configuration file is json.

5. **Application Model and User's view of system -**



Key elements

1. **Application Config file** - Contains info about the application used i.e., type of sensor used, name of sensor, controls, executables - all the scripts used in application.
2. **Sensor Config file -** contains details about the sensor used by the platform like sensor name, id, type, output type, output range, network details etc.
3. **Application Platform -** Sends config file to the platform and receives output from the platform.
4. **Sensor Registration -** if the application needs new sensors, then it will request the platform to add it, and the sensor registration module will store all information about the new sensor in the database and register it.
5. **Platform manager -** Manages the flow of the application and when a new application is received, provides it with resources.

Various Users

1. **Application developer** - develops applications, its algorithms and what all applications are going to do.
2. **Platform Admin** - Admin registers the sensors on platform.
3. **User -** Users who will be using the application.

Key Steps in whole application development Process -

Develop:

- Understanding the platform and develop application based on a Config file containing -
    - Application Name
    - Application Dependencies
    - List of Algorithms in Applications
        - Algorithm Name
        - Algorithm Id
        - Type and number of Sensors used by algorithm
        - Action performed by algorithm

Algorithm Script -

- Contains info about all the sensors to be used based on type.
- What elements are to be controlled.
- What type format of data is needed.
- Processing and interference from the data received.
- Notification to be sent.

- First a <app_name>.zip file must be provided by application developers which contains two pieces of information like app_config.json and scripting files. The app_config.json contains information mentioned above and scripting files contain actual algorithms.
- The developed application must be compatible with our platform.

Deploy and configure:

- Now these details can be used and our sensors should be registered and then details like the running of sensors is given by the developer according to which our sensor will run and now when the app dev wants any sensor data then he/she will request for that .
- And then sensor manager will provide the corresponding data from that sensor sensor manager and sensor manager will pass on data to the deployer which runs that on our algorithm and sends the output to the action handler, which will further process the output in a way that depends on sensor type.

Run and monitor:

- Contain info like when to run a particular sensor or whether it will run always or on schedule and monitoring of the system.
- Application will be executed on a docker instance with the dependencies running.
- Output of an algo instance, if it is not recurring, can be seen on going to the particular algorithm instance's page.
- Monitoring will be done on platform services and in case of a services failure the application will handle it with the fault tolerant system.

6. Key Data Structures -

Sensor Information Structure

- Sensor ID
- Sensor Type
- Location
- Sensor Status
- Sensor Input Data Type
- Sensor Output Data Type
- Sensor Data Rate

Scheduling Information

- Schedule Type
- Schedule Algo

- Schedule Start Time
- Schedule Stop Time

User Information

- User Password
- User email
- User Role

Algorithm Information

- Algorithm Name
- Algorithm Id
- Algorithm Sensors
- Algorithm Output Type

Registries and Repositories -

- Scheduling Registry - To store all schedules provided by the user.
- Application Repository - To store all the applications/algorithms.

7. **Interactions and Interfaces -**

Interaction with other modules -

1. Scheduler will send the details of the algorithm to the deployer which has to be executed. Apart from algorithm/application it will also share details regarding the dependencies that have to be installed before the request has to be executed. Application Manager will interact with scheduler's registry to store/update schedules and with deployment manager to get status of algorithms.
2. Applications running inside the docker containers inside the servers will request for sensor data from the Sensor manager using the SDK provided by the platform.
3. Applications running inside the docker containers can send sensor control actions/email/sms to the Action manager with the help of SDK provided by the platform.
4. Deployer will send its health status to the Monitoring module for fault tolerance.

8**. Persistence -**

- **Deployment Information -** The Requests for deployment is stored in a DB collection
- **Application Status -** The status of the application is updated periodically and is stored in the DB collection.

- **Deployment Server Repository -** The status of the deployment servers and their metadata is periodically monitored and is updated in the DB collection .

Hence even if the deployment manager goes down, the above mentioned points will help in maintaining fault tolerance and persistence at the component and platform level