

Group7 - Team1 Design Document

Members - Chitra Kumari, Akshat Gupta, Aayushi Nigam

Components - Scheduler, Deployer, Load Balancer, Platform initializer

May 8, 2021

Contents

1 Overview	1
2 Block Diagram	2
3 Technologies Used	4
4 Environment Used	4
5 List of subsystems	5
5.1 Scheduler	5
5.2 Deployment Manager	5
6 List of sub-modules	5
6.1 Platform initializer service	5
7 Actors Involved	6
7.1 Platform Deployer	6
7.1.1 Flow with respect to the platform deployer	6
7.2 Configurer	7
7.2.1 Flow with respect to the configurer	7
8 Services	8
8.1 Platform initializer	8
8.2 Scheduler	9
8.3 Deployer	10
8.4 Load Balancer	10
9 Key Data structures	12
9.1 Platform initializer	12
9.2 Scheduler	12
9.3 Deployer	12
9.4 Load Balancer	12
10 Lifecycle of the module	12
11 File Formats	13
12 Registry and Repository	15
13 Interaction with other components	16
13.1 Interaction between Application Manager and Scheduler	16
13.2 Interaction between Scheduler and Deployment Manager	16
13.3 Interaction between Deployment Manager and Sensor Manager	16
13.4 Interaction between Deployment Manager and Node Manager	16
13.5 Interaction between Platform Initialiser and Scheduler/Deployer	17
14 Persistence	19
14.1 Scheduler	19
14.2 Deployment Manager	19

15 Test Cases	20
15.1 Test cases to test the team's modules	20
15.1.1 Scheduler	20
15.1.2 Deployer	20
15.1.3 Load Balancer	20

1 Overview

Our team deals with 3 components - **Scheduler**, **Deployer** and **Load Balancer** (which will be a sub-component of the deployer), and 1 service - **Bootstrapper**.

Platform initializer is used to provision VMs, deploy various components of the platform on the VMs with minimal manual intervention.

User's request to deploy an application is sent to the scheduler which passes the application(or algorithm) to be run to the deployer at the scheduled time.

Deployer after balancing the load among the nodes, makes sure that the app(or algorithm) runs on the appropriate node (or virtual machine). There is also a heartbeat manager associated with every component to keep a check on the health of each component (i.e. to make sure that platform doesn't go into absurd state if any component fails).

2 Block Diagram

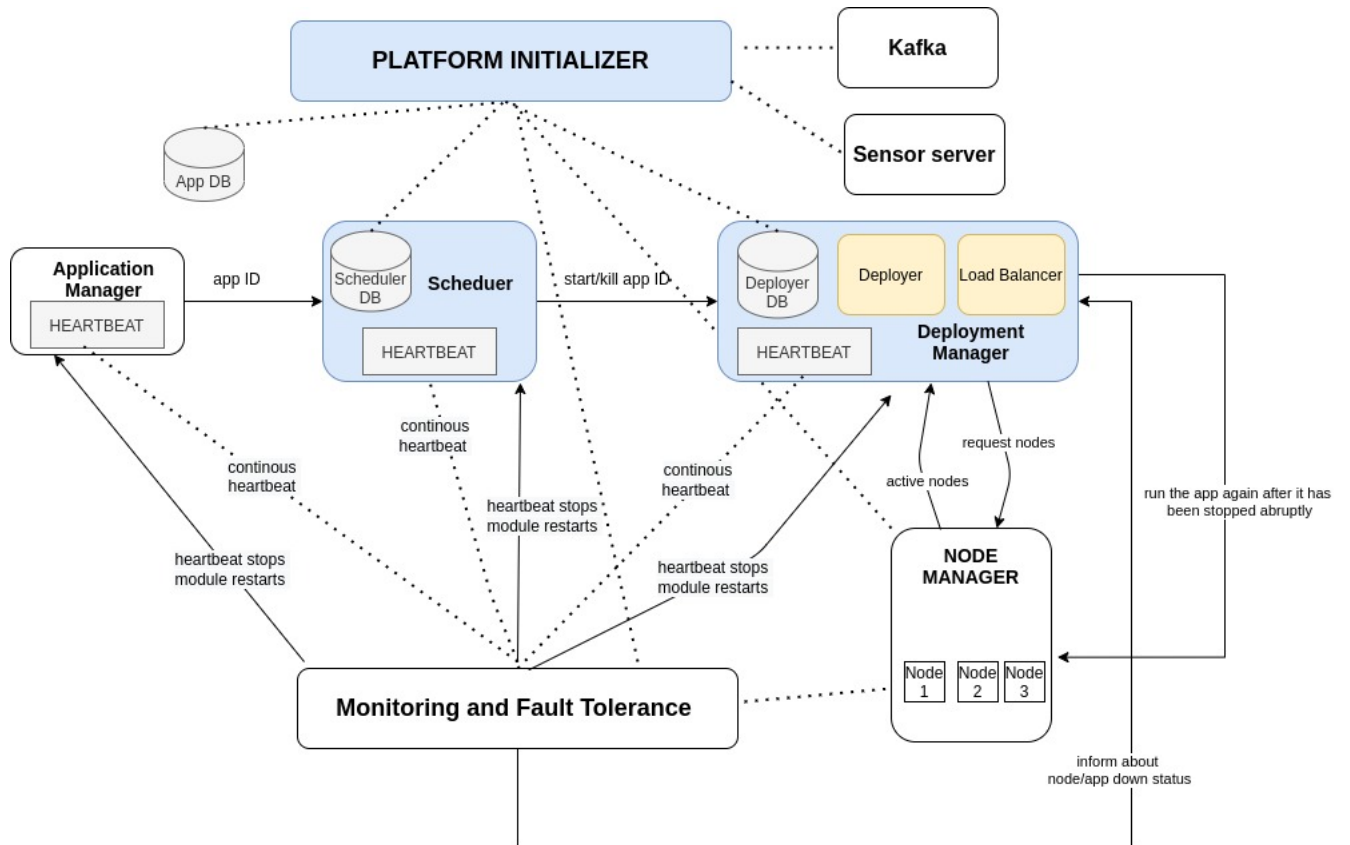


Figure 1: Block diagram for the components involved

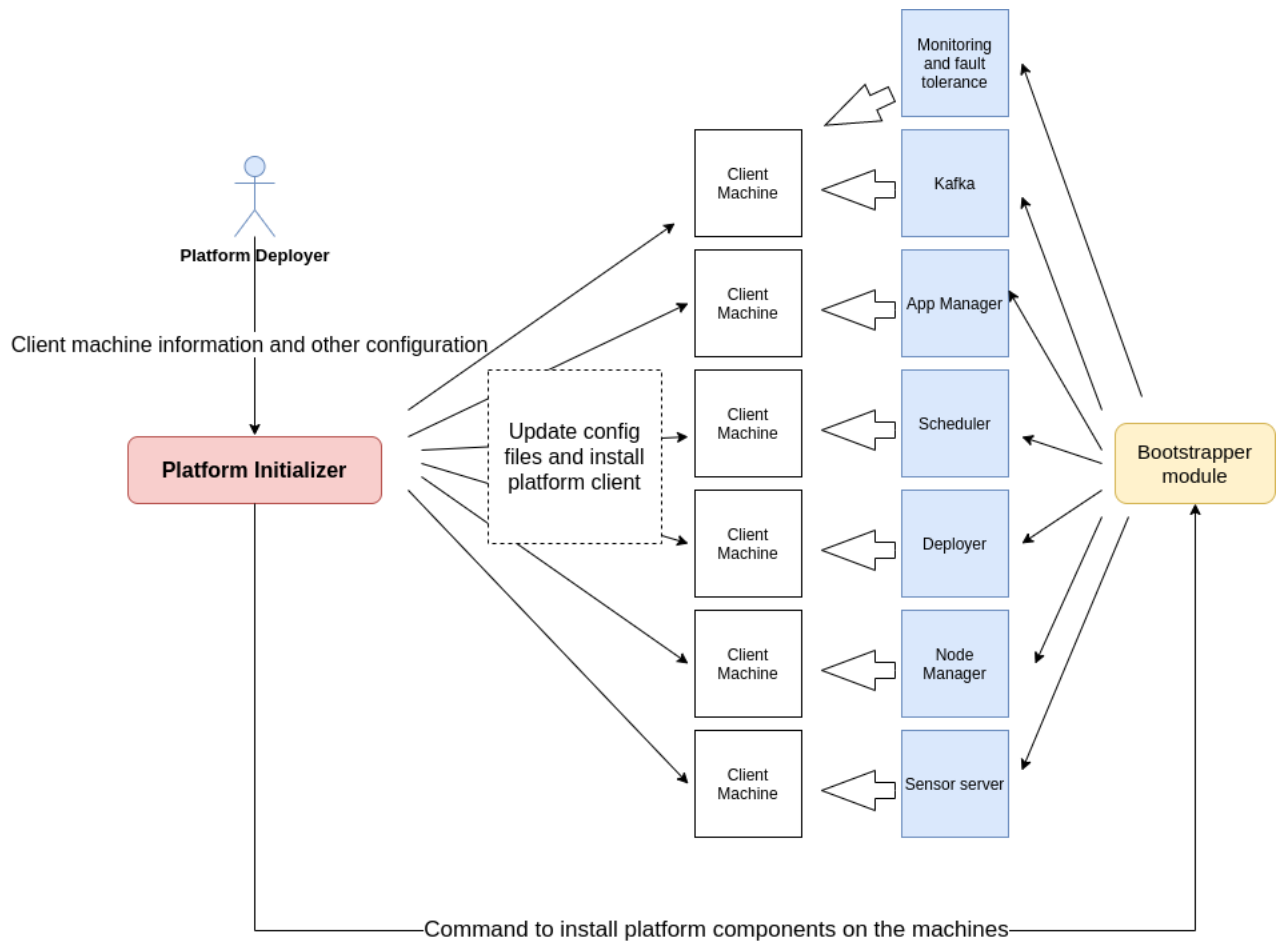


Figure 2: Block diagram for the platform initializer

3 Technologies Used

- **Python framework** - overall development
 - Presence of Third Party Modules
 - Extensive Support Libraries
 - User Friendly data structures
- **Apache Kafka** - for communication
 - Scalable
 - Fault Tolerant
 - Publish-Subscribe Model
- **MySQL** - for database
 - Data Security
 - High Performance
 - On demand scalability
- **Docker** - for platform independence and portability
 - Consistent and isolated environment
 - Mobility
 - Easy collaboration
- **Microsoft Azure** - for distributed environment
 - High Availability
 - Good Scalability Options
 - Cost Effective

4 Environment Used

- OS - Linux 64-bits

5 List of subsystems

5.1 Scheduler

- In memory heap
- Scheduler DB
- Heartbeat Manager

5.2 Deployment Manager

- Heartbeat Manager
- Load Balancer
- Deployer DB

6 List of sub-modules

6.1 Platform initializer service

- Prerequisites installer
- VM provisioner and config updater
- Platform client installer
- Platform components bootstrapper

7 Actors Involved

7.1 Platform Deployer

Actor involved for the **Initializer service** is the **Platform Deployer**, who deploys the platform on the client system / machines. The platform configurator updates the addresses of the client machines, if they already exist, or provisions them using the service, if they don't exist. The bootstrap service then takes over to handle the client machines, and deploy various components of the platform on different machines, in a distributed manner.

7.1.1 Flow with respect to the platform deployer

- The platform deployer configures the information of the client machines on the bootstrap service.
- If the client machines do not exist, deployer can also configure the initializer service to create distributed machines for client.
- The platform initializer then configures the machines to run the platform components and initializes them on the machines in a distributed manner

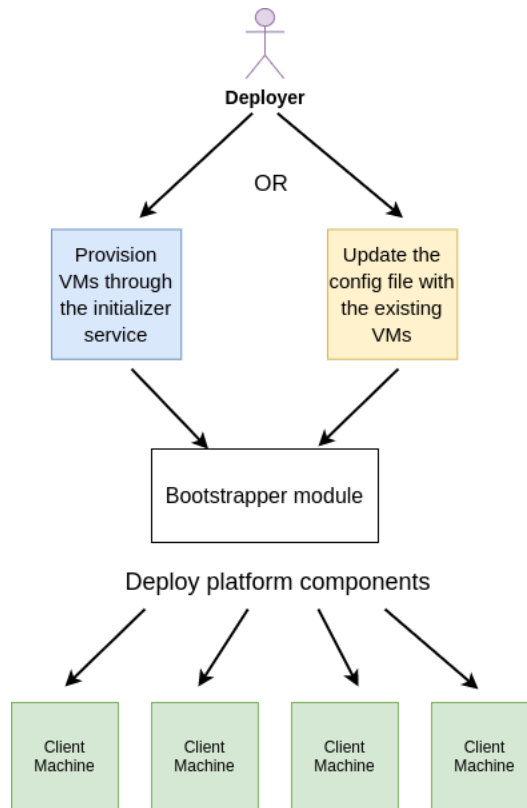


Figure 3: Deployer interaction with platform

7.2 Configurer

Actors involved in these components is the **Configurer**. Configurer will bind the physical instances of the sensor at configuration time. During this time, configurer will provide scheduler with the scheduling information. This scheduling info includes how many instances to create, and for each instance, set its run mode, i.e, whether it should run on demand, periodically, continuously etc.

7.2.1 Flow with respect to the configurer

- The app developer deploys the application on the platform.
 - The configurer then contacts the scheduler and decides on the application that he/she wants to configure
 - The configurer provides information to the scheduler regarding the number of instances to create for a particular application, or the jobs to be run by the scheduler.
 - This information can be provided by the configurer in two ways.
 - * Through the UI dashboard of the scheduler
 - * By uploading the configuration files.
- Here, we have provided configurer with UI to provide the information

Home

Appinfo(id, Name):

App id:

Location Information

Room no:

House No:

Street:

City:

Execution Information

Date:

Start Time:

Duration(in minutes):

Repeation needed? Yes: ☒ No: ☐

After how much time?

Year	Month	Day	Hour	Minute
<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>	<input type="text" value="0"/>

Figure 4: Scheduler UI

8 Services

8.1 Platform initializer

- The platform initializer initializes the client machines so that they are able to run the platform components. The platform components are independent of the number of machines the client has, and can run all on one machine, or even if they are distributed.
- Platform initializer then builds the configuration files, sends them to every machine, and install platform client on them

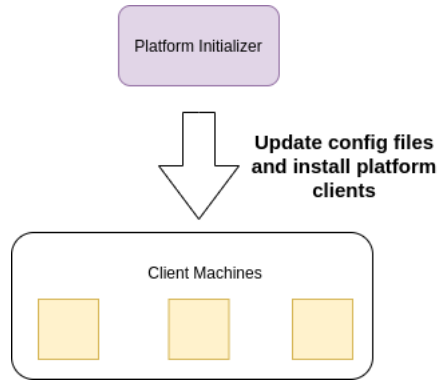


Figure 5: Installation of platform clients

- After installing the platform clients, the initializer decides which service to run on which machine, and deploys the service on the machine, with the help of bootstrapper.

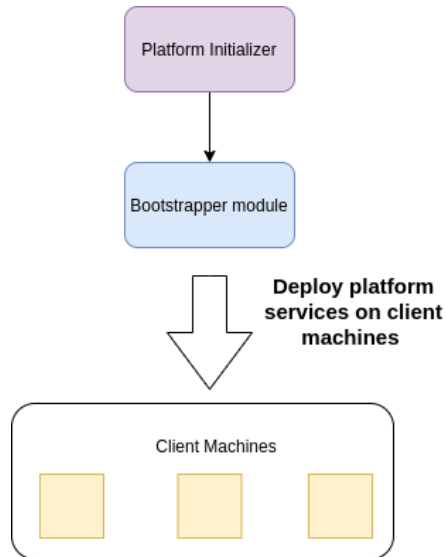


Figure 6: Deployment of platform components

- Various platform components that are deployed by the bootstrapper are as follows
 - **Application Manager and UI**
 - **Scheduler**
 - **Deployer**
 - **Node Manager**
 - **Sensor server**
 - **Fault tolerance and monitoring**
- The bootstrapper module makes use of docker to deploy services on the client machines. It builds the docker image on the master client system, uploads it to the central server, and downloads and runs on client machines.

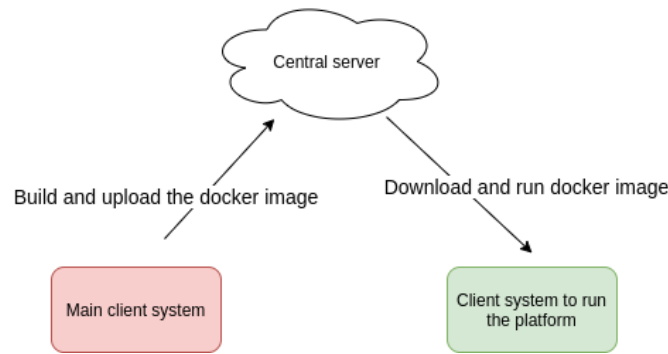


Figure 7: Deployment of platform components

8.2 Scheduler

- The scheduler will receive application instance id from application manager. Based on this id, scheduler will fetch the scheduling info from the app repository. The information fetched are:
 - **start date and time:** the date and time when the application should start running
 - **end date and time:** the date and time when application should be stopped
 - **duration:** the time duration for which the application should run
 - **repetition:** if the application should run periodically
 - **interval:** the time period after which the application should again start running
- The application instances are placed in the priority queue as per their start times
- The applications are popped out of priority queue as per their scheduled time and a notification is sent to the deployer of deployment manager to start the application.
- Information sent to deployment manager are:
 - **application instance id:** a unique id of the instance of the application to be run
 - **command(start/kill):** if the application is to be run, a start command is sent, otherwise if the application has finished its running time duration then kill command is sent to deployment manager.

- **type**: whether the application is to be run on a standalone machine or shared machine

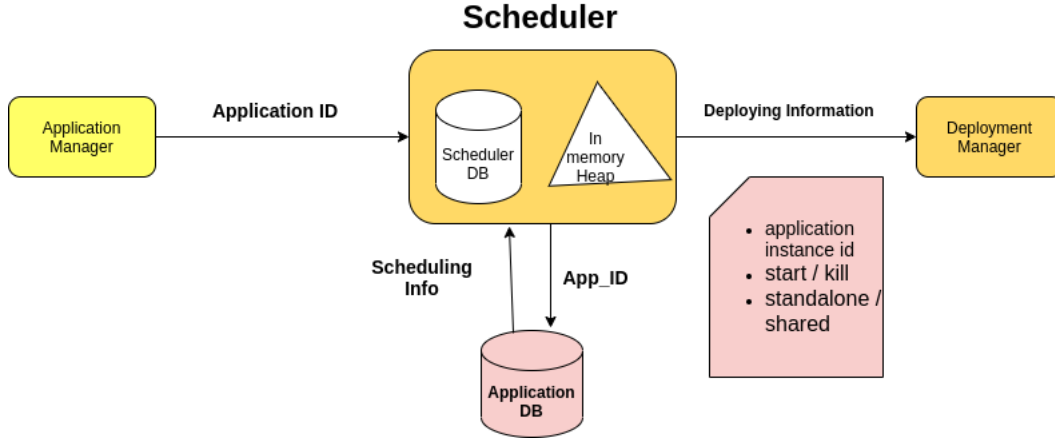


Figure 8: Scheduler Service

8.3 Deployer

- The deployment manager checks if the job requires a standalone environment, or a shared environment.
- Accordingly, it either demands a new node from the node manager, or tries to run the job on one of the existing active nodes, list of which it retrieves from the node manager.
- In case of shared environment jobs, the deployer makes use of Load Balancer, to select the node with least node.
- Once the job or algorithm is done running, and the node is no longer being used, it is returned to the node manager.
- If any application instance stops running, the fault tolerance manager sends a message to the deployer and deployer handles this case as follows :
 - Deployer checks the node status on which the application instance was running (stored in the shared database between node manager and deployment manager)
 - If the node is active and running, deployer will start the application on the same node.
 - Else, it will request the nodes from the node manager and follows the same procedure of deploying a new app instance.

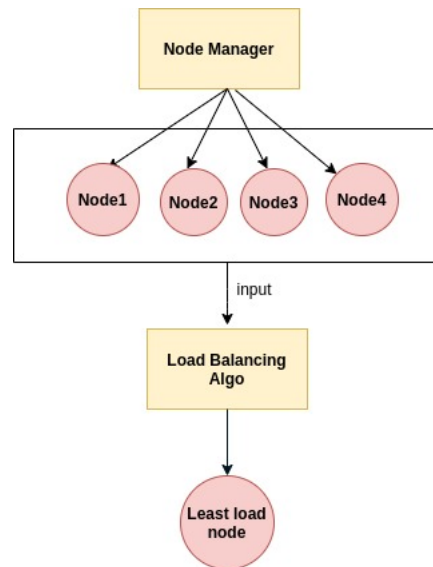
8.4 Load Balancer

- We use two metrics to calculate the load on the machine.
 - Load on the CPU
 - Percentage of RAM in use.
- These metrics are retrieved from the nodes using a library called paramiko.

- We use the harmonic mean of the above two values to calculate the load on a particular machine, and select the node with the least load.

$$Load = \frac{2 \times Load\ on\ CPU \times RAM\ usage}{Load\ on\ CPU + RAM\ Usage}$$

- Any one of the following load balancing algorithms can be used:
 - **Round Robin** - Round-robin load balancing is one of the simplest and most used load balancing algorithms. Client requests are distributed to application servers in rotation.
 - **Least Connection** - Least Connection load balancing is a dynamic load balancing algorithm where client requests are distributed to the application server with the least number of active connections at the time the client request is received.
 - **URL Hash** - URL Hash is a load balancing algorithm to distribute writes evenly across multiple sites and sends all reads to the site owning the object.
 - **Source IP hash** - Source IP hash load balancing algorithm that combines source and destination IP addresses of the client and server to generate a unique hash key. The key is used to allocate the client to a particular server.



9 Key Data structures

9.1 Platform initializer

- Map: Mapping of platform services with the client machine they are running on. The data is stored as files, and other configurational details, like database config, kafka config etc.

9.2 Scheduler

- Heap: In memory data structure for storing the scheduling info.
- Queue: For storing the incoming jobs, before they've been added to the heap.
- Scheduler DB: used to store scheduler state in order to recover scheduler after a failure

9.3 Deployer

- Map: For storing the mapping between job-ids and node-id

9.4 Load Balancer

- Heap: For storing the load on every node returned from the node manager, and to get the node with least load.

10 Lifecycle of the module

- All the components of this module will be started by platform initialiser.
- Each component has a heartbeat manager that keeps track of its health and monitors it.
- If a component is down/fails, heartbeat manager sends a signal to monitoring and fault tolerance module.
- Fault tolerance module then makes sure that the failed component resumes its work from its last saved state.

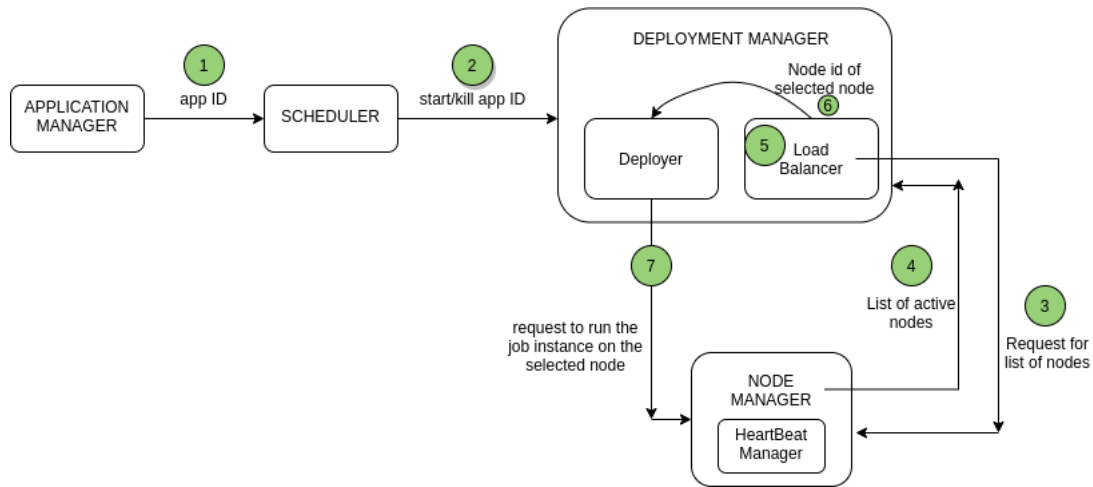


Figure 9: Scheduler and Deployment Manager Lifecycle

11 File Formats

Various config files created by platform initializer have following formats

- Services-config file: Tells about where each service is deployed, and the name of the machine.

```
{
  "monitoring": {
    "username": "nisarg",
    "ip": "52.172.183.104",
    "vm_name": "VM2"
  },
  "deployer": {
    "username": "nisarg",
    "ip": "52.172.144.235",
    "vm_name": "VM1"
  },
  "sensor_server": {
    "username": "nisarg",
    "ip": "52.172.144.235",
    "vm_name": "VM1"
  },
  "ui_appmanager": {
    "username": "nisarg",
    "ip": "52.172.144.235",
    "vm_name": "VM1"
  },
  "kafka": {
    "username": "nisarg",
    "ip": "52.172.144.235",
    "port": 9092,
    "vm_name": "VM1"
  }
}
```

Figure 10: Fields inside service config file

- DB config file: Tells about database that platform uses

```
{
  "host" : "karan.mysql.database.azure.com",
  "user" : "user",
  "password": "password",
  "database": "ias"
}
```

Figure 11: Fields inside DB config file

- Kafka config file: Tells where the kafka is deployed

```
{  
  "username": "nisarg",  
  "ip": "52.172.144.235",  
  "port": 9092,  
  "vm_name": "VM1"  
}
```

Figure 12: Fields inside kafka config file

- Repo config file: Tells where the app repo is stored

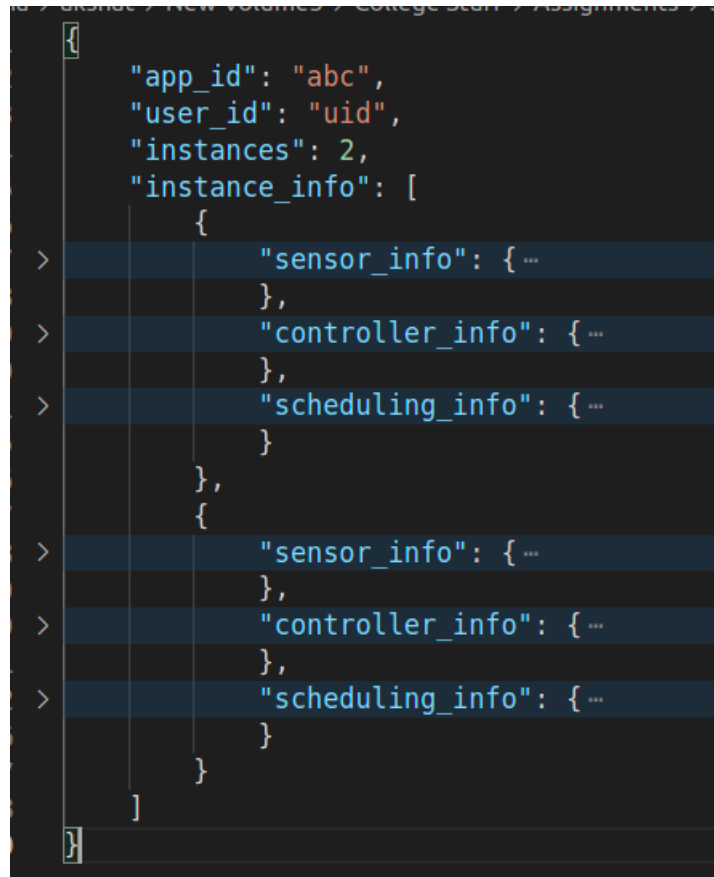
```
{  
  "c_str" : "url",  
  "s_name" : "ias"  
}
```

Figure 13: Fields inside repo config file

12 Registry and Repository

Registeries will be used in the following places:

- Scheduler: Here a repository will be used to store information about various application instances, which is provided by the configurer.



```
[
  {
    "app_id": "abc",
    "user_id": "uid",
    "instances": 2,
    "instance_info": [
      {
        "sensor_info": { ...
      },
      {
        "controller_info": { ...
      },
      {
        "scheduling_info": { ...
      }
    ],
    {
      "sensor_info": { ...
    },
    {
      "controller_info": { ...
    },
    {
      "scheduling_info": { ...
    }
  }
]
```

Figure 14: Information stored at scheduler

- Deployment Manager: Here a registry will be used to store runtime information like application instance id and the node id on which this application instance is running.



Figure 15: Information stored at deployer

13 Interaction with other components

Figure 2 shows the interaction between various components.

13.1 Interaction between Application Manager and Scheduler

- Application manager on receiving request for deployment from configurer sends app instance id to scheduler, in order to schedule the application
- Scheduler will retrieve the scheduling infomation of the application instance from the app DB with the help of app instance id.
- With the help of the retrieved scheduling information, scheduler will place the app instance at appropriate place in its priority queue
- Interaction between app manager and scheduler takes place through a single one-to-one kafka topic.

13.2 Interaction between Scheduler and Deployment Manager

- When the scheduling time of an app instance is reached, it is popped out of priority queue.
- Now scheduler notifies the deployer to start the app instance by sending its app instance id and a command to start or kill the app instance.
- The communication between scheduler and deployer takes place through a single one-to-one kafka topic.

13.3 Interaction between Deployment Manager and Sensor Manager

- The deployer part of Deployment Manager will contact the Sensor Manager in order to get a list of sensor nodes of the sensor types required to run the application.
- After getting this list from the Sensor Manager, the deployer will bind the sensors used in application with the actual sensor instances.

13.4 Interaction between Deployment Manager and Node Manager

- The Load balancer part of Deployment Manager will contact the Node Manager in order to get a list of active nodes.
- After getting this list from the Node Manager, the Load Balancer will calculate the load of each node using Load Balancing Algorithm.
- A node with lowest load will be selected.

- The deployer will contact the Node Manager with the selected node on which the application with application id is to be run.

13.5 Interaction between Platform Initialiser and Scheduler/Deployer

- Platform initialiser will initialise the scheduler and the deployer to get them up and running.

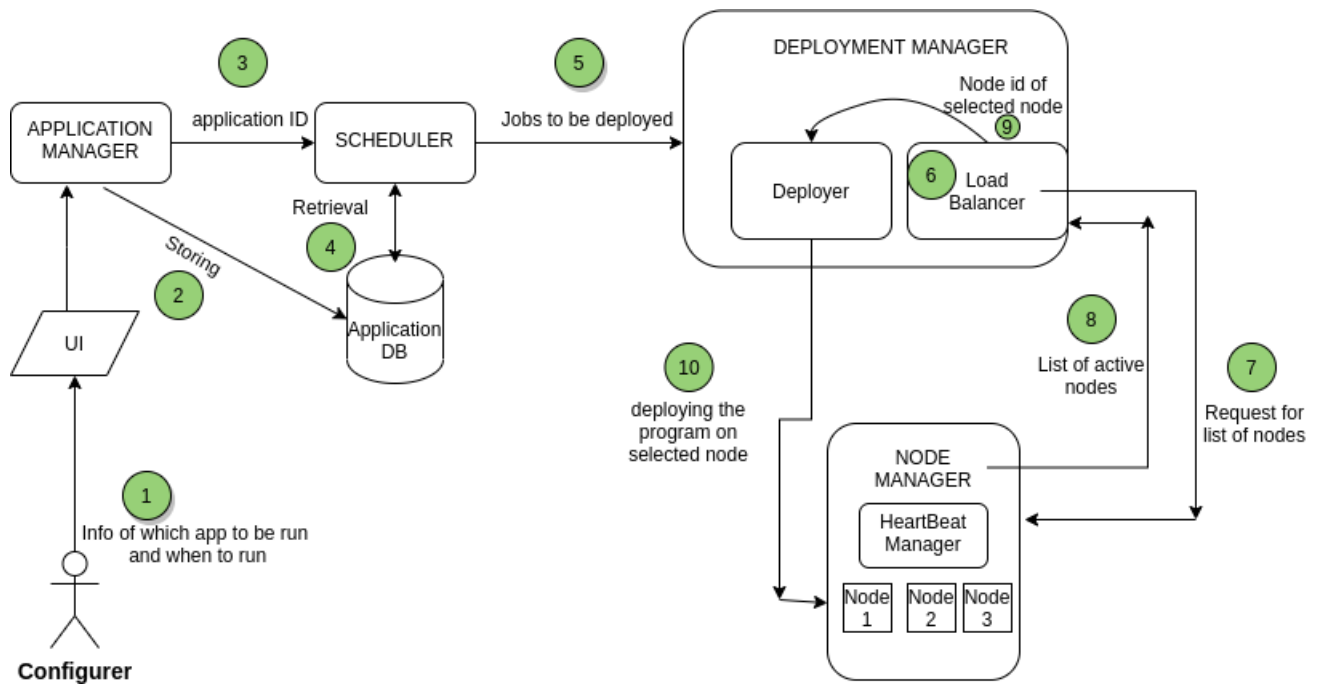


Figure 16: Interactions among different components

14 Persistence

14.1 Scheduler

- Scheduling info along with the currently running job ids need to be saved so that whenever the scheduler fails, it can be restarted without causing any problems in the scheduling of applications.
- A copy of the priority queue of applications also needs to be saved in the database.
- To maintain persistence in Scheduler, a scheduler DB is used.

14.2 Deployment Manager

- Node id and application instance ids of the running jobs need to be saved in the database.
- Status of the nodes is also saved to make sure that app is not deployed on an inactive node.

15 Test Cases

15.1 Test cases to test the team's modules

15.1.1 Scheduler

- On receiving request from app manager, the scheduler should fetch scheduling information from app DB and should validate it.
- If the scheduling info is correct, the app instance should be placed at the appropriate position in the job queue.
- Scheduler should send all the relevant info to the deployer like - app instance id, command to start/kill, whether the app instance be run on standalone or shared machine.
- When the scheduler is restarted after a failure, it should correctly re-schedule all the jobs.

15.1.2 Deployer

- Deployer on receiving a job from the scheduler should correctly retrieve the config files from the repository
- On retrieving the config files, it should analyse it and retrieve a node from the load balancer.
- On receiving the node, it should start the job on that node, and return the job id to the scheduler, node to the node manager, and job-node mapping to the application manager.
- On receiving the kill job information from the scheduler, it should retrieve the required node from the load balancer, and kill the particular job.
- On receiving the node down/app down message, the deployer should be able to start the application again on the running node.

15.1.3 Load Balancer

- On receiving the request from the deployer, it should retrieve a list of all available nodes from the node manager.
- On retrieving the list of available nodes, it should see if the node required is for shared environment, or standalone environment.
- If the shared environment is required, it should correctly calculate the load on each of the node, and return the appropriate node to the deployer.