IoT Platform For Monitoring End Devices and Controlling Them Through Applications

Group 1, Team 4

May 8, 2021

Neerja Gangwar(2019201020), Jyoti Gambhir(2019201032), Apurvi Mansinghka(2019201093)

Mentor and Professor : Prof. Ramesh Loganathan

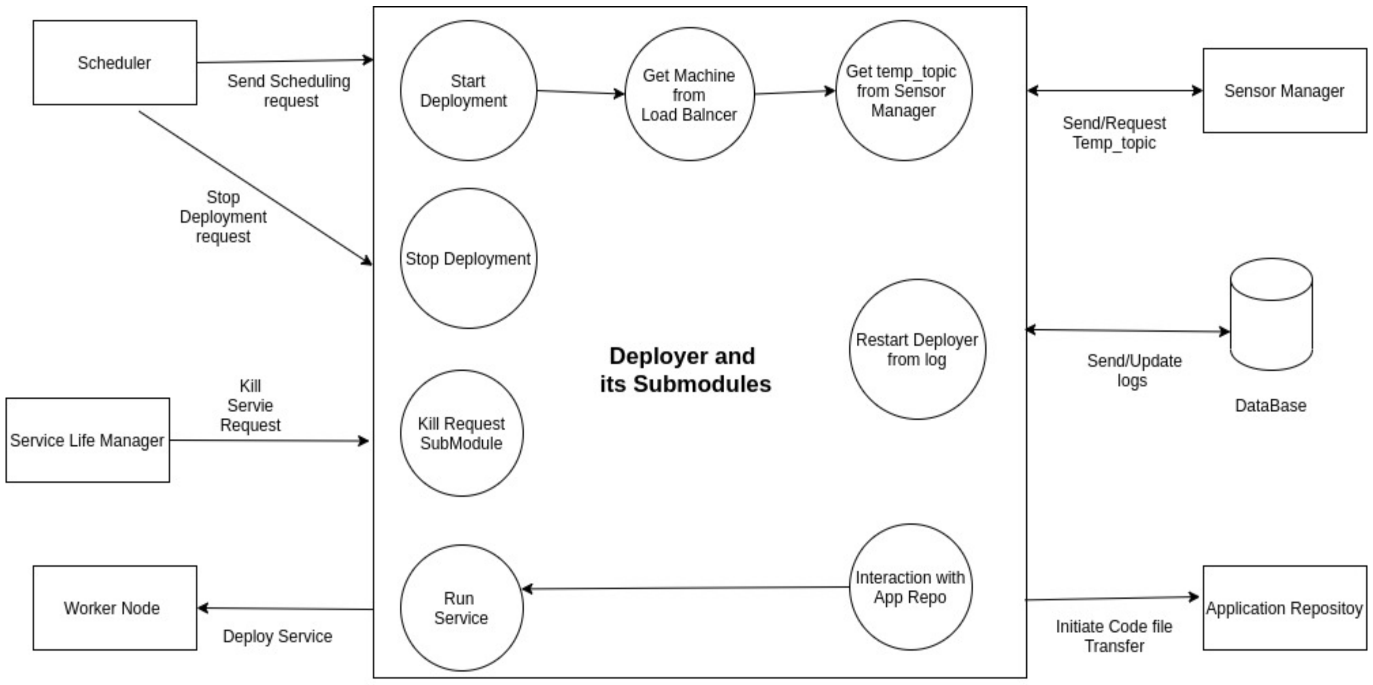
TA: Pratik Tiwari, Shubham Agarwal, Jay Krishna

Contents

1 Functional Overview of Team Modules

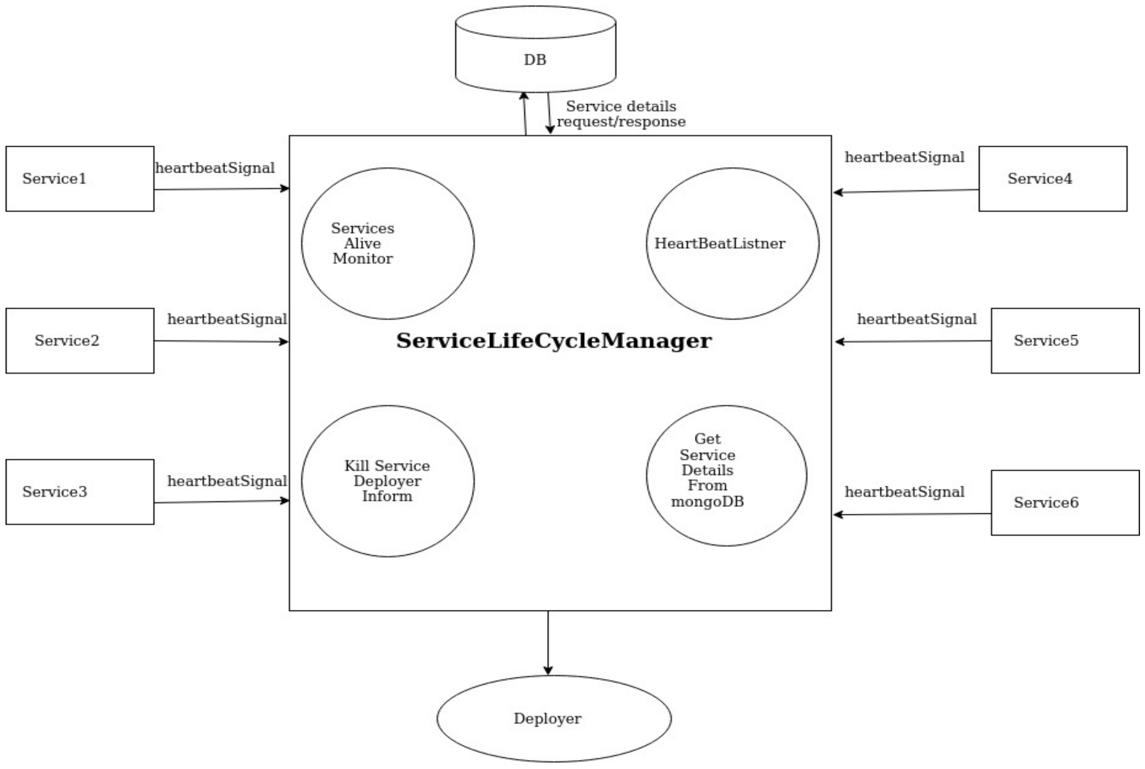
1.1 Deployer

Deployer is responsible for running the scheduled application service on one of the up and running worker nodes. It interacts with multiple modules such as scheduler, load balancer, Application Repository, service life manager and worker node to setup the enter environment and deploying the service. It also notify the scheduler to reschedule a service in case of service failure.



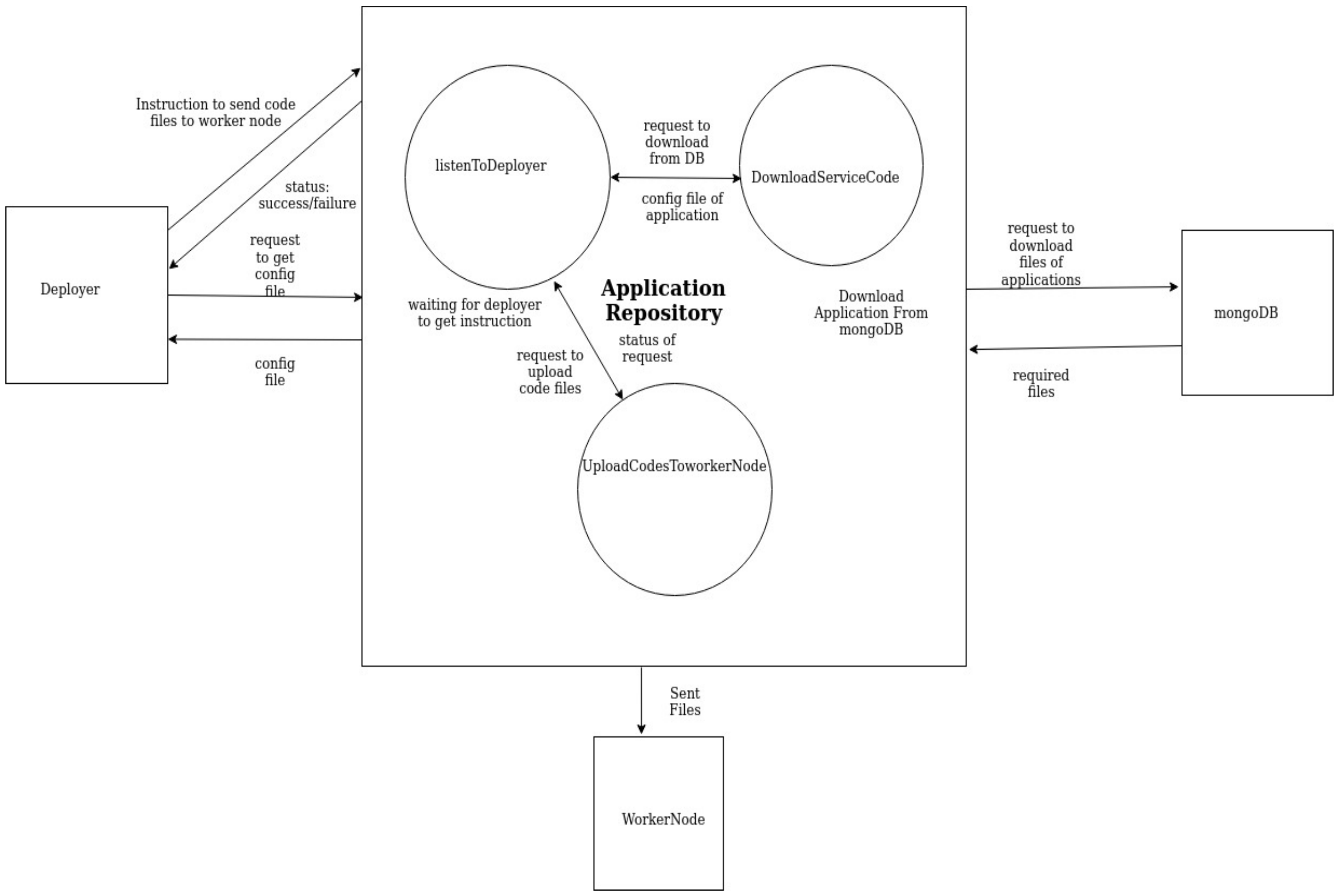
1.2 Service Life Manager

Service Life Manger is monitoring the working of each of the running application services. It notifies the deployer about the failed services which further interacts with other modules to take necessary action. Heartbeat technique is used at the service life manager to handle fault tolerance of services.



1.3 Application Repository

All the code and configuration files will be uploaded by the application developer on and stored in the application repository. Deployer will fetch the application config file for the application at the time of deploying from the repository.On getting schedule request deployer will instruct this module to send the program/code files to a specific worker node.


Instruction to send code files to worker node
status: success/failure
request to get config file
config file
Deployer
listenToDeployer
request to download from DB
config file of application
DownloadServiceCode
waiting for deployer to get instruction
Application Repository
request to upload code files
status of request
Download Application From mongoDB
UploadCodesToworkerNode
request to download files of applications
required files
mongoDB
Sent Files
WorkerNode

2 Use cases

2.1 Deployer

1. Deploy the application service on worker node.
2. Initiates the transfer of the required code files helper code to the worker node.
3. Deployer stops the services when instructed by the scheduler at the scheduled time.

2.1.1 Types Of User

1. Scheduler :- It is responsible to schedule the application service by sending request to deployer.
2. Sensor Manager :- sensor manager sends temp topic for the application service.
3. Application repository :- It sends the configure file of the application to the deployer.

2.2 Service Life Cycle Manager

1. All the application services send heartbeat to the life cycle manager to indicate that they are alive.
2. Life cycle manager monitor each service, if some service goes down it sends the notification to deployer, scheduler and sensor manager to update their respective databases.

2.2.1 Types Of User

1. deployer :- When some service goes down, service life cycle manager sends the signal to deployer to update its database and notify other modules as well.
2. services :- services send their heartbeat to service life cycle manager.

2.3 Application Repository

1. It downloads the zip folder of the code and configure files from the application database, when the deployment starts.

2. It sends the configure file as a response to a deployer request.
3. It provides the required code at the worker node before deployer runs the service. scheduled time.

2.3.1 Types Of User

1. Deployer :- Application repository sends configure file of the application service to the deployer.
2. Application Developer :- It sends all the required files of the application to the Application repository.

3 Test Cases

3.1 Test Cases for Modules

3.1.1 Deployer

Simulation : Created a test interaction file which simulates the functionailty of sensor manager, load balancer, application repo, scheduler. Simulated a worker node container to run the application.

Test case 1:
i/p : Request from scheduler to start deployment of a service with all correct format.
o/p: Deployer extract details and forwad it getMachine submodule.

Test case 2:
i/p : Request from scheduler to start deployment of a service with incorrect format.
o/p: Deployer sends status zero to scheduler and the error message.

Test case 3:
i/p: Load balancer responds with machine name of available worker node.
o/p: Deployer forwards the request to further sub modules to run the file.

Test case 4:
i/p : Fail to connect to Load balancer or get status zero, i.e., worker nodes are not available.

o/p: Deployer waits for some time and resend the request to load balancer.

Test case 5:
i/p : Sensor manager responds with the requested temp topic.
o/p: Deployer forwards the request to further sub modules to run file on worker node.

Test case 6:
i/p : Sensor manager responds with error message.
o/p: Deployer waits for some time and resend the request to sensor manager.

Test case 7:
i/p : Request from SLM to kill the service if worker node is down.
o/p: Deployer delete the entry of the service from its data structure and inform scheduler to reschedule the service.

Test case 8:
i/p : Request from scheduler to stop deployment when service is running.
o/p: Deployer login into the worker node using ssh and kill the service.

Test case 9:
i/p : Request from scheduler to stop deployment of a service which has been killed.
o/p: Deployer performs a check to see if service is in its running queue.If not it sends killed status to scheduler.

Test case 10:
i/p: Restarting the previously scheduled services according the failed state. o/p: Deployer stores the log of each service, to restart it checks the last state and restarts the deployment process accordingly.

3.1.2 Service Life Manager

Simulation: Created a test interaction file which simulates the functionailty of sensor manager, load balancer, application repo, scheduler.And simulated deployer which deploys multiple application services running on worker nodes.

Test case 1:

i/p : Heartbeat of the services.
o/p : Compares the heartbeat count and notify deployer if service count is zero.

Test case 2:
i/p : Service is dead but it has the entry in service database.
o/p : Service life cycle manager compares the service life cycle manager data structures with service database and deletes the conflicting entries.

Test case 3:
i/p : Separation between stopping service command and dead services
o/p : Stopping service command will clear the database entry, so if service is not responding, then service life cycle manager will check the persistent database before instructing the deployer.

Test case 4:
i/p : Previously running services in database(If service life cycle goes down).
o/p : It compares the state of the services present in database and restore its data structures,if found service has state as 6.

Test case 5:
i/p : Service stops sending heartbeat to the service life cycle manager
o/p : SLM sends the notification to the deployer to clear its data structures and notify other connected modules.

Test case 6:
i/p : Instance when deployer goes down, and SLM is trying to connect to the deployer to send the notification of the killed service.
o/p : In the data structure present at the SLM for the killed service, it set it as zero so that it can send notification to the deployer when it gets up.

Test case 7:
i/p : Service goes down before sending the heartbeat to the SLM.
o/p : Data structure of the SLM will get updated by running queue of the deployer.

Test case 8:
i/p : Mulptiple services sending heartbeat at the same time to the SLM.
o/p : SLM simultaneously updates its data structures.

Test case 9:
i/p : Mulptiple services go down simultaneously.
o/p : SLM updates its data structures and sends the notification to the deployer simultaneously.

Test case 10:
i/p : Service has restarted by the scheduler. o/p : The data stuctures of the SLM will get updated by the heartbeat.

3.1.3 Application Repository

Simulation : Create a test interaction file (tester.py) to simulate functionality of deployer.Simulated a worker node container to run the application. Simulated a script to upload zip to database as the APP UI module.

Test case 1:
i/p : Request from deployer to send config file of a particular application and config file is present.
o/p: gets folder from the database, unzip and send the config json to deployer.

Test case 2:
i/p : Request from deployer to send config file of a particular application but config file is not present.
o/p: sends error report to the deployer with missing config file message.

Test case 3:
i/p : Request from deployer to transfer code files to worker node and Zip folder is present in the database.
o/p: Unzip the folder, ssh into the worker node and recursively copies all the code files of service and helper files of platform.

Test case 4:
i/p : Request from deployer to transfer code files to worker node and Zip folder is not present

in the database.
o/p: App Repo sends unsuccessful status report to deployer with zip folder missing error .

Test case 5:
i/p : Request from deployer to transfer code files to worker node but helper files are not present .
o/p: App Repo sends unsuccessful status report to deployer with helper files missing error.

Test case 6:
i/p : Request from deployer to transfer code files to worker node but worker is not up and running .
o/p: App Repo sends error report to deployer to resend the request.

Test case 7:
i/p : Request from deployer to transfer code files to worker node but worker node is not up and running .
o/p: App Repo sends error report to deployer to resend the request.

Test case 8:
i/p : Request from deployer to transfer code files of an application already present in the worker node.
o/p: App Repo does not rewrite the files and send status to deployer file already present.

Test case 9:
i/p : Request from deployer to transfer code files of an application already present in the worker node.
o/p: App Repo does not rewrite the files and send status to deployer file already present.

Test case 10:
i/p : Request from deployer to transfer code files of an application but the source code folder is not present .
o/p: App Repo sends error report to deployer.

3.2 Integration Testing

Test case 1:
i/p : Input user login email id, password.
o/p : Login successful if valid login email id and password, else Incorrect login details entered.

Test case 2:
i/p : Application.zip file
o/p : Validation of directory structure. Upload if appconfig.json and src folder present in zip file else invalid.

Test case 3:
i/p : Request.json for scheduling the application.
o/p : Application scheduled if valid else error message to reschedule the application.

Test case 4:
i/p : Connection Refused from Application Repository while sending config file to deployer.
o/p : Retry connecting after 3 sec.

Test case 5:
i/p : Connection Refused from Load Balancer while sending the worker node to deploy the application.
o/p : Retry connecting after 3 sec.

Test case 6:
i/p : Status 0 from Load Balancer while sending the worker node to deploy the application.
o/p : No worker node available currently. Retry connecting after 3 sec.

Test case 7:
i/p : Connection refused from Application Repository while sending file through ssh to worker node.
o/p : Retry connecting after 3 sec.

Test case 8:
i/p : Status failed from Application Repository while sending file through ssh to worker node.
o/p : Issue while connecting worker node.Retry connecting after 3 sec.

Test case 9:
i/p : Status success from Application Repository while sending file through ssh to worker

node.
o/p : Files sent successfully to worker node.

Test case 10:
i/p : Connection failed at Sensor Manager for fetching temporary topic.
o/p : Retry connecting after 3 sec.

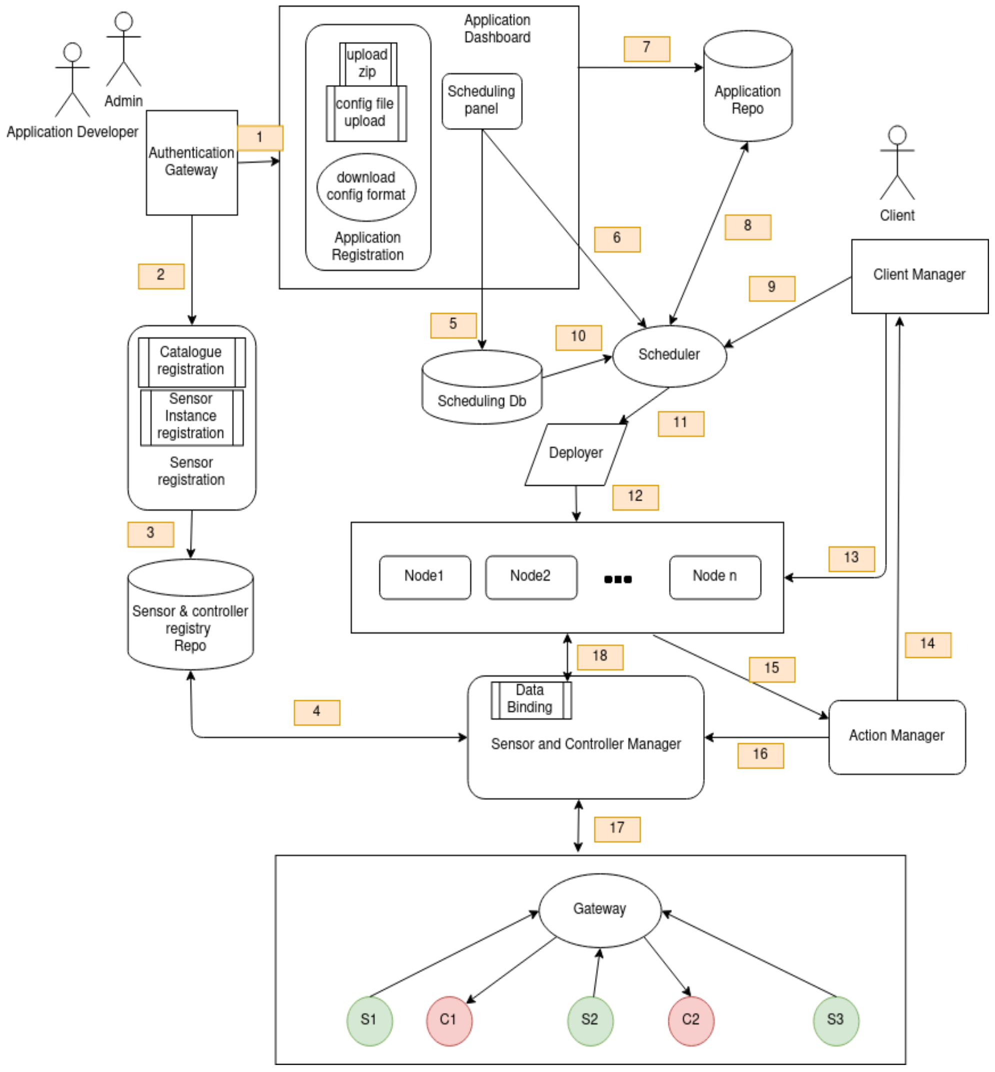
Test case 11:
i/p : Error at Sensor Manager for fetching temporary topic.
o/p : Temporarily unable to create temp topic.Retry connecting after 3 sec.

Test case 12:
i/p : status successful sent to scheduler when application deployed successfully.
o/p : Successfully deployed. Continue deploying a new one.

Test case 13:
i/p : json sent to action manager.
o/p : Notify the users through mail or message as described in request.json.

4 Solution Design Considerations

4.1 Design Big Picture



1. After Authentication, developer accesses the dashboard to register his application through zip file and also provides information about the scheduling of these application.
2. Again, after authentication developer can register the sensor, download catalogue(the format in which he has to upload his/her sensor information, however, sensor type and

data type of the sensor will be pre-defined in the platform)

3. Sensor after registration its information will be stored in repository(can be any kind of file-system), also, this module will provide access to controller manager.
4. Sensor controller provides access to controller manager through which controller can be accessed.
5. Dashboard will also provide interface to developer to provide scheduling information of the application which will be stored in scheduling DB.
6. Correspondingly along with scheduling db, the information will be provided in the scheduler.
7. Application source code zip file and the configuration files are stored on the Application repository.
8. Scheduler reads the application's source code and configuration file stored in the Application repository to pass it along for deployment.
9. If the application which the user wants to access it not deployed on any of the hosts, Client Manager will invoke the scheduler to immediately schedule a deployment into one of the hosts.
10. In case of some failure or a platform reboot, the scheduler will read all the scheduling information from the Scheduling database.
11. Scheduler sends the source code and the configuration file of the particular application to the deployer for it's deployment into the host machines.
12. After setting up the Deployment environment on the host machine(s) allocated by the Load balancer, the application's instance will be executed on those host machine(s).
13. Client Accessing the code deployed on hosts.
14. Action manager sending response or notification depending on the type of service expected and mentioned in the configuration file.
15. Hosts sending the response / output to the action manager to perform necessary action.
16. Action manager sending response to control the controller.
17. Sensors sending data to the sensor manager / Controller manager sends signal to manipulate the controller.
18. Sensor manager fetching sensor data (real time as well as stored) and input them to the code running on the nodes.)

4.2 Approach for communication and connectivity

1. We will use Flask API for communication between the different the modules.
2. The flow of information will be in the form of JSON request-response.
3. We will use messaging-queue as the communication medium.

4.3 Registry & Repository

4.3.1 Registry

- Sensor Registration
- Application Developer Registration
- Application Upload

4.3.2 Repository

- **Application Repository**:
- **Sensor Repository**:
- **Scheduling Database**:
- **Platform Repository**:

5 Application Model and User's view of system

1. After Authentication, developer accesses the dashboard to register his application through zip file and also provides information about the scheduling of these application.
2. Zip folder is validated for config file and src folder(which contains code files for the application) and is stored in the Application repository..
3. After validation developer can register the sensor, download catalogue(the format in which he has to upload his/her sensor information, however, sensor type and data type of the sensor will be pre-defined in the platform)

4. Sensor after registration, its information will be stored in repository(can be any kind of file-system), also, this module will provide access to controller manager.Sensor controller provides access to controller manager through which controller can be accessed.
5. Dashboard will also provide interface to developer to provide scheduling information of the application which will be stored in scheduling DB.
6. Correspondingly along with scheduling db, the information will be provided in the scheduler.
7. Scheduler reads the application's source code and config file stored in the Application repository to pass it along for deployment.
8. If the application which the user wants to access it not deployed on any of the hosts, Client Manager will invoke the scheduler to immediately schedule a deployment into one of the hosts.
9. In case of some failure or a platform reboot, the scheduler will read all the scheduling information from the Scheduling database.
10. Scheduler sends the source code and the config file of the particular application to the deployer for it's deployment into the host machines.
11. After setting up the Deployment environment on the host machine(s) allocated by the Load Balancer, the application's instance will be executed on those host machine(s).
12. Client access the application deployed on hosts.
13. Hosts sends the response / output to the action manager which further sends the response to controller to perform the necessary action.
14. Action manager will send response or notification depending on the type of service expected and mentioned in the config file.
15. Sensors sends data to the sensor manager / Controller manager which in turn sends signal to manipulate the controller.
16. Sensor manager fetches sensor data (real time as well as stored) and inputs them to the code running on the nodes.

6 Key Data Structures

6.1 Deployer

1. Running Service Queue
2. JSON format requests
3. JSON format response

6.2 Service Life Cycle Manager

1. Running Service Queue
2. Services heartbeat Dictionary
3. Presistent Data :- mongoDB (Services Information)

6.3 Application Repository

1. Presistent Data :- mongoDB (Services Information)

7 Interaction Interfaces

7.1 APIs

7.1.1 Modules to Modules Interactions

Scheduler to Deployer interaction :-

startdeployement

i/p : Scheduler sends POST request in JSON format containing service name, user id, app id, action details, service id, place id.
o/p : JSON response containing success/failure status.

stopdeployement

i/p : Scheduler sends POST request in JSON format containing user id, app id, service name and service id. o/p : JSON response containing sucess/failure status.

Deployer to Application Repo interaction :-

send config file

i/p : Deployer sends application name to app repo to download from database.
o/p : Config file of the application to deployer.

send files machine

i/p : Deployer sends machine name, machine password, machineIP, app id, service name, service id, action details to the app repo to send all the files on the worker node. o/p : JSON response containing sucess/failure status.

Deployer to Sensor Manager interaction :-

sensor manager

i/p :Deployer sends request to sensor manager containing service id.
o/p :Temp topic from sensor manager.

Deployer to Load Balancer interaction :-

load balancer

i/p :Deployer sends request to load balancer containing service id.
o/p :Gives worker node or error status if no node present.

8 Persistence

8.1 Deployer

- Deployer logs each running service along with the state at which it has completed.
- Deployer has 6 stages: scheduling request received,get config from app repo, get worker node from load balancer, get details from sensor node, send files to work node and run files.
- Whenever Deployer starts, it reads log data and restart each service from the next state.

8.2 Service Life manager

- It maintains logs of services which are running on the deployer.

8.3 Application Repository

- Application source code zip file and the configuration files are stored on the Application repository.
- Application Manager checks structure of config JSON file(Application config), if correct structure, store it in Application Repository.
- Application Repository will be used by the Deployer at the time of Deployment of Applications, gets the relevant data (application code, config, etc.)