Internals of Application Server

Application Deployment Platform

Under the guidance of Prof. Loganathan

Group 1

INTRODUCTION:

The ADP simplifies the process of deploying and managing digital applications across various domains. It offers seamless integration capabilities, effortlessly connecting to diverse data sources and leveraging the oneM2M platform for comprehensive data collection. Designed with user experience in mind, the ADP boasts:

- **Flexible configurations:** Tailor the platform to meet your specific needs.
- **Robust security measures:** Ensure the safety and integrity of your applications and data.
- **Efficient resource utilization:** Optimize resource allocation for cost-effectiveness and performance.

Streamlined Application Management:

The ADP streamlines the entire application lifecycle, from deployment to management. This empowers you to:

- **Effortlessly set up applications:** Get your applications up and running quickly and easily.
- **Simplify application management:** Manage all your applications from a centralized location.
- **Maintain diverse applications:** The ADP is compatible with a wide range of digital solutions.

Focus on User Experience:

The ADP prioritizes a user-friendly experience, allowing you to:

- **Intuitively configure settings:** Easily tailor the platform to your specific requirements.
- **Monitor application performance:** Gain insights into application health and performance.
- **Ensure data security:** Benefit from robust security features to protect your valuable data.

By leveraging the ADP, you can streamline application deployment and management, fostering greater efficiency and innovation.

REQUIREMENTS

1. Overview

a. Introduction

- We are using **Machine-to-machine (M2M)** communications for automated data transmission and measurement between electronic or sensory devices. For gathering data and storing data we get from M2M communication we will use ThingsBoard. **ThingsBoard** is an open- source IoT platform for data collection, processing, visualization, and device management.
- We will need a **sensor manager** to manage our sensors and **sensor stream** to convert the datawe get from the sensor into a stream to handle large numbers of files or data. **Stream handler** is required to handle the incoming stream of data from the sensor stream whenever a requestto get the data is made.
- **Workflow manager** consists of a pool of sequential microservice pipelines. Every sequence is arranged in some fashion to fulfill a desired service. Whenever any request comes from the request handler, workflow manager will check available microservices required to execute the desired workflow from the Microservice Manager. Microservice Manager will provide the instance of that microservice to the workflow manager so that it can execute the desired service.
- **Load balancer** is used to balance loads among the various modules. Whenever for a service, the number of requests crosses a particular threshold, load balancer will create/ divert the requests to new instances of that service to balance the stream of requests.
- There are a few ways to make requests to our application to make use of microservices. One way to make use of APIs, so that external applications can make requests to our platform. The other way is that we have predefined requests which users can upload on our application and then the **scheduler** will perform those requests like some requests that need to perform on a regular basis in a timely fashion.
- We will need **Kafka** for communication between various components of our platform and is used to pass pipelined data also known as topics throughout our platform via the help of a processing queue. It allows the microservices to get real time data with updated values and zero fault tolerance.

b. Scope: The project is intended to design and develop a platform where users can run/ host their IOT based application. The platform will have the capability to interact with various sensors independent of its type. We are using oneM2M platform to gather data, so the platform is independentof the protocol via which data is coming to the platform. The platform is capable of understanding the application requirement dynamically via config files and selecting the most efficient workflow(s) for building it.

2. Intended Use

a. Intended Use: The primary purpose of our IoT platform is to enable organizations to deploy and manage large-scale IoT solutions in a secure and scalable manner. The platform connects devices and applications, manages devices and data, analyzes data, and provides security and scalability features. Overall, our IoT platform provides an end-to-end solution for deploying and managing IoT solutions that can provide real-time insights, optimize processes, and enable automation.

b. Assumptions and dependencies: An IoT platform has a complex architecture that relies on various assumptions and dependencies to function effectively. Some of the key assumptions and dependencies of our IoT platform include:

1. The devices and sensors used in any IoT solution can communicate with each other through standard protocols.
2. For the platform to offer useful insights and guide decision-making, the information gathered from gadgets and sensors is accurate and trustworthy.
3. It can integrate with services and other third-party applications to provide more functionality and value.
4. The services and the module maintaining and running the services would be hosted in cloud VMs, which also are dynamically scalable depending upon the load and requirements as maintained by the platform.

3. System Features and Requirements

a. Platform Requirements

i. Deployment of the platform: Deployment of a platform involves the process of making the platform available and operational in a production environment. The platform should provide capability of deploying the supported types of applications onto servers that are catered to a specific configuration. The server should be capable of providing high-end resources which in turn would allow the platform to be efficient and scalable. The platform should also provide necessary information about the modules, environment variables and theirrelative paths.

ii. Different actors on the platform: Actors on an IoT platform include input and output devices, physical devices, and software applications. They receive data, process it, and initiate appropriate actions. Examples include a smart thermostat adjusting temperature and a software application sending alerts for maintenance. Actors may include application developers, platform configurers, and end-users specifying sensor type, number, and location.

iii. Applications overview of the platform: This IOT based platform provides a foundation of hardware, software, and cloud services that can be customized and tailored to meet the needs of various industries and domains. Developers can use this platform to create innovative solutions that leverage real-time data collection and analysis, remote monitoring and control, and automation to achieve various

business objectives.

Here are some example applications that can be built on top of this IoT platform:

1. Smart Home Automation: An IoT platform can be used to control various smart devices in a home such as lights, thermostats, security cameras, and appliances, allowing homeowners to remotely monitor and control their home environment.
2. Smart Agriculture: An IoT platform can be used to monitor soil moisture, temperature, and other environmental factors, allowing farmers to optimize crop yields and reduce water usage.
3. Healthcare Monitoring: An IoT platform can be used to collect health data from wearable devices and other medical sensors, enabling remote health monitoring and personalized healthcare applications.

b. Functional Requirements

i. Application:

1. Interaction with sensors: APIs enable developers to access registered sensors by passing sensor IDs, and the platform handles interactions and returns data in a usable format. The sensor manager module streamlines this process by managing sensor discovery, configuration, and data retrieval, serving as an abstraction layer between sensors and applications.

2. Development of application on the platform: Our IoT platform provides infrastructure and services to manage and process data from connected devices, but the application logic and UI are developed separately. The platform offers APIs for the applicationto interact with services such as retrieving data from sensors, storing data in the cloud, and triggering alerts based on sensor data. Applications can be deployed on the platform, which hosts and maintains them to ensure availability and performance.

3. Identification of sensors for data binding: Sensors registered on IoT platforms have unique IDs and metadata, used to identify and select sensors for data retrieval. The sensor manager module, running within the platform, manages interactions between applications and sensors, handling multiple simultaneous requests. The module identifies the relevant sensor based on metadata in the request, initiates a data binding operation, and returns the requested data to the application or node.

4. Data Binding to the application: The data binding process involves retrieving data from sensors, preprocessing the data, and storing the data in a format that can be easily consumed by the application. The IoT platform and sensor manager play a crucial role in facilitating this process and ensuring that the data is transferred and stored efficiently and securely.

iv. **Starting services:** Scheduler start the services according to the start_time programmed.

v. Communication model:

1. Communication model must be able to **ingest messages** from various modules and store themfor processing.
2. **Reliable delivery** of messages to the appropriate requesting module.
3. **Scalable:** Must be able to handle high volumes of data and support a large number ofconcurrent users and applications.
4. **Identity and access management:** The Model must use various authentication and authorization services, including their access to resources.
5. **Inter-Module Communication:** The Model must have the feature of transferring messages between modules.

vi. Server and node manager: The deployer initiates a request to the node manager for starting a new server node whenever a new app is deployed. If the app startup process is successful, the node manager module communicates the IP and port details of the new server back to the deployer, and send a mail via the notification service to the deployer.

vii. Deployment of application on the platform:

1. **Compatibility:** The platform must be compatible with the application and its components,including the operating system, programming language, and dependencies.
2. **Monitoring and Logging:** The platform must provide monitoring and logging capabilities to ensure the application is performing as expected and to detect and troubleshoot issues.
3. **Scalability:** The platform must be able to handle changes in the workload and traffic, as wellas provide scalability options such as auto-scaling.
4. **Reliability:** The platform must be reliable and available, with high uptime and minimal downtime, This can be achieved through use of Load Balancer, Requirement Manager
5. **Security:** The platform APIs must have a validation check via the use of API keys. We use LDAP server and JWT tokens to ensure security at each step.

viii. Registry & repository:

1. **App repository:** The application manager will keep track of the running applications with their data stored in a database or data matrix. This data matrix must contain the application instance ids, server instance on which they have been ran, current state of the application
2. **Sensor Repository:** Sensor Management Module contains a list of registered sensors on the platform. This list is stored in DB. The information contains meta information related to each sensor.

ix. Load Balancing: To implement load balancing for microservices in a platform, we can follow below process:

1. **Choose a load balancing method**: There are several load balancing methods (like Round- robin load balancing, least connections load balancing, IP hash load balancing, etc.), we can choose the one that is best suited as per requirement.
2. **Select a load balancer**: There are several load balancers available (like Apache Load Balancer, HAProxy Load Balancer etc.) that can be used to implement load balancing for microservices.
3. **Configure the load balancer**: Once we select the load balancer, we can configure it to distribute incoming requests to each microservice. This may involve setting up routing rules, configuring SSL certificates, and other configuration options specific to your load balancer.
4. **Set up service discovery**: We need to ensure that each microservice can find and communicate with other microservices in the platform. This may involve setting up a service registry or implementing a service mesh.
5. **Monitor performance and availability**: We can Implement monitoring tools to track the performance and availability of each microservice and the load balancer. This allows us to proactive maintenance and updates to prevent downtime.
6. **Test and optimize**: Once load balancing is set up, we need to test the platform to ensure that each microservice is being utilized effectively and that performance and availability are optimized. We can modify load balancing settings as required to improve performance.

By following these steps, you can ensure that each microservice is distributed evenly across multiple servers, improving performance, and minimizing downtime.

x. **Interactions between modules:** In an application platform, modules interact with each other to provide a complete and cohesive solution to the end-users. The interactions between modules can be categorized into three main types: communication, data sharing, and control:

1. **Communication**: Modules may need to communicate with each other to exchange information, trigger actions, or coordinate activities. This communication can happen in several ways (like message passing, or remote procedure calls).
2. **Data sharing**: Modules may need to share data with each other to perform their respective functions. This data can be shared through shared memory, file systems, or databases. The sharing of data needs to be done in a controlled and secure manner to prevent unauthorized access or modification.
3. **Control**: Modules may need to coordinate their activities to achieve a specific goal or meet a certain requirement. This coordination can be achieved through various control mechanisms, including synchronization, concurrency, and task management.

To ensure effective interaction between modules in an application platform, we need to follow thesesteps:

1. Define clear interfaces and protocols

2. Use standard communication mechanisms
3. Implement security measures

4. Monitor performance and scalability

xi. Packaging details: Packaging an application involves bundling all the necessary files into a single package that contains the executable, config files and dependencies.

xii. Configuration files details: Configuration files, either in XML or JSON format, will define parameters, options, settings, and preferences for an application. Initially it will contain default settings which could be later modified to change the behavior of the application.

xiii. Interaction of different actors with the platform: Actors could be either input devices such as sensors or cameras or output devices, such as displays or actuators. Actors on an IoT platformare responsible for receiving data, processing it, and initiating appropriate actions based on the results of that processing. These devices or applications can perform various actions based on the data collected from sensors or other devices connected to the IoT platform.

c. Non-Functional Requirements

i. Fault tolerance

1. **Platform**

   Redundancy - This means having multiple instances of the same component, service, or application running simultaneously, so that if one instance fails, the others can take over the workload and ensure that the system continues to operate. We might use the techniques below to achieve redundancy.

2. **Application: Error handling** in a way that the user experience is smooth and the application does not crash under any circumstances.

ii. Scalability:

Platform & Application: To achieve scalability we use:

- A modular architecture
- Employ horizontal scaling
- Implement asynchronous

processingWe would be using **Kafka**
which enables:

- Distributed Architecture
- Replication
- High Throughput

iii. Accessibility of data

1. Application: Access data using **REST APIs**.
2. Sensors: We use **OneM2M** which is a standard for Machine-to-Machine (M2M) communications that aims to provide a common framework for collecting,

exchanging, and

managing data between different devices, networks, and applications. OneM2M provides a standardized approach to handling data from different devices and sensors.

ThingsBoard can integrate with **oneM2M** by using its **MQTT** .To receive data via **MQTT**, ThingsBoard needs to be configured with the **oneM2M MQTT** broker address, username, andpassword. Once the connection is established, ThingsBoard can subscribe to the desired topics to receive data from **oneM2M** devices. The data received via **MQTT** can then be processed and stored in ThingsBoard's database for visualization and analysis.

iv. Specification about application:

- **Reliability:** This refers to the application's ability to perform as expected and avoid unexpected failures.
- **Performance:** Performance requirements describe how well the application should perform interms of speed, throughput, and response time.
- **Usability:** Usability requirements describe how easy the application should be to use and howwell it should meet the needs of its users.

v. UI and CLI for interaction: Terminal based UI for interacting with platform.

vi. Security - Authentication and Authorization

- Password Based
- LDAP

vii. Monitoring: Provision, monitor and control your IoT entities in secure way using richserver- side APIs provided by ThingsBoard.io

viii. Persistence: We are looking forward to below approaches:

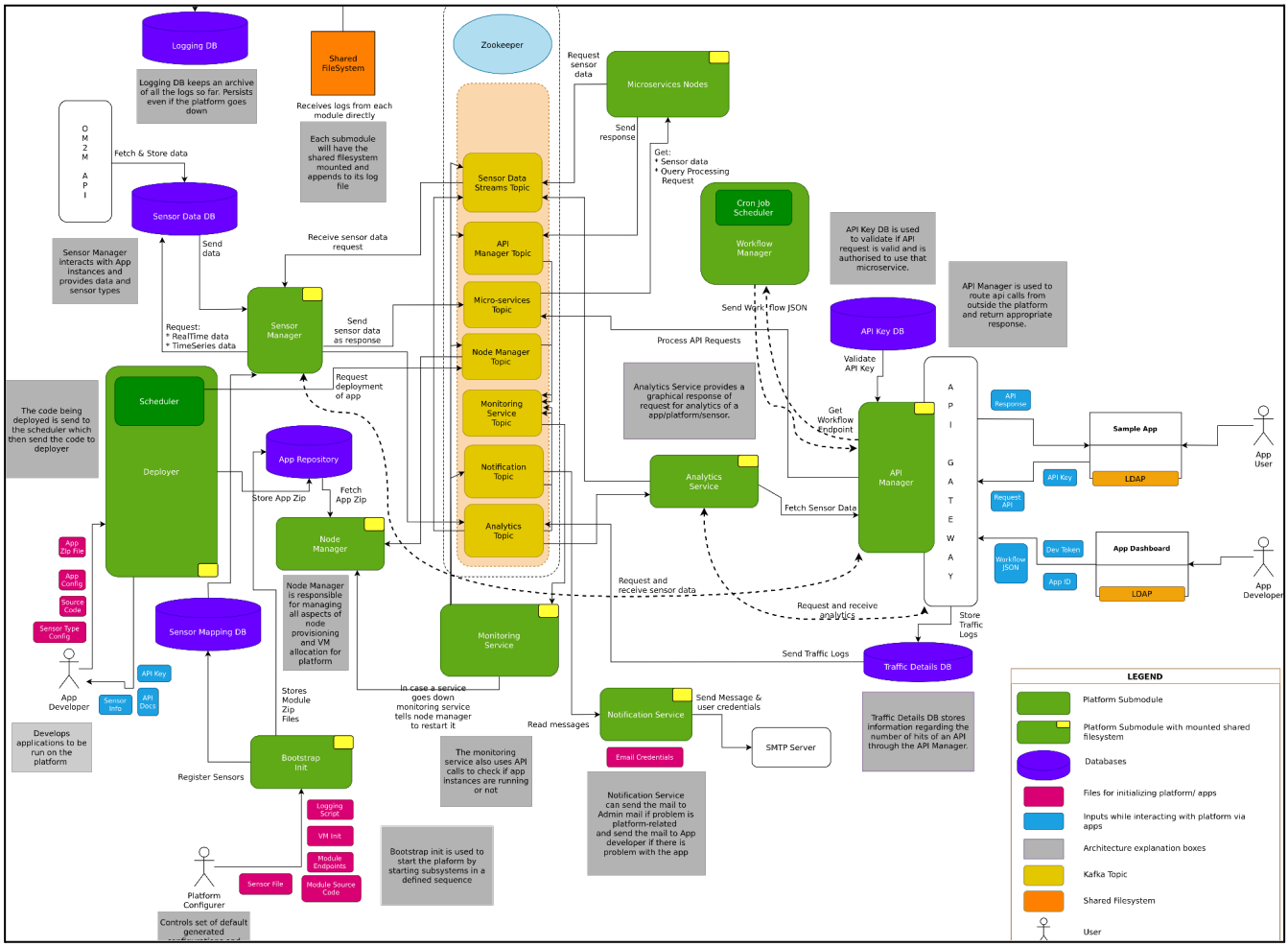
- File-based Persistence
- Relational and non Relational (but schema based) Databases

d. External Interface Requirements:

1. OAuth for authentication purpose
2. APIs for managing bare metal resources from AWS /Azure platform.

4. List the key functions:

a. A block diagram listing all major components:



Platform major components interaction

b. Brief description of each component:

- **Refer overview 1.a**

c. List the 5 major parts each team will take one part:

1. Monitoring Service and Load Balancing Service
2. Node/VM Manager
3. API Manager,Communication Manager and Analytics Module
4. Deployment Manager and Bootstrap Service
5. Sensor Manager

5. Use cases:

a. What users can do with the solution:

1. Our Platform provides a platform for developing and managing Internet of Things (IoT) devices and solutions. Here are some things that users can do with our Platform:

2. Connect and manage IoT devices: Provide a way to securely connect and manage IoT devices,including device registration, authentication, and communication.
3. Collect and analyze data: Provides tools for collecting, storing, and analyzing data from IoT devices, including real-time analytics and machine learning.
4. Build custom applications: Provides tools and frameworks for building custom applicationsand workflows that interact with IoT devices and data.
5. Monitor and manage IoT solutions: Provides monitoring tools for performance and health ofIoT solutions, including alerts and diagnostics.
6. Scale and optimize solutions: Our Platform provides tools for scaling and optimizing IoT solutions, including automatic scaling and load balancing.

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

- Device manufacturers: These are companies that create IoT devices such as sensors, controllers, and gateways. They use our platform to connect their devices to the cloud and provide their customers with a secure, scalable, and reliable IoT solution.
- Solution architects: They design and develop IoT solutions for various industries such as healthcare, manufacturing, and transportation. They use our platform to build, deploy, and manage IoT solutions that collect and analyze data from connected devices to gain insightsand improve business operations.

In summary, the users of our platform can come from a variety of domains and roles, such as device manufacturers, solution architects, IT administrators, and business owners. They use our platform to develop, deploy, and manage IoT solutions, collect and analyze data from connected devices, and gaininsights to improve business operations.

c. Usage scenarios:

- Smart Home Automation: Smart home automation systems can be created to remotely control devices such as thermostats, lights, and locks. For instance, using Amazon Alexa or Google Assistant, users can control their home devices through voice commands.
- Connected Vehicles: With our platform, connected vehicles can be built to collect and analyze data from sensors, cameras, and other sources. The data collected can help car manufacturers understand customer driving patterns, identify potential issues, and provide predictive maintenance services.
- Agriculture: Our platform can be used to improve crop yield and reduce water waste in agriculture by providing insights into crop health and moisture levels. Farmers can use our platform to monitor soil moisture levels, temperature, and humidity to optimize crop growth.

6. Primary test case for the project:

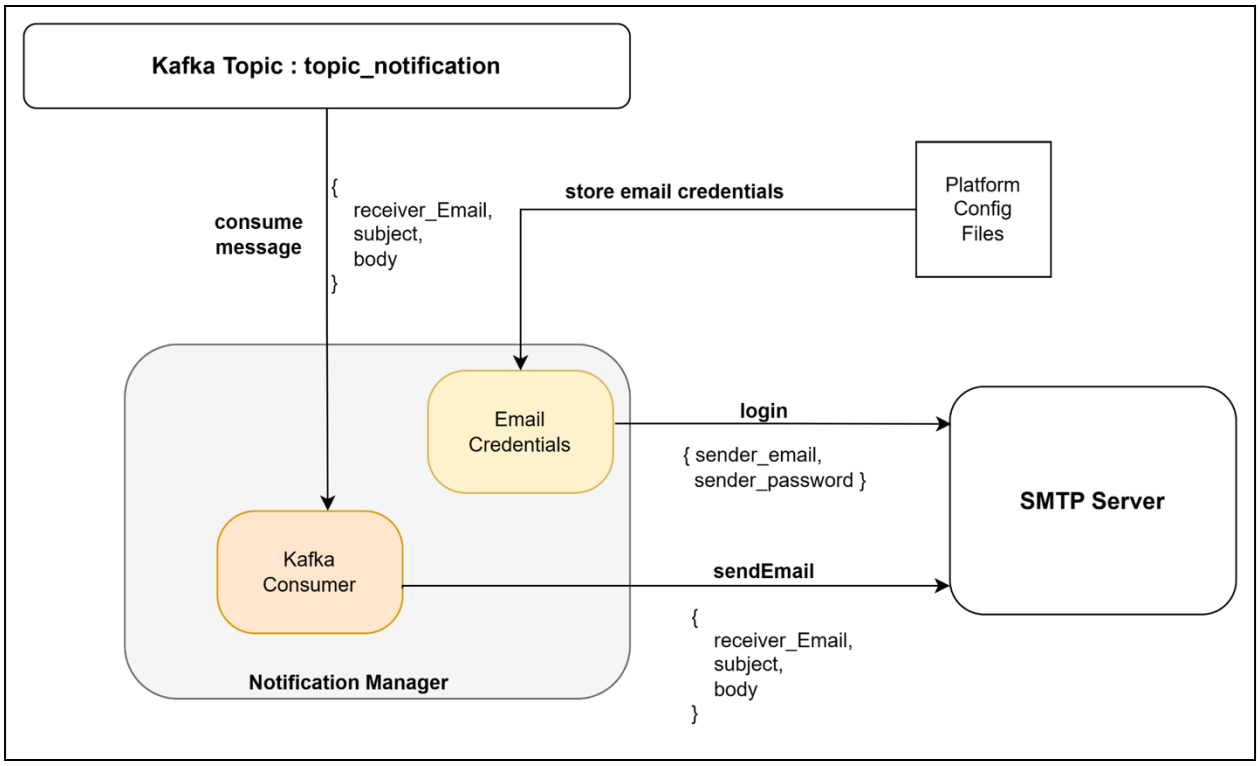
a. **Name of use case: SMART LIGHTING**
b. **Domain/company/environment where this use case occurs:** Home automation, Classrooms automation.
c. **Description of the use case purpose, interactions, and what will the users benefit:** Smart Lighting enables users to control their home lighting system using their smartphones or voice assistants, providing convenience and energy efficiency. The system can automatically turn on and off the lights based on time, presence detection, or ambient light levels, ensuring energy conservation and cost savings.
d. **What is the "internet angel" around these things:** The internet angel around Smart Lighting is the wireless connectivity and automation enabled by IoT devices and platforms, which help users remotely monitor and control their lighting system from anywhere, using cloud-based applications.
e. **Information model:** The system collects data from various sensors, such as motion sensors, light sensors, and occupancy sensors, to determine the lighting requirements in a room. The user can also set the lighting schedule or create personalized lighting scenes, which can be stored in the system's database.
f. **How is location and sensory information used:** The system uses location data to determine the user's presence and turn the lights on or off automatically. The sensory data collected fromvarious sensors, such as motion sensors and light sensors, is used to adjust the lighting levelsin real-time.
g. **Processing logic on the backend:** The backend processing logic involves data analysis and machine learning algorithms that analyze user behavior and preferences to optimize the lighting system's performance. The system can also perform predictive maintenance, detectingand alerting users to any faults or errors in the system.
h. **User's UI view- what is their UI, where, and all will they do with these Systems:** The user can access the Smart Lighting system through a mobile app or a voice assistant such as Alexa or Google Assistant. The UI displays the lighting schedule, personalized lighting scenes, and real-time lighting levels in different rooms. The user can also customize the lighting levels or turn the lights on/off manually using the app or voice commands.

7. Subsystems

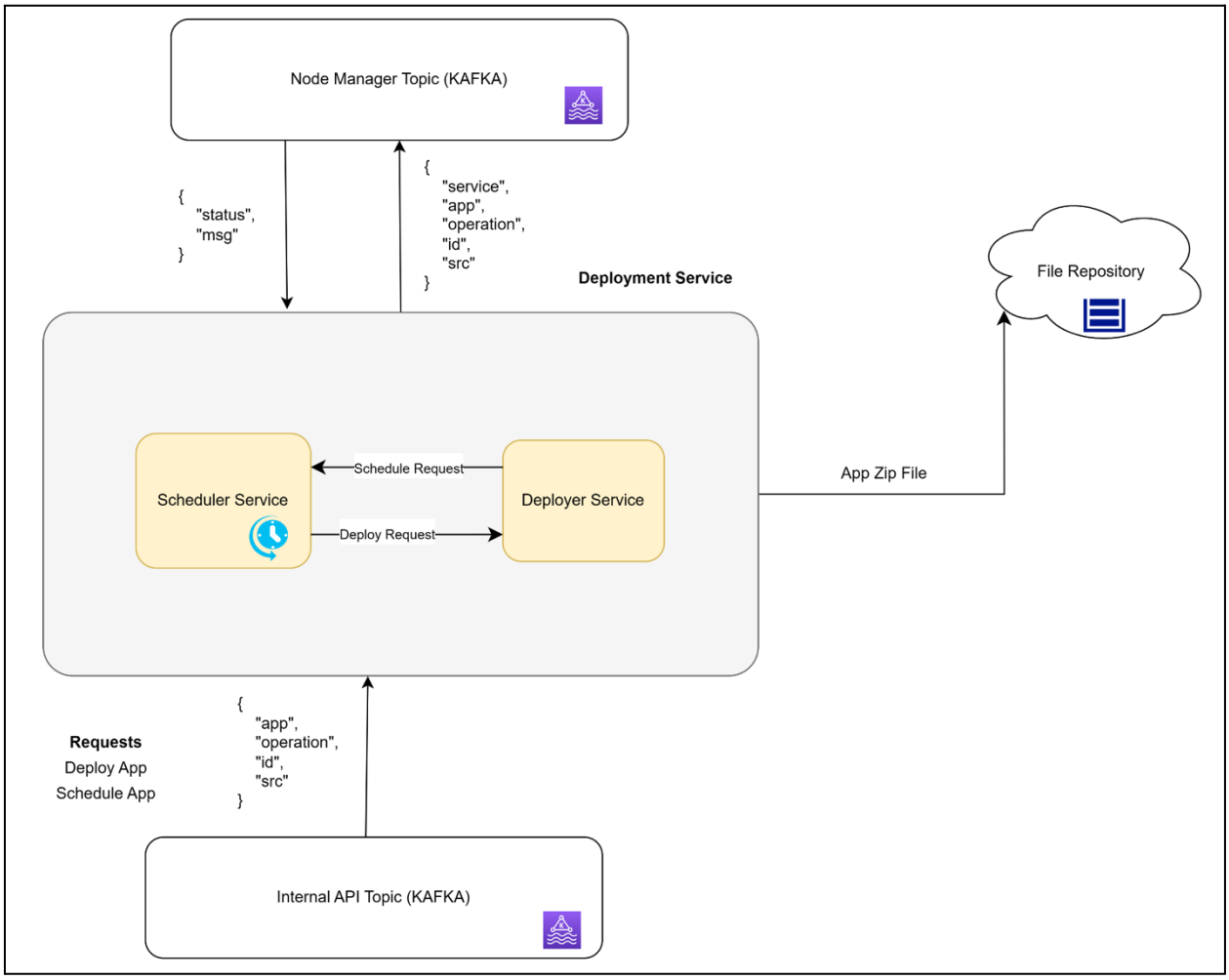
a. Key subsystems in the project:

1. Monitoring Service and Load Balancing Service
2. Node/VM Manager and Deployment Manager
3. API Manager and Communication Manager
4. Deployment Manager and Bootstrap Service
5. Sensor Manager

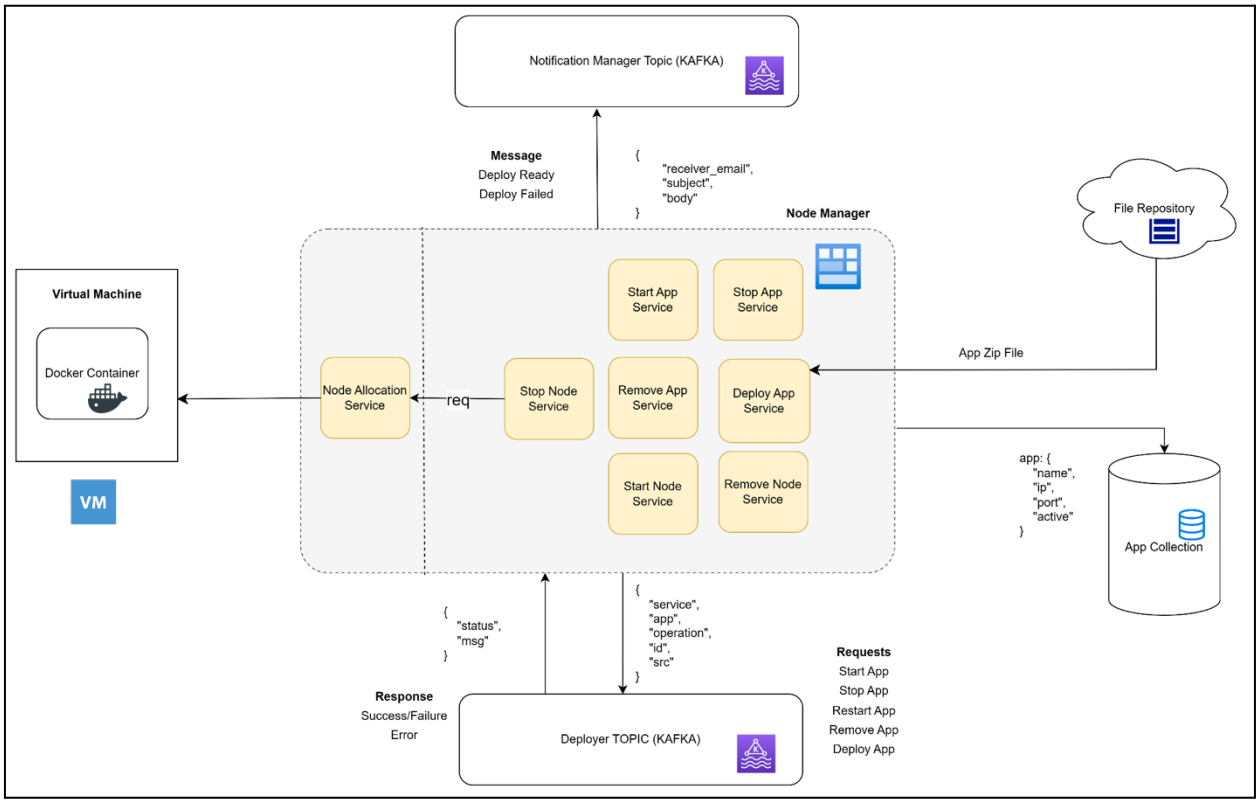
b. A block diagram of all subsystems:



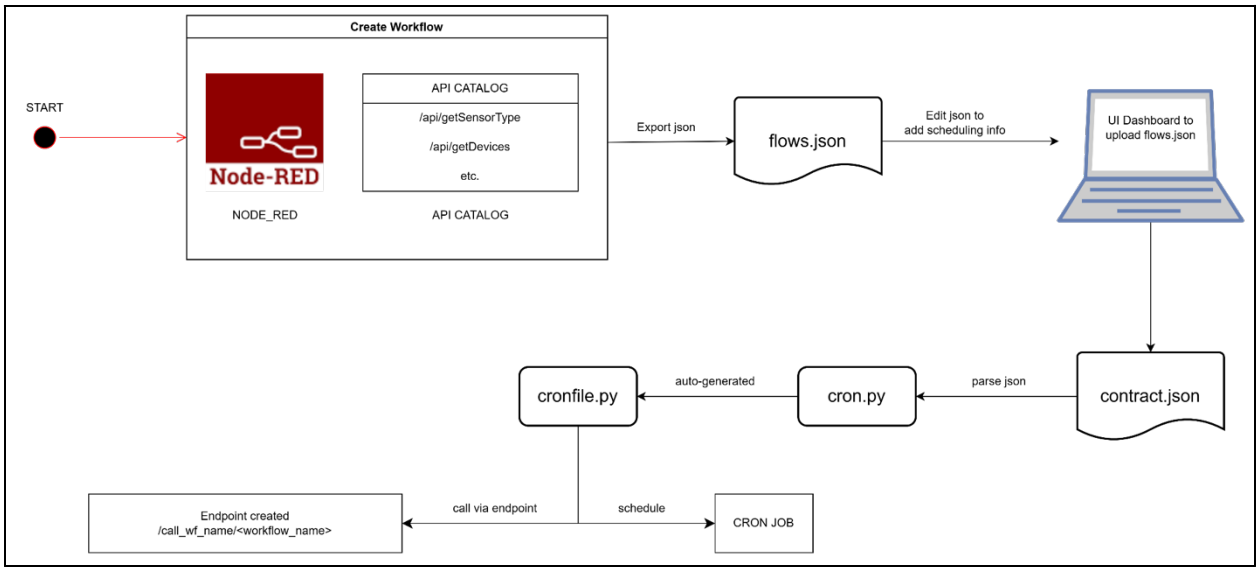
Monitoring Service



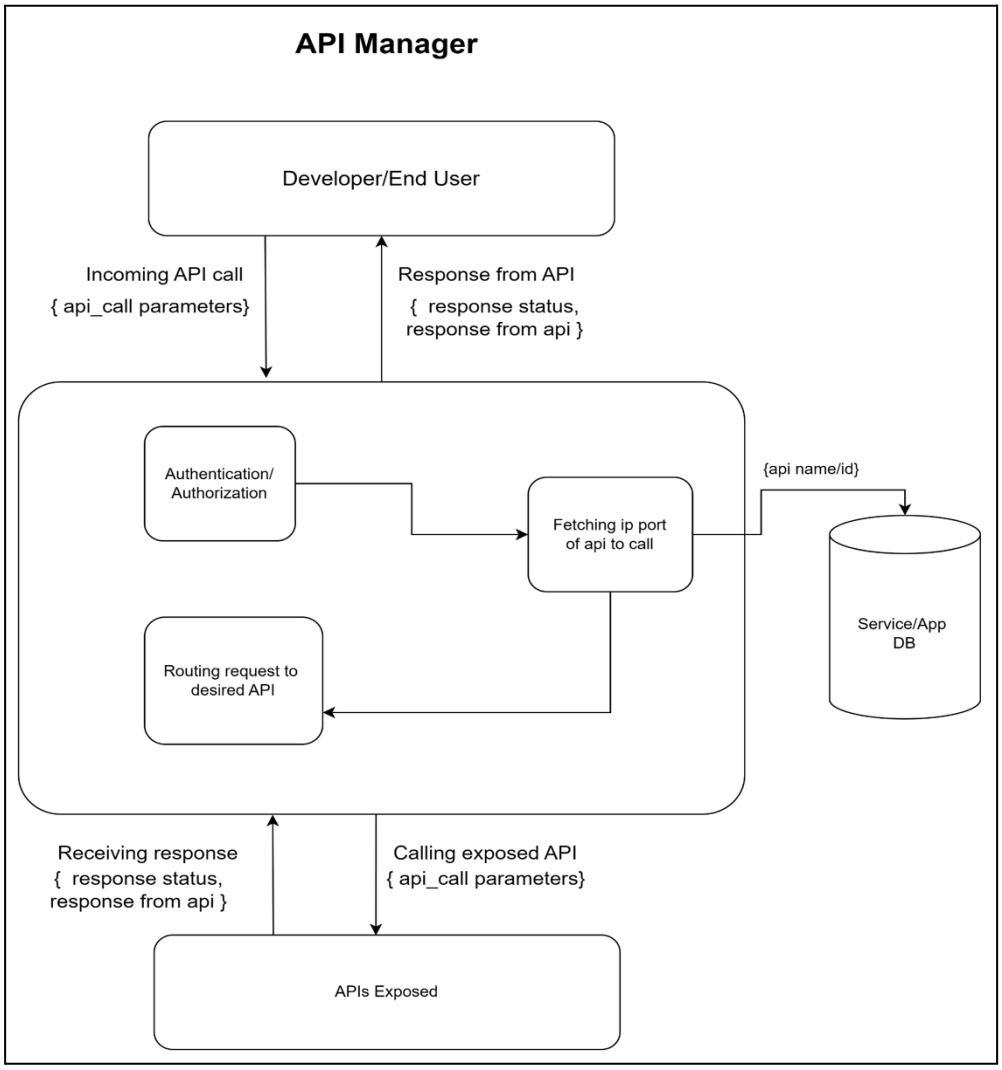
Deployment Module



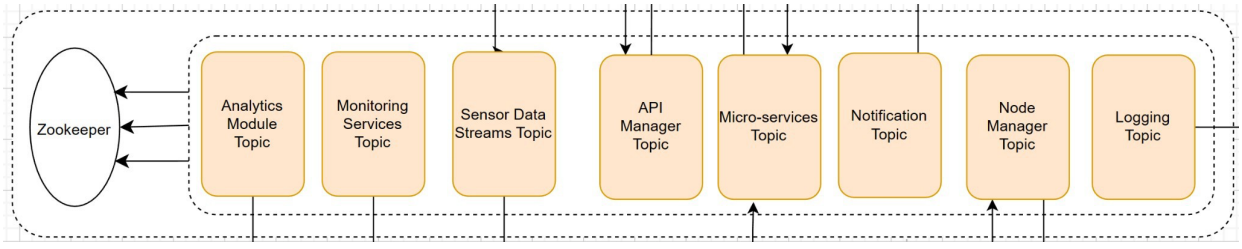
Node Manager



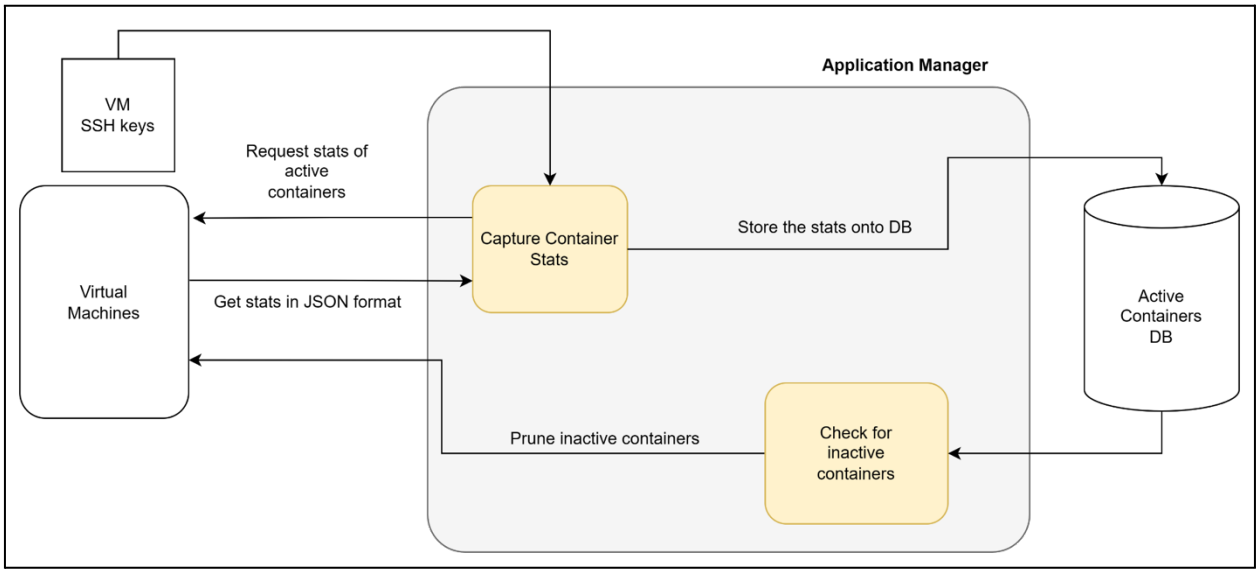
Workflow Manager



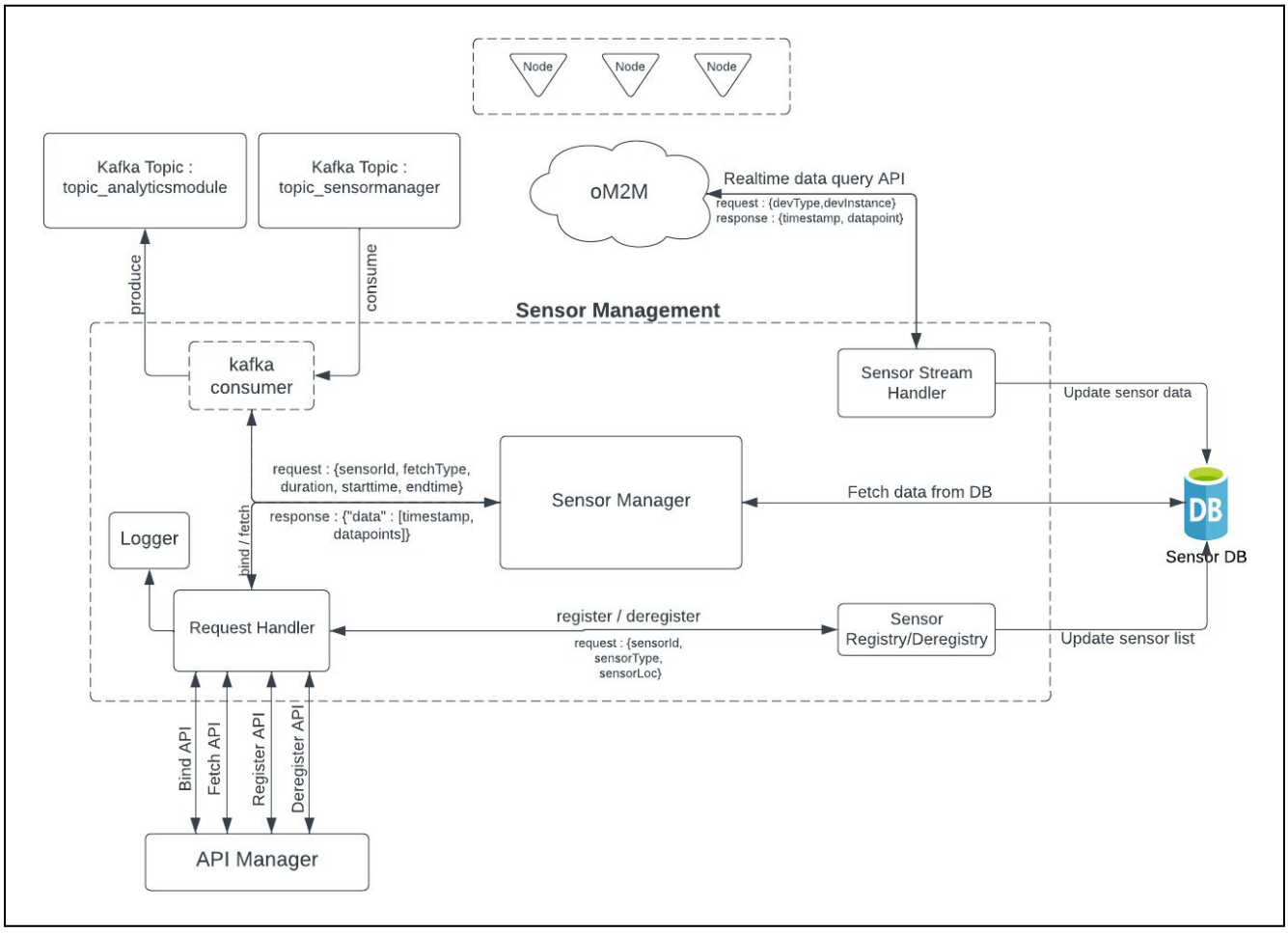
API Manager



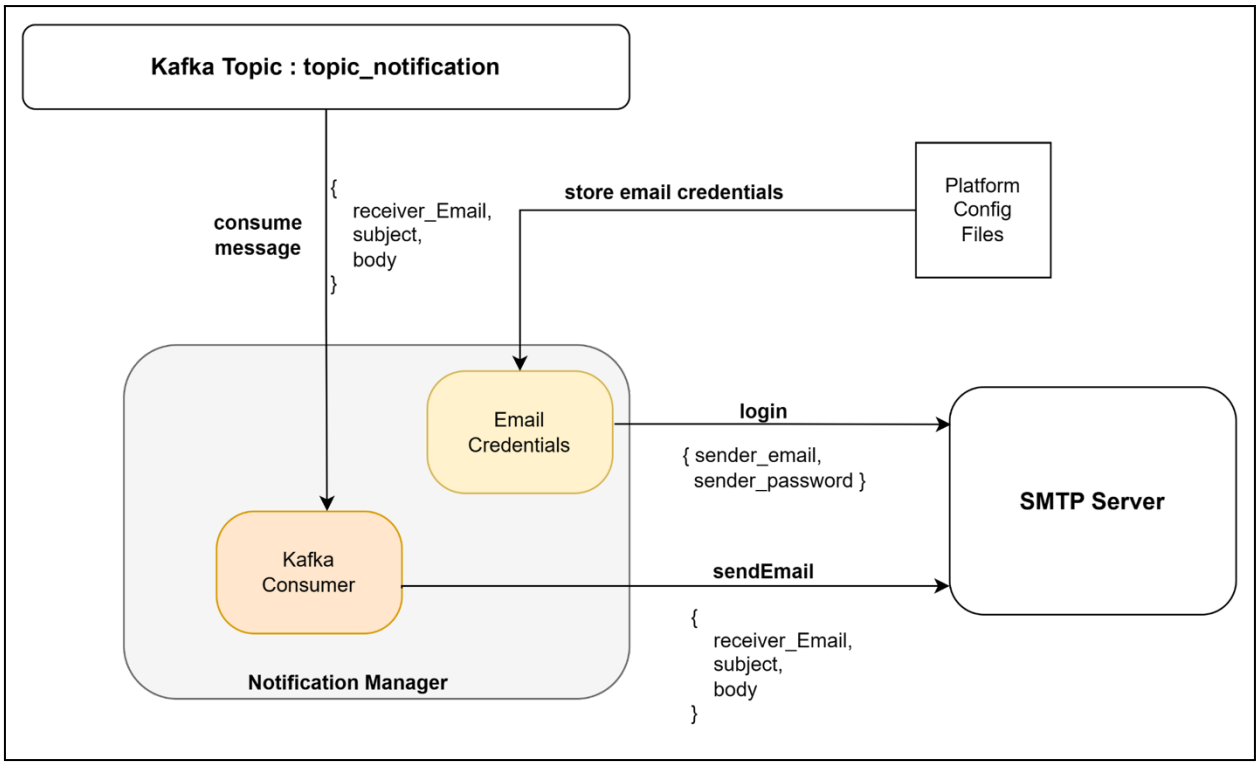
Communication Manager



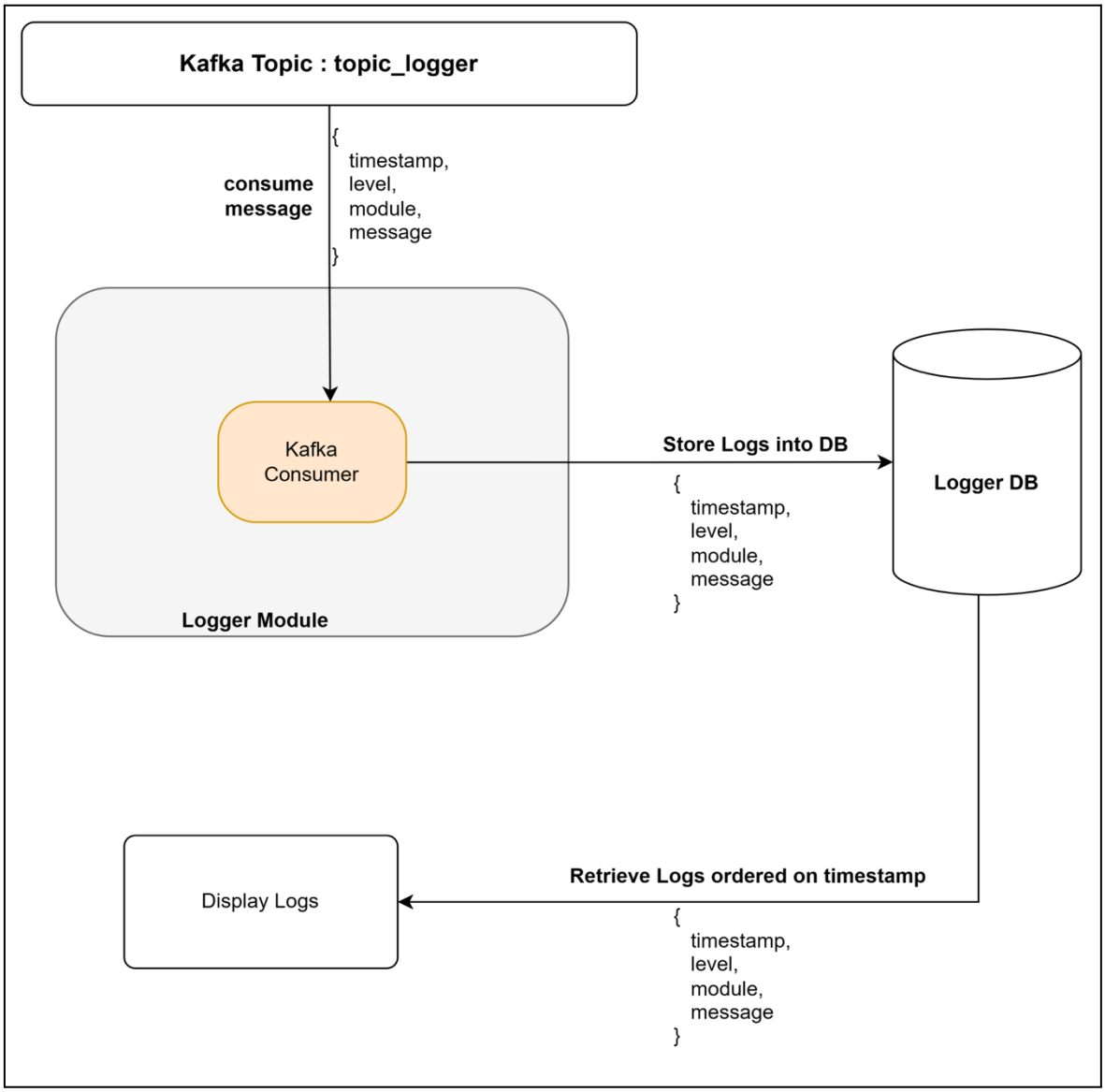
Application Manager



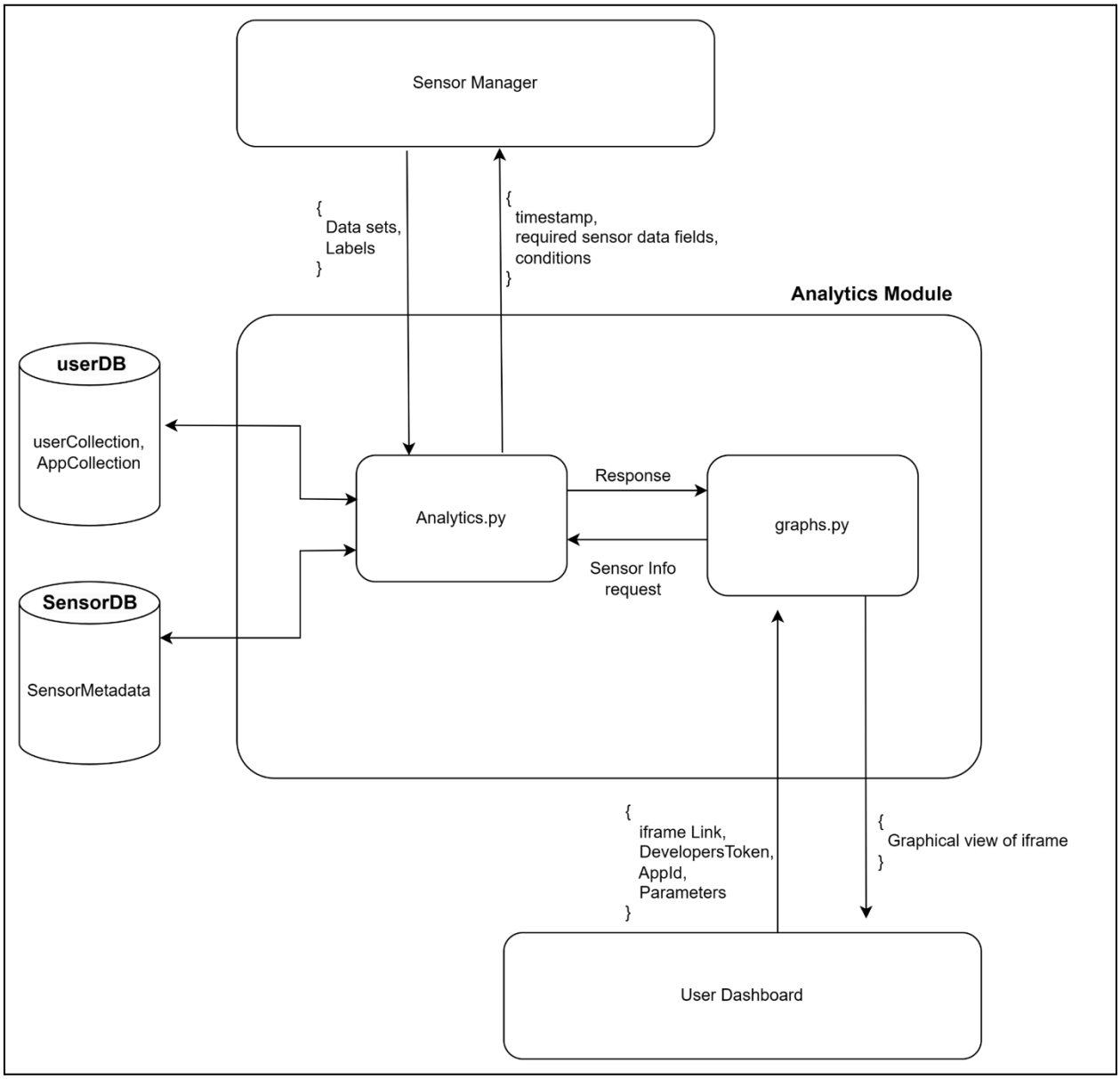
Sensor Manager



Notification Service



Logging Service



Analytics Module

c. Interactions involved across these subsystems:

Interaction of Sensor Management Module: Sensor management module interacts with monitoring service, Communication Manager. The Monitoring Service remains in constant connection with the Sensor Management Module and keeps checking the health status of module's subcomponents. Communication Module provides a channel for other modules of system such Node VM Manager etc., to perform API request on module's Endpoint through communication manager.

Interaction of Communication Module: Communication Module will interact with every other module and will facilitate passing of messages between them these messages can be APIrequests, API responses, Binary Files etc.
In general, the Communication Module interacts with: Sensor manager, API manager, App Deployment Manager, Bootstrap Service, Load Balancer, Monitoring Service, Service Lifecycle Manager.

Interaction of API manager: API manager will interact with communication manager. Node V/m Manager Module to retrieve the API Endpoints of Different Modules, perform Request and Return response through Manager, respectively.

Interaction of App Deployment Manager: App Deployment Manager will interact with Node V/M Manager to deploy the Application.

Bootstrap Service: Bootstrap services initiates the system once at the start of its execution, it interacts with every other module to start the module's respective required components.
In general, the Bootstrap Service interacts with: Sensor manager, API manager, Communication Module, App Deployment Manager, Load Balancer, Monitoring Service, Service Lifecycle Manager.

Load Balancer: Load Balancer interacts with Node V/M manager, API manager to distribute the load.

Monitoring Service: It interacts with every other module to ensure no module has stopped working. In general, the Monitoring Service interacts with: Sensor manager, API manager, Communication Module, App Deployment Manager, Bootstrap Service, Load Balancer, Monitoring Service, Service Lifecycle Manager, Node V/M Manager.

Node V/M Manager: Node V/M Manager interacts with Node V/M manager, API manager to distribute the load.

Deployment Manager: Deployment Manager interacts with Node V/M manager to facilitate its service to that module.

d. External interfaces with the system

Bootstrap Service: CLI
Deployment Manager: CLI or
GUI **Platform Services:** REST
APIs **Platform App:** UI

e. Registry & Repository:

1. **App repository:** The application manager will keep track of the running applications with their data stored in a database or data matrix. This data matrix must contain the application instance ids, server instance on which they have been ran, current state of the application
2. **QoS repository:** The Monitoring Service will take response from each of the microservices and store the QoS value extracted from the response.
3. **Sensor Repository:** Sensor Management Module contains a list of registered sensors on the platform. This list is stored in DB. The information contains meta information related to each sensor.
4. **Blob Storage:** A shared filesystem that can be mounted to execute code on an newly provisioned VM and update the contents on the fly. Also help maintain the runtime logs.

8. Brief overview of each Subsystems:

a. List of the subsystems:

1. Monitoring Service and Load Balancing Service
2. Node/VM Manager
3. API Manager and Communication Manager
4. Deployment Manager and Bootstrap Service
5. Sensor Manager

c. Functional Overview of each subsystem:

1. **Monitoring Service and Load Balancing Service:**

   The monitoring service tracks platform service performance (response time and availability), sends notifications, and creates new instances of services/apps. It also provides analytics and reporting on usage and resource consumption for data-driven decision-making. Load Balancer ensures uniform load and optimal deployment of applications on available nodes. It selects the least active node based on CPU/RAM usage and requests more nodes during high workload.

2. **Node/VM Manager :**

   Node Manager is responsible for initializing all nodes in a distributed system and ensuring they start with the correct configuration parameters. It also monitors node status and reports any errors to the appropriate module.

   Deployment manager is the entity that enables the platform to allocate any number of resources that is required for the initialization of a service and also once the service has been killed deallocates the resources and does the clean-up activities.

3. **API Manager and Communication Manager:**

   The API Manager controls access to APIs, routes external requests to the appropriate API, and enforces rate limits. It allows developers to manage and select APIs for specific functionalities.

The Communication Manager facilitates data exchange between devices and different components of an IoT platform. It acts as a bridge between devices and the platform, enabling communication and data transfer. It also manages communication between different platform modules using a standardized protocol.

4. **Deployment Manager and Bootstrap Service:**

   The Application/Deployment Manager module sets up application instances on the platform, maintains the status of running applications on the platform and enables application deployment.

   Bootstrap service is the initial component when setting up an application hosting platform. It identifies the required files and initializes the necessary services to establish and sustain the platform's operation.

5. **Sensor Manager:**

   The Sensor Management Module facilitates sensor registration, identification, and live streaming. It offers services for registration, identification number retrieval, live streaming of sensor data, and retrieval of data for a specified time interval. The module includes a Logger for logging incoming and outgoing traffic and an API Gateway for requesting the service.

d. List of services/capabilities in the part:

1. **Monitoring Service and Load Balancing Service:**
   - Monitor Server/Services Performance
   - Monitor Application Performance
   - Analytics and Reporting
   - Load Balancing Service

2. **Node/VM Manager and Deployment Manager:**
   - It can trigger or kill an instance of virtual node (VM).
   - Allocation / Deallocation of resources for services.

3. **API Manager and Communication Manager:**
   - Routes request from Application to appropriate API end points.
   - API Analytics & Traffic Management
   - Communication Manager enables Communication between different modules.

4. **Deployment Manager and Bootstrap Service:**
   - Deploys an application with a given set of application files from the application developer.
   - Bootstrap service is responsible for bringing the platform to life.

5. **Sensor Manager:**
   - Registration/Deregistration service for sensors.

- Retrieval of Identification No. from geolocation and providing live stream of data for that sensor.
- Integration with API Gateway providing service endpoints

g. Interactions between this and other parts. Nature/purpose of interactions, interchange info/services:

Interaction of Sensor Management Module: Sensor management module interacts with monitoring service, Communication Manager. The Monitoring Service remains in constant connection with the Sensor Management Module and keeps checking the health status of module's subcomponents. Communication Module provides a channel for other modules of system such Node VM Manager etc., to perform API request on module's Endpoint through communication manager.

Interaction of Communication Module: Communication Module will interact with every other module and will facilitate passing of messages between them these messages can be APIrequests, API responses, Binary Files etc.
In general, the Communication Module interacts with: Sensor manager, API manager, App Deployment Manager, Bootstrap Service, Load Balancer, Monitoring Service, Node Manager.

Interaction of API manager: API manager will interact with communication manager. Node V/m Manager Module to retrieve the API Endpoints of Different Modules, perform Request and Return response through Manager, respectively.

Interaction of App Deployment Manager: App Deployment Manager will interact with Node V/M Manager to deploy the Application.

Bootstrap Service: Bootstrap services initiates the system once at the start of its execution, it interacts with every other module to start the module's respective required components.
In general, the Bootstrap Service interacts with: Sensor manager, API manager, Communication Module, App Deployment Manager, Load Balancer, Monitoring Service, Service Lifecycle Manager.

Load Balancer: Load Balancer interacts with Node V/M manager, API manager to distribute the load.

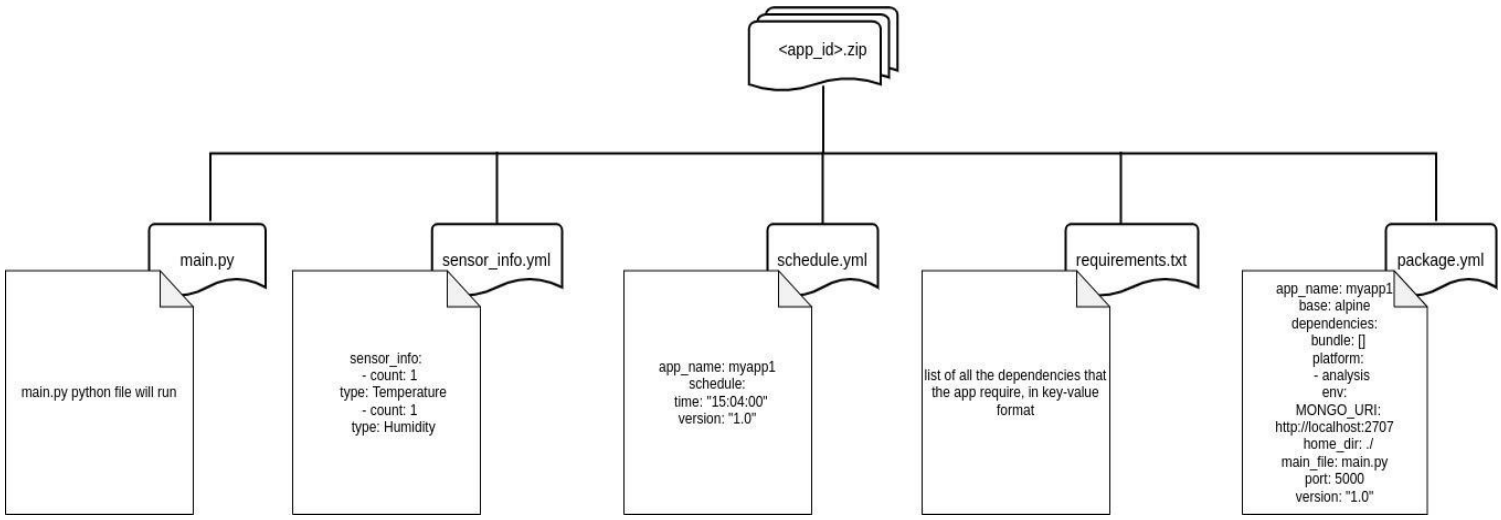
Monitoring Service: It interacts with every other module to ensure no module has stopped working. In general, the Monitoring Service interacts with: Sensor manager, API manager, Communication Module, App Deployment Manager, Bootstrap Service, Load Balancer, Monitoring Service, Service Lifecycle Manager, Node V/M Manager.

Node V/M Manager: Node V/M Manager interacts with Node V/M manager, API manager todistribute the load.

Deployment Manager: Deployment Manager interacts with Node V/M manager to facilitateits service to that module.

h. App Zip Info:

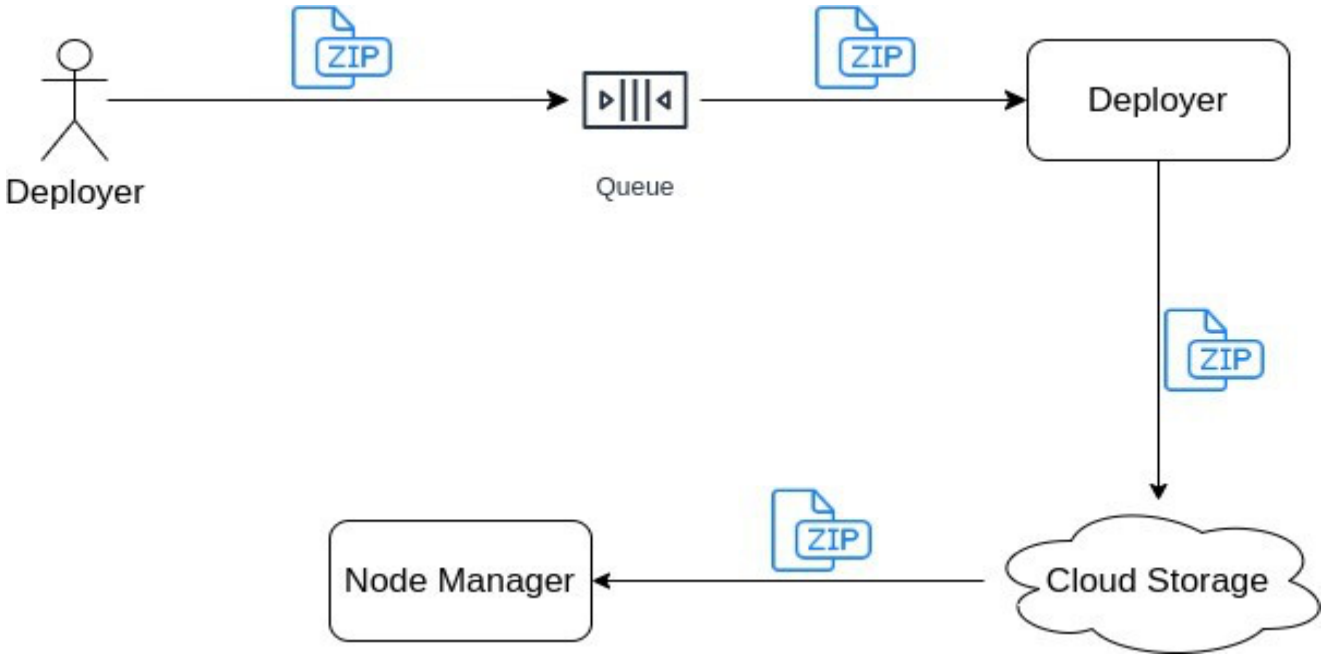
A basic example of the data that is contained inside the <app_id>.zip folder that the user will upload to deploy their application.



The user (App Developer) when wants to deploy their application on our platform they will have to upload a zip file containing the following files:

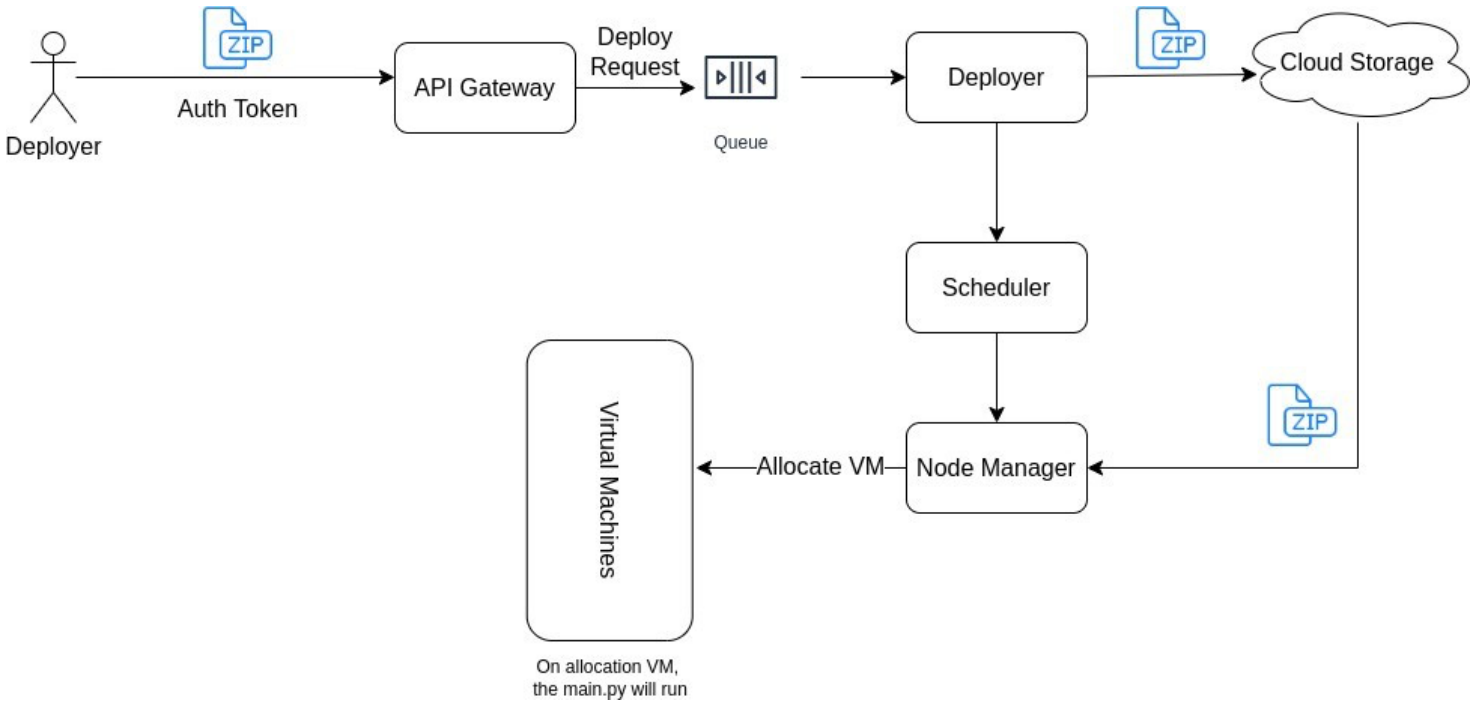
- **main.py** : It is the python script that will execute first on our platform which defines the application.
- **sensor_info.yml** : This is another mandatory file that the user has to upload in the zip folder. This file will contain the sensor detail that the application will use and the count which defines the number of instances of that particular sensor the app requires.
- **schedule.yml** : This file contains the scheduling information for the application. It will contain the app_name with the scheduled timing when the application is supposed to be deployed on our platform.
- **requirements.txt** : This .txt file contains the information regarding the list of all dependencies that the application will require.
- **package.yml** : This file contains the configuration information for the application. It contains information like application name, docker image information, dependencies that the application requires to run, database storage uri (where the application can store its data), port information etc.

Info where the <app_id>.zip goes:



- The zip folder when uploaded, it first goes to the queue from where the deployer will get the app files.
- The deployer then uploads the file to the cloud storage for future access.
- The zip is then accessed by the Node Manager while assigning VM for the application.

i. Sample App Deployment :



Steps involved in how a sample app will be deployed on our platform:

- First the user will be authorized via the token generated from their user id and password.
- If the user is authorized to deploy an application on our platform, then they can upload anapp.zip file which contains the mandatory files that our platform requires to build the application.
- The zip can be uploaded using deploy API on our platform.
- The deployer now must look for the uploaded files in the queue which it will upload on the cloud storage (here Azure) and then forwards the app information to the scheduler toschedule the app's deployment.
- The scheduler will check on the deployment time for the application which is provided inthe schedule.yml file in the app.zip folder.
- Now when the timer reaches the scheduled time for an application, the deployment request will be sent to the Node Manager which is responsible for providing VM for theapplication to run.
- After an application is allocated a VM on our platform, it will deploy the application byrunning the main.py script and creating the endpoint for the sample application.