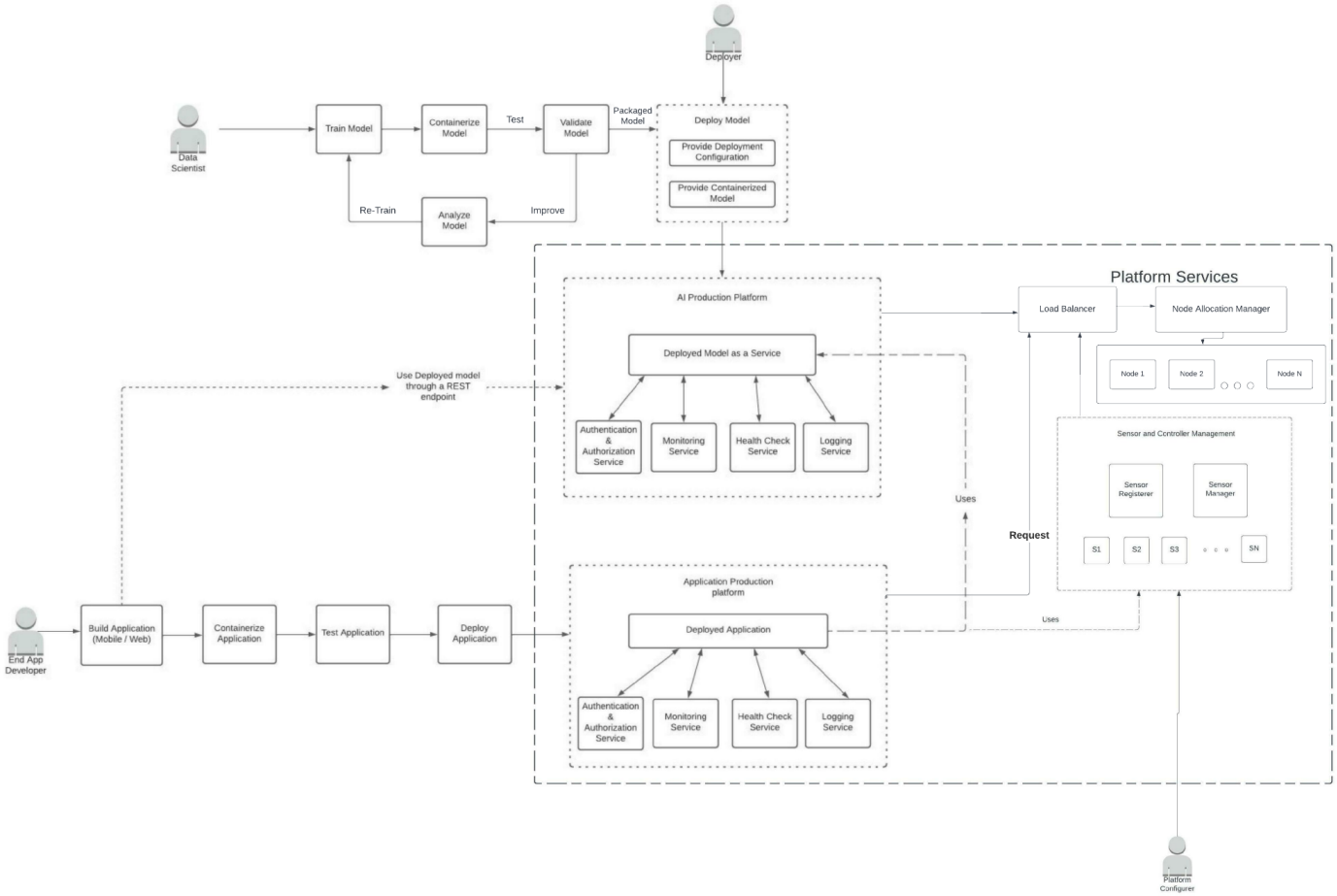
the platform and then for each type they can register instances for those sensors and controllers. After this the end application developer can deploy his/her application using the deployment service of the platform and bind that service with respective sensor/controller types using data uploaded in their config file. At the end of this process, the end user can view all the deployed applications and the available sensor/controller instances based on the bound type and then choose the sensor instances according to his/her preferences. The sensors/ controllers would be managed by the Sensor- Controller service and the deployment would be managed by the deployment service and the node manager service.



2. Use Cases/Requirements (Very detailed documentation of requirements. For the group documentation it will be about overall platform while in the Team documentation it will be more focused about specific components)

- Platform Initializer
    - Initialize a number of Virtual machines / containers on the Host as configured by the configurator.
    - Sets up the environment on VMs for module deployment.
    - Deploys and starts all the services.

- Request Manager
    - Providing GUI to users to interact with platform services, enabling:
    - App developer Uploading Application
    - Data Scientist Host AI models
    - Platform Configurator Manage Sensors
    - End User to consume different services provided by hosted applications.
    - Managing users
    - Making appropriate calls to end points

- Deployment manager
    - Provides UI to upload pickle, config, contract files.
    - Generates docker file and wrapper file
    - Deploys docker and wrapper file on a VM(Azure).

- Node manager
    - Responsible for initiating node instances
    - Responsible for init file generation

- Server Lifecycle manager
    - Manages deployment servers and their status
    - Responsible for fault tolerance of nodes
    - Responsible for initiating nodes with necessary packages and modules

- Load Balancer
    - Responsible for finding out the best node to deploy the application.
    - Looks for relevant stats and returns chosen node's ip address for deployment.

- Authentication & Authorization Manager
    - Creating Database of each of the 4 user types
    - Providing end points

- Sensor Manager
    - This module is responsible for handling the sensors, starting from registering it for the very first time to fetching and preprocessing data and sending it to the application.
    - Module also deals with the registration and configuration of the controllers.

- Scheduler
    - Responsible for handling requests
    - Schedules pending jobs from the queue and sends them to the deployment module.

- Monitoring Manager
    - Responsible for continuously monitoring the status of all the other modules.
    - Logs are taken care of by the logging service. It checks if the log file content has been updated or not.

3. Test cases (will clarify what your module will do. Build on your operations and use cases listed)

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

- List the cases- simulate what the other modules/users can do with your module
    - **Data scientists** can upload pickle, config and contract files.
    - **App developers** can upload source files, config and contract files.
    - **Platform configurer** registers sensor types into the platform and then he/she can register sensor/controller instances corresponding to those types, alongwith information about those instances like location.
    - **End user** will be able to view all the applications that are deployed on the platform and then he/she can select an application which they want to use and also select the sensor/controller instances based on their locations which they want to bind with that application.

- At least 10 test scenarios. Give name, and a brief 3-5 line description. Inputs. Outputs. Invocation mechanism. Expected behavior. Other external considerations (say, when to start a container or how many to start and such- as needed by the test)
- This should be based on the use cases listed in the Team's requirement

3.2 Overall project test cases (relevant to the module)

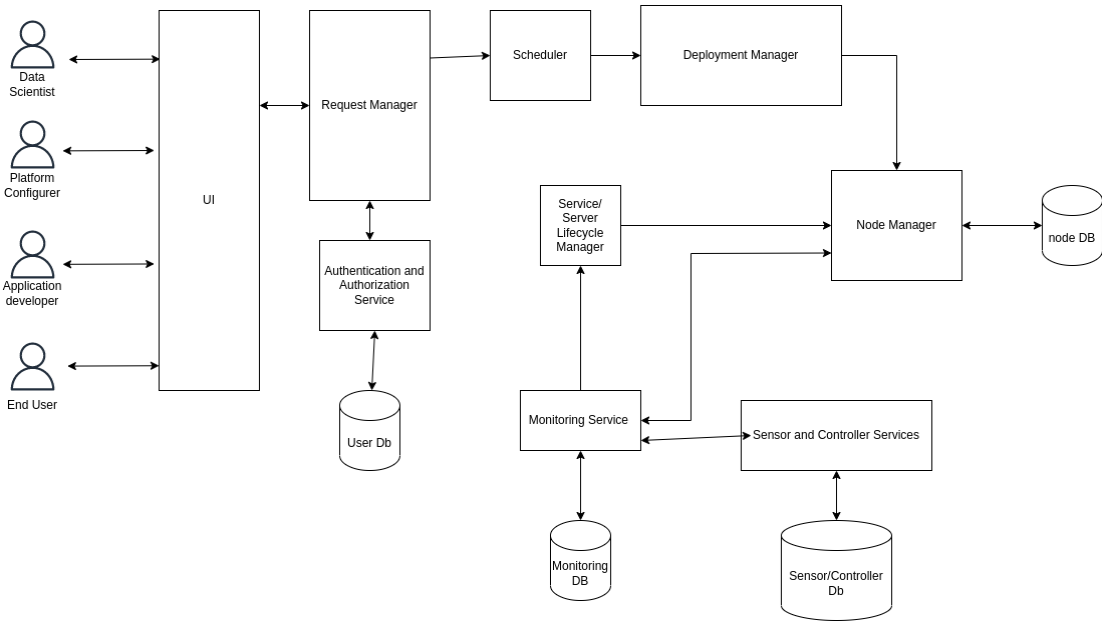
- Our team began by creating and training four models to be deployed on the platform. We then tested these models using pickle files on a Flask server, creating different routes for each model.
- For each model, we developed contracts for data pre- and post-processing. Config files were also created in collaboration with the deployment team based on the dependencies encountered throughout the project.We also worked with the deployment team to wrap the model (pickle file) and enable it to function as a service. End user was given an interface (command line for testing) to use the listed model

of their choice from a list of four. End applications were developed for each model, with sufficient user interactions providing prompts for data input.

4. Solution design considerations

(Group level- to be discussed by all four teams, and captured in all four team-design documents).

4.1 Design big picture



4.2 Environment to be used

- Operating System : Linux, Ubuntu 20.4 LTS

- Nature of Interaction: The platform services can be accessed by the users of the 4 different roles though a web GUI in a secured way.

  Inside the platform the module microservices would interact with each other using a messaging system with technologies like Apache Kafka or using REST API endpoints.

  The sensor instances would send data to the platform through Apache Kafka.

- Environment for deployment: The Nodes for deployment would be created using docker containers inside Azure VMs.

- Unified Database used by the Platform: We shall use Azure SQL database or MongoDb Database for creating the registries and data entry tables for the platform.

4.3 Technologies to be used

- Python
  - Most of the implementation would be done based on python as it provides the language constructs that satisfies most of our requirements for the integration and implementation of the modules and also it provides efficient ways to integrate with other technologies like Azure services, database technologies etc. that we would be using in our project.
  - We would use Python as the base technology for the implementation.

- Flask
  - Flask provides compatible ways to integrate with python for creating API endpoints.
  - We would use Flask to create all our API and UI endpoints.

- Apache Kafka
  - Kafka would be used as the messaging queue for interaction between the modules.

- Docker
  - Docker provides ways to package and containerize applications using the commands. As it is an efficient way to interact with the remote virtual machines using commands so docker is our choice of technology for containerizing the applications and models and host them in the remote VMs.

- Bash
  - We plan to use bash scripting for initiating the system commands in the platform modules.

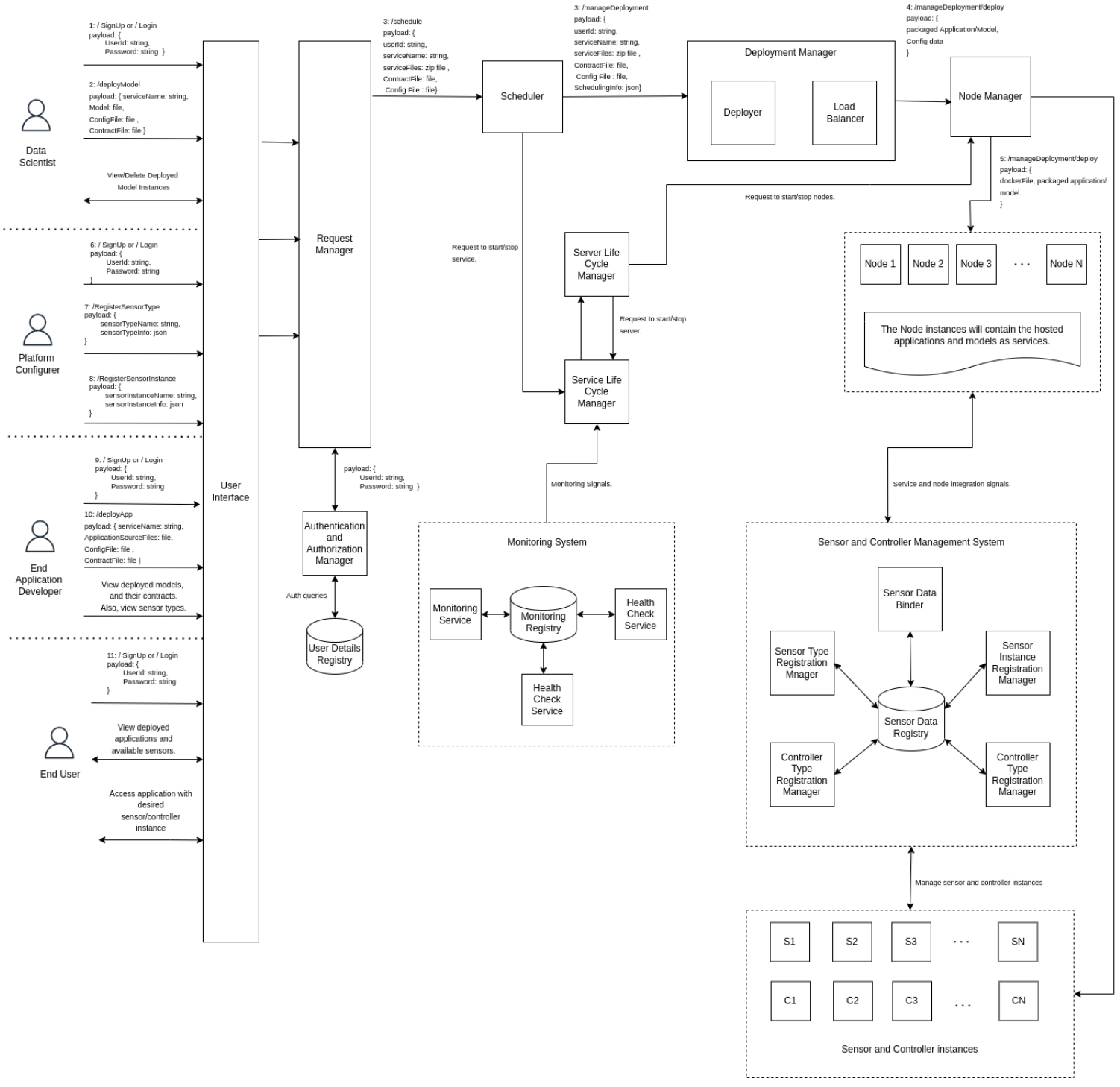
- HTML, CSS, JS, Bootstrap
  - For our unified UI, we will be using frontend technologies like HTML, CSS, JS and Bootstrap to create the dashboards and other UI interfaces.

- Azure Services
  - *Azure VMs* : We plan to use Azure VM resources for our hosted remote machine which would act as the servers and nodes for the service deployment.

- Database Technologies

  - For implementing the unified database of our platform, we plan on using either Azure SQL or MongoDB depending on our needs during implementation.

4.4 Overall system flow & interactions



4.5 Interaction with Sensors

Intended Use & Scope

We begin with the registration of all the sensors, providing unique ID to each, that could be used for data input to any of the AI Model deployed onto the platform.

The sensor along with its id is then available on the platform to be made use of. The user gets to choose the sensor for data collection in his model.

Sensors are configured accordingly post registration, and then the data is being fetched from sensors using Kafka. Also, data collected from the application is being sent to the controller specified by the user.

To make things smooth for the application developer, this module hides the complexities of sensor and controller technologies providing a single format using abstraction.

This module intends to provide scalability to our project wherein we can add the sensors and/or controllers to the list of our existing ones, as and when required.

Assumptions & Dependencies

Before the setup of the platform, all the sensors are available to be fetched in case of their registration.

Whenever a new application executes, the sensors required by it are up and running in normal fashion.

Configuration files sent by the user for any sensor must be in the fixed format as required.

Functional Requirements

Registration:

Sensor Registration – On initialization of the platform, the sensors required by the application need to be registered. Also, in the case of new addition of sensors we need to register the newly added onto our database.

Controller Registration – All the controllers that ought to collect data from the deployed application, must be registered beforehand in the same fashion as the sensors.

Configuration:

Sensor Configuration – All the sensors registered need to be configured in a fashion that they could take the input and provide the data in some specific format for all the application.

Controller Configuration – In a similar way, we need to configure the controller types and all their instances.

Sensor Identification:

Upon registration of the sensor onto the platform, all its details are saved in the repository. We provide all the sensors with some unique id based on its properties, which are different for each.

Application Manager then uses this unique id to identify the sensor based on the parameters/ properties provided by the user, after the sensor manager receives a data binding request.

Data Binding:

After the required sensor has been identified, data is to be sent in the specified – fixed format.

Certain node will be servicing some instance of the application which made the request to the sensor manager with a sensor id.

Using the ids, the required sensor(s) for receiving/sending data are made available. The data is then sent in the specified format

4.6 Scheduler

The Scheduler module is responsible to run/send jobs to deployment manager periodically at predetermined intervals.The scheduler will receive the scheduling information in the form of a HTTP request giving the information related to the user from the request manager.

4.7 Load Balancing:

As the application request is received by the deployer. It gets the free nodes from the platform for deployment. A load balancing algorithm is then applied, and the active workload is calculated for each of the available nodes by noting the CPU usage and the RAM usage of each. In case of high workload, the load balancer requests for more nodes from the manager.

4.8 Interactions between modules: :

Each module would be hosted as a microservice on a node and the interaction between these modules would be done using REST endpoints and messaging system implemented using a technology like Apache Kafka.

4.9 Platform bootstrap

Initialize a number of Virtual machines / containers on the Host as configured by the configurator based on

the configuration files which will contain information about which service will run on which machine.

4.10 Request Manager handling authentication and authorization

Handles all the requests generated from outside of the platform and delegating it to the adequate microservice.

It also performs authentication / authorization through Authentication & authorization manager by making an appropriate call to an end point

5. Application Model and User's view of system

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

The platform will provide an environment for the users to connect to / use location sensitive sensors, controllers and also AI models from their own applications. It will provide an interface in the form of a dashboard which will expose the utilities in a user-friendly way. Data scientists can deploy their AI models in this platform in a secure manner. They can also configure the deployment (for eg. they can choose the number of instances of the app to be deployed, the environment of the server where it would be deployed etc.). After the models are deployed, the end app developers can use these hosted models in their applications and after the application is created and built, they can use the platform to deploy it in a secured and configured manner as well. The end app developer can also use the registered sensors and controllers on the platform in their application to fulfill their application's purpose.

The deployed instances of the models and applications on the platform would also be constantly monitored and tracked using monitoring, health check and logging services. The usage of the server machines containing the deployed instances would be made secure using an authentication and authorization service.

- Structure of the files

Data Scientist : For deployment, the data scientist can upload a pickle file of their model, a contract file in python (Contract.py) which contains preprocessing and post processing functions and a config file containing deployment information in json format.

Application developer: They can upload a zip file containing their source files, config file containing information about the deployment and sensor types and contract files containing the information about how to use the application.

- Application artifacts

Design Artifacts:

- The design documents containing the design of the platform's implementation.

Development Artifacts:

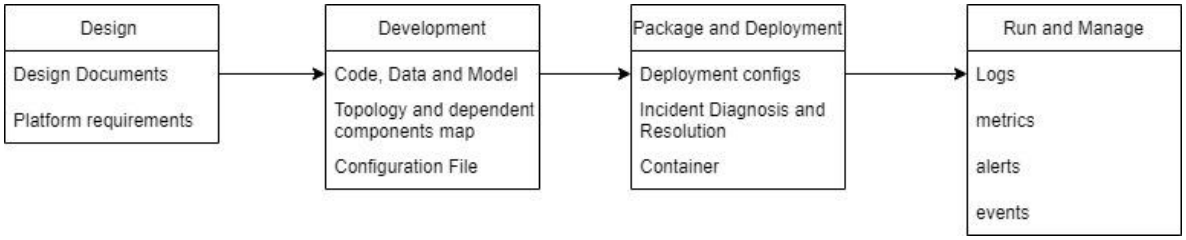
- The code files, data files, configuration files that are part of the implementation.

Deployment Artifacts:

- Deployment configs, DockerFile, Container of the packaged application or model wrapped as a service.

Runtime Artifacts:

- Logs, monitoring artifacts, events and alerts.



User admin interactions

- How will the user use this system

The different users can use this platform by signing up to be able to login again using their credentials. Depending on the role the particular user has selected, the user will be given a list of options.

For application developers - List of models uploaded by the data scientist would be visible along with the option of uploading his applications.

For data scientist - List of models uploaded by him would be visible along with the option of uploading his models.

- What are How to deploy

Based on the config files and the contract files, the user will deploy the models and applications on the platform, with the help of the deployment manager. A dashboard would ask the user to upload the models and applications.

6. Key Data structures

Persisted data (registry, rules, et al)

User database containing tables - End user , Application developer, Data scientist , Platform configurer. This is part of the request manager module for validating and authorizing the users.

Models database containing tables - Models, Applications. This is for storing the files uploaded by the application developer and data scientist.

Config file - For the platform configurer which includes the information of which services shouldn't run on which machines while platform bootstrapping.

7. Interactions & Interfaces

- *APIs*

(Should include more detail than what was in Team's requirements)

- User interactions
- API'S handled by the request manage

To manage the user login and sign up:-

- { /signup_DS, /signup_AD , /login_DS , /login_AD }

To validate the user:-

- { /authen_AD , /authen_DS , /adduser_AD, /adduse_DS }

To get the model and app info and upload new models and apps

- { /getapps , /getmodels , /uploadDS , /uploadAD , /addapp, /addmodel }

The services handled by these API endpoints would be enabled by interaction with the database.

- *Module to Module interactions*

- The request manager would call the exposed API to the scheduler for passing the information regarding user requests.

- The deployment manager will interact with the scheduler to get details like userid, password and scheduler information.

- The deployment manager will interact with the monitoring service to get status of the tasks of an application and display the results.

- The running application will interact with the sensor manager to get data from the sensor.

8. The modules; that the four team will work on

- Scheduler, Monitoring System (Logging, Monitoring, health check) : **Team 1**
- Deployment Manager, Node Manager, Load Balancer, Server Lifecycle Manager : **Team 2**
- Sensor and controller management : **Team 3**
- Request Manager, UI, Authentication and Authorization Service, Platform Initializer: **Team 4**

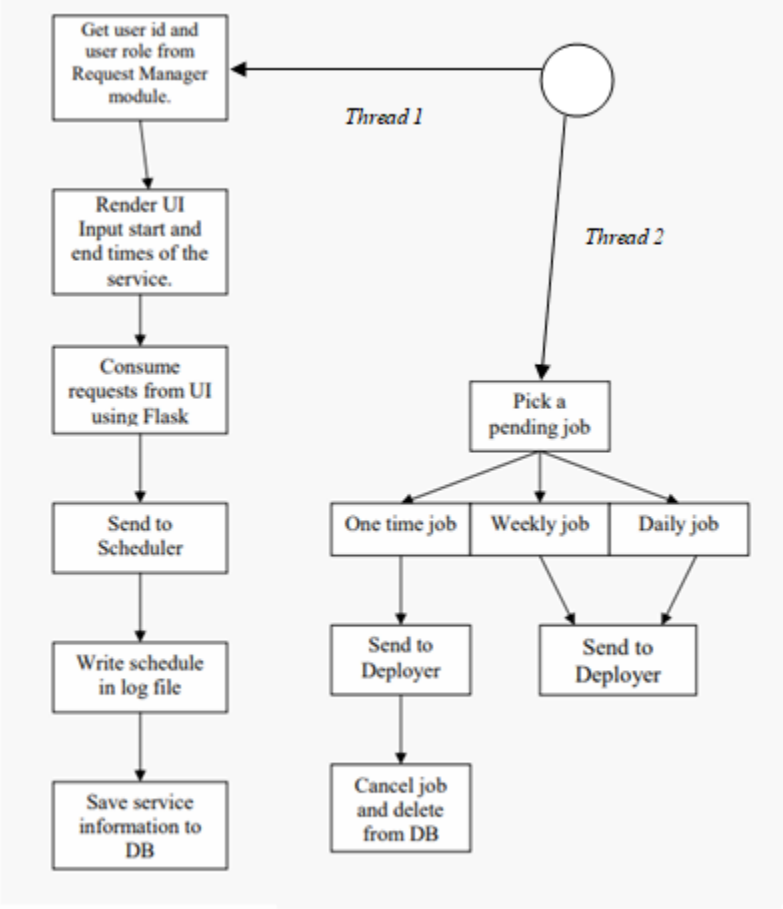
Low Level Design (for each of the modules)

Scheduler

Lifecycle of the module

1. Gets authentication details from the Request Manager module.
2. Get user ID, application ID, start time, duration, days, and a zip folder containing all source files of the service from the front end.
3. Data input by the user is first validated and then sent to the flask backend.
4. One thread receives scheduling requests from the front end and another runs pending requests.
5. Job is sent to the deployer on time according to its schedule.

Block Diagram



Diag1. Overall Flow

Description

The frontend was written in HTML, CSS, and Javascript. The data input by the user is first validated and then sent to the flask backend. We have used MongoDB to store the scheduling requests and Flask to establish a connection with the front end. We have created two threads; one for receiving scheduling requests from the front end and another for running pending requests. Scheduling requests can be of three types: One-Time jobs, daily jobs, and weekly jobs. In the case of one-time jobs, they would be scheduled for the earliest day among the chosen days. We consume the following data from the frontend - user ID, application ID, start time, duration, days, and a zip folder containing all source files of the service. We also have a logs file to log whenever a new job is scheduled along with the period, days, and time, and whenever a job is sent to the deployer.

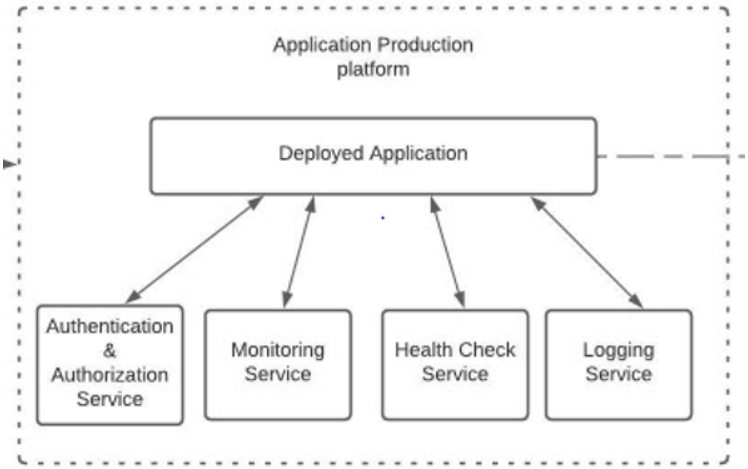
Monitoring System

The monitoring module will be connected to most of the other modules of the platform that include- Scheduler, Sensor, load balancer, deployer, node manager. It monitors them periodically using heartbeat messages and reports and takes appropriate action in the case when there is an error encountered.

Submodules

- Monitoring Service
- Health Check Service
- Logging Service

Block diagram



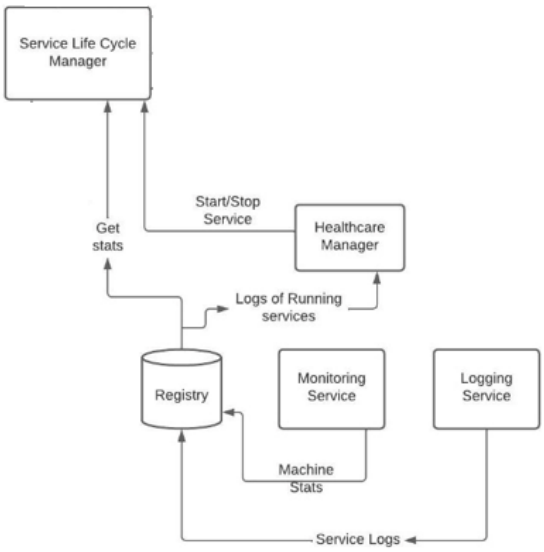
Interactions among sub-modules

The monitoring system is responsible for continuously monitoring the status of all the other modules. And the Health Check service checks the health of each module by periodically sending in the requests, attempting connection. The logs are taken care of by the logging service. It checks if the log file content has been updated or not.

- **Monitoring Service:** The monitoring system is responsible for continuously monitoring the status of all the other modules. It is also responsible for verifying if an instance of the module itself is working or not. It detects and reports any abnormal behavior of any module and the node on which it is running.

- **Health Check service:** The Health Check Service is the load balancer. It checks the health of each module by periodically sending in the requests, attempting connection, or sending a ping to them and getting back the status. It distributes the workload among all the running instances of the service.

- **Logging Service:** It is a log monitoring service that checks if the log file content has been updated or not. It can take the log file path and the log message as the input and check whether the service has written the specified log message onto the log file or not.



-

Request Manager

1. Lifecycle of the module

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

Providing GUI to users to interact with platform services, enabling:

App developer Uploading Application

- Providing contract
- Proving forms for Uploading application
- Taking Scheduling details.

Data Scientist Host AI models

- Providing contract
- Proving forms for Uploading model
- Taking Scheduling details.

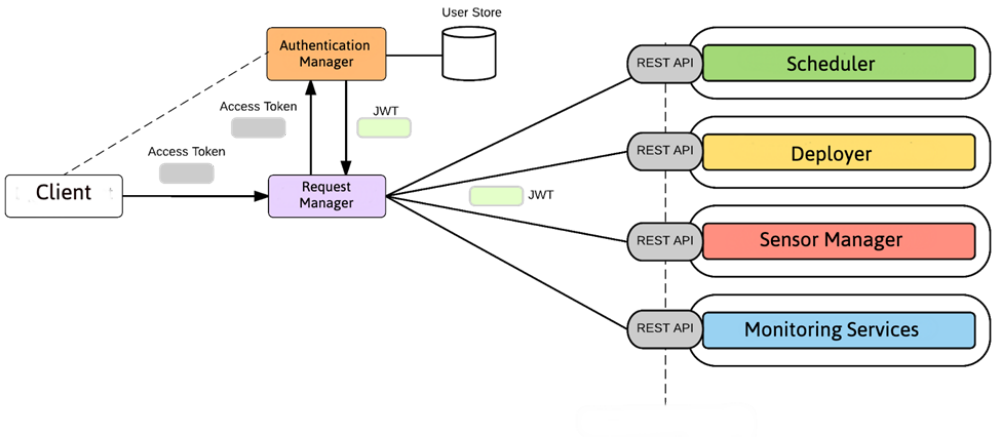
Platform Configurator Manage Sensors

- UI to add new sensor types
- UI to Manage logs/view status.

End User to consume different services provided by hosted applications.

- UI to let end - user select application and sensor instances
- Letting end - user access application end-points

Block Diagram



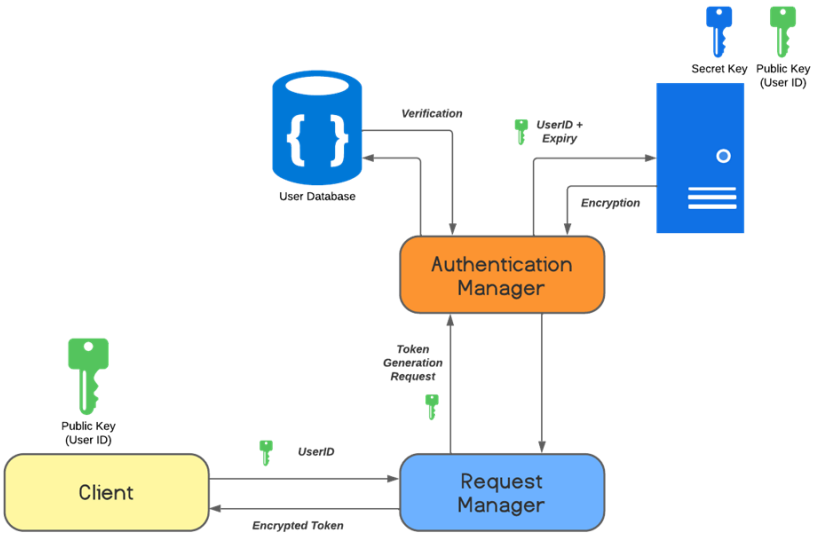
Authentication & Authorization Manager

1. Lifecycle of the module

- This module is responsible for authenticating and authorizing the requests. For that it, maintains the user databases, generate Tokens for logged in users and verifies Token on requests.

Block Diagram

- Token Generation



Platform Initializer

1. Lifecycle of the module

- This module is responsible for initializing all the proposed microservices up and running on a set of Host machines. It installs the prerequisites on the Host machines and sets the environment for modules deployment, and configures all the different modules.

Requirements:

- Requires to be configured according to the type of Host, number and type of services .
- Initialize a number of Virtual machines / containers on the Host as configured by the configurator.
- Sets up the environment on VMs for module deployment.
- Deploys and starts all the services.

Deployment Manager

1. Lifecycle of the module

- How will the module/its components be started, monitored and stopped (if applicable).

  The deployment manager will interact with the scheduler to get details like userid, password and scheduler information. It will interact with the monitoring service to get status of the tasks of an application and display the results. The running application will interact with the sensor manager to get data from the sensor.

- How will any user's application components need to be made available/deployed/setup on this module.

  - **Data scientists** can upload pickle, config and contract files.
  - **App developers** can upload source files, config and contract files.
  - **Platform configurer** registers sensor types into the platform and then he/she can register sensor/controller instances corresponding to those types, alongwith information about those instances like location.
  - **End user** will be able to view all the applications that are deployed on the platform and then he/she can

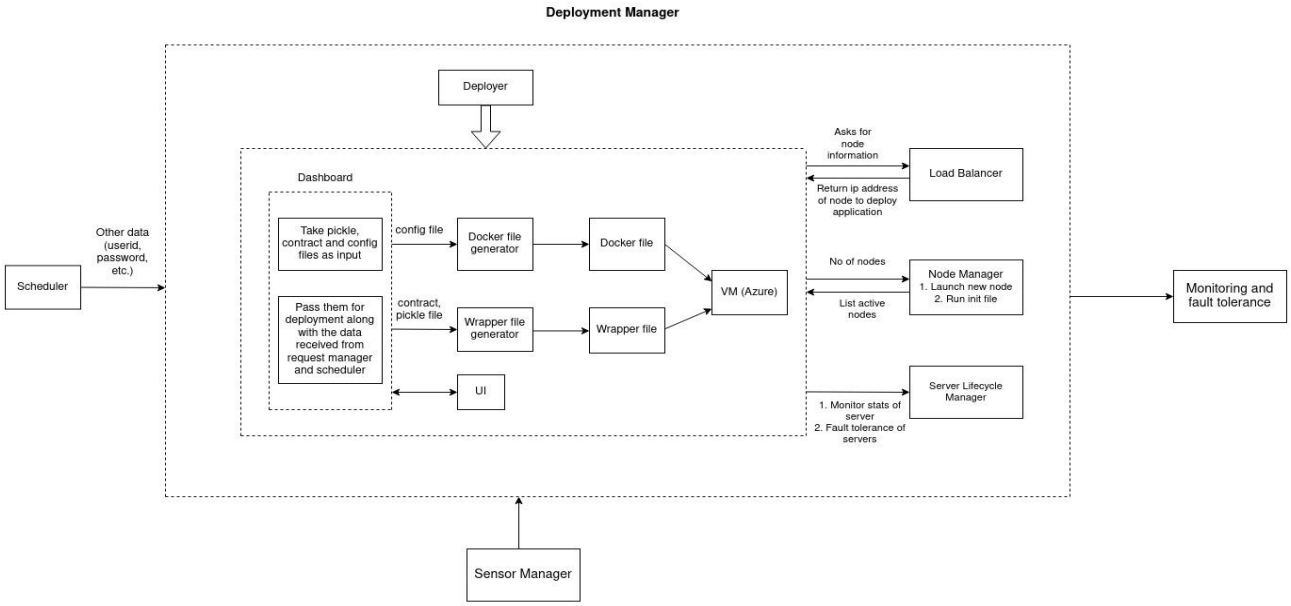
select an application which they want to use and also select the sensor/controller instances based on their locations which they want to bind with that application.

- Any other considerations (based on the module's functionality)

2. List the sub modules

The submodules are deployer, load balancer, server lifecycle manager and load balancer.

A block diagram:



Describe the interactions among sub-modules:

After receiving data like userid, password and scheduler info, the deployer takes input of pickle, config, contract and source files via dashboard of UI. Then it generates docker and wrapper files and deploys the wrapper file using the docker file onto the VM(Azure). The deployer asks the load balancer to choose the best node to deploy the application on. The load balancer returns the chosen node's ip address to the deployer. The Node manager launches new nodes and runs init file on them. These nodes are used for deployment. The Server lifecycle manager monitors stats of servers and makes them fault tolerant.

3. Brief overview of each sub module

- For each sub module:

  #(parts of the whole solution

  #give a meaningful name for each part)

- Describe the functionality of the module (7-10, bullet list)

Deployer -

- The deployer takes pickle, config, contract files as input and generates the docker file and wrapper file to be deployed on the VM (Azure).
- The UI will be used for uploading the pickle, config, contract files in case of data scientist and source files, config and contract files, in case of application developer.
- Other than serving the deployment requests, it is responsible for monitoring the applications and managing their life cycles.

Load Balancer -

- This module is responsible for finding out the best node to deploy the application.
- It looks for relevant stats like cpu usage, and finally returns the chosen node's ip address to the deployer for deployment of applications.

Server Lifecycle Manager -

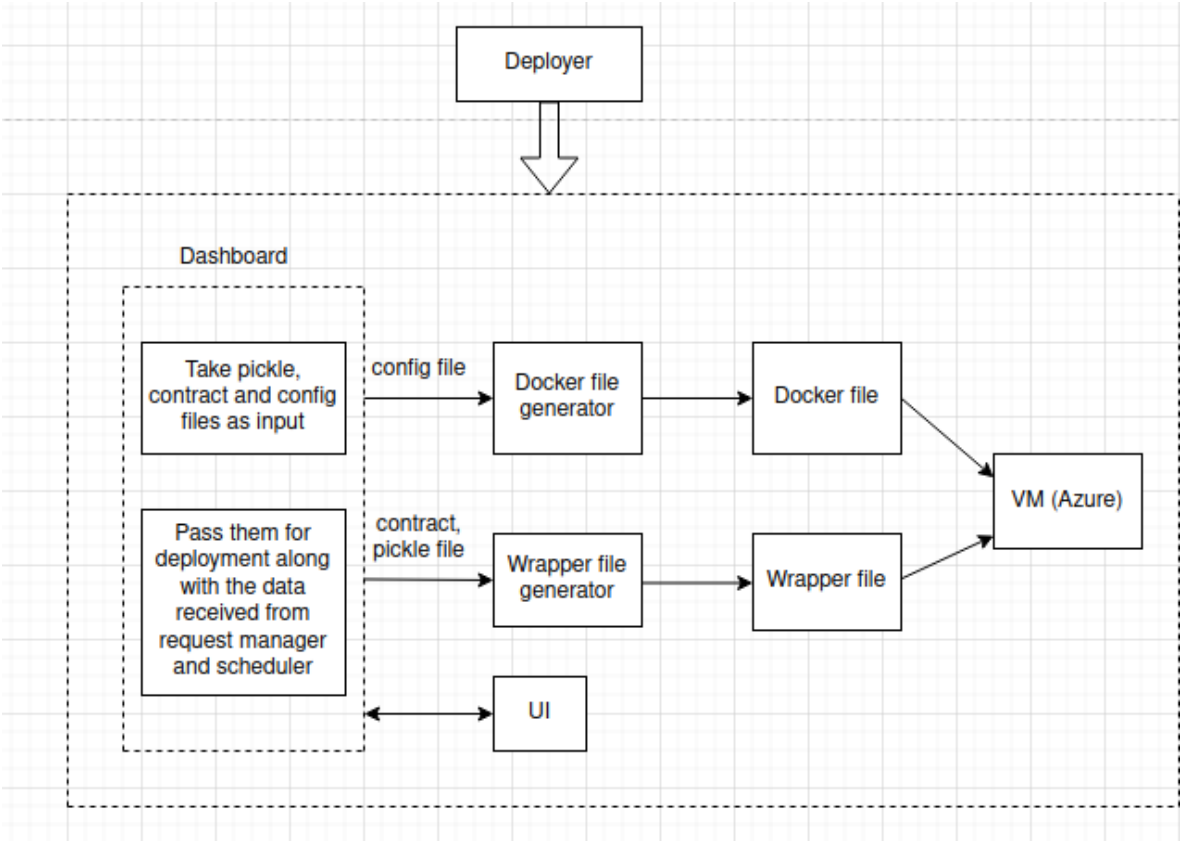
- This module manages the deployment node servers and their status.
- It is responsible for the fault tolerance of the nodes and initiating the nodes with necessary packages and modules.

Node Manager -

- It is responsible for initiating node instances as requested by the deployment manager.
- It is also responsible for init file generation which helps in communication with the scheduler, deployer and sensor manager.

- A brief block diagram of the internals (constituents)

Deployer



- List the interactions (with which other module, external entities, file or network, comm. Framework et al)

The deployment manager will interact with the scheduler to get details like userid, password and scheduler information. It will interact with the monitoring service to get status of the tasks of an application and display the results. The running application will interact with the sensor manager to get data from the sensor.

- APIs & Classes
  - <u>APIs:</u>

    /manageDeployment - This API endpoint can be used to deploy an application or model.

    /manageDeployment/deploy - This API endpoint will be used internally by other modules to access the node manager for managing the node instances.

  - <u>Classes:</u>

    DockerFileGenerator - This class is responsible for generating the docker file from the config file for a service.

    WrapperClassGenerator - This class is responsible for creating an api endpoint wrapper

over a model. It reads a model from a pickle file and then creates an endpoint over it.

DeploymentManager - This class uses the node manager to deploy the application or model wrapper class.

NodeManager - This class would setup the node instance and then upload the generated DockerFile and source files into the node (remote VM) after which it would create a docker image and run a docker container in the VM to host the service remotely.

4. Interactions between sub modules

- Interactions among the sub-modules

  After receiving data like userid, password and scheduler info, the deployer takes input of pickle, config, contract and source files via dashboard of UI. Then it generates docker and wrapper files and deploys the wrapper file using the docker file onto the VM(Azure). The deployer asks the load balancer to choose the best node to deploy the application on. The load balancer returns the chosen node's ip address to the deployer. The Node manager launches new nodes and runs init file on them. These nodes are used for deployment. The Server lifecycle manager monitors stats of servers and makes them fault tolerant.

5. Interactions between other modules

The deployment manager will interact with the scheduler to get details like userid, password and scheduler information. It will interact with the monitoring service to get status of the tasks of an application and display the results. The running application will interact with the sensor manager to get data from the sensor.

Sensor Management

1. Lifecycle of the module

- This module is responsible for handling the sensors, starting from registering it for the very first time to fetching and preprocessing data and sending it to the application. Module also deals with the registration and configuration of the controllers.

Interactions between other modules

A. **Application Manager·** It provides the unique IDs of the sensor(s) which it requires to interact with for data collection. And the sensor manager provides the information for the specified sensor(s) which is then stored at application's

end.

B. **Node Manager & Nodes** - Node manager assigns a node for each application instance. Node might interact with sensor manager to send and/or receive the data.

C. **Monitoring System**- The Health Check Service of the Monitoring System keeps a check on Sensor and Controller Manager, which is being done by using a heart-beat signal.

D. **Fault Tolerance** - Fault tolerance module interact with Sensor manager module and in case of any discrepancy or fault, the instance of fault tolerance module related to Sensor or Controller manager gets triggered.

Block Diagram