IAS PROJECT

REQUIREMENT DOCUMENT

GROUP 3

TEAM 1
Jatin Sharma (2022201023)
Harsimran Singh (2022201049)
Adarsh (2022201081)

TEAM2
Ujjwal Prakash (2022202009)
Priyank Mahour (2022201047)
Sayantan Trivedy (2022201019)

TEAM 3
Yash Pathak: 2022201026
Vivek Kirpan: 2022201071
Prasad Kawade: 2022202006

TEAM 4
Udrasht Pal (2022201020)
Vikram Raj Bakshi (2022201044)
Dishant Sharma (2022202019)
Nikhil Chawla (2022201045)

TEAM 5
Harshit Kashyap (2022201050)
Prashant Kumar (2022201058)
Rishabh Gupta (2022202011)

1. OVERVIEW

a. INTRODUCTION

The aim of this platform is to provide an environment that simplifies the creation of applications that connect to and use location-sensitive IoT devices. It also offers the ability to run user-defined procedures that leverage such devices.

The platform provides a client interface for uploading applications, connecting to suitable IoT devices, and running them on required sensor data. The client is also given a UI-based dashboard for uploading and running applications, a set of APIs, and an algorithm scheduler service for running applications.

Once an application is running, the platform provides several capabilities to users, including a binding service for connecting sensors to algorithms, a deployment service for running executables in response to algorithm-generated events, a notification service for pushing notifications to registered users, an event service for controlling IoT devices based on algorithm-generated events, and a scheduling service for periodically running applications.

Overall, this platform aims to make it easier for developers to manage sensor data from IoT devices and create effective applications by offering a comprehensive set of services and tools.

b. SCOPE

The platform should be able to provide the following deliverables:

- Enable application developers to create an application that can connect and use IoT devices with ease.
- Manage sensor data and use it effectively.
- UI and APIs for running applications.
- Provide various services such as Deployment service, Notification service, Scheduling service.
- Perform remote actions on the IoT devices and provide a dashboard to monitor the devices.

2. INTENDED USE

a. INTENDED USE

The intended use of the IoT platform is to provide developers and users with a platform that simplifies the process of utilizing sensor data from IoT devices and creating effective applications. It aims to offer an environment where users can easily connect to location-sensitive IoT devices, manage sensor data, and run their applications.

The platform provides sensors and an interface to allow applications deployed on it to communicate with the sensors. Users do not have to worry about configuring the sensors, managing communication mechanisms, load balancing, node management, and other essential features of application development and deployment. Instead, users can deploy instances of the application they want to use with minimal effort. When a user submits a request to deploy an application or use an existing instance, the platform takes care of all the necessary processes to make the application usable by the user.

b. ASSUMPTIONS AND DEPENDENCIES

Assumptions -

- The IoT platform assumes that the sensors/devices it will be connecting to have the necessary hardware and firmware to enable communication with the platform.
- The sensor data will be processed through OneM2M Server (APIs)
- The application developers should provide a configuration file in the provided format.
- The platform assumes that users will have some understanding of how to use the platform's APIs and dashboard to develop and deploy their IoT applications.

Dependencies -

- The IoT platform depends on the availability and functionality of the sensors/devices it will be connecting to for data acquisition and communication.
- The platform depends on the availability of suitable infrastructure and resources (such as servers, storage, and networking) for hosting and running the platform and the applications deployed on it.
- The platform depends on the security of the underlying systems and infrastructure to protect the confidentiality and integrity of the data being transmitted and processed.

3. SYSTEM FEATURES AND REQUIREMENTS

a. Platform Requirements

i. **Deployment of the Platform :** Deployment of the platform is complex and important and needs to be done based on the functional and non-functional requirements. We need to have an estimate and a clear understanding of the types and volume of sensors that will connect and the scale of the subsequent platform.

We can deploy the IoT platform on-premises, cloud (IaaS) or in a hybrid model where we leverage the advantages of both.

This may involve setting up servers, configuring networking and security settings, and ensuring that the platform is properly integrated with other systems in your IoT ecosystem.

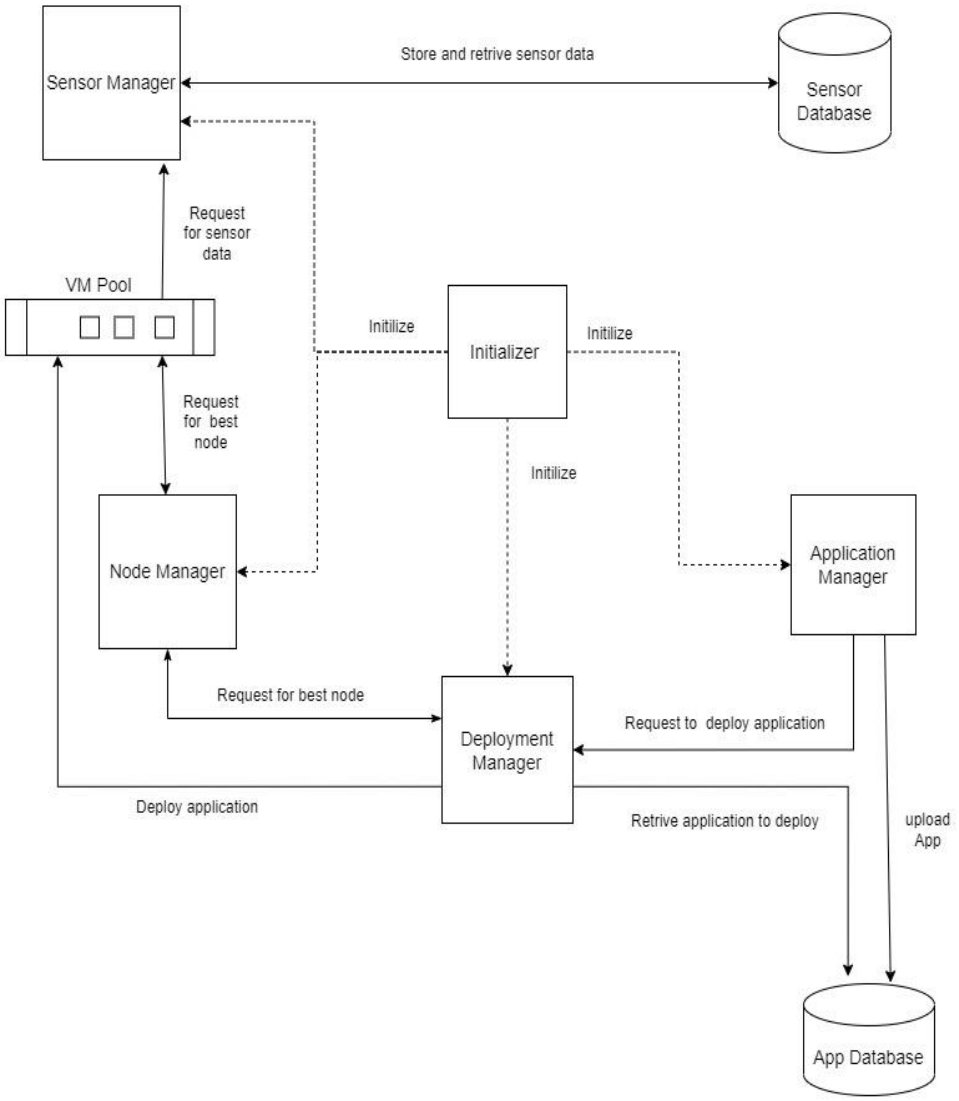
Once we start building the platform, we'll need to test it. Thus, we can deploy a test environment as discussed above and can run all test cases on it. Once all the development is done, we can have a production environment. This environment will be available for end users to use.

Monitoring and maintenance will also be a part of the deployment process as it ensures that the platform is running effectively and efficiently.

ii. Different actors on the platform

- IoT devices
- Platform Administrators
- Platform Developers
- Platform Maintainers
- END Users
- Application Developers

iii. Application Overview of the platform



b. Functional Requirements

i. Application

1. **Registering sensors:** When a sensor is added to an IoT platform for the first time or when the platform is initialized, it is necessary to register the sensor in order to retrieve data from it. The registration process involves storing information about the sensor, such as its sensor ID, input/output type, data rate, and location, in the platform's repository.

2. **Interaction with IoT sensors:** Each sensor has unique characteristics that enable it to be identified uniquely.

   OneM2M brings together all components in the IoT solution stack. It avoids reinvention in favor of reusing existing technology components and standards. oneM2M's architecture defines a common middleware technology in a horizontal layer between devices and communications networks and IoT applications. The OneM2M standard provides a universal M2M service layer that can be integrated into various hardware and software to connect these IoT sensors. The horizontal architecture of OneM2M allows for common service functions to be provided, enabling multiple applications to use a common framework.

3. **Development of Application on the platform:** Application manager of the platform provides details of the platform. Application developers use this information to create applications over the platform. Application developer creates and provides configuration files to the platform based on which an instance is created. Application manager receives this request and saves the configuration files. These files are forwarded to other managers during the communications and they start working accordingly. For eg: Sensor manager will parse Sensor Config file and create schemas, tables, etc for these sensors. After the configurations are instantiated, application meta and scheduling information needs to be provided.We can also select any particular services that need to be used by the Application (APIs) or workflows.
4. **Development of Workflow by the Application Developer:** The application developer will make use of the documentation provided and make a workflow.json which will be deployed and given as an end point to the application developer which he can use further.
5. **Identification of sensors for data binding:** During the registration of a sensor, all of its important details are stored in the platform repository. Additionally, each sensor has unique properties that allow it to be identified uniquely.When the sensor manager receives a data binding request, it uses the provided parameters to identify the corresponding sensor and sends the requested data in the specified format for binding.
6. **Data Binding to the application:** If an application instance on a node requires data from a sensor, it will communicate with the sensor manager and provide the sensor's ID. The sensor manager will use the ID to locate the correct sensor and send the requested data to the application in the required format. Conversely, if the

application needs to send data to the sensor, it will follow the same process.

ii. **Scheduling on the platform:** The scheduler retrieves the configuration files for an application or procedure from the repository and validates them. After validation, it checks the start time, end time, and priority of the job, and adds them to a priority queue. It can have other details as well like authorisation, etc. It is added to the queue only once, while in the case of a cron job, it is added repeatedly until it finishes. When the job's start time arrives, it is sent to the deployer to be executed.

iii. **Acceptance of scheduling configuration:** The scheduling information in the form of a meta file is provided by the application developer and stored in the repository. This meta file can be in JSON format. When the application needs to be executed, the Application Manager retrieves the application and the meta file and passes them to the Scheduler. The Scheduler then parses the meta file, extracts the necessary scheduling information, and validates it.

iv. **Starting and stopping services:** After the Scheduler accepts and validates the scheduling information, it checks the start time, end time, and priority of the job and adds them to a priority queue. For normal jobs, the job is added to the queue only once, while for cron jobs, it is added repeatedly until its end time.

Once the job's start time arrives, it is sent to the deployer for execution. Once the job's end time is reached, the scheduler sends a signal to the deployer to stop the job.

v. **Communication Model:** Our platform requires several types of communication:

- Sensors produce a continuous stream of data that applications consume as needed.
- Nodes, services, and applications need to be continuously monitored for fault tolerance.
- There is a need for inter-module communication (such as communication between the deployer and node manager).

To facilitate asynchronous continuous data communication, we can use Kafka. For inter-module communication or intra-module duplex communication, we can use socket communication.

vi. **Server and service life cycle:** The Server Life Cycle Manager is responsible for managing the running server's life cycle and keeping track of available and free nodes. It acts as a node manager and provides the Service Lifecycle Manager with a list of active nodes or a new node when requested. It also takes care of currently occupied nodes.

The Service Lifecycle Manager serves as a deployment manager that receives service details from the scheduler. It communicates with the Server Life Cycle Manager to obtain the address of the node or server where the service will be executed. Once it obtains the address, it deploys the service on that specific node or server.

vii. **Deployment of the Application on the platform:** After the Scheduler hands over the job to the Deployer, the Deployer assesses whether a standalone environment or shared environment is required for the job. If a new node is needed, the Deployer requests one from the Node Manager.

It selects an active node to run the job based on the list provided by the Node Manager. For shared environment jobs, the Deployer employs a Load Balancer to pick the node with the least amount of load. When the job or algorithm finishes running and the node is no longer in use, it is returned to the Node Manager.

viii. **Registry and Repository:** Registry and Repository will be used to store the information for different platform modules like User data, Application data, Node Info etc.

This will also be used to store queries, artifacts and code of the applications.

ix. **Load Balancing:** Load balancing is the process of distributing a set of tasks over a set of resources, with the aim of making their overall processing more efficient. Load balancers are used to increase the capacity (concurrent users) and reliability of applications. Its purpose is to ensure that incoming traffic is efficiently distributed among multiple nodes or servers to prevent overloading any single node.

First the application manager gets a request to deploy the application. Then the application manager sends a message to the deployment manager to deploy the application. The deployment manager then deploys the application on VM having best health (health information is provided by node manager). Till now only the original instance of the application is deployed. Its end point is not yet known by anyone except the deployment manager.
Now the deployment manager sends the ip:port of the application, name of the application, name of the docker_image of the application, etc to the load balancer.
The load balancer upon receiving this information creates a config file for this application which contains the original endpoint of this application

and the ip:port of the proxy that it creates for it. This is the final end point of the deployed application. This end point is sent to the deployment manager which is further sent to the application manager. Now this endpoint is sent to the end-user.
The load balancer also updates its config files AppDetails.json (which contains all the details about the deployed application, like its end point, number of instances, container ids, etc), VmDetails.json(which contains details of all the VMs in the VM pool like their public ip, the apps they host, the port number available for a new app to be deployed over it), AppHealth.json (which contains name of all the apps, and a list corresponding to each app which has the CPU and RAM usage of all the containers of that app), Nginx.json (which contains the latest port on which we can create a new NGINX proxy server).

x. Interactions between modules:

1. **Application Manager and Scheduler:** The Scheduler gets information from the UI to schedule the app. This contains the start_time (time from now in seconds after the app has to be deployed) and the end_time (time from now in seconds after the app has to be stopped) . The application details are already known because we have already selected the application which is to be scheduled.
   Now there are two functions which are run in separate threads in the scheduler. One is to send start_app message to the application manager and second is to send stop_app message to the application.
   These threads are delayed as per the start and end time provided by the user.
2. **Application Manager and Sensor Manager:** The sensor manager is responsible for sharing the sensor data with the application manager, which will then store it in the application database along with other relevant application information and passed to the scheduler in the form of metadata with application's other information.
3. **Scheduler and Application Manager:** Upon receiving the scheduling information from the UI, the scheduler sends a message to the application manager to start or stop the app at the specified time. Then the deployment of the application is done as usual.
4. **Deployment Manager and Node Manager:** The deployment manager needs the ip and port of a VM to deploy an application. This is the VM which should be the healthiest. The deployment manager sends a message to the node manager to send the address of the VM with best health. The node manager replies to this by sending the ip and port of the VM with best health. It

iterates over all the VMs present in the VM pool and checks the health of the machines and finds the one with best health.

5. **Sensor Manager and Nodes:** If there is an application instance running on a node, it may need to establish communication with the sensor manager to exchange data with one or more sensors. This could involve sending or receiving data streams.
6. **Monitoring and Fault tolerance and other modules:** The module responsible for monitoring and fault tolerance interacts with other modules like the Scheduler and Sensor Manager. It performs periodic checks on these modules to ensure that they are functioning correctly. If any issue or fault is detected, the corresponding instance of the fault tolerance module is triggered, and an alternative node is assigned to run the affected module.
7. **Platform Initializer and all other modules:** Platform initializer will initialize all the modules to get them up and running.

xi. Packaging details:

- **Application zip:** This collection comprises configuration files and scripts.
- **Controller instance configuration zip:** This collection of files consists of configurations for controller instances.
- **Controller type configuration zip:** This set of files includes configurations for various types of controllers.
- **Sensor instance configuration zip:** This collection of files consists of configurations for sensor instances.
- **Sensor type configuration zip:** This set of files includes configurations for various types of sensors.

xii. Interaction of different actors with the platform:

1. **User**
    a. The actor in question will make use of all the applications and their associated services on the platform.
    b. To utilize the application and its services, the actor is required to complete the registration and authentication process on the platform.
    c. It is the responsibility of this actor to initiate the application that has been deployed by the actor-Application developer.
2. **Application developer**
    a. When deploying the application on the platform, the actor must include important information such as sensor data and other necessary details in the configuration file.
    b. The actor has the capability to ask the analytic module to examine the current status of the application that they have deployed.

c. If deemed necessary, this actor will perform the necessary updates to the application.

3. **Platform admin**
    a. The platform administrator will be responsible for registering sensors.
    b. The platform administrator has the ability to request the current status of the platform from the analytic module.
4. **Platform developer**
    a. The individual or group identified as this actor is responsible for initiating and terminating all major components within the platform.

c. Non-Functional Requirements:

i. Fault Tolerance:

1. **Platform:** Fault tolerance is a crucial aspect of an IoT platform. The platform should be able to detect and recover from failures in components like the scheduler, deployer, and application manager.
2. **Application:** The applications should also be designed to handle failures, with multiple instances running in parallel and a failover mechanism to switch to a backup instance in case of failure. This ensures uninterrupted service delivery to the users even in the face of failures.

ii. Scalability:

1. **Platform:** Scalability is an important aspect of an IoT platform, and our platform is designed to be highly scalable. It can register any number of sensors to the platform, making it flexible to handle a large amount of data.
2. **Application:** If an application instance is consuming resources above a specified threshold, it will be shifted to a new node at runtime, which exclusively belongs to that instance. This allows for efficient use of resources and ensures that the application can scale with demand.

iii. Accessibility of data:

1. **Application:** Application should be able to access sensor data. In the application, sensors are specified in the configuration file and the application is able to access their data.
2. **Sensors:** Sensors can receive data from the application based on events. The platform ensures that data is accessible to the appropriate applications and sensors.

iv. Specification about Application:

Applications built using the IoT platform will specify which sensors they will use in the configuration file.

Applications can also define their own algorithms to process the sensor data and generate useful insights.

v. **UI and CLI for interaction:** The IoT platform provides a UI-based interface for the application developer to upload sensor and controller configuration files. The application developer can upload the application code and the required sensor information in a zip file through the UI. The user can deploy the desired application at any place and time using the UI, and can also view notifications for different applications deployed by them. Additionally, the platform may also support a CLI (command-line interface) for advanced users who prefer to interact with the platform using commands.

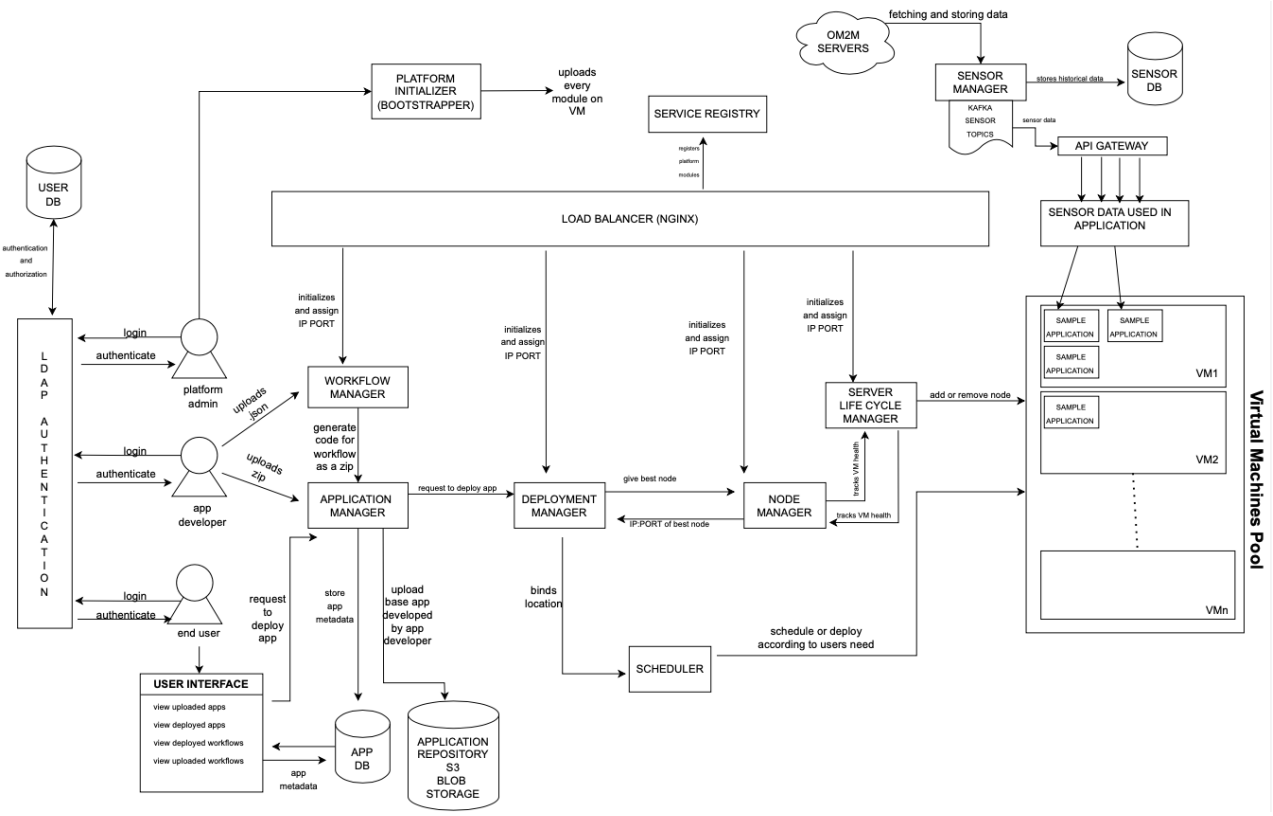
vi. **Security:** Authentication and Authorization
    1. **Authentication:** To access the platform, users need to authenticate themselves with the authentication manager. Once authenticated, users can access the applications or services on the platform.
    2. **Authorization:** The platform admin can view the current status of the platform. Application developers are authorized to view all the status information of the applications they have deployed on the platform.

vii. **Monitoring:** Monitoring keeps track of the health and performance of the platform and the applications running on it. This includes monitoring the system resources, such as CPU and memory usage, network connectivity, and sensor data. The platform provides real-time monitoring and alerts for any issues or anomalies that may occur. The monitoring system generates reports and logs to help diagnose any issues that may arise.

viii. **Persistence:** The IoT platform stores data related to application and platform subsystems in a Relational Database to ensure persistence. In case of any faults or discrepancies detected in the modules, the platform can retrieve the state from the database and the fault tolerance module can re-initialize the affected module. This ensures that the system can recover from errors and continue to function without any loss of data.

4. The Key functions

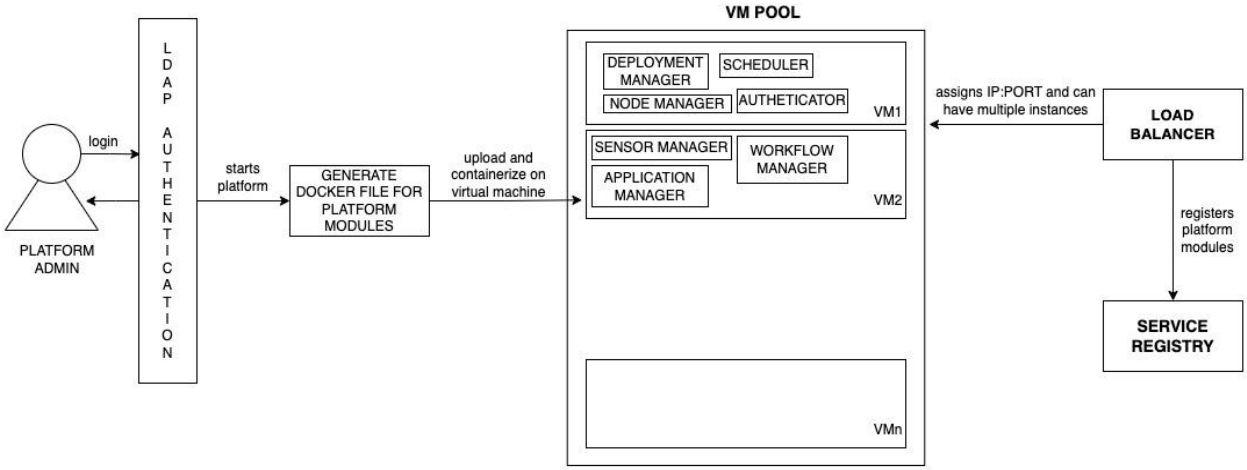
a. A block diagram listing all the major components:



b. Description of each component:

i. Platform Initializer:

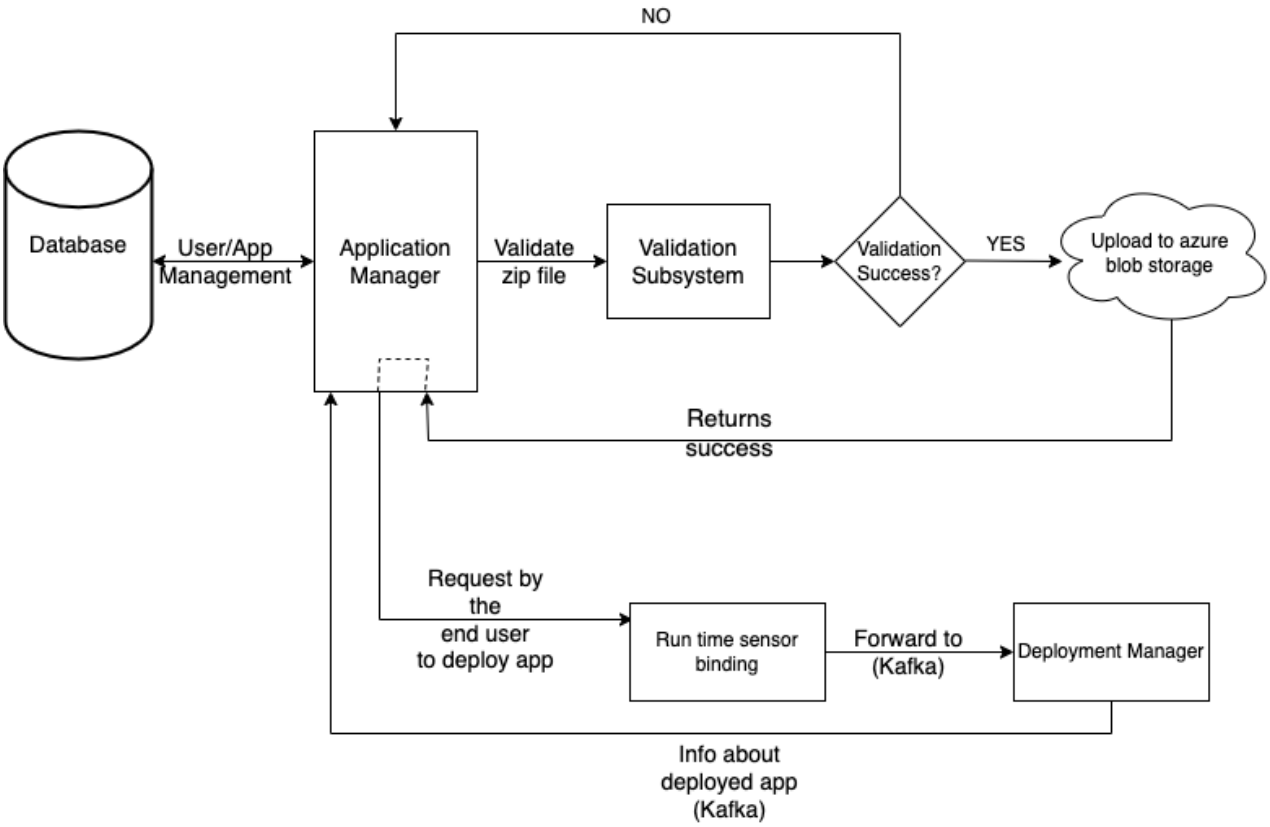
PLATFORM INITIALIZER



Platform initializer can be considered as a bootstrapper module. It reads the configuration file when it is running and gets the information of all modules. It initializes all the modules (services or various managers) using the gathered information after which the algorithm can be executed.

Platform initializer will create the necessary infrastructure to support our IoT platform. It will achieve this by initializing virtual machines on the host and setting up the environment on virtual machines.

ii. Application Manager:

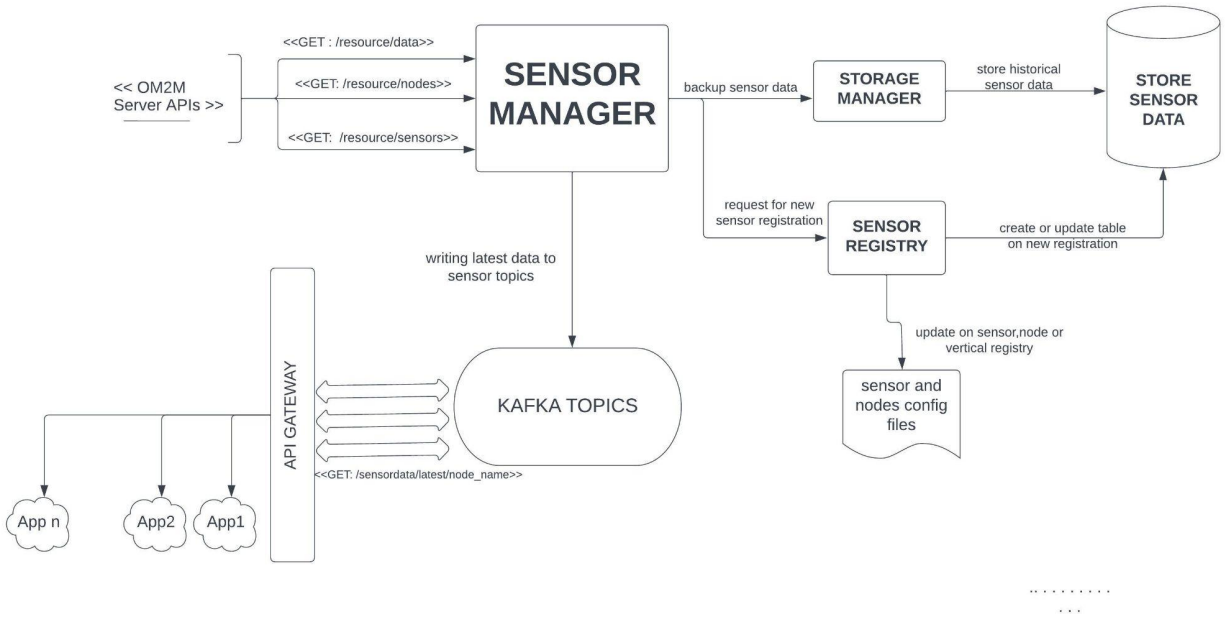


In this system, various modules work together to manage the deployment and execution of applications on a platform. The authentication module verifies actor credentials, while the platform configurer adds sensors and controllers to the platform. The application manager validates uploaded files and deploys applications based on user preferences, which can be scheduled for execution using a scheduler. Finally, notifications from the application are sent to a notification page for user access.

Application developers can use this module for sign up and login. After login, The developer can upload the algorithm config zip file. The module will validate the structure of the files. It is also used to show all sensory data like sensor type list and input sensor list.

iii. Sensor Manager:

SENSOR MANAGER



This module is responsible for enabling communication between applications deployed on our platform and external sensor infrastructure. Its main functionality includes registering new sensor types and nodes, retrieving data from external data sources, identifying application requirements, binding sensor data with applications, and storing sensor data in a database.

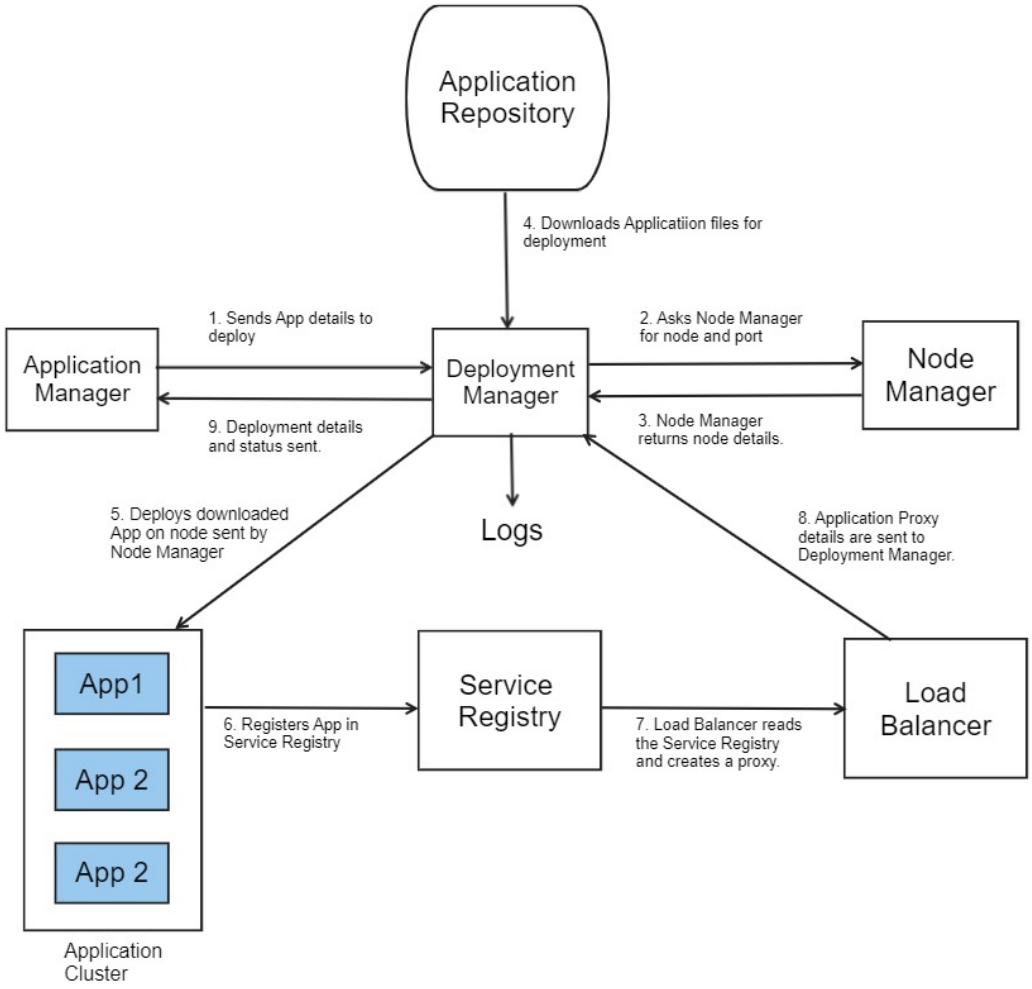
The Sensor Manager module retrieves data from external sources via APIs (in this project OM2M is used). The collected data is stored in JSON format to enable configuration with applications at runtime. This allows for the binding of sensor data with applications. The module plays a vital role in registering new sensor types and nodes and storing sensor data in a database.

The Sensor Manager module transfers collected data to Kafka topics. Applications access data through an API gateway. This allows applications to retrieve their specific data in real-time and make use of the latest sensor readings. By utilizing Kafka topics, the system can efficiently manage and distribute sensor data to the appropriate applications.

System administrators can register new sensor types and nodes using the Sensor Manager module. Upon making the request, the Sensor Manager communicates with the Sensor Registry to create new tables in the sensor database. Additionally, the Sensor Manager generates sensor and node

configuration files to ensure that the new sensor configs are properly integrated into the system. This allows for efficient management and integration of new sensor types and nodes into the system.

iv. Deployment Manager:



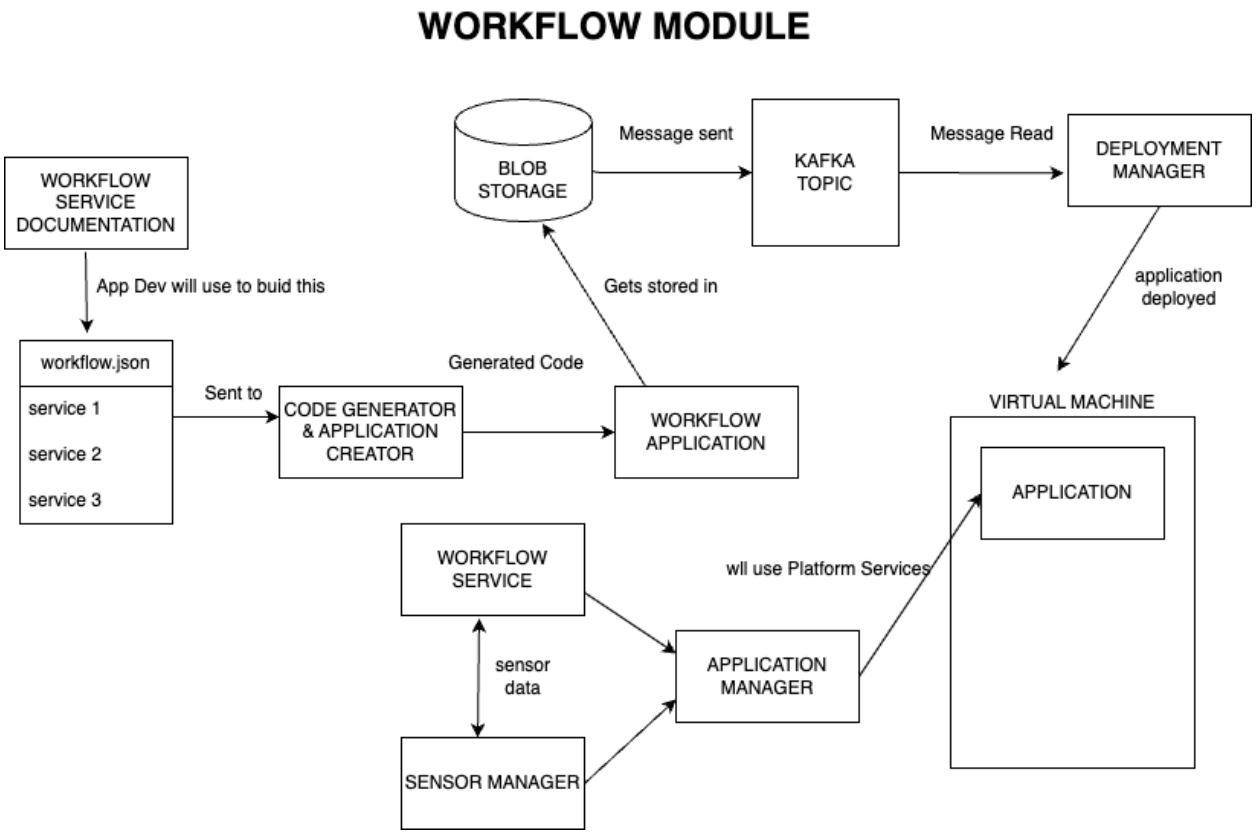
This manager is responsible for deploying / creating an instance for a user application based on the procedure and the configuration provided by the user. The application manager forwards the application instance ID or algorithm to the scheduler. The scheduler then schedules the deployment at the designated time and sends it to the deployment manager.

The deployment manager balances the load among the nodes and ensures that the application or algorithm is running on the correct node or virtual machine. If a node is available, the algorithm will be executed

on that node else the deployer will request for a new node to the node manager and then run the algorithm on the newly allocated node. Load balancer will request for nodes to the node manager.

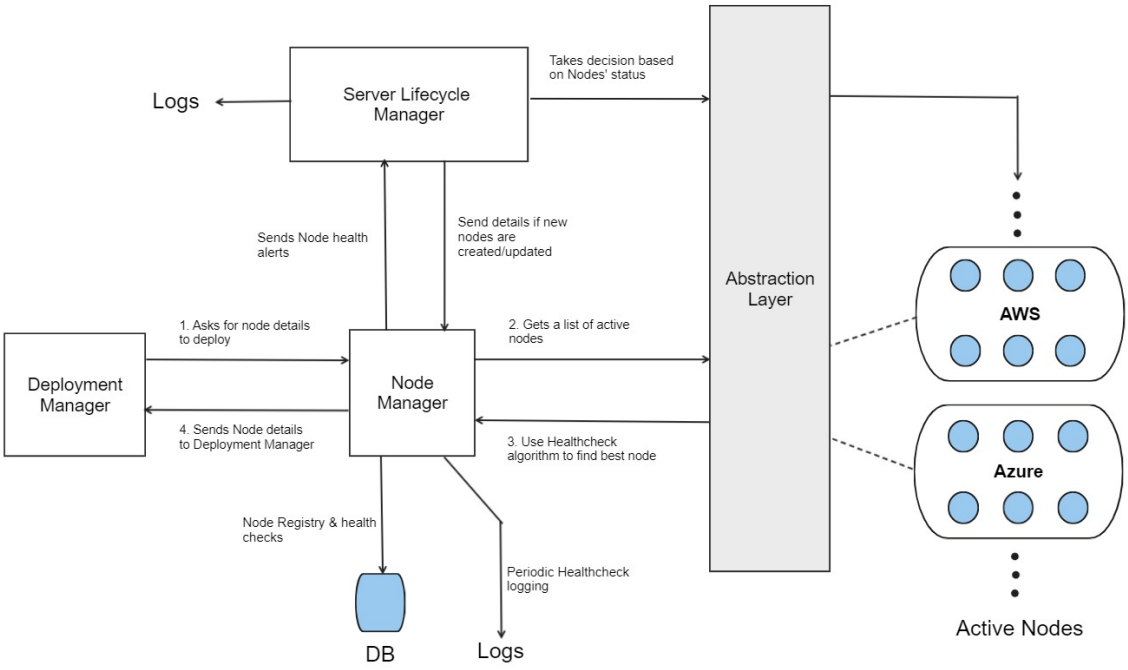
The deployment manager acts as an intermediary between the scheduler and node manager to start or stop applications. The node manager is responsible for managing active nodes and ensuring that load is balanced efficiently. Once the appropriate nodes are identified, the application is deployed based on configuration provided by the configure . Overall, this approach enables efficient resource utilization and streamlined application deployment.

v. WorkFlow Manager:



The Application Developer will make use of existing API i.e Platform Services to create a workflow.json and which will be sent to the Code Generator module. The Code Generator module will generate a code , and store that in the shared Blob Storage. Now, that will be picked by the Deployment Manager module and it will be deployed. The end point of the deployed workflow will be given to the Application Developer and can be used in the application for further use.

vi. Node Manager:



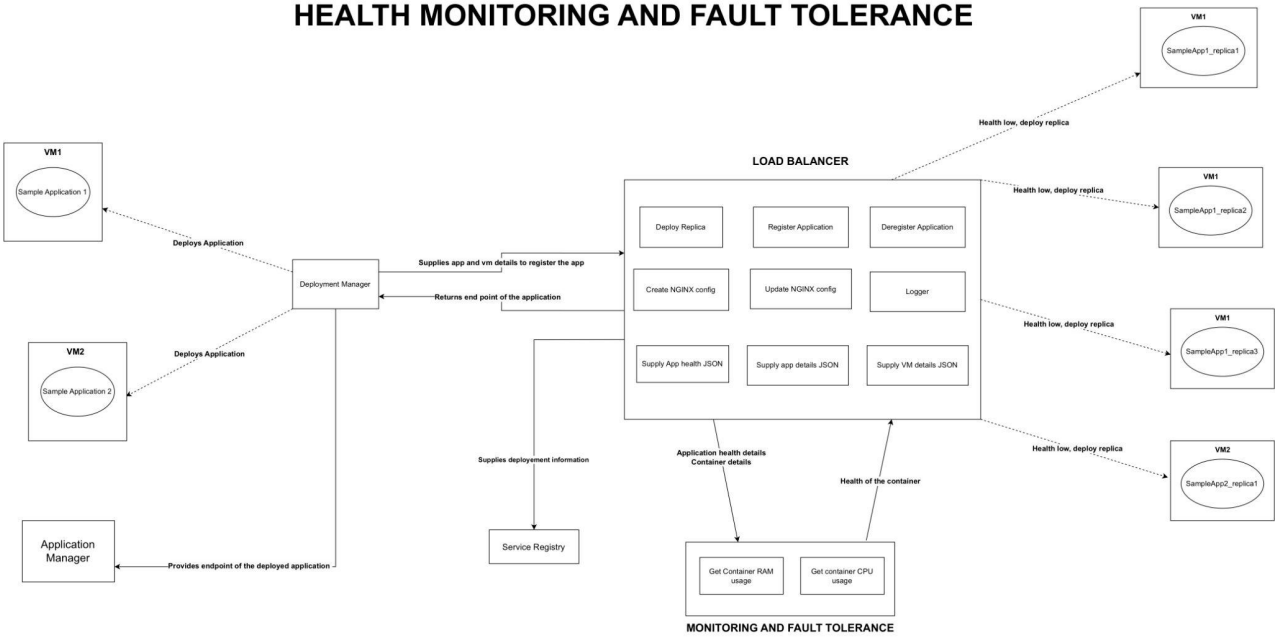
This module is responsible for managing and allocating new nodes, which are virtual machines, based on requests and managing their status in a cloud infrastructure system. It initiates node instances as requested and required by the deployment manager and manages nodes/instances while communicating with other managers.

Node manager provides node details, including a unique node ID, when the node request is made. Once the appropriate nodes are identified, the application is deployed based on configuration provided by the configure. Overall, this approach enables efficient resource utilization and streamlined application deployment.

Node manager allocates new nodes to the load balancer for execution of an algorithm and sends a list of active nodes to the load balancer. It will start the new node and load the init file to the node. Init file is responsible for communication between node and load balancer. It can allocate one or more nodes based on requirements of load balancer. If the available nodes are not sufficient to satisfy the request, the module creates a new node.

vii. Load balancer and Monitoring and Fault Tolerance Module:

LOAD BALANCER
and
HEALTH MONITORING AND FAULT TOLERANCE



First the application manager gets a request to deploy the application. Then the application manager sends a message to the deployment manager to deploy the application. The deployment manager then deploys the application on VM having best health (health information is provided by node manager). Till now only the original instance of the application is deployed. Its end point is not yet known by anyone except the deployment manager.

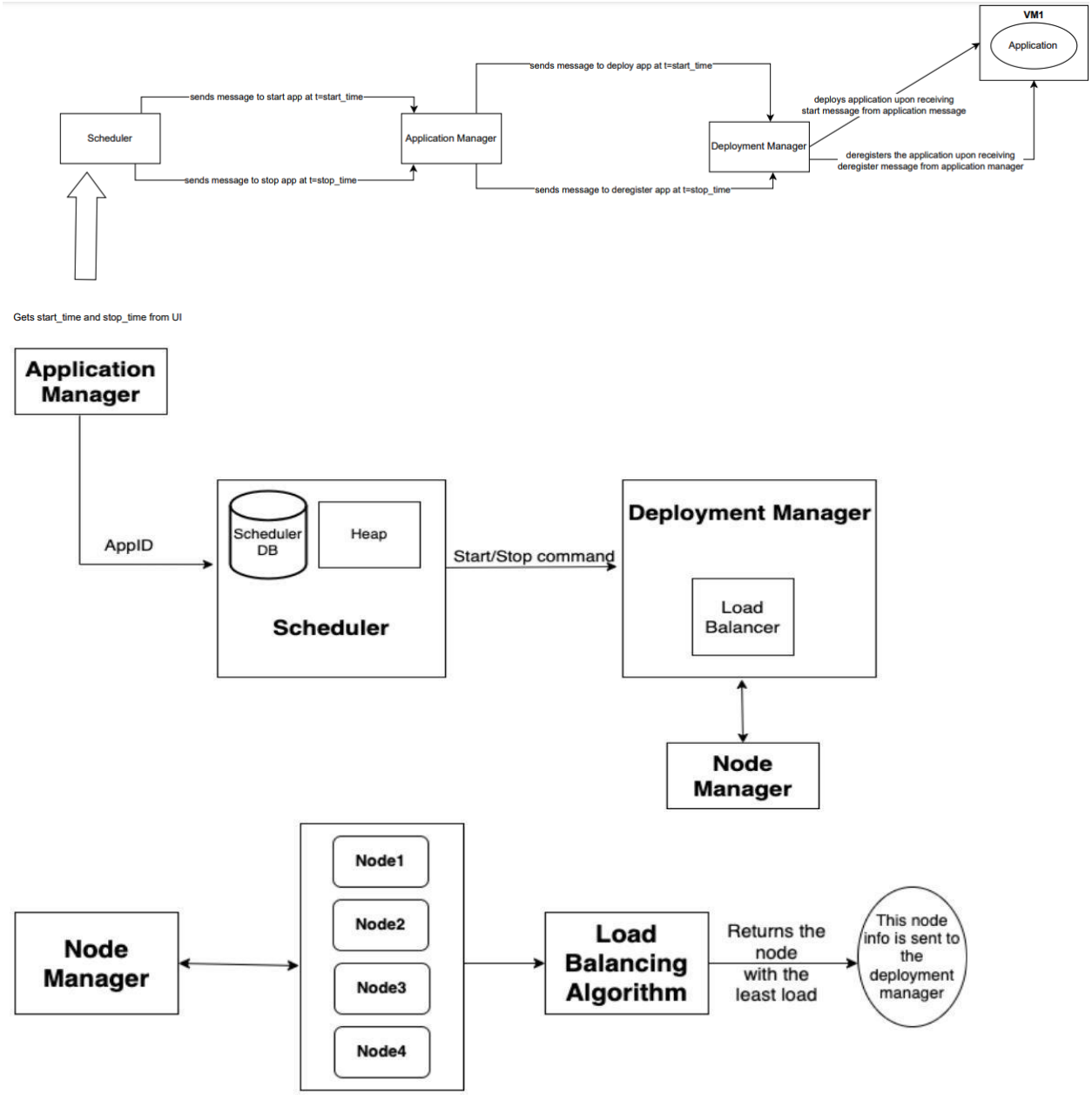
Now the deployment manager sends the ip:port of the application, name of the application, name of the docker_image of the application, etc to the load balancer.

The load balancer upon receiving this information creates a config file for this application which contains the original endpoint of this application and the ip:port of the proxy that it creates for it. This is the final end point of the deployed application. This end point is sent to the deployment manager which is further sent to the application manager. Now this endpoint is sent to the end-user.

The load balancer also updates its config files AppDetails.json (which contains all the details about the deployed application, like its end point, number of instances, container ids, etc), VmDetails.json(which contains details of all the VMs in the VM pool like their public ip, the apps they host, the port number available for a new app to be deployed over it), AppHealth.json (which contains name of all the apps, and a list

corresponding to each app which has the CPU and RAM usage of all the containers of that app), Nginx.json (which contains the latest port on which we can create a new NGINX proxy server).

viii. Scheduler:



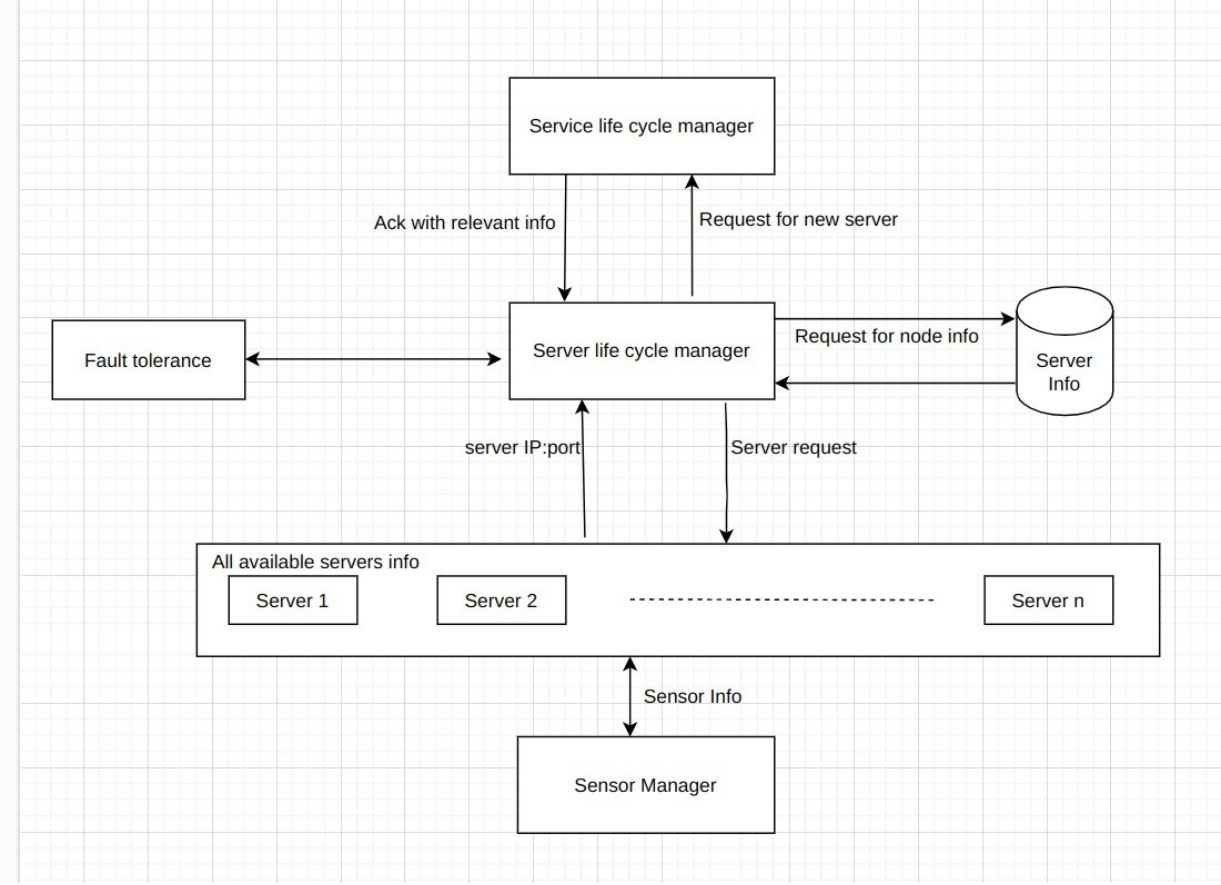
Scheduler is responsible for running/sending jobs to the deployment manager periodically at predetermined intervals. When the scheduler receives an application instance ID from the application manager, it uses that ID to obtain scheduling information from the application repository. Scheduling information can contain schedule start time, frequency, procedure,

authorization and whether the application needs to be repeated, along with the interval for each repetition.

The scheduler then places the application instances into a priority queue according to their start times. As each scheduled time arrives, the corresponding application is retrieved from the priority queue and a notification is sent to the deployer or deployment manager to initiate the application.

After this, it sends this information to the deployment manager for running the application. The information sent to the deployment manager includes the application instance ID, a command to either start or terminate the application, and whether the application should be executed on a standalone machine or a shared machine.

ix. System Life Cycle Manager:



The system life cycle manager is responsible for ensuring that the platform's components are integrated, scalable, and can support a large number of devices and users. They must also ensure that the platform is secure, reliable, and can handle large amounts of data generated by IoT devices.

x. **Platform Manager:** This manager provides CLI/UI to the users to initialize their applications. Initializing the application can have multiple steps like authentication, authorisation, uploading configs, etc. It also provides the users with the APIs that are exposed by platform. Users can also create specific workflows based on their application's requirements.

xi. **Notification Manager:** This manager is responsible for creating and sending notifications with relevant information. This can be triggered by various managers or events. The notification manager helps users make informed decisions about their IoT devices and maintain a secure and reliable IoT ecosystem.

5. Use cases

a. List what the users can do with the solution

i. Control sensor based on the data generated and result analysis

ii. View real-time and historical sensor data

iii. Set thresholds and alerts based on sensor data to trigger actions or notifications

iv. Monitor and manage device connectivity and status

v. Analyze data trends and smart decision based during algorithm execution

b. Who are the types of user's name domain/vertical, role, what they are trying to do when they need the system?

i. User's name: - IoT device manufacturers

Domain/Vertical: - Manufacturing, Technology

Role: - Data Analysts, Business Analysts

ii. User's name: - Facility managers

Domain/Vertical: - Facilities

Management, Operations.

Role: - Facility Managers, Maintenance Technicians

iii. User's name: - Agriculture producers

Domain/Vertical: - Agriculture, Farming

Role: - Farmers, Agronomists

iv. User's name: - Environmental researchers

Domain/Vertical: Environmental Science, Research.

Role: Environmental Scientists, Researchers

v. User's name: - Logistics and transportation companies

Domain/Vertical: - Logistics, Transportation

Role: - Fleet Managers, Dispatchers

c. Usage scenarios:

i. **Mess Overcrowding Management:** This technology could be used in busy cafeterias, food courts or mess halls where there are long lines of people waiting to be served. The system can be set up to monitor the number of people present in the space and alert staff when the number of people reaches a certain threshold, indicating overcrowding. This can help staff manage the flow of people, avoid overcrowding and maintain a safe and comfortable environment for everyone.

ii. **Personalized Healthcare:** This technology can be used in healthcare settings to provide personalized treatment and care to patients. The system can use sensors and wearable devices to monitor patients' health and collect data about their symptoms and conditions. This data can be analyzed by healthcare professionals to create customized treatment plans that are tailored to each patient's specific needs.

iii. **Smart Parking Solution:** This technology can be used in busy parking lots or on-street parking spaces to help drivers find available parking spots quickly and easily. The system can use sensors to detect the presence of vehicles and provide real-time information about available spots to drivers through a mobile app or digital signage. This can help reduce traffic congestion and improve the overall parking experience for drivers.

iv. **Automated Street Light:** This technology can be used in urban areas to reduce energy consumption and improve public safety. The system can be set up to automatically adjust the intensity of the streetlights based on the time of day, traffic flow, and weather conditions. This can help reduce energy waste and provide adequate lighting for pedestrians and vehicles.

v. **Smart Home Security:** This technology can be used in residential homes to enhance security and provide peace of mind to homeowners. The system can include sensors, cameras, and smart locks that are connected to a central hub and can be controlled remotely through a mobile app. This can help detect and prevent intrusions and provide alerts to homeowners in case of suspicious activity.

vi. **Supply Chain Management:** This technology can be used in logistics and supply chain operations to improve efficiency and reduce costs. The system can include sensors, RFID tags, and other tracking technologies that allow companies to monitor the movement and location of goods throughout the supply chain. This can help optimize inventory levels, reduce waste, and improve delivery times.

6. Test Case for the Project

a. Test Case - 1:

i. **Name of use case:** Smart Building Automation

ii. **Domain/company/environment where this use case occurs:** Commercial buildings, office complexes, and other large public spaces.

iii. **Description of the use case purpose, interactions, and what will the users benefit:** The purpose of this use case is to automate the management of a commercial building by connecting various IoT devices, such as sensors for temperature, occupancy, and lighting, and integrating them with an algorithm that can automatically adjust settings based on user needs and preferences. Users, such as building managers, benefit from increased efficiency and cost savings, as well as improved comfort and productivity for building occupants.

iv. **What is the "internet angel" around these things:** The internet of things (IoT) plays a critical role in this use case by connecting all the various devices and sensors, allowing for seamless communication and data sharing.

v. **Information model:** The information model for this use case includes various sensors and devices that provide data on building occupancy, temperature, lighting, and other factors. This data is then processed and analyzed by an algorithm that adjusts settings and sends notifications to building managers and occupants.

vi. **How is location and sensory information used:** Location and sensory information are used to determine occupancy levels and adjust settings accordingly. For example, if a room is empty, the lights and temperature can be adjusted to save energy, while if a room is occupied, the settings can be adjusted to ensure maximum comfort and productivity.

vii. **Processing logic on the backend:** The backend processing logic includes an algorithm that analyzes data from various sensors and devices and adjusts settings based on user needs and preferences. The algorithm also takes into account various factors such as time of day, occupancy levels, and weather conditions to make intelligent decisions.

viii. **User's UI view:** what is their UI, where and all will they do with these systems: The user interface for building managers includes a dashboard that displays real-time data on building occupancy, temperature, and lighting, as well as notifications and alerts for any issues or problems. The UI also allows for manual adjustments and customization of settings, as well as the ability to view historical data and trends over time. The UI can be accessed from any device with internet access, allowing for remote management and monitoring of the building.

b. Test Case - 2:

i. **Name of use case:** Precision Agriculture

ii. **Domain/company/environment where this use case occurs:** Agriculture industry, farms, and crop fields.

iii. **Description of the use case purpose, interactions, and what will the users benefit:** The purpose of this use case is to improve crop yield and quality by connecting various IoT devices, such as soil moisture sensors, weather stations, and drones, and integrating them with an algorithm that can provide farmers with actionable insights and recommendations. Users, such as farmers, benefit from increased efficiency and cost savings, as well as improved crop yield and quality.

iv. **What is the "internet angel" around these things:** The internet of things (IoT) plays a critical role in this use case by connecting various sensors and devices, allowing for seamless communication and data sharing.

v. **Information model:** The information model for this use case includes various sensors and devices that provide data on soil moisture levels, weather conditions, crop health, and other factors. This data is then processed and analyzed by an algorithm that provides insights and recommendations to farmers.

vi. **How is location and sensory information used:** Location and sensory information are used to determine soil moisture levels and weather conditions, as well as to monitor crop health and growth. This information is then used to provide insights and recommendations to farmers on how to optimize irrigation, fertilizer application, and other farming practices.

vii. **Processing logic on the backend:** The backend processing logic includes an algorithm that analyzes data from various sensors and devices and provides insights and recommendations to farmers. The algorithm takes into account various factors such as weather conditions, soil moisture levels, crop growth stages, and historical data to make intelligent decisions.

viii. **User's UI view:** what is their UI, where and all will they do with these systems: The user interface for farmers includes a dashboard that displays

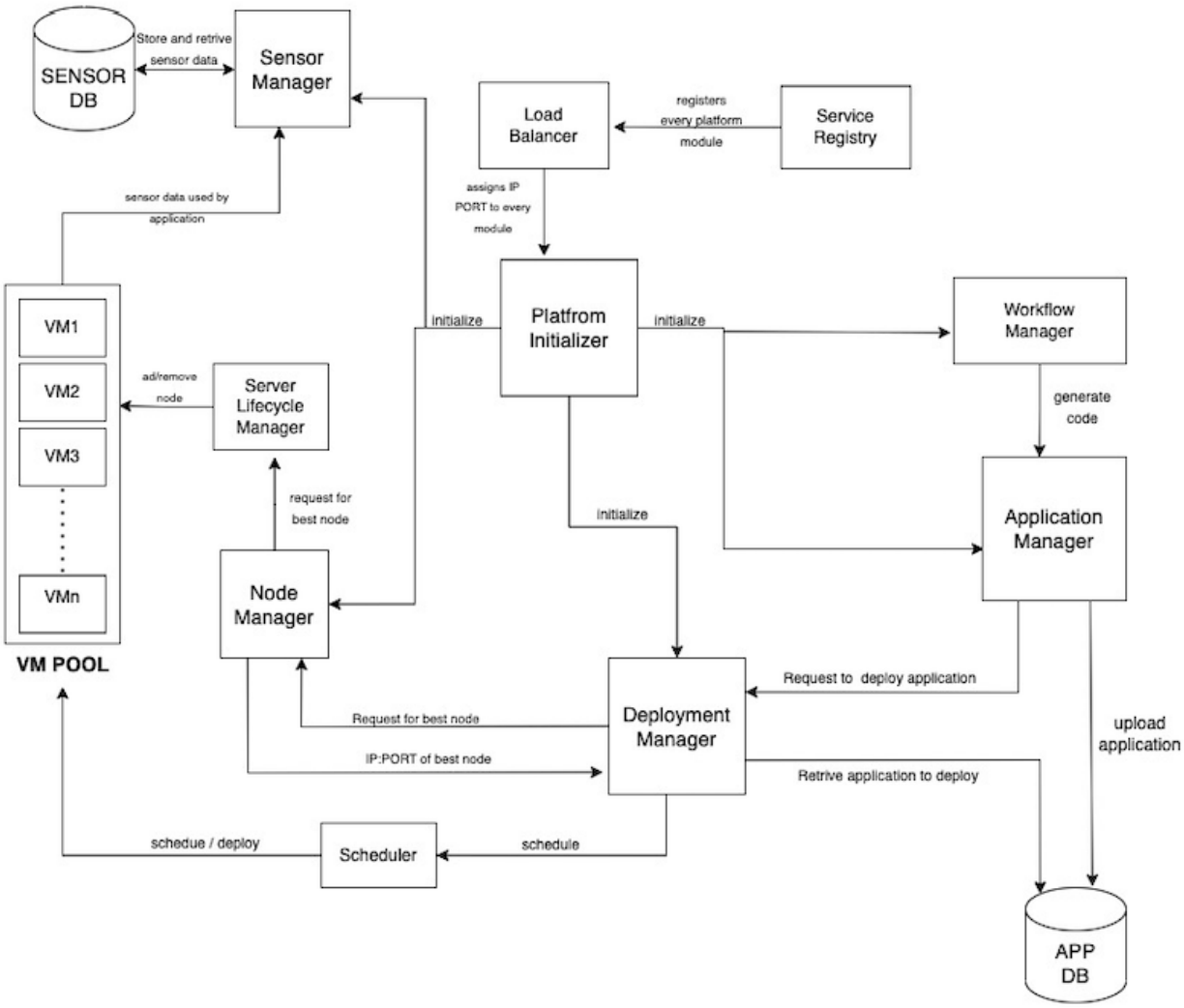
real-time data on soil moisture levels, weather conditions, and crop health, as well as insights and recommendations for optimizing farming practices. The UI also allows for manual adjustments and customization of settings, as well as the ability to view historical data and trends over time. The UI can be accessed from any device with internet access, allowing for remote management and monitoring of the farm.

7. Subsystems

a. Key subsystems in the project

- Platform initializer
- Scheduler
- Monitoring & Fault tolerance
- Application Manager
- Deployment Manager
- Sensor Manager
- Node Manager

b. A block diagram of all the subsystem



c. Interactions involved across these subsystems

i. Platform Initializer

Platform Initializer module is responsible for initializing all available platform modules, such as the Repository, Scheduler, Deployment Manager, Sensor Manager, and others, when the platform is launched.

ii. Scheduler

Scheduler module is responsible for collecting scheduling information from the developer regarding the algorithms provided. It is also responsible for initiating and terminating the scheduled algorithms at specific times specified by the user.

iii. Monitoring & Fault Tolerance

The Monitoring & Fault Tolerance module is responsible for detecting and responding to any failures that may occur within an application. This may involve reinitializing failed nodes or modules, triggering alerts to notify

system administrators, or implementing workarounds to keep critical services running.

iv. Application Manager

Application Manager module consists of two components: a Dashboard user interface for uploading and deploying user applications, and a set of APIs provided to users for deployment. This module serves all requests initially made by the application developer, including login, storing and retrieving algorithms and meta files in .zip format, etc.

v. Deployment Manager

Deployment Manager module is responsible for deploying algorithm instances stored in the repository to run, using various configuration files that determine what packages need to be deployed and on which system the algorithm should be run. It initializes the algorithm with configuration data provided in the algorithm configuration file.

vi. Sensor Manager

This module is responsible for binding the correct sensor data stream with the corresponding algorithm. It provides services such as "Sensor Manager" & "Add Sensor" through the communication module. It identifies IoT devices from the
sensor database based on the specified parameters.

vii. Node Manager

Node Manager module is responsible for initiating node instances as requested by the Deployment Manager.

d. Protocols & Mechanisms

- Kafka
- MQTT
- All these subsystem communicate with each other via their respective exposed APIs

e. External interfaces with the system

- User Dashboard

f. Registry and Repository

- User Database
- Application Database
- Sensor Database
- Monitoring Database