Internals of Application Server

Distributed IoT Platform

Group Design Document

Group 5

Table of Contents

Introduction

IoTs are now playing an important role handling the environments intelligently with convergence of sensor devices. Physical deployment of devices like sensors to connect devices remotely for seamless functioning and ease of operations. A platform bridges the gap between device sensors and data networks. An IoT platform is a set of components that allows developers to spread out the applications, remotely collect data, secure connectivity, and execute sensor management.

Use cases/Requirements

- Config files containing data about sensor types, instances, id's, geo-location
- Algorithm to be executed having particulars about classes and functions to get the sensor data and use it in the way to generate the specific output required by the end user.

Test cases

Test cases for modules

- **Deployer**
    a. Integration check with load manager
    b. Server connection check to get values from each node for heartbeat monitoring service
    c. Test case to validate load on server freeness
    d. Integration check application and sensor id binding
    e. Any node server goes down or needs to be unallocated.
- **Scheduler**
    a. Checking various priorities.
    b. Checking job type
    c. Testing different time instances. (start/end)
    d. Testing various intervals/durations.

- **Application Manager**
    a. Testing validator logic
    b. Is it correctly placing script files in respective directories
- **Bootstrap**
    a. Can it load the whole system in order or sequence
    b. Can it recognise the config file
    c. Can it make the whole system executable by installing all the required basic softwares.
    d. Can it deploy the whole platform by accessing a remote server.
- **Sensor Manager**
    a. Sensor Installation: A new instance must not get installed if the type of that sensor is not registered on the module
    b. Every algorithm gets the data from the sensor that was asked only.
    c. The information model of the sensor type is being maintained.
    d. The sensor instances are getting filtered correct based on location or placeholder id
    e. The sensor data is provided only when required.
- **Communication Module**
    a. Check if it's used for communication between the modules then is it able to take data from one module and send it to the other.
    b. How it will manage different types of sensors data.
    c. How a dynamic topic will be created and sensors will be binded at runtime.
    d. If load on a particular topic increases then how does it manage the load and do balancing of load on topic.

Test cases for overall module (group specific)

1. **Authentication**
    a. Validate Username and password
    b. Check access permissions of the user
2. **Deployment**
    a. Check the deployment status of an application model based on the end point.

3. **Notification service**
    a. Checking if the user is receiving notification about the model result or not

Solution design considerations

Design big picture

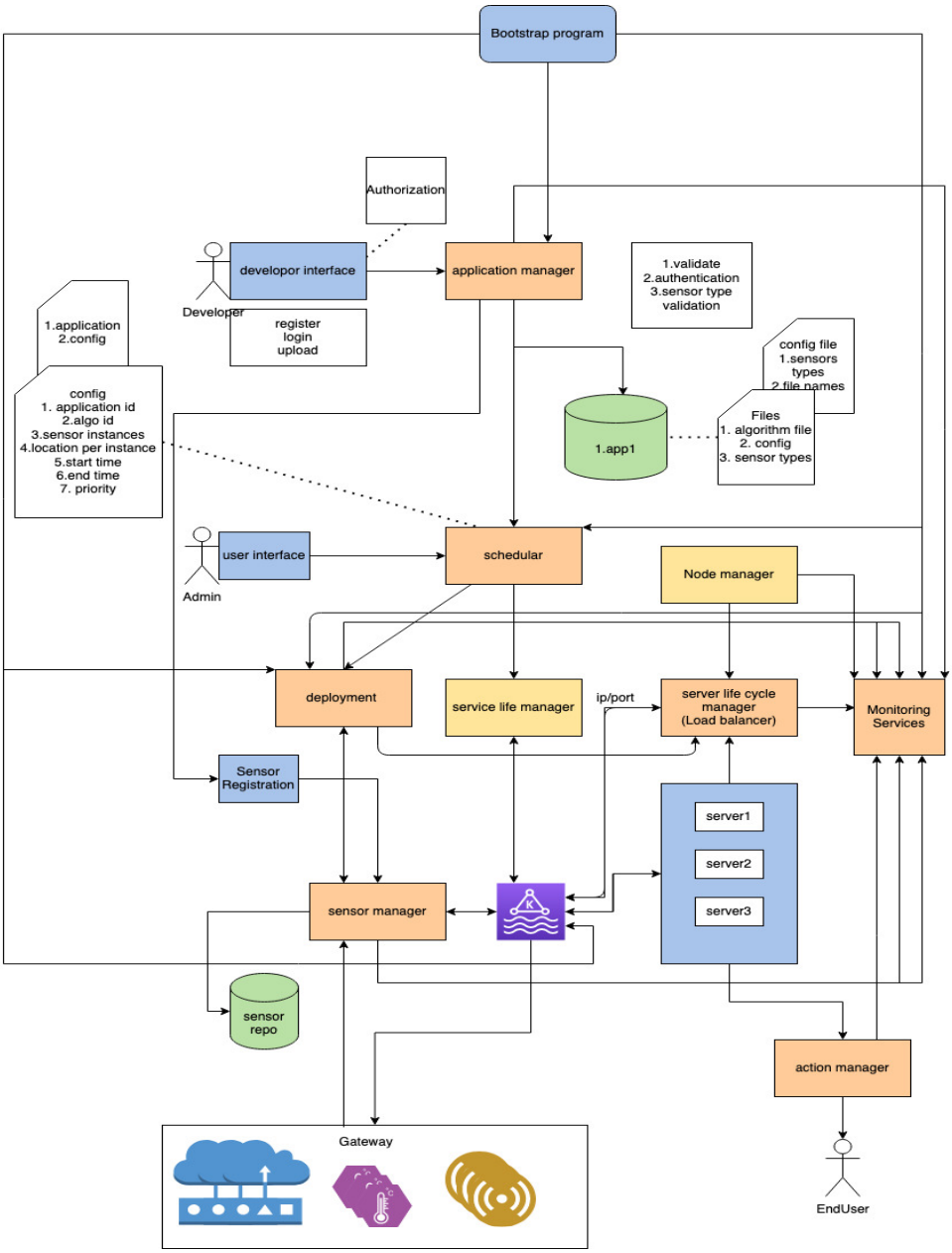


Fig: Block Diagram

Environment to be used

- 64-bit OS (Linux)
- Minimum RAM requirement : 8GB
- Docker image used for model transfer.

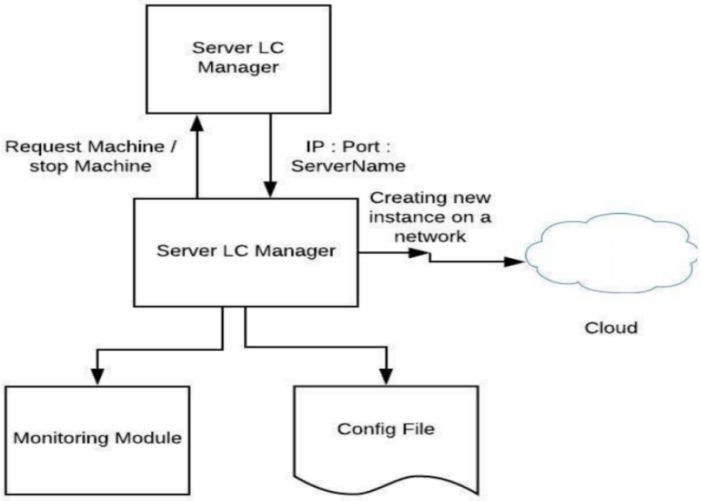
Technologies to be used

- Kafka
- Docker
- Flask
- Python
- Bash Scripts
- MongoDB
- Sensor simulator

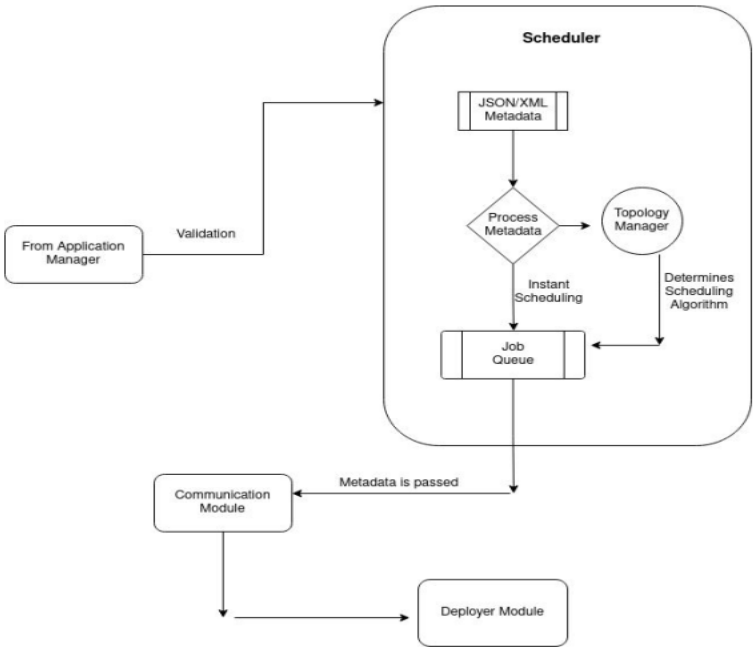
Overall system flow & interactions

- At first platform Initializer will start and get information of all the modules from the config file, and then It will basically initialise the modules from which it got the config file.
- User (application developer) will be authenticated using UI
- User will execute its algorithms and make a config file of the same and then further this config file will be validated by the platform
- After this the algo will be deployed by scheduler and it will assign some node for the job.
- The result of the algorithm execution will be sent to the action module which will communicate to various sensors depending upon the algorithm output.

Server & service lifecycle



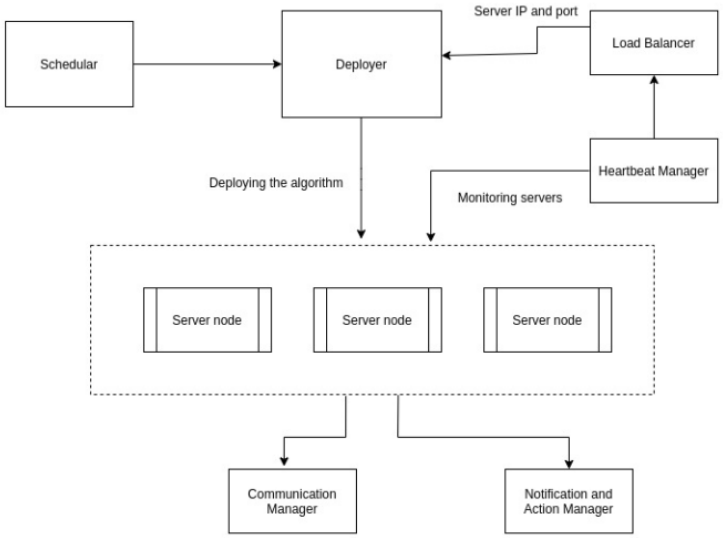
Scheduler



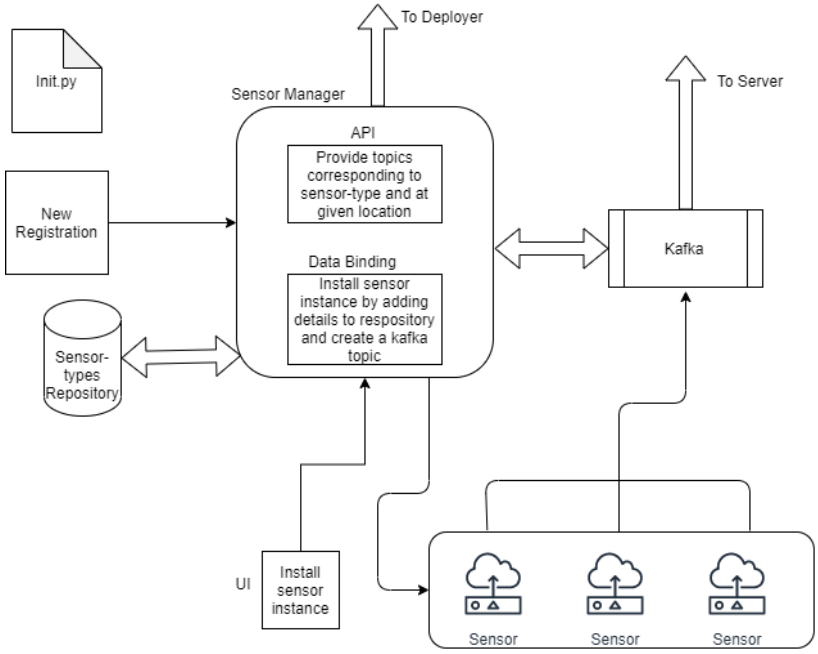
Load Balancing

- Scheduler module accepts the information from the application manager which validates the information of the user and now the scheduler will handle its scheduling information given by the user.
- The scheduler will assign the job to the deployer.

- Heartbeat monitor will determine current server status and if status is fine will proceed.
- Load balancer component will assign the appropriate server node.
- Deployer will execute the specific algorithm for the application.



Data Binding



Interactions between modules

Refer to block diag.

The Kafka cluster stores streams of records in categories called topics. A topic is a category or feed name to which records are published. Topics in Kafka are always multi-subscriber; that is, a topic can have zero, one, or many consumers that subscribe to the data written to it.The communication module will create Topics for each component as well as sensor and then whoever want to communicate with a component will write on the corresponding Topic and a component will receive the message sent to it by reading the data on its Topic.

Wire and file formats

Most of our communication between modules happens with the help of RESTful services. As our module consumption/release of inbound/outbound data uses the RESTful services internally we are in front of JSON.
Our important data stores in document DB (MongoDB) in the form of JSON and are easy to control programmatically.

Communication management

Communication module is the core part of the IOT platform. It acts as the central module between deployment manager and sensor manager which will help in binding the streaming sensor data with active algorithms(running instances). We are planning to build the communication module using stream - processing framework Kafka. Kafka server will be used for all communication within modules (example sensor manager to repository or Node etc.). The Information Model for Communication system will be based on input and output APIs.

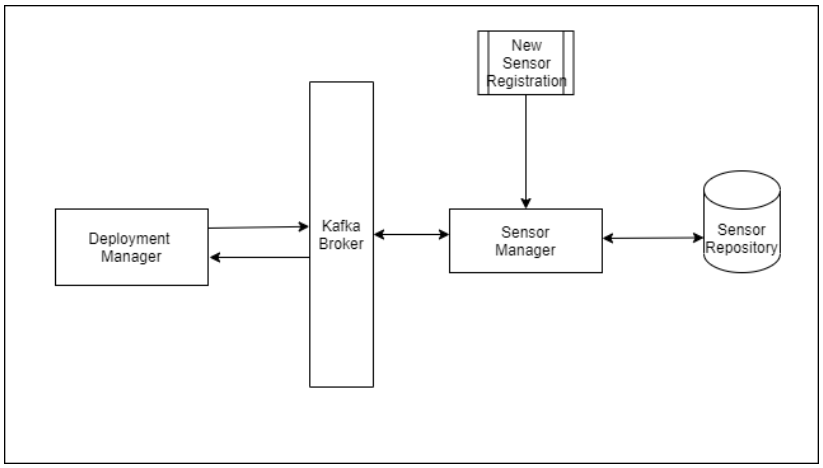


Fig.: Communication module

Application Model and User's view of system

Overview of the application dev model

- Application developer interacts with our authentication service and login with his credentials
- If user is new he/she must use our auth service to register himself and auth data resides in mongodb
- Once app developer logins our system will provide an interaction menu to reach our repository service which get holds of our applications which are bundled and hosted in local repository
- CLI provides developers to upload new algorithms that are developed locally and at the same time dev can list out to see the currently hosting algorithms. Moreover, legacy algorithms can also be removed.
- End user now registers himself and uploads the config file of the sensor types instances
- Scheduler then binds the sensor instances and pass it to the deployer
- Deployer then run the algorithm notify the user when meets with given condition or perform certain action if the application is of action type

Application artifacts

Application Artifacts in our development model

In general artifacts in the application development model can be API, interfaces, application config files, scripts, table definitions, configurations, UI elements, and etc.
Files -

1. Application.config - dependencies required by app.config, config settings like OS, Platform.
2. Application.meta - metadata file is a xml formatted type containing the information about scheduling. let's say, location, start_time, end_time, temperature etc.
3. Platform configuration file – JSON file containing the list of environment server urls(host/port), db information, sensor registries it might need for many operations like while bootstrapping or can be used as topology when killing/stopping and starting the platform services.
4. Schedule.config - contains information like at which rate the job trigger

API

- Sensor API – This can help identify the gateway, config, actions, type, manufacturer_details, output format.
- Database Driver API – Api helps working with DDL, DML operations and along with that we can get data based on conditions, and aggregated filtered data.
- Kafka Driver - we are using kafka for queue/topic so using it our system will find out about feed generated by sensors.

Modules

Modules play a crucial role in application development platforms. Every module is lightly coupled with little light tagging for serving the purpose of part of the platform lifecycle. Some of the modules that are indulged are.

1. Document database like mongoDB for sensor registration, while holding the meta data, configuration of sensors and it can take up the sensor produced data.
2. Kafka to send and receive response/requests and kafka connect.
3. Authentication service across the platform.
4. Gateways – route messages from sensors and do some actions on top of it.

Interfaces

1. Session based login with authentication and provides an authorization of to access flows or modules in a platform.
2. Sensor registration and configuration
3. An interface to access sensor data from mongodb.
4. Web service interface like REST API to schedule the algorithm, so it gives the query-based data extraction from the database.
5. We can develop UI for developers to invoke to get live sensor data, status of nodes, does the deployment based on the post request, algorithm upload etc.

User admin interactions

- Check server load status
- Check logs
- Register new users/application developers
- Register sensors
- Monitor overall health status

Key Data structures

- Queue (for scheduling)
- Hash Map (mapping Ip to the CPU usage RAM usage of the nodes)
- Heap (for priority scheduling)

Interactions & Interfaces

- App manager - Scheduler
- Scheduler - Deployer
- Deployer - Communication Module
- Communication Module - Sensor Manager
- Scheduler - Server Life Cycle
- Server Life Cycle - Load Balancer & Fault tolerance
- Authentication - to all end parts of deployment and registration

Persistence

Our IoT platform is persistent because of high availability that our node manager service creates a runtime node incase if it finds out service down with help of a heartbeat monitor.
Our platform is also elastic as it scales the servers to run applications in case the load on the server beyond the threshold.

The modules that the four teams will work on

S No	Team	Modules
1.	Team1	• Communication Module • Sensor manager • Data Binding
2.	Team2	• Scheduler • Monitoring
3.	Team3	Deployment Manager • Deployer Server Lifecycle Manager • Load Balancer

		• Heartbeat monitor • Node Manager Sample application
4.	Team4	• Application Manager • Platform Initialiser