Team 3
Module Requirements

PREPARED FOR

IAS PROJECT - GROUP 4

IIIT-Hyderabad

PREPARED BY

TEAM 3

Dhruv Sachdev (2020201060)

Nisarg Sheth (2020201049)

Pujan Ghelani (2020201083)

Sub Modules implemented by our Team

1. Sensor Binder
2. Scheduler
3. Deployer
4. Load Balancer
5. Application Level Monitoring
6. Service Lifecycle

Application Scheduling and Deployment

Introduction

- **Sensor Binder:** This module will identify required sensor in the application using deployConfig.json file. It will also bind those sensors with the application.

- **Scheduler**: Scheduler takes as an input the time and the intervals at which certain programs are to be run from a configuration file. Job of the scheduler is to start and stop jobs/programs according to the requests received. It sends the requests of starting and stopping jobs to the deployer as it is the deployer module which actually deploys programs on computing instances.

- **Deployment Manager**: Deployment Manager is responsible for deployment of programs on computing instances. It receives these requests from the scheduler. The deployment manager has to recognize the sensors which are associated with the application. It also needs to determine the packages and the environment that needs to be set up for the application.

- **Load balancer:** There will be a system called load-balancer which would decide which computing instances should it deploy the programs directed by the scheduler.

- **Application level Monitoring :** In this module, we are monitoring application by checking application status codes. This module also fetches the statuses all the running containers and sends it to the dashboard for monitoring purposes. The user can also request the logs of a particular container via this module.

Functional Overview and Interaction Between Modules

1. **Sensor Binder:**
    - Sensor binder receives a request from platform manager in the form of deployConfig.json file.

    - We use the filters given in the config file which are used to decide which sensors should we bind to the application.

    - The module looks into the database where the details of sensors are stored to find the ones matching to our criteria.

- It throws an error if it does not find the sensors to bind required by the application.

- Else, it generates an instance id and forwards the deploy config file to scheduler which handles from there on.

2. **Scheduler:**

    - Scheduler receives the application id and the location of the configuration file in the application repository from the platform manager.

    - Scheduler will fetch the configuration file and it would have to verify if the configuration in the file are in the correct format and according to the constraints of the platform(if any).

    - Scheduler will have to store the scheduling details in a list and set some form of cron job which would grab the necessary files and pass it to the deployer at the appropriate time/interval.

    - It will also send a request to stop the application to the deployment manager either at the scheduled end time or an interrupt received from the platform manager.

    - It will do so by maintaining two priority queues, one queue will be used to handle the start time of the scheduled applications and the other queue will be used for handling the stop times.

3. **Deployer:**

    - Deployment Manager receives the application id and the location of the configuration files in the application repository from the scheduler.

    - It will have to locate and identify the sensors which are associated with the application that is to be deployed. It would have to communicate with sensor manager to get the information about the sensors.

    - Deployment manager will have a sub system called a load balancer which will receive the number of nodes required for the application from the deployer. It will try to find the required number of nodes with least load and will create nodes if there aren't enough nodes available. It would return the details of the nodes bound with the application to the deployer.

4. **Load Balancer:**

- Load Balancer keeps receiving information about the load(RAM usage, CPU Usage) on computing instances from the monitoring module so that it can make decisions to balance the load.

- Deployer will setup the environment and packages on the nodes provided by the load balancer and then start the application program on those nodes.

5. **Application level Monitoring**
    - It will monitor various application statuses and If it is one of the following 137,143,133 status codes, it shows that the application was stopped/exited abruptly by any reason like shutting down of the machine or any external interrupt. In that case, our module will detect such cases and send the restart request and restart the application on the machine suggested by load balancer.

    - Dashboard will ask this module about container logs detail and It will provide required container logs to dashboard.

6. **Server lifecycle:**

    - Each server will have a kafka topic on which it will send its status every 30 seconds.

    - The Server lifecycle will consume from each server's topic and if it does not receive anything for 80 seconds, it assumes the server is down and will restart the server.

    - If any applications were deployed on the server app monitoring module will detect it and after the server is restarted it will re-run all the applications that were running at the time of crash.

    - If any platform services were deployed on the server service lifecycle manager will detect it and after the server is restarted it will re-run all the services that were running at the time of crash.

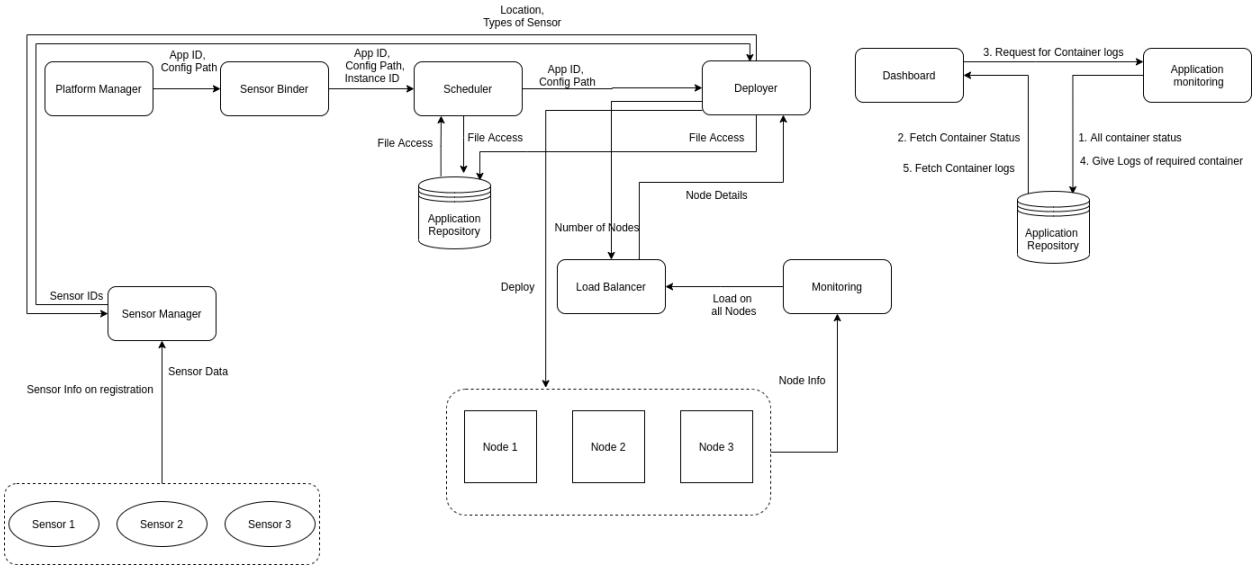
    - Service Lifecycle will fetch a list of all containers running in the platform from MongoDB.

    - Every service of the platform is running in a docker container. It will only consider service containers from a list of containers and will check if the container is running fine using container logs. If it finds out that service has crashed, it will restart a service and update it in MongoDB collection (service_status) with a new container id.

Roles and Responsibilities

1. **Sensor Binder**
    a. Identifies and binds the required sensors with the application.
2. **Scheduler:**
    a. It has to start or stop jobs according to the schedule requests.
    b. It accomplishes these tasks with the help of the deployer.
3. **Deployment Manager:**
    a. Deployer has to deploy or stop the programs on the nodes when requested by scheduler.
        i.
4. **Load Balancer**
    a. Load Balancer has to assign the computing instances for an application by using an load balancing algorithm when requested by the Deployer.
5. **Application level monitoring**
    a. Continuously monitor application status and provide container logs to the dashboard.

Block Diagram

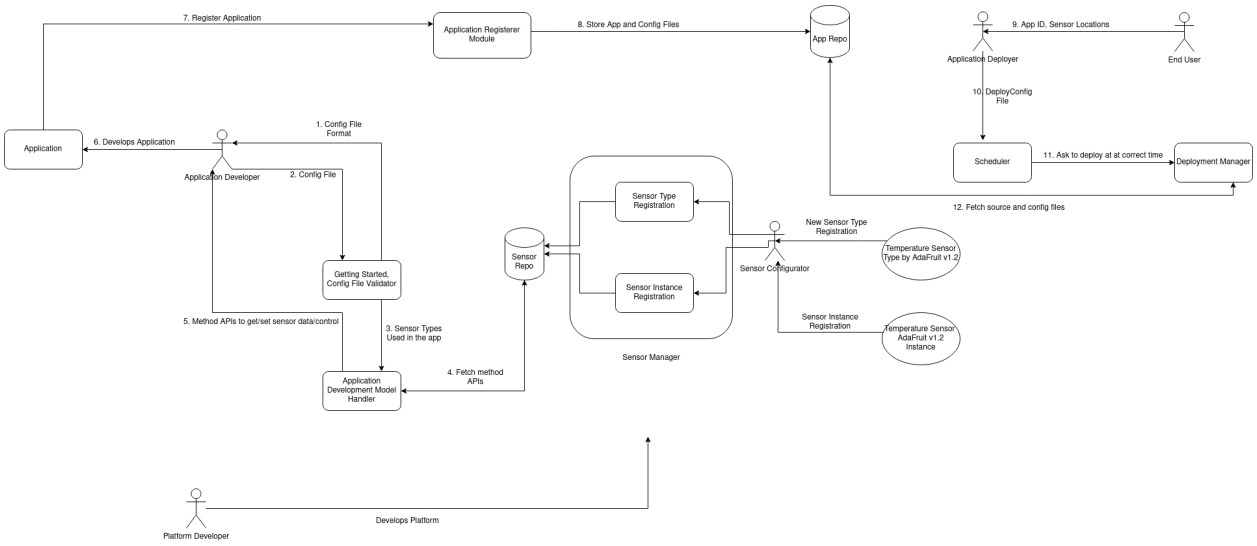


Link for above Diagram:
https://drive.google.com/file/d/1fV1lFgRyLO8SQyL-tTXlv7z_vwWNVbCn/view?usp=sharing

Flow Diagram for Application Development Model

Link to the flow diagram:
https://drive.google.com/file/d/1kTuKxr758avz_JfU4kjWfnfwnOeFOvL6/view?usp=sharing

It might not be visible properly in the document here. So, you can go ahead and open the above link.



Communication Between Actors and Platform and Responsibilities of various components in the Application Development Model

1. **Platform Developer** develops the platform.

2. The **Application Developer** downloads an example config file and the format of the config file that is to be provided by the application developer to the platform.

3. The **Application Developer** provides the config file which contains the type of sensors used, number of sensors used, controls of the sensors, scripts in the application and the input output of those scripts.

4. Once the config file is validated, sensor types used/needed in the application are given to the Application Development Model Handler which would fetch the method APIs to control those sensor types.

5. Application Development Model Handler gives those method APIs back to the Application Developer.
   The scripts are supposed to take command line arguments which would represent the kafka topics assigned for each sensor that is used in the algorithm. So, the **application developer** simply has to read from the designated topic for getting sensor data and it does not have to communicate with the sensors at all.

6. **Application Developer** develops the application and uploads/registers the application afterwards.

7. The source code files and the config file are stored in the Application Repo.

8. New sensor types and sensor instances can be registered and it is done by the **Sensor Configurator** via Sensor Manager.

9. Sensor Manager stores the details of the sensor types and/or instances in the sensor repository.

10. **End User** chooses to avail services of some application available on the platform. End User will have to communicate that with the **Application Deployer** and he/she will make a DeployConfig file on the user's behalf which will contain the location of the sensor instances that the application has to bind to.

11. The application is scheduled and deployed according to the details provided in the DeployConfig file by the scheduler and deployment manager respectively.

Different Users and Their Roles

Application Developer: To develop the application/algorithm according to the application development model given by the **platform developer**. He/She also has to provide a config file which would contain details about sensor types, number of sensors, controller types, etc. used in the application.

Platform Developer: The one who designs the **application development model** and develops various components of the platform which would enable an application developer to register an application and the end user to run these applications on the platform. Various components include sensor registration module, monitoring, deployer, scheduler, load balancer and ensures availability, security etc of the platform. Platform developer also has to give the format of the config file to the application developer that is to be given for application registration.

Application Deployer: This guy actually deploys the application on specific instances of sensors of the type that the application is made for. Gives a **DeployConfig** file or selects the

sensor instances via UI and asks for deployment of the application instance.

Sensor Configurator: Registers the new sensor types when they are not already defined on the platform. When a new sensor type is to be registered, details about that sensor type have to be provided via a config file. Details like the data that it records, data rate, controls of the sensors, method APIs have to be specified in the **SensorTypeConfig** file.

He/She also registers new sensor instances pertaining to specific sensor types. When a new sensor instance has to be registered, its type, location, ip address, port, etc have to be specified via a **SensorInstanceConfig** file.

“End User/Customer”: The one who wants to use the services of some application(s) available on the platform. Example, a patient wants/needs services of the bed in the hospital and various predictions that are made using some algorithms using sensor data from bed and room. **Application Deployer** deploys the application and builds a **DeployConfig** file based on the location of the sensors provided by the end-user.

<u>Sample UseCase:</u>

1. **End to End farm management System App(Primary)**. :To increase the productivity of agricultural and farming processes in order to improve yields and cost-effectiveness with sensor data collection by sensors like soil moisture, temperature sensor,ph sensor,water level sensor.We will use sensor data collection for measurements to make accurate decisions in future. It can be used as a reference for members of the agricultural industry to improve and develop the use of IoT to enhance agricultural production efficiencies.

   We have used two algorithms.
   watercontent.py, airconditions.py

   The sensor types -
   air-condition-sensor - measuring the temperature of the soil
   soil-moisture-sensor - measuring the humidity of the sensor and turning off or on the sprinkler based on the threshold.

Dashboard Screenshots