IAS

Team Design

Group - 3 Team - 2

Service lifecycle manager,

Server lifecycle manager

Deployment Manager

Submitted by:

Shrayans Jain
(2019201086)

Shubham Agrawal
(2019201085)

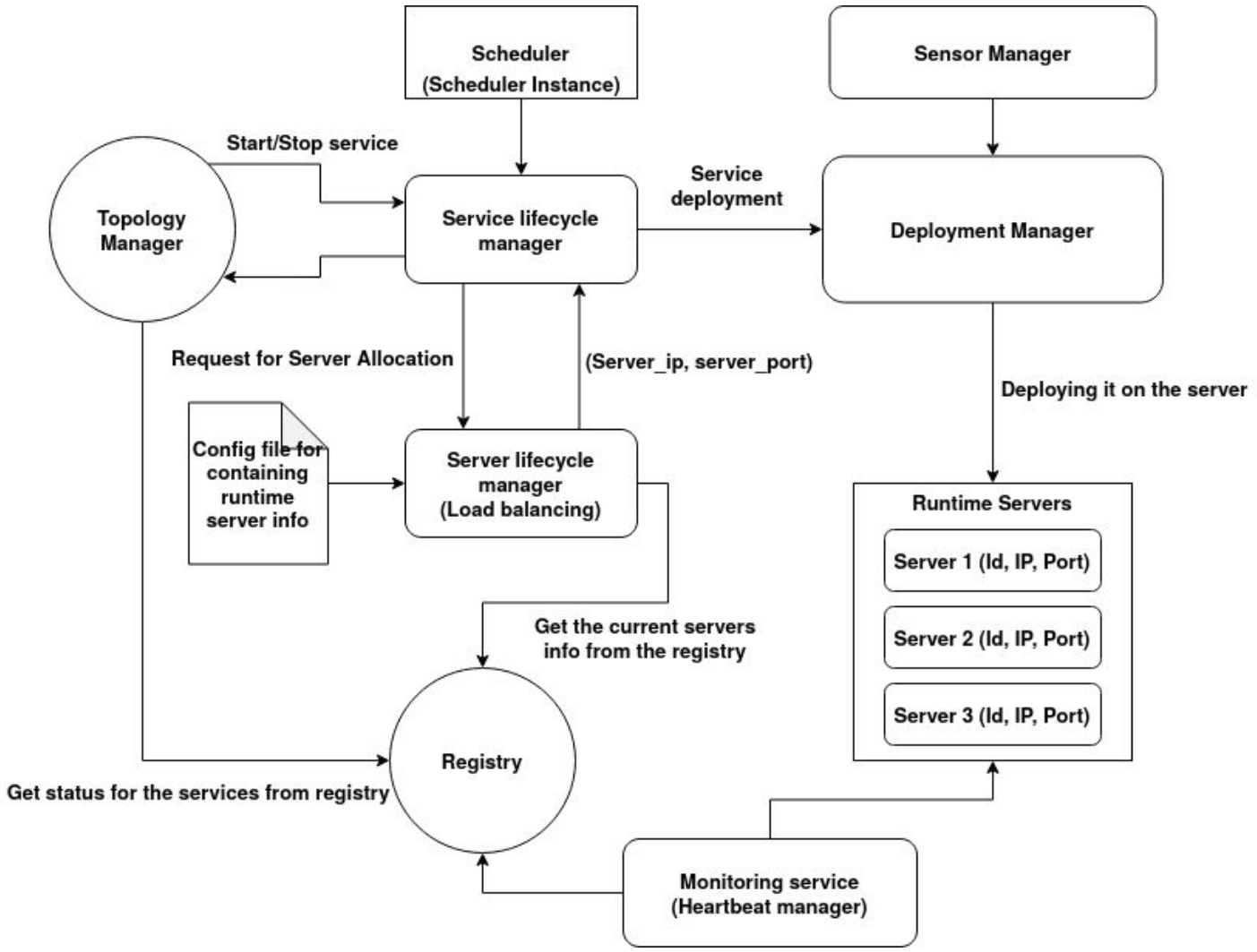
1. Introduction of Project:-

Definition: A Distributed Platform which allows a user to build smart city applications, code their own algorithm or use from a set of algorithms which platform offers and deploy applications. Platform deals with different types of sensors and algorithms.

Scope: Our platform provides a set of independent services that the app developer can use for app development with his own custom code updates. The platform provides various independent services like Security service (Authentication and Authorization), Build and Deployment capabilities , Logging and monitoring, Notification and Actions Service, Algorithm Runtime, Load balancing, Resource management, Scheduling service and repository to store data.

2. Design

Big picture



❖ Server Lifecycle Manager

➢ Server Life cycle manager manages the life cycle of the running server. It has a list of available ip and port and also it contains a Load balancer as a sub component. Whenever a service life cycle manager communicates with Server LC manager, it sends an ip and port of node that have a minimum load to service life cycle. Server lifecycle manager will get the server's status from the registry. Monitoring service which has the heartbeat manager (to get the information about the available servers whether alive or not), It will update the registry.

➢ It will check which node is having less load which is updated in the registry by monitoring service and After applying the load balancing algorithm, it will give the response back to the service lifecycle manager. The response will contain the assigned Server Ip and Server port.

➢ State of each node is saved, so if any node fails then according to the last saved state it is resumed.

❖ Service Lifecycle manager

➢ Service lifecycle manager will receive details of service in json format from the scheduler. It will then communicate with the server LC manager to get the IP/Port of server on which it will run it. Topology manager will give instruction of start/stop for any service to service LC manager.

➢ After that it will give details to the deployment module to deploy the service on the runtime server.

➢ Functionalities of this module:

- Run a Service instance
- Kill all Service instances running on single machine input (IP, username)
- Kill a single Service instance input (ipaddress, hostname, serviceid)

❖ Deployment manager

➢ **Deployment manage**r will bind the sensors to the algorithm instance. When the sensors are binded according to the geolocation provided by the user and then the algorithm instance can now access the stream of data related to that sensor and then run it on the server chosen by the server runtime manager.

➢ When the service is deployed, the deployment manager ping service lifecycle manager with the ack that “Service Deployed successfully”.

➢ All the communication is done by kafka and the sensor will produce the data on the sensor topic.

➢ Then finally that algorithm is run on that node and pushes its output on the message queue.

➢ **Action server** will perform the necessary action(Email, sms, etc).

3. How service will run?

- **Request Manager** will receive requests from users via application interface. Now the request manager will send a token (userID, application name, service name) to the **service lifecycle manager** . Now the service lifecycle manager will send these details to the authentication server.
- Now the **authentication server** will check whether this user can use this application's service or not. For this authentication server will look into the registry and will respond back to the service lifecycle which will eventually respond back to the request manager.
- Now in case of a valid user, the request manager will create a token (containing algorithm name for that service, priority if any, dependencies if any, sensor type and location for that service , action, etc..) related to service from algorithm repository.
- Now the request manager will send these details to the **scheduler**. Now the scheduler will solve dependencies and will schedule the algorithm's instance accordingly. Now what to schedule along with the token will be sent to the server lifecycle.
- Now the server life cycle will assign a server , where this service's instance will run. It will add details like port number of server ,its ip address to the token. Now it will pass this token to the service lifecycle manager.
- Now the service lifecycle manager will send this token to the **deployment manager** .
- Now it's the deployment manager's job to parse the token and find details like sensor type and location. After getting these results deployment manager will ask sensor manager to provide sensor id's list , which deployment manager wants. For this deployment manager will look into the **sensor repository.**
- Deployment manager after receiving a sensor id list will bind this list with a token and will send this token to the **runtime server** (i.e the one whose details were present in the token received by the service lifecycle).
- Now the runtime server will read the token. It will find algorithm names and sensor ids for that algorithm. It will ask for data from those **sensor's topic** . These sensor's topic will be managed by a kafka broker.

- ❏ These topics will send data to the runtime server. Now the runtime server will run the algorithm with data as runtime argument .
- ❏ The output generated by running that algorithm will be sent to the **action server** along with a token.
- ❏ Action server will read output sent by runtime server. It will also read a token to know what action to take for that output and for that service.
- ❏ Action server will check the mapping of this action in the **action repository** to know which algorithm to run to serve this action.
- ❏ Now the action server will run an action algorithm with output received from the runtime server as runtime arguments and the user will get the result.
- ❏ There is another job which will be done by an action server. Based on the output, Action server will check for thresholds associated with service. After knowing this , if there is any discrepancy then it will inform the control unit to turn off/on some sensors.

4. Technologies used

- NFS: Network file sharing
- Bash shell scripts for automation
- Python framework is used to develop a platform
- MongoDB

5. Load Balancing

Here the server is assigned to a particular task(or algorithm) using a load balancing algorithm. Servers are assigned dynamically with respect to the parameters of selecting a server:

1. **CPU Utilization**

2. **Memory Utilization**

3. When we want both then we take the harmonic mean of CPU and Memory Utilization.

$$par = 2/(1/cpu + 1/mm)$$

We can use any of the following **Load Balancing algorithms:**

• **Round Robin** – Requests are distributed across the group of servers sequentially.

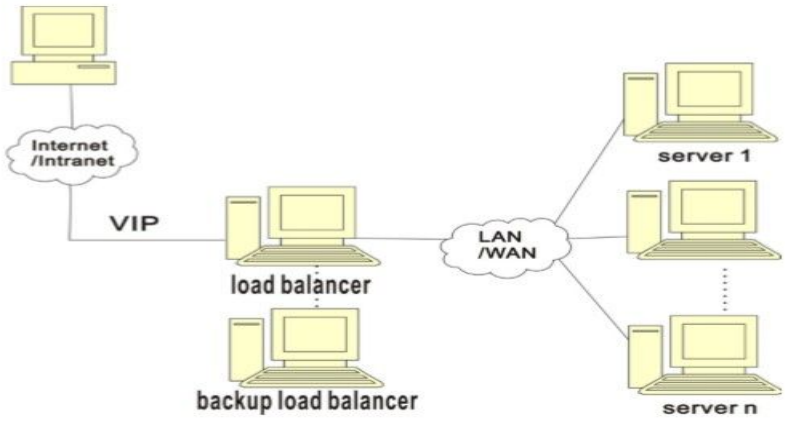
• **Least Connections** – A new request is sent to the server with the fewest current connections to clients. The relative computing capacity of each server is factored into determining which one has the least connections.

• **Least Time** – Sends requests to the server selected by a formula that combines the fastest response time and fewest active connections. Exclusive to NGINX Plus.

• **Hash** – Distributes requests based on a key you define, such as the client IP address or the request URL. NGINX Plus can optionally apply a consistent hash to minimize redistribution of loads if the set of upstream servers changes.

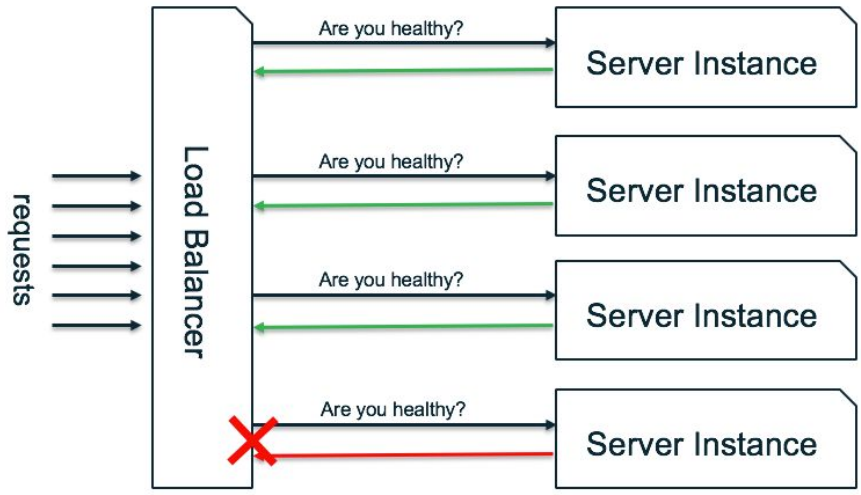
• **IP Hash** – The IP address of the client is used to determine which server receives the request.

• **Random with Two Choices** – Picks two servers at random and sends the request to the one that is selected by then applying the Least Connections algorithm (or for NGINX Plus the Least Time algorithm, if so configured).



Monitoring Service

- A service instance will continuously send a signal to monitoring service.
- Monitoring service will update the machine stats in registry.
- Health checker will get info about running instances from the same registry.
- It will check all instance data.
- If any instance is not functioning properly it will handle it accordingly.
- If an instance gets killed somehow it will schedule it again.



6. Metadata

For Runtime Server:

```
Metadata of RunTime Server:

{
    {
        "Action_id":
        "User_id":
        "Output":
        "Node_id":
        "Execution Time":
        "Algo_id":
    }
}
```

Metadata for Service Life cycle manager:

```
{
    userd_id:'u1_user'
    app_id:'a1'
    start_time:
    end_time:
    duration:40
    dependency:
    action:'none'
    notification:{
        type:'email'
        body:'"lala"'
    }
    location:'nilgiri'
    service_path:'a/a1'

}
```

7.Persistency

- Our platform is persistent because of its high fault tolerance, whenever a node is down, then the services running on that node are shifted to some other node.
- The platform will be persistent with the use of NFS, databases and services that maintain the state of the platform.
- In case a node crashes, the system brings it back in the last state on some other node.
- In case any system or application service or running model crashes then it will be brought back by healthCheck service.

8. Test Cases

8.1 For Server Lifecycle Manager:

1. All the Servers are busy

   When all the servers are busy, then which server to be selected to run the service.

2. Requested server available or not

   Requested service node is not available, then the appropriate error message should be sent.

3. No server is active

   Suppose all the servers are not active, then the first task is to up the servers and then run the service.

4. Different Load balancing algorithms

   Try for different load balancing algorithms like:

- Round Robin
- Least Connections
- Least Time
- Hash
- IP Hash
- Random with Two Choices

8.2 For Service Lifecycle Manager :

1. Service is running or not

   To check whether the node is active or not

2. Running Service is crashed (or killed) unexpectedly

   Kill the service manually via a command, So our service lifecycle should run that service on the other server.

3. Running server is crashed (or killed) unexpectedly

   Manually stop the server, So our service lifecycle manager should run all services of that server on some other server. (Service state is saved in the registry).

8.3 For Deployment Manager

1. Server assigned by server lifecycle manager is not active

   Error message should be sent to the service lifecycle manager.

2. Error While binding the sensor to the algorithm instance

   Report error message to the sensor manager.

3. Deployed service server goes down

   Send the message to the deployment manager and the deployment manager will report it to the service lifecycle manager.