IAS PROJECT

DESIGN 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. INTRODUCTION TO THE PROJECT

With growing IoT Application respective device data is also growing at a similar rate. Each sensor has potential data which can be used to make important decisions. 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.

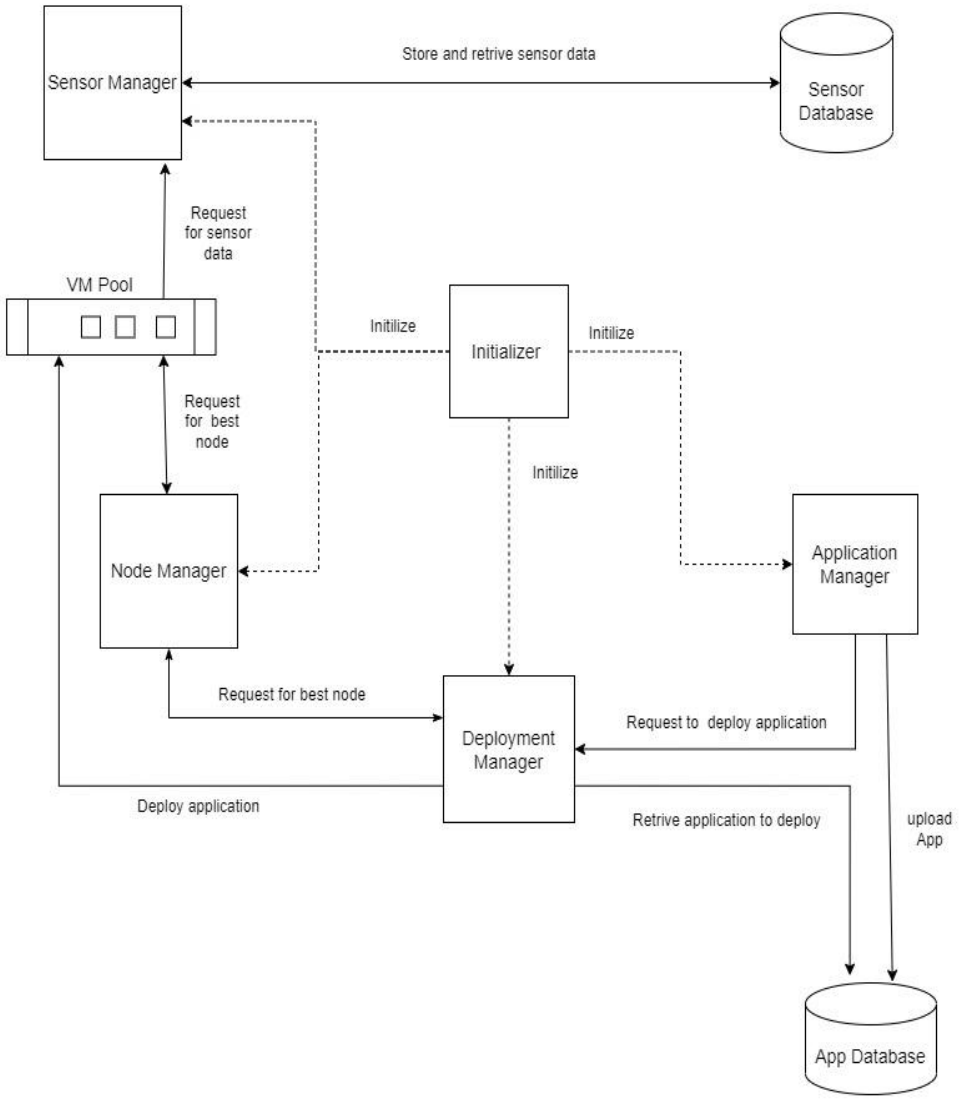
The intended use of the IoT platform is to provide developers and users with a platform that simplifies the process of utilizing sensor data from IoT devices and creating effective applications. It aims to offer an environment where users can easily connect to location-sensitive IoT devices, manage sensor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

Functional overview of Team's module

The platform is initialized using the platform initializer module. Platform initializer will pick the code dumps of various app modules from the platform repo and deploy them on some nodes. Application Developer comes with the configuration files and the python files with the algorithms. There are various modules as shown above such as Sensor Manager, Deployment Manager, etc.

Corresponding to the zip files provided by the app developer various code dumps will be made available at the appropriate nodes. The end user will provide various scheduling information depending upon its use-case and this will run at appropriate VMs.



2. USE CASES/REQUIREMENTS

Application developers can create and deploy IOT applications on their platform. There can be multiple applications running at the same time

What the users and application developers can do with the solution

- Control sensor based on the data generated and result analysis
- View real-time and historical sensor data

- Set thresholds and alerts based on sensor data to trigger actions or notifications
- Monitor and manage device connectivity and status
- Analyze data trends and smart decision based during algorithm execution

Smart Home - An IoT platform can enable homeowners to remotely control and automate various devices in their home, such as lighting, thermostats, refrigerator, washer, and dryer, dishwasher air conditioner and security systems.

Smart City - Environmental Real-time data like temperature,Humidity,etc will be stored continuously in the database. Researchers can analyze those data, based on which they can predict events. proximity and location-sensor can be used to find nearby public-wifi and can be used in automated traffic control system.

Event Crowd Control - With multiple sensors installed in an event (convert,exhibition etc) and the insights coming from the IOT platform, we can control the crowd, reduce waiting time and efficiently manage parking and food services.

Industrial automation - Our platform can be used to deploy applications concerning industry management across locations. Applications in this domain can be used to manage and monitor production, make smart decisions using distributed sensor networks, manage deliveries using GPS, vehicle health monitors etc.

3. TEST CASES

3.1 Test cases- used to test the team's module

Test 1: Authentication Module testing

Input : Login configuration of some new and some existing user

Output: Module should allow login to existing user but should restrict new or unauthorized access without sign-in.

Purpose : because it is necessary to allow only valid developers/users to our platform for managing system resources efficiently and providing prompt services to our user.

Test 2 : Testing of input file validation from application developer

Input : Zip file from application developer

Output : Valid or Invalid

Purpose : The purpose of this testing is ensuring that application developer is uploading config and programs file in right format or not

Test 3 : Application manager commands testing.

Input : Basic test program to test proper functionality of application manager.

Output : Output log file containing execution time and output of each command.

Purpose : The purpose of this testing is ensuring that application manager is performing its task and sending proper command to each of its subsequent module.

Test 4 : Scheduler commands testing.

Input : Basic test program to test proper functionality of scheduler.

Output : Output log file containing the application instance id, start time and end time of all application instances .

Purpose : The purpose of this testing is ensuring that scheduler is scheduling instances of application according to their priority and is able to maintain that queue of instances even after scheduler failure.

Test 5 : Deployer testing.

Input : Basic test program to test proper functionality of deployer.

Output :Deployer should respond on new job deploying command and job kill command and it should also interact with load balancer to get information of healthy nodes .

Test 6 : Load Balancer testing.

Input : List of test programs to test proper efficiency of the deployer.

Output : Load balancer should return details of best possible healthy nodes to deploy all these input programs after getting deployment requests from the deployer.

Purpose : The purpose of this testing is ensuring that Loadbalancer is efficient and properly interacting with node manager.

Test 7 : Node manager testing.

Input : Basic program to check fault tolerance of node manager against sudden failure .

Output : Node manager should be able to recreate a new image of all the containers from inactive nodes to active node and should be able to get health of nodes properly.

Test 8 : Sensor manager testing.

Input : List of ip, port and unique sensor/controller id to test proper efficiency of sensor manager.

Output : Successful connection between sensors/controllers and sensor manager/ controller manager using given ip, port.

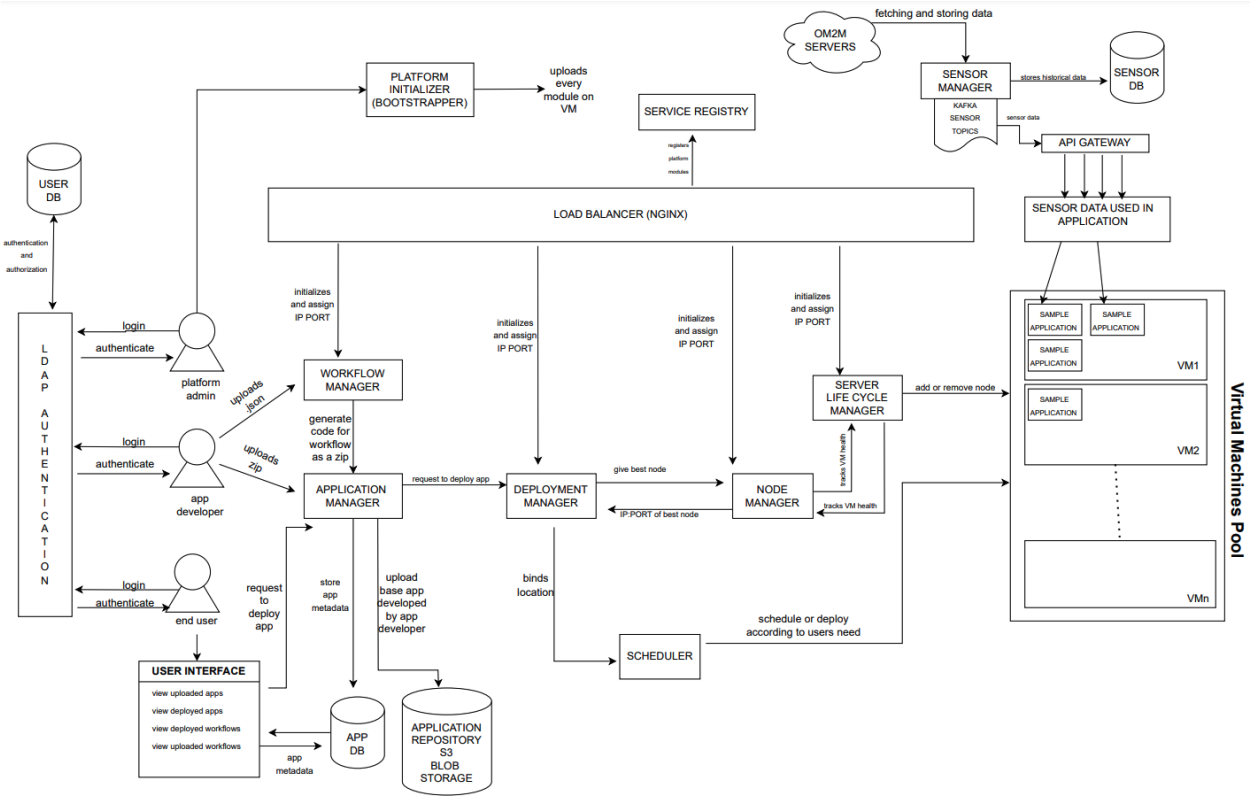
Test 9 : Fault tolerance testing

Input : sudden failure of platform due to any uncertain reason

Output : The module should reinitialize the application on a different node chosen after recovery and should maintain previous state before failure.

4. SOLUTION DESIGN CONSIDERATIONS

4.1 Design big picture



1. **Application Manager**: Responsible for managing the lifecycle of applications running on the IoT platform. This includes tasks such as starting, stopping, scaling, and monitoring the performance of applications.
2. **Platform Developer(configurer or admin)**: Develops and maintains the IoT platform on which the applications run. This includes designing and implementing the underlying infrastructure, defining APIs for application development, and ensuring the platform is scalable, secure, and reliable.
3. **Application Developer**: Develops applications that run on the IoT platform. This includes writing code, integrating with APIs provided by the platform, and ensuring the application is scalable, secure, and performs well.
4. **Scheduler**: Schedules and manages the execution of tasks and processes running on the IoT platform. This includes managing resources, coordinating workflows, and ensuring timely execution of tasks.

5. **Deployment Manager:** Manages the deployment of applications onto the IoT platform. This includes packaging and distributing the application, configuring the platform for the application, and ensuring the deployment is successful.
6. **Platform Load Balancer**: Distributes traffic across multiple instances of an application running on the IoT platform. This ensures that the platform can handle high volumes of traffic and that the application performs well under load.
7. **Fault Monitor**: Monitors the platform and applications for faults and errors. This includes detecting and diagnosing issues, notifying relevant parties, and taking action to resolve the issue.
8. **Node Manager**: Manages the virtual nodes on which the services are running. This includes configuring and managing resources ensuring availability, and monitoring performance.
9. **Authentication Manager**: Manages the authentication and authorization of users and applications on the IoT platform. This includes ensuring secure access to resources and data, enforcing security policies, and managing user roles and permissions.

4.2 Environment to be used

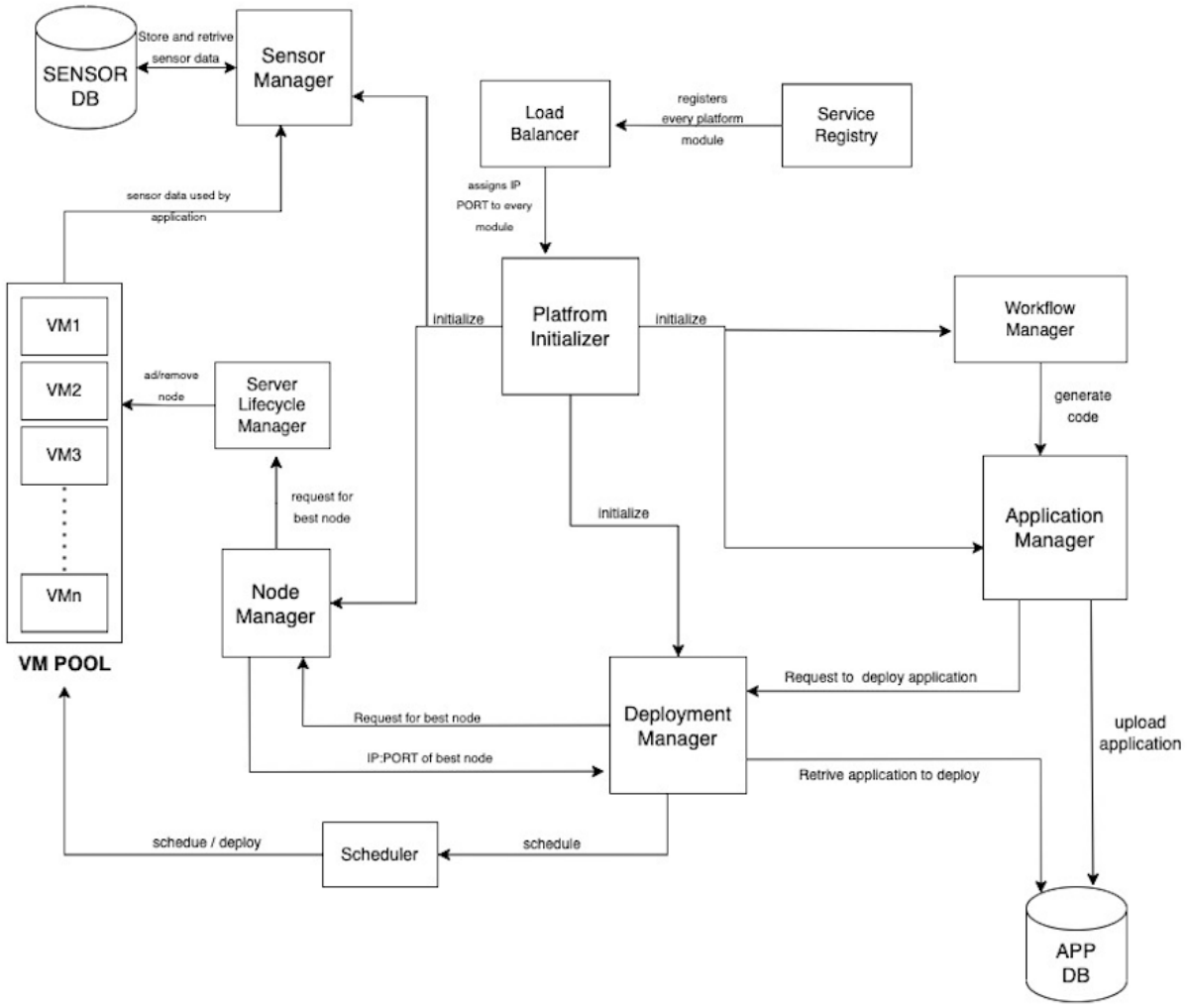
- OS – Linux/windows /macOS
- Docker container

4.3 Technologies Used

- Python - base tech stack
- Python frameworks - Flask for overall development and REST APIs
- Docker
    - Mobility - Ability to run anywhere

    - Isolation - It ensures your applications and resources are isolated and segregated.
- MySQL - for database(store data)
    - Data Security
    - High Performance
- Apache Kafka - for communication(distributed streaming)
    - Producer-Consumer Model
    - Scalable
    - Fetching real time data

4.4 Overall system flow & interactions



4.5 Interaction with Sensors

Senser data will be coming through ONEM2M gateway. The sensor data will be fetched by sensor manager and will be shared between other modules or platform services through Apache kafka. The sensor data will also be stored in sensor database.

4.6 Approach for communication & connectivity

Kafka is used for real-time streams of data, to collect big data, or to do real time analysis (or both). Kafka is used with in-memory microservices to provide durability and it can be used to feed events to CEP (complex event streaming systems) and IoT automation systems.Kafka is often used in real-time streaming

data architectures to provide real-time analytics. Since Kafka is a fast, scalable, durable, and fault-tolerant publish-subscribe messaging system.

4.7 Registry & repository

- Application DB - It stores the artifacts(zip and config files) received from the application developer.
- Deployer DB - used by the deployment manager to store information about the devices being managed. It typically contains information such as the device's unique identifier, its current software version, its location, and its current configuration settings.
- User DB - to store end user and application developer credentials and other related details
- Monitoring & Logging DB - store the data of sensors and control operations of important checkpoints
- Sensor DB - to store the sensor data (type,reading etc)

4.8 Scheduler

- The scheduler in this system receives an application instance ID from the application manager.
- Using this ID, the scheduler fetches scheduling information from the application repository, such as the start time, end time, duration, whether the application should repeat, and the interval of repetition if necessary.
- The application instances are then placed in a priority queue based on their start times.
- As the scheduled time for each application instance arrives, it is popped out of the priority queue and a notification is sent to the deployment manager or deployer to start the application.

4.9 Load Balancing

- In our system, we retrieve two key metrics from the nodes to calculate their load: the CPU load and the percentage of RAM used.
- These metrics are obtained using the paramiko library.
- To determine the load on a particular machine, we calculate the harmonic mean of these two metrics. This allows us to obtain a more balanced and accurate assessment of the machine’s load.

- We then select the node with the least load to assign new applications to, ensuring that the workload is distributed evenly across the nodes in the system.

4.10 Interactions between modules

- **Interaction between application manager and scheduler**
  - ➔ When a request for deployment is received by the application manager from the configurer, it sends the application instance ID to the scheduler so that the application can be scheduled.
  - ➔ The scheduler then retrieves the scheduling information for that application instance from the app database using the app instance ID.

- **Interaction between deployment manager and scheduler**
  - ➔ Once the scheduled time for an application instance has been reached, the scheduler removes it from the priority queue.
  - ➔ The scheduler then sends a notification to the deployer containing the application instance ID and a command to either start or kill the instance. This communication between the scheduler and deployer occurs through a Kafka topic.

- **Interaction between platform initializer and scheduler**
  - ➔ Plaform initializer will initialize the scheduler and the deployer to get them up and running.

- **Platform initializer will interact with all the platform modules** (node manager, application manager , sensor manager and other modules) to track their health.

- **Interaction between deployment manager and scheduler**
  - ➔ Upon receiving requests from the application manager, the scheduler will add jobs to its priority queue and send a notification to the deployment manager via a one-to-one topic containing job information.
  - ➔ To initiate the application, the scheduler will communicate with the deployer in the Deployment Manager. The scheduler will create a list of jobs to be executed and add them to the job queue, which the deployer will retrieve.

- **Interaction between Deployment Manager and Node Manager**
  - ➔ The Load Balancer component of the Deployment Manager communicates with the Node Manager to obtain a list of active nodes. Once the Node Manager provides this list, the Load Balancer uses a Load Balancing Algorithm to determine the load on each node.
  - ➔ Based on the load calculations, the Load Balancer selects the node with the lowest load. The Deployer then contacts the Node Manager with the information about the selected node, where the application with the corresponding application ID is to be run.

- **Interaction between platform initializer and deployment manager**
  - ➔ Plaform initializer will initialize the scheduler and the deployer to get them up and running.

5. APPLICATION MODEL AND USER'S VIEW OF SYSTEM

5.1 Overview of the application dev model (dev steps/flow)

The platform is initialized using the platform initializer module. Platform initializer will pick the code dumps of various app modules from the platform repo and deploy them on some nodes. Application Developer comes with the configuration files and the python files with the algorithms. There are various modules as shown above such as Sensor Manager, Deployment Manager, etc.

Corresponding to the zip files provided by the app developer various code dumps will be made available at the appropriate nodes. The end user will provide various scheduling information depending upon its use-case and this will run at appropriate VMs.

Files Structure

Application Developer provides *zip file which will have python code and the configurations files in *.json extension.
Moreover, the user will provide majorly the start and end time of the application and other scheduling info.

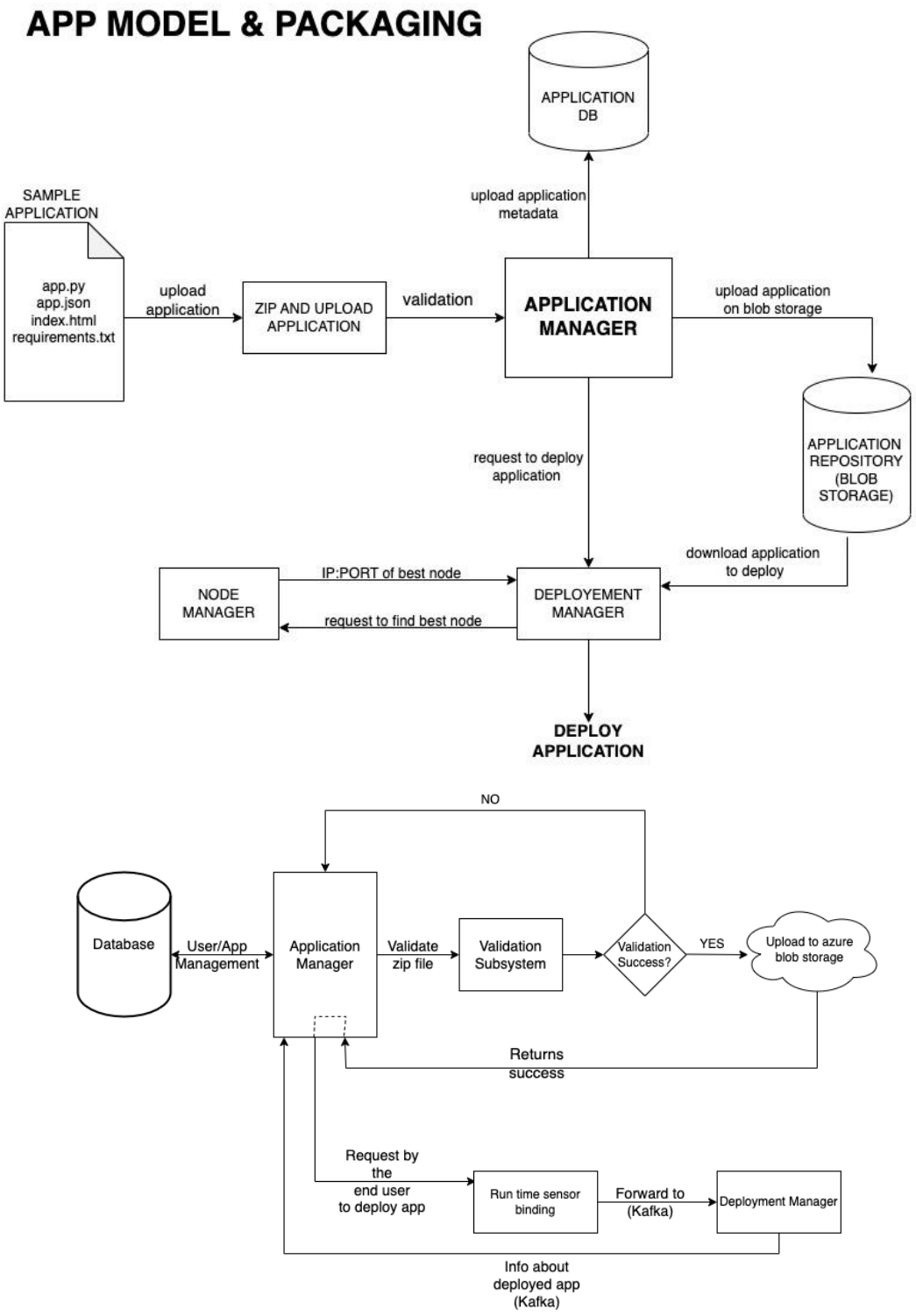
Deploy/Setup

The whole process of deploying the application has been described above, just to brief it up, whenever the end user (user of the application developed by the Application Developer) wants to use the application, he/she will be providing the scheduling information. For that period of time, an instance of the application will be deployed on the virtual machine which has the maximum health.

5.2 Application artifacts

- Packaging files
- Configuration files
- Logging
- Monitoring
- Notification and alerts
- Docker files
- Deployment configuration
- Application code

5.3 Application Model and Packaging



5.4 User admin interactions

The different user will login into the platform and depending on the level of access

and type of role, there will be multiple options. The application developer will upload the configuration file and the script(or program) of the application that he/she is trying to built. The user can see sensor related data and other results of the application along with notification and alerts.

6. KEY DATA STRUCTURES

Platform Initializer will be using a map which will store which platform service is running on which machine.

Scheduler will be using heap and queue to scheduling any job.

User DB - for storing details and credentials of end user and application developer. It is part of the application manager module for validating and authorizing the users.

Application DB - for storing the metadata of the applications created by the application developer.

7. INTERACTIONS AND INTERFACES

APIs

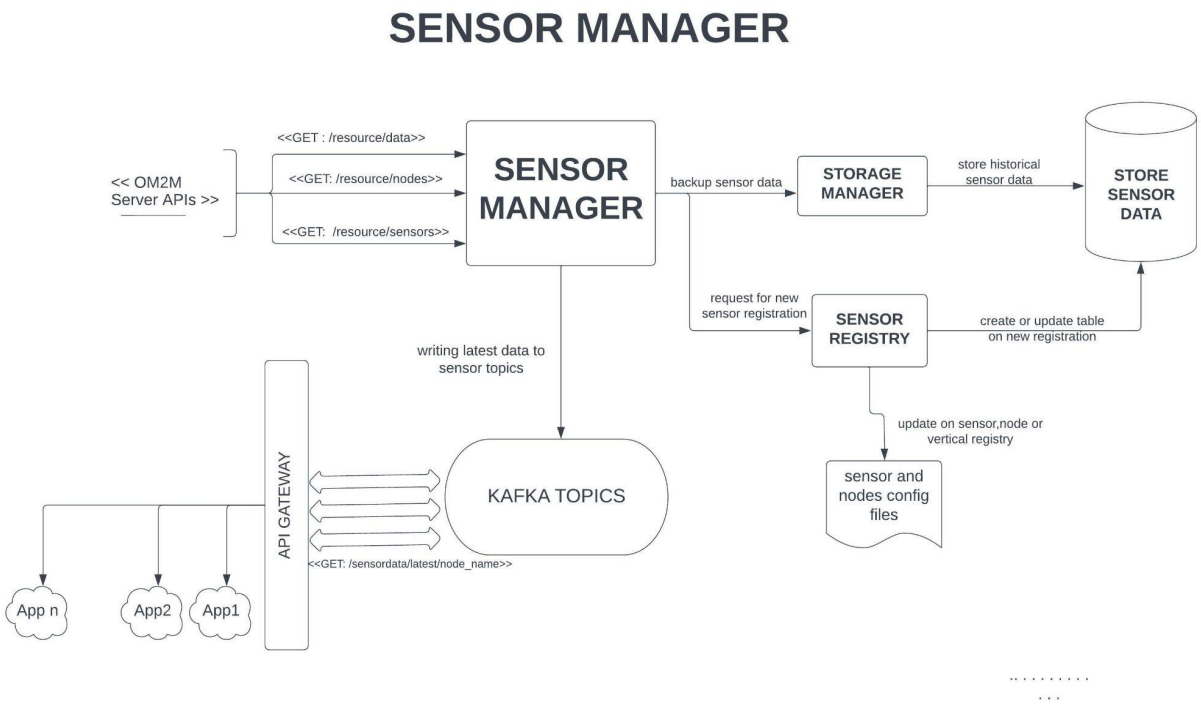
- GET /active-node - get list of active node
- POST /node/:id - create a new node
- GET /sensor-data - get sensor data of all the deployed sensors
- POST /controller-data/:id - set controller data
- POST /register - create an account for application developers and end users
- POST /login - login route for application developers and end users
- POST /logout - login route for application developers and end users
- POST /upload-file - API end points to upload the packaging and configuration file

8. PERSISTENCE

All the persisted data in different databases and all the other configuration files.

9. VARIOUS MODULES

9.1 Sensor and Controller Manager



Block Diagram of sensor and Controller Manager

- The Sensor Manager module enables communication between applications and external sensor infrastructure.
- The module registers new sensor types and nodes and retrieves data from external data sources via APIs (in this project OM2M is used).
- Collected data is stored in JSON format to enable configuration with applications at runtime and allow for binding sensor data with applications.
- The module stores sensor data in a database and transfers collected data to Kafka topics for efficient management and distribution to the appropriate applications.
- Applications access data through an API gateway, allowing them to retrieve their specific data in real-time and make use of the latest sensor readings.
- 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.

- The Sensor Manager also generates sensor and node configuration files to ensure proper integration of new sensors into the system.
- The efficient management and integration of new sensor types and nodes into the system is a key feature of the Sensor Manager module.

9.2 Nodes and Node Manager (VM's):

The Nodes and Node Manager (VM's) module is responsible for handling the allocation and management of virtual machines (nodes) as needed. When a request for a node is made, this module provides information on the available nodes, including their IDs. If there are no nodes available to fulfill the request, a new one will be created. Additionally, the module is responsible for monitoring the status of nodes, whether they are currently active, free, or down.

Lifecycle

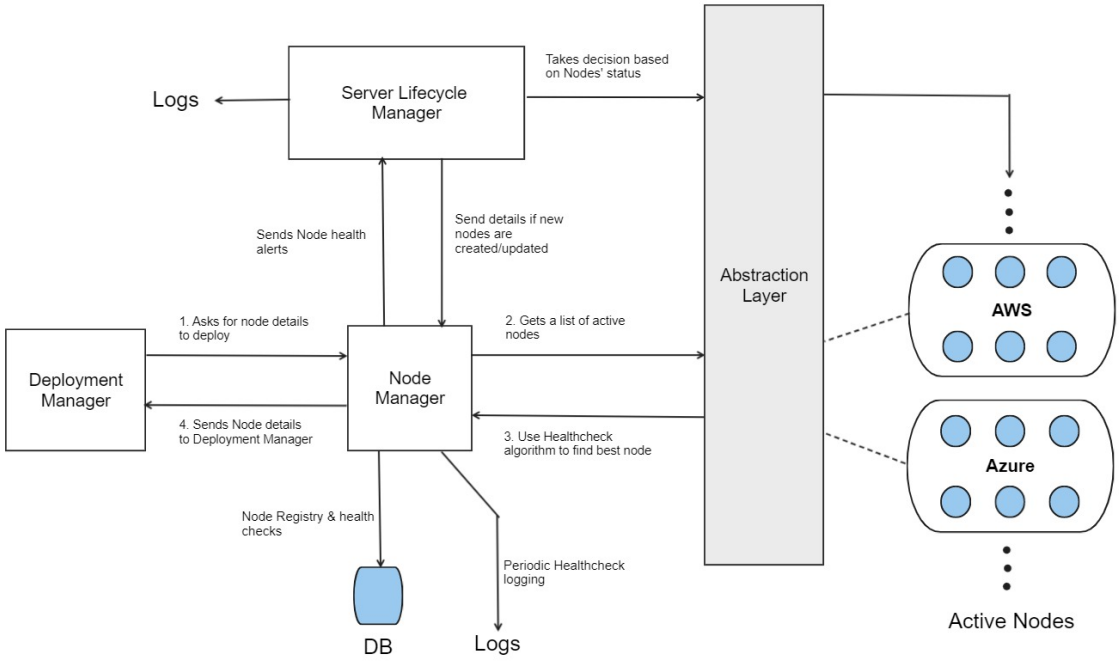
- When the platform is initialized, the node manager module starts and k number of virtual machines (nodes) are initialized.
- When a node request is made by the deployer, information about the available nodes is provided.
- The requested node is then used to run the user's application.
- The nodes are continuously monitored for any failures using the Health client.
- The node manager and nodes continue to exist till the platform is running.

Sub Modules

- Nodes: These are the entities where application instances or platform components are executed. They may be physical servers or virtual machines. Nodes can also communicate with the sensor manager to send/receive data from sensors.
- Node Manager: This module is responsible for managing information about the nodes and providing a new node to the application developer when necessary.

Functionality

- Generating a list of currently active virtual machines or nodes if requested.
- Provisioning a new virtual machine or node as needed by the deployer.
- Managing the node registry.
- Shutting down a virtual machine.

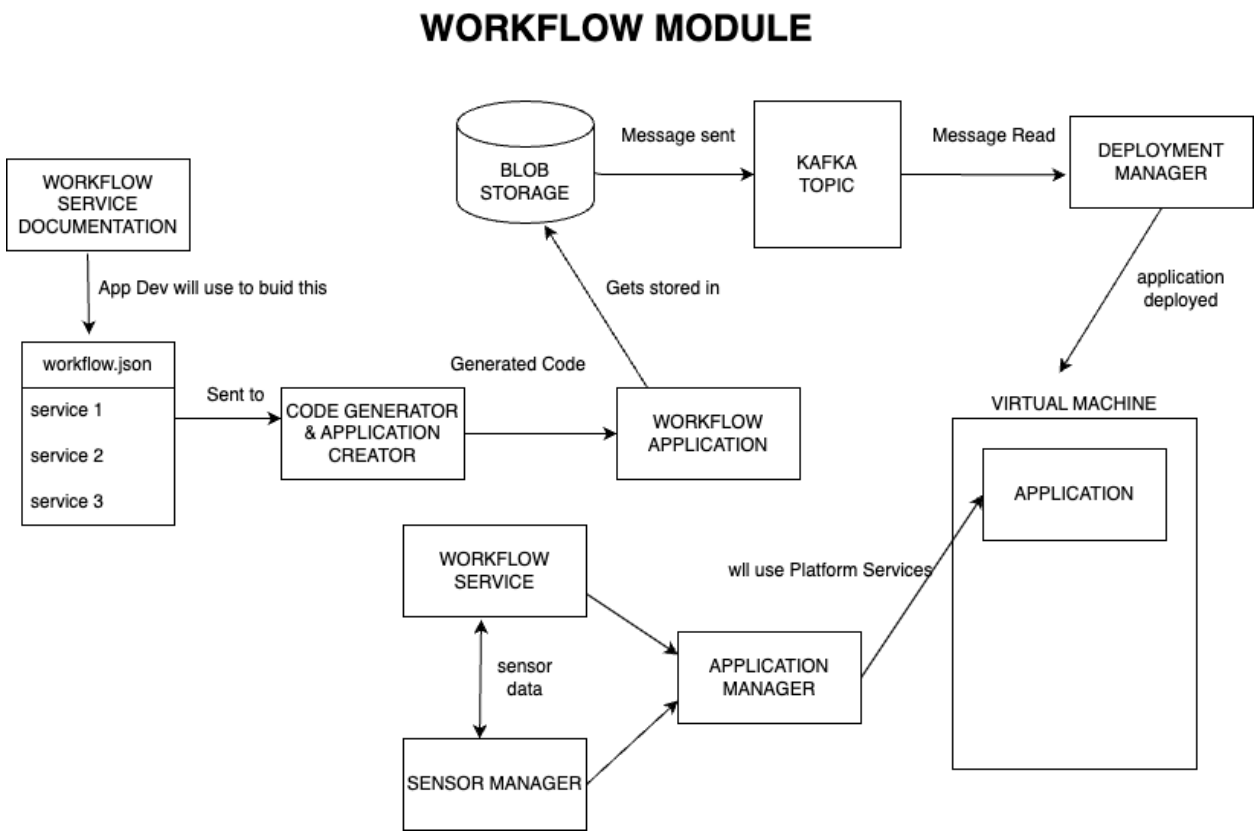


Block Diagram of Node Manager

Interaction with other modules

- Interaction with Monitoring and Fault Tolerance - The Node Manager sends health messages to the Monitoring and Fault Tolerance module. Periodic checks are performed to ensure that the modules are functioning properly. If any issues or faults are detected, the Fault Tolerance module's instance is notified.
- Interaction with Deployer - Requests the Node Manager for a new node when the workload on the currently active nodes increases.

9.3 Workflow Module



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.

Lifecycle:

- When the platform is initialized the Code Generator submodule will be ready to accept the workflow.json file
- After generating code , this will be consumed by Deployment Manager and in return it will send the end point to access that

Functionality

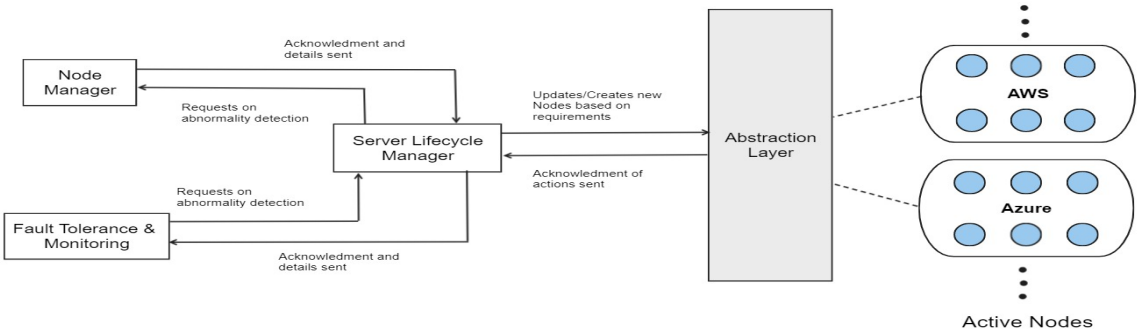
- It serves the purpose of low code by allowing the application developer to just mention the flow of the code.
- It provides the end point of the workflow which can further be used in any application code thus making it reusable.

9.4 Monitoring and Fault Tolerance

The Monitoring and Fault Tolerance module continually checks the platform components for any failures and ensures that the system operates seamlessly even if one or more components fail. It monitors platform modules, application instances, and nodes using heartbeat messages. If a component is not working correctly, the module either initializes a new node with its corresponding applications or reinitializes the application on the same node.

Lifecycle

1. The fault tolerance module starts as the platform initializes.
2. Every new application and node that starts registers with the fault tolerance module.
3. The module continuously monitors other applications and nodes for failures using the Heartbeat client.
4. The Fault Tolerance module exists until the Platform is turned off.



Sub Modules

- Health Client: Every platform module/service (scheduler, deployer, node manager, etc.) must run a Health client API so that the Monitoring and Fault

Tolerance service can identify the status of these services continuously and restart them if they are down for any reason.

APIs provided:

health (): Sends health to the Monitoring Module in the following format:

Node: node-<ip>_<port>

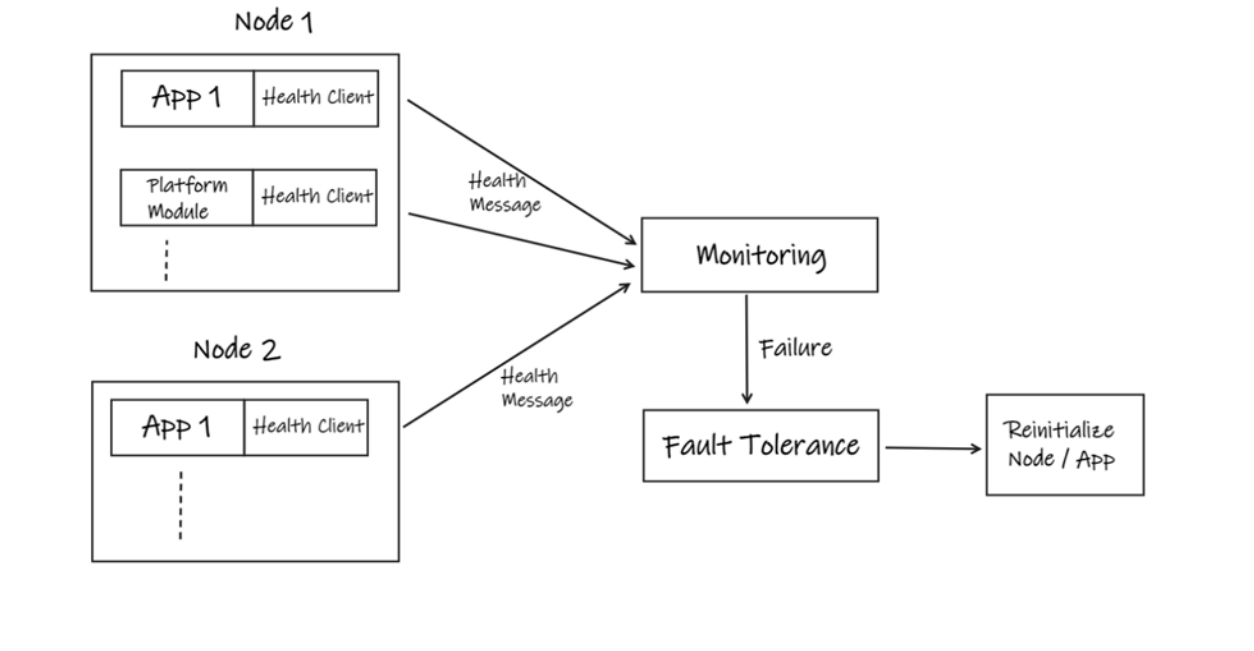
Application: application-<app_instance_id>

Services: service-<service_name>

- Health Manager - The Health Manager is responsible for monitoring the status of each service. In case a service stops abruptly, the Health Manager generates an error message indicating which service has stopped.
- Fault Tolerance - Upon receiving information of a module or node being down, it reinitializes the module of node using its initialization configuration details.

Functionality

- Platform: It checks whether all components of the platform, such as the scheduler, deployer, and application manager, are functioning correctly. In case of any failure, it reinitializes the component.
- Application: It checks whether the application instances are functioning correctly. If they are not, it reinitializes another application instance. If an active node goes down, it reinitializes all the modules and applications running on that node on a different node.



Block diagram of fault tolerance

Interaction with other modules

- Interaction between monitoring sub-module and fault tolerance sub-module - The monitoring module continuously checks the health of all the running nodes. If a node fails to send its health, the monitoring module calls the Fault Tolerance module to reinitialize that node.
- Interaction of Monitoring and Fault Tolerance with other Modules - The Monitoring and Fault Tolerance module interact with various modules, including - Scheduler ,Sensor Manager, Load Balancer, Deployer, Application Manager, Node Manager, Nodes and Application instances
- Periodic monitoring is performed to detect any faults or discrepancies in these modules' operation. If a problem is detected, the Fault Tolerance module's instance related to that module is triggered, and an alternate node is assigned to run that module.

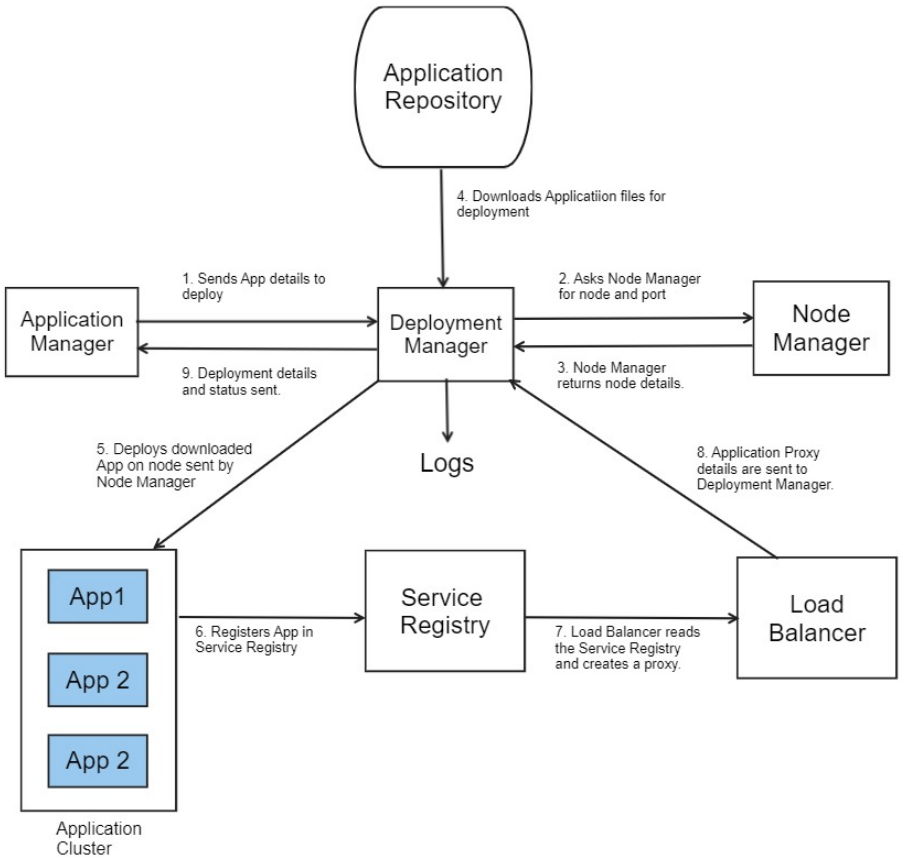
9.5 Deployment Manager

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 runs on the correct node or virtual machine.

Lifecycle

- This module follows a specific lifecycle where its components are initiated by a platform initialiser.
- Each component is equipped with a heart manager that monitors its health and status. In the event of a component failure, the heart manager sends a signal to the monitoring and fault tolerance module.
- The fault tolerance module then takes action to ensure that the failed component can resume its work from its last saved state, ensuring continuity of the system.



Block diagram of deployment manager

Subsystem

Load Balancer- The purpose of this module is to determine the optimal node for deploying the application. It gathers relevant statistics, such as CPU usage, and uses this information to select the most suitable node. Once a node has been chosen, the module provides the IP address of the selected node to the deployer, which can then proceed with the deployment of applications.

Deployer - A deployer is a tool that takes input files such as pickle, config, and contract files, and generates a docker file and wrapper file that can be deployed on a VM. Users can upload these input files via a UI, depending on whether they are data scientists or application developers. In addition to handling deployment requests, the deployer also monitors applications and manages their life cycles.

Health manager - health manager checks periodically the life of all the running nodes and it is associated with every component to keep a check on the health of each component (i.e. to make sure that platform doesn't go into an absurd state if any component fails).

Interaction with other modules:

- Interaction between platform initializer and all other platform modules - Platform initializer will interact with all the platform modules (node manager, application manager , sensor manager and other modules) to track their health.

- Interaction between deployment manager and scheduler
    - ❖ Upon receiving requests from the application manager, the scheduler will add jobs to its priority queue and send a notification to the deployment manager via a one-to-one topic containing job information.
    - ❖ To initiate the application, the scheduler will communicate with the deployer in the Deployment Manager. The scheduler will create a list of jobs to be executed and add them to the job queue, which the deployer will retrieve.

- Interaction between Deployment Manager and Node Manager
    - ❖ The Load Balancer component of the Deployment Manager communicates with the Node Manager to obtain a list of active nodes. Once the Node Manager provides this list, the Load Balancer uses a Load Balancing Algorithm to determine the load on each node.
    - ❖ Based on the load calculations, the Load Balancer selects the node with the lowest load. The Deployer then contacts the Node Manager with the information about the selected node, where the application with the corresponding application ID is to be run.

- Interaction between platform initializer and deployment manager
    - ❖ Plaform initializer will initialize the scheduler and the deployer to get them up and running.

9.6 Scheduler and load balancer

The scheduler will retrieve scheduling information for an application instance from the app repository using its ID, as provided by the application manager. Once the information is retrieved, it will be sent to the deployment manager to initiate the application's execution. Load balancing is the method of assigning a group of tasks to a group of resources to improve overall processing efficiency. Load balancers are implemented to enhance application capacity (simultaneous users) and reliability.

Lifecycle

- The platform initializer will launch all components within this module.
- Each component includes a health manager that monitors its status and overall well-being.
- If a component fails or becomes inactive, the health manager will notify the monitoring and fault tolerance module.
- The fault tolerance module will then ensure that the failed component resumes its operation from its previously saved state.

Subsystems

- Memory heap (queue)

- Scheduler DB
- Load balancer
- Deployer DB

9.7 Platform Initializer

Lifecycle

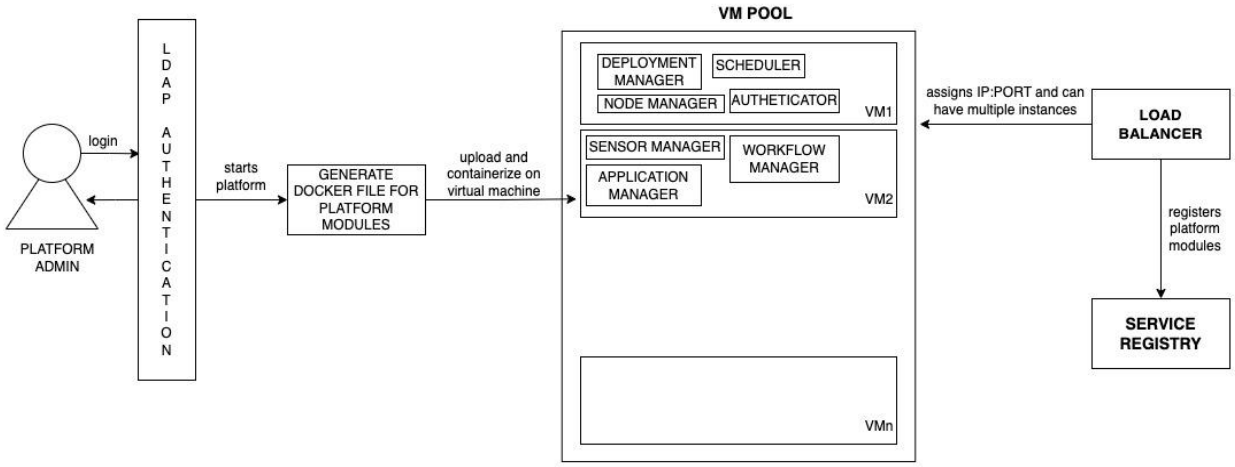
Platform Initializer can be considered as an Bootstrapper module. This component will initialize all the other module(services or various manager).Platform initializer will create the necessary infrastructure to support our IOT platform. The platform initializer will achieve this by initializing virtual machines on the host and sets up the environment on virtual machines. It deploys all the module.

Services and Subsystems

The services of the platform initializer are:

- Installs the required dependencies on the host machine(virtual machines in our case)
- Initialise and start all the platform services(modules and managers)
- Tracks and maintains the health of various platform services through kafka
- Components that will be initialized on the platform
    1. Application Manager
    2. Scheduler
    3. Application Deployer
    4. Node Manager

- The platform initializer can run all the platform services (or modules) on one single virtual machine or then can be deployed on multiple virtual machines.



Block diagram of Platform Initializer

Interaction with other modules:

Interaction between platform initializer and deployment manager - Plaform initializer will initialize the scheduler and the deployer to get them up and running.

9.8 Application Manager : Request Manager, Authentication & Authorization Manager

Lifecycle

Application Manager is responsible for handling all the requests generated from outside of the platform and delegating it to the adequate microservice. It acts as a single point of contact for all the outsiders. It is also responsible for providing the UI to the users.

This module will take configuration files from Application developers and Platform Configurers through UI. We will validate the structure of zip files through the validation module. We will provide Authentication through UI. Users can login

and deploy application they wanted, they can also see the notifications for applications.

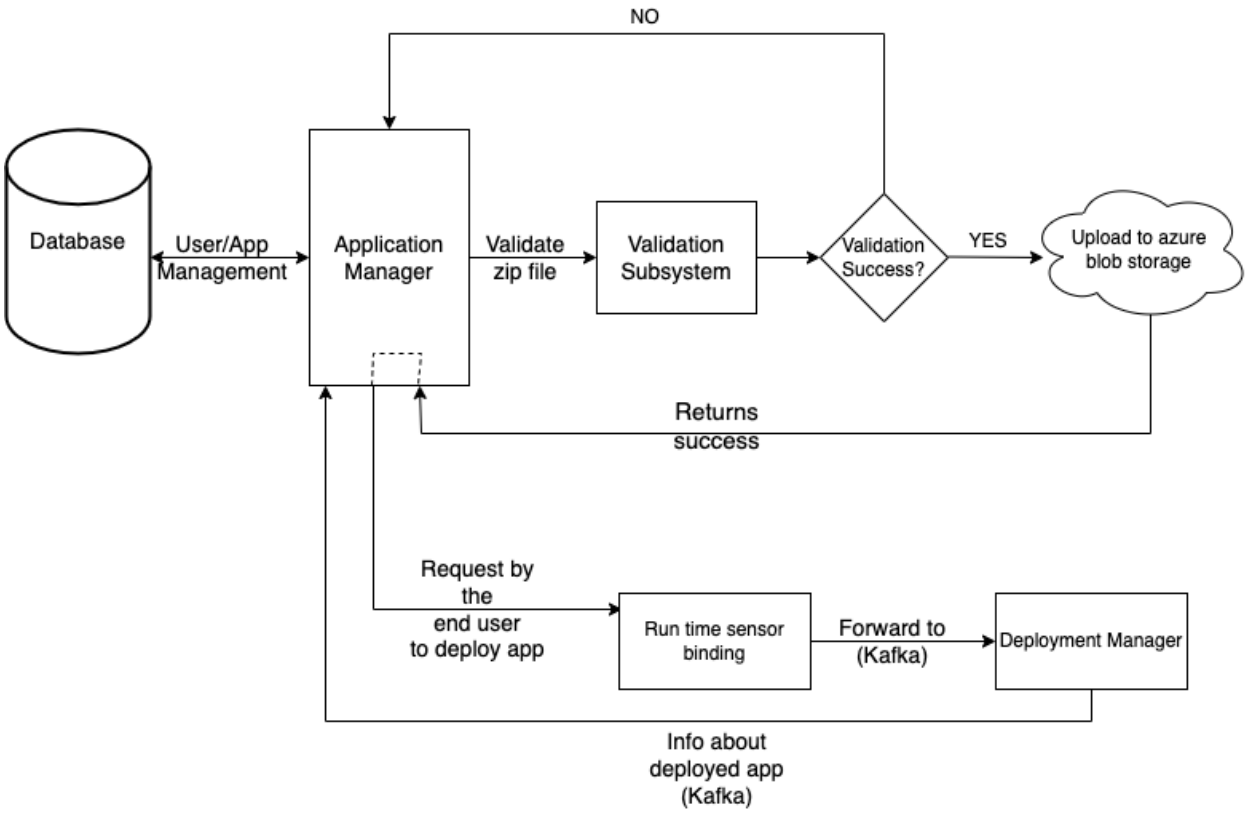
- Sign up : Application developer and end users can use this module for sign up
- Login : Application developer and end users can login through the login module.
- Dashboard : Application developer can upload application(pthon script) and meta files in zip file on the console. He can download the format of meta file for the algorithm from the console.

Authentication and Authorization Manager module is responsible for authenticating and authorizing the requests. For that it maintains the user databases and generate Tokens for logged in users

Subsystems

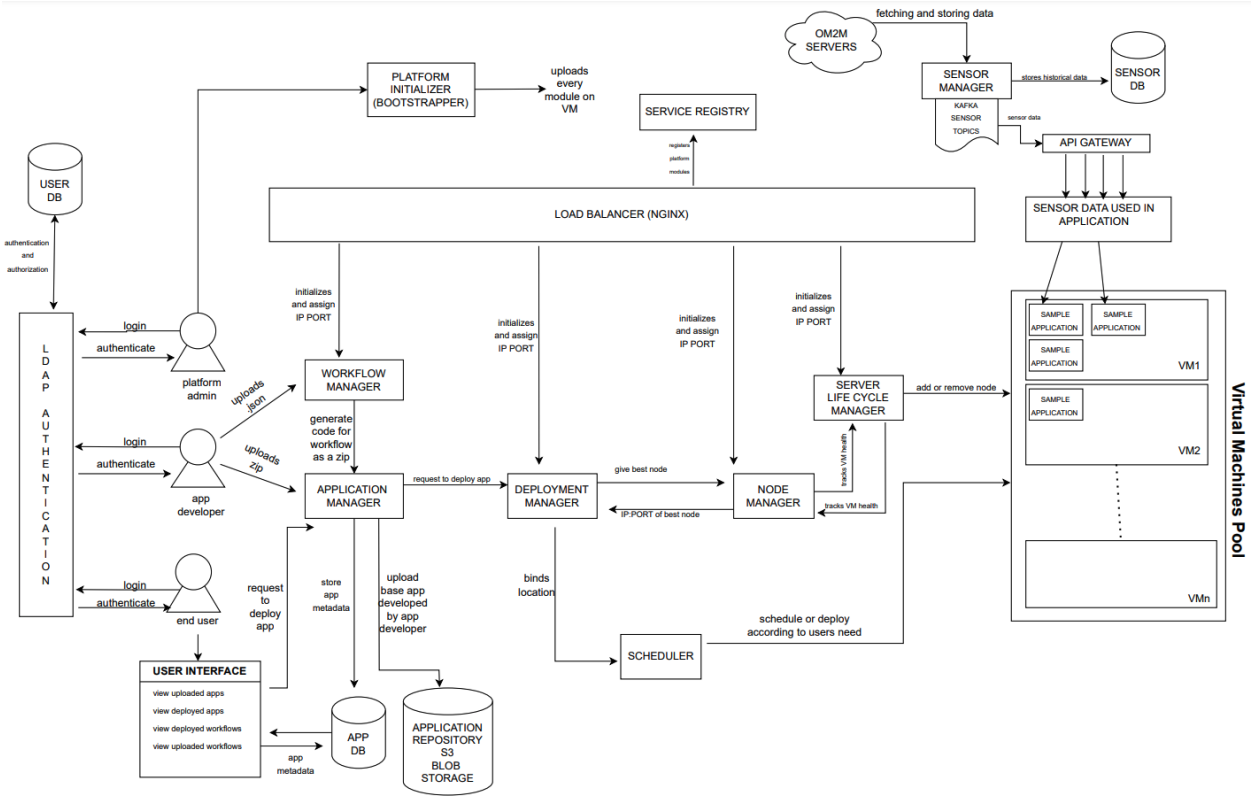
Request Manager **-** It takes care of the various requests coming from different actors on our Platform. Firstly, A zip file will come from the application developer which contains the Python code and some config files. It needs to be deployed at various nodes. Now this module, also handles the request from Platform configurer for adding various information regarding the new sensors. Lastly the request comes from end user for deploying the application/ using the application. All requests have a well defined workflow to execute.

Authentication and Authorization Manager **-** This deals with handling all the authentication info from every possible actor. This handles the very first thing that happens before any request. For this it maintains several databases for perform the authentication. So the main function of this subsystem is to deny any unauthorized access.



Block diagram of Application Manager

Interaction with other modules:



- **Application manager with Scheduler** - Application Manager will provide Scheduler with the application and the meta file of the application, where information related to scheduling are present (start and end times, job type etc)

- **Application manager with Health Management Subsystem** - The monitoring and fault tolerance module interacts with Application manager, Authentication Manager. For all these modules, periodic monitoring will be provided through heart beat messages.

- **Application manager with Sensor Management Subsystem** - Sensor manager will provide sensor's information to the application manager which will be stored at application database along with other application

information and passed to scheduler in form of meta data with application's other information.

- **Application manager with Controller** - Manager Control manager will send Controller's information to the application manager and that information will be stored at application database along with the other app information

- **Application Manager with Deployment Manager Subsystem -** The application manager after receiving the config and the code files from the application developer needs to deploy the application on an appropriate VM. For this it needs to interact with the above mentioned subsystem.

- **Application Manager with Health And Load Management Subsystem** - For deploying an application, the application manager needs to know the location of the virtual machine for deployment. The Health and Load Management Subsystem will provide the id of the VM with maximum health.