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 Overview

1.1 Introduction (Definition of what constitutes in this project)

This is an IoT based platform where multiple types of IoT Applications can be deployed and monitored through web/desktop/mobile based application. The platform is developed so that it eases the work of developers handling each sensors separately, here the sensors management and controlling is kept abstract from users perspective.

1.2 Scope

Scope of the project is to build end to end platform for managing IoT devices where users can deploy their applications, collect data from platform, scale the platform to support high number of concurrent users, build dashboards to get insights from the data.

2 Intended Use

2.1 Intended Use

Consider a use case of taxi in super-app(kind of like everything can be deployed) as an IoT device, in this different types of services/features for taxi can be deployed like taxi-ride, food delivery, grocery pick and drop etc.

2.2 Assumption & dependencies

Assumption: Basic assumption at this point of time, that all the sensor supports REST/HTTP protocol for request and response through the data, for the time being only python codes can be deployed.

Dependencies: Any libraries which makes the work easier are going to be used, however not API's/libraries which uses the entire component of the platform.

3 System Features and Requirements

3.1 Modular Requirements

3.1.1 Deployment of the platform

Platform initializer will start the platform modules as per various parameters provided by the platform Configuration file.

3.1.2 Applications overview on the platform

The applications that will be running in the platform will interact will various IoT sensors. Those sensors will be registered in the platform i.e the sensors will collect real world data and provide those data to the sensor server module of the platform. The platform will provide those data to the application and then the application will perform tasks according to the analysis of the data.For example like smart college campus. It contains various sub systems like Smart Parking(Monitoring of parking spaces availability in the campus), Smart Lighting(Intelligent and weather adaptive lighting in campus lights), Temperature control in different labs, classes and server rooms, fire alarm system etc.

3.2 Functional Requirements

3.2.1 Starting and Stopping services

When a request is received to start a service from scheduler, deployer requests machine name from load balancer, application files from application repository and sensor topic from sensor manager initiates the application. At the end time, scheduler will hit stop deployement on deployer to stop the service.

3.2.2 Service Life cycle

All the services send heartbeat to the service life Manager. If heartbeat is not received till 1 min, then deployer is informed about the service.

3.2.3 Deployment of application on the platform

After receiving machine name from load balancer, application files from application repository and sensor topic from sensor manager and installing all the dependencies, deployer initiates the application.

3.2.4 Interactions between modules

Interaction between Scheduler is through Flask API start deployment at Deployer. Interaction between Application Repository and Deployer is through Flask API sendconfigfile and sendfilestoMachine. Service Life Manager hits killed service API on deployer when a service is identified to be dead.

3.2.5 Configuration files details

Json files is created to send notification details to action manager.

3.3 Non - Functional Requirements

3.3.1 Fault tolerance

If any of the modules Application Repository Service, Deployer, Service Life Manager goes down, it's identified by the fault tolerance module and is initiated. Each modules maintaines logs in mongodb for each service. Thus on restarting it reads the logs and get restored to the previous state.
For Application, Service life manager maintains heartbeat count for each service as soon as any service goes down, heartbeat count is no more increment and being set to 0 helps SLM to identify that service has stopped. It informs the deployer. Deployer in turn send message to Scheduler to re schedule the application.

3.3.2 Scalability

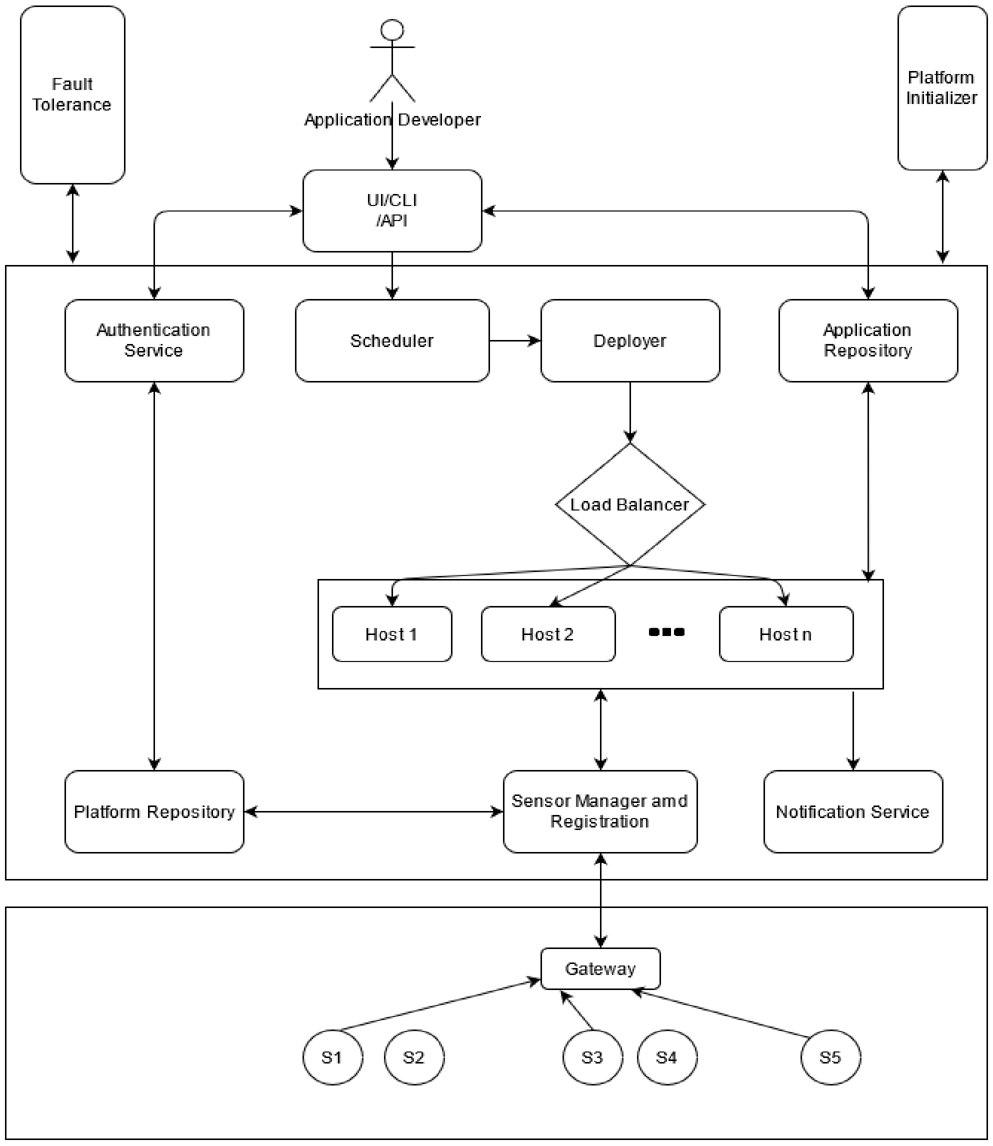
On increasing data storage limits, to allow for scalability in this regard, we will be using distributed databases. To allow for different application instances to be run for the users, we will provide and array of different servers that will host these applications. A load balancer will be in place to balance the load on these servers.
The applications that are deployed on our platform are also distributed in nature so they can also be scaled horizontally. We can have different machines hosting the instances of the same application to allow for good scalability. The application configuration file will be used to spin up a new instance of the given application. The choice of load balancer is still unclear but there needs to be some kind of load balancing for requests that are incoming to the application.

3.3.3 Persistence

If any of the modules Application Repository Service, Deployer, Service Life Manager goes down, it's identified by the fault tolerance module and is initiated. Each modules maintaines logs in mongodb for each service. Thus on restarting it reads the logs and get restored to the previous state.

4 Key Functions

4.1 block diagram listing all major components



Block Diagram of platform

4.2 Description of each component

4.2.1 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.

4.2.2 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.

4.2.3 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.

4.3 4 major parts

4.3.1 Deployer

Deployer is responsible for running the scheduled application service on one of the up and running worker nodes.

4.3.2 Service Life Manager

Service Life Manger is monitoring the working of each of the running application services.

4.3.3 Application Repository

All the code and configuration files will be uploaded by the application developer on and stored in the application repository.

5 Use Cases

5.1 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.

5.1.1 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.

5.2 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.

5.2.1 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.

5.3 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.

5.3.1 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.

6 Test Cases for the project

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.

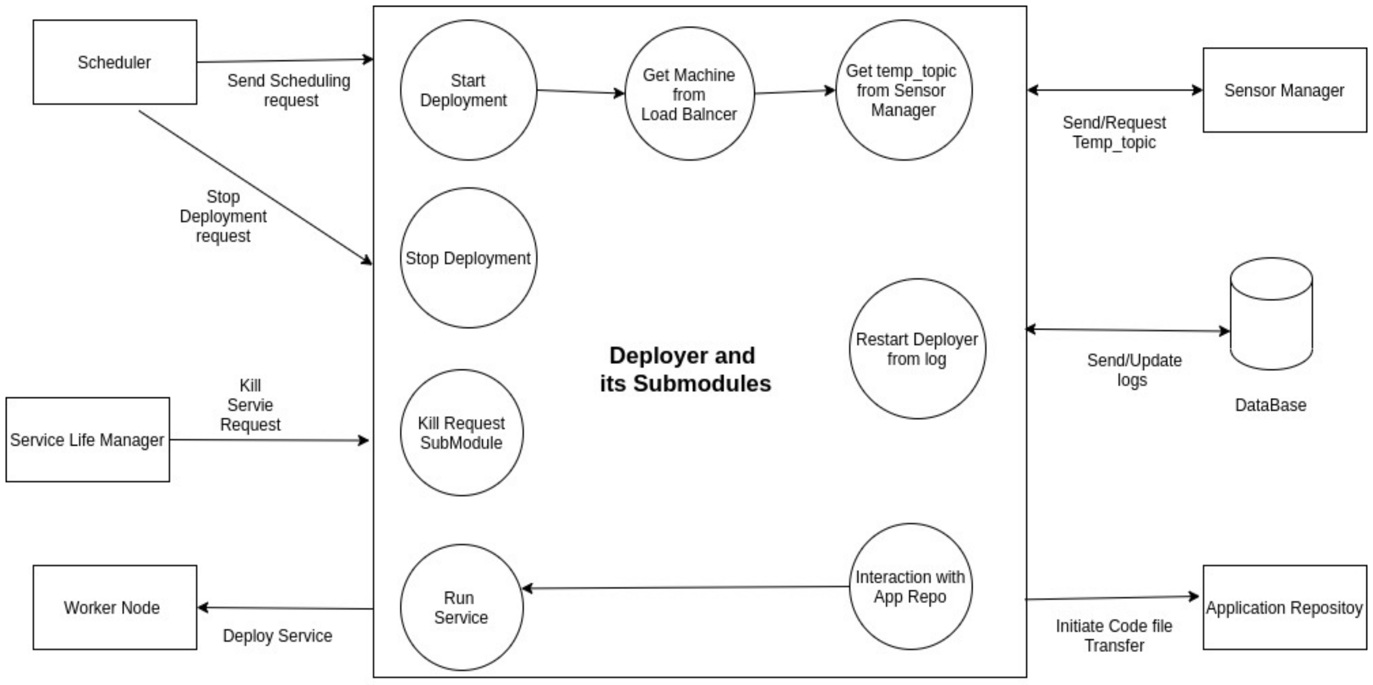
Test case 12:
i/p : status successful sent to scheduler when application deployed successfully.
o/p : Successfully deployed. Continue deploying a new one.

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

7 Subsystems

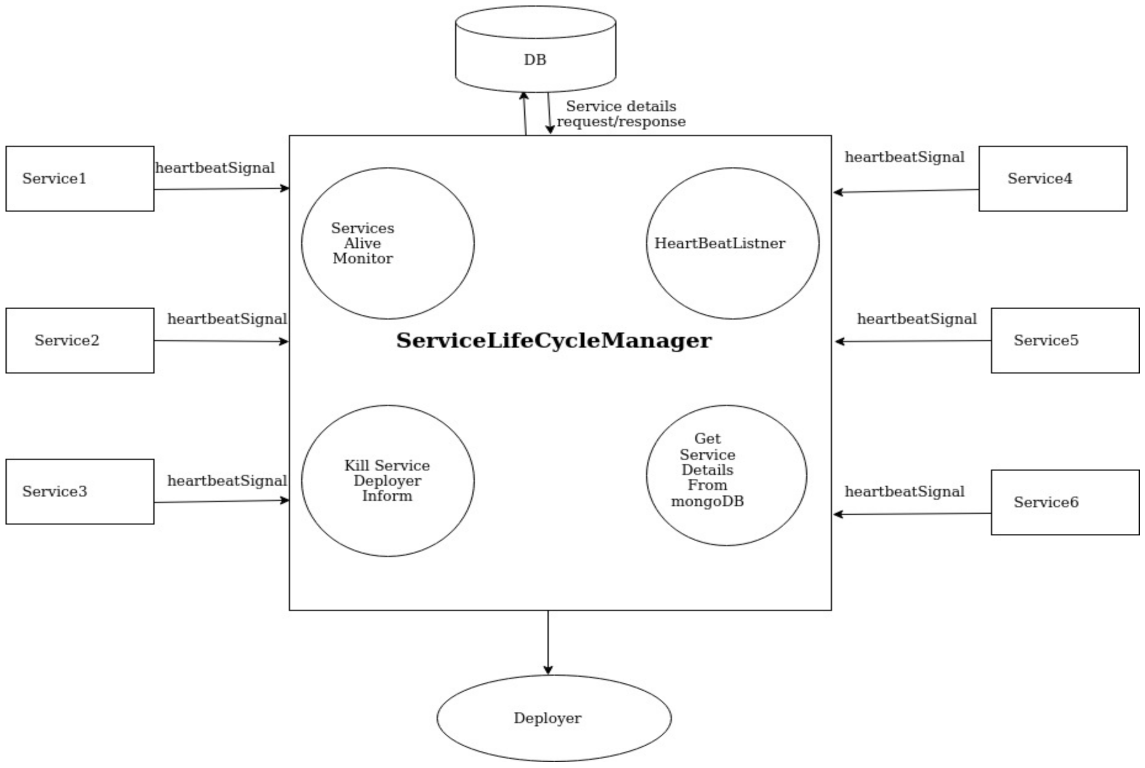
7.1 Deployer

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.



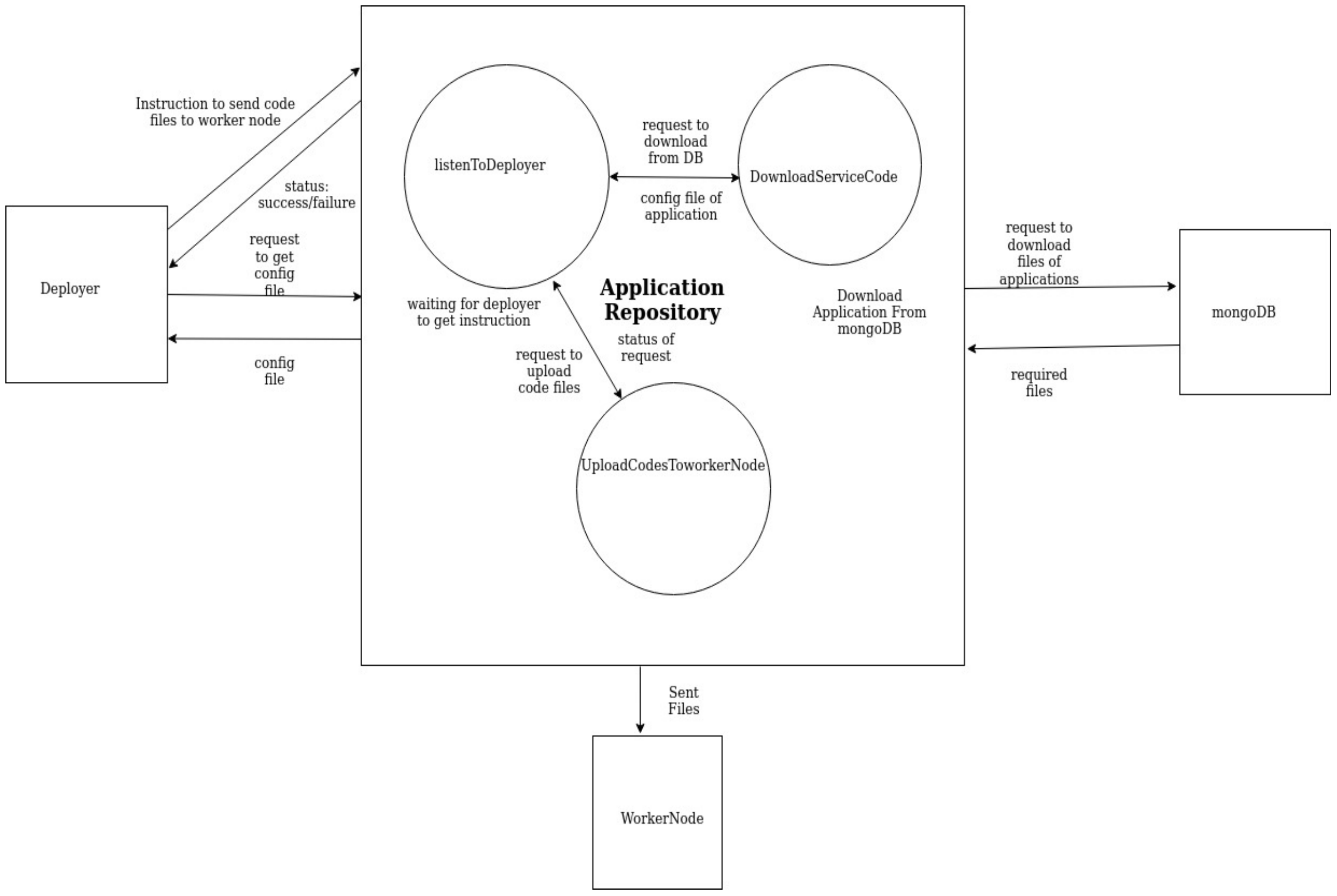
7.2 Service Life Manager

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.



7.3 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
Download Application From mongoDB
status of request
request to upload code files
UploadCodesToworkerNode
request to download files of applications
required files
mongoDB
Sent Files
WorkerNode