DESIGN SPECIFICATION FOR SERVERLESS PLATFORM

IIIT-H IAS Project Group

April 29, 2017

Submitted By-
Hemant, Vini, Kanishtha, Riddhi, Siddhartha,
Dhawnit, Vikas, Ishan,Jayant, Shikha

TABLE OF CONTENTS

1. INTRODUCTION

A serverless architecture is basically a new design pattern that needs less effort in building your own server components.We have referred AWS lambda implementation for the development of the platform. This is an application platform as a service (aPaas) platform whereby an app developer can build and deploy app as a set of independent services that run in response to events and auto-scale for app user. Our platform provides a set of independent services that the app developer can use for app development with his own custom code updations. The platform provides various independent services like Security service(Authentication and Authorization), Data Service(Configurable Service), File Service, email service… etc to the application developer. In-order to deploy an application, the developer has to break his application in multiple independent stateless and autonomous functions. Platform implicitly connects every service(independent function provided by the user) to an input queue and an output queue of respective services. Automatic load balancing (spawning more instances of a service) is also provided and is based on the incoming requests for a service. Load is distributed on number of machines(scaling up and down of machines on the basis of the load on the platform). App developer has to simply write plain java code and can use our class A services' functions whenever needed. He just has to bundle up all independent .java code files in a Jar and deploy on the platform along with some config files. Config files contains the configuration for configurable services like security service and data Service.

2. FUNCTIONAL OVERVIEW

Our platform provides the following independent/autonomous services to the app developer who will be using our platform:

- **API gateway :** API gateway is the single endpoint of our platform. All the requests from outside the platform hit the gateway, which are then directed to the appropriate handlers and response is sent back to the user.

- **Authentication service :** Authentication service checks for the valid user using the pre-build LDAP based authentication mechanism used in platform and send the authentication token back to the user for future use.

- **Authorization service :** Authorization service checks whether the authenticated user is authorised for a requested service or not.If the user is authorised then only the request is carried forward.

- **Internal message queues :** It provides an asynchronous mechanism for inter-communication between services by a platform transparent addressing mechanism ("SvcQueueServiceName").

- **Load balancing :** It ensures scaling up and down of platform based on the load on the

platform.It does scaling both at machine level and service level and thus guarantees responsiveness.

- **Logging :** Provide an easy to use, general purpose logging system that keeps track of all events associated with a particular service.It all provides logs for the whole platform as well.

- **E-mail service :** All services may make use of the inbuilt email service to send emails.

- **File system service :** The service provides functions for creation, deletion, updation of transient files.

- **Data service :** It is configurable service.It provides an abstraction of a general purpose database frontend that fits the data as a service model.

- **Service deployment frontend :** The frontend is a single endpoint for the deployment of services and its management. The user is not concerned with background activities involved in hosting and running the service.

3 TEST CASES OF THE PLATFORM

A) API Gateway:

1. Wrong API end-point.
   Input: Enter wrong API end-point
   Expected output: Appropriate error message

2. Unauthorized user accessing our API.
   Invocation method : Request to api from unauthorized user
   Expected output : Request not granted.

3. Authorized user accessing end-point
   Invocation method: Request to api from authorized user
   Expected output: Request granted

4. Authenticated user but unauthorized service.
   Invocation method: Request to api from authorized user for a service which they have no permissions.
   Expected output: Request not granted.

5. API Gateway down
   Invocation method: Request to api while api gateway is down.
   Expected output: Proper error message.

6. Multiple requests at the API
   Invocation method: Request to api from 2-3 user simultaneously.
   Expected output: All Request handled successfully.

7. Stress testing.
   Invocation method: Request to api from large number(1000-2000) of users.
   Expected output: All request handled successfully.

B) Load Balancer:

1. Requested Service is not loaded.
   Invocation method: Request a service that has not been loaded.
   Expected output: Service is loaded successfully.

2. Requested Service is not available.
   Invocation method: Request a service which is not available in the platform.
   Expected output: Appropriate error message generated.

3. All machines are fully loaded.
   Invocation method: Request service when all server are fully loaded/busy.
   Expected output: Wait for servers to response.

4. Requested Service is already loaded.
   Invocation method: Request a service which is loade.
   Expected output: New instance of the service is created and requests are processed and response is returned.

5. No server is up and running.
   Invocation method: Request a service first time.
   Expected output: Server is loaded and instance of service is executed.

C) Data Services

1. Requested data is available.
   Invocation method: Request for the available data.
   Expected output: The request will be served properly.

2. Requested data does not exist.
   Invocation method: Request for data that does not exist.
   Expected output: The request can’t be completed as data is not available.

3. Requested table is not available.
   Invocation method: Request for data in non-existing table.
   Expected output: The requested data can’t be returned.

4. Unauthorized data access.
   Invocation method: Data is requested by user who is not having access to that data.
   Expected output: The data will not be returned.

5. Database connection errors.
   Invocation method: The data is requested from the database but connection can’t be established.
   Expected output: The data will not be returned.

6. SQL injection.
   Invocation method: request for non authorized data by appending trivial condition,

for ex (Table.name=”vikas”’or’1’=’1)
Expected output: the data corresponding to valid name should be appear

7. Requested data updation.
Invocation method: Request for data updation.
Expected output: data will be updated in the database successfully.

D) File Services

1. Authorized file access.
Invocation method: The user requests for an authorized file.
Expected output: The file will be served to the user successfully.

2.Unauthorized file access.
Invocation method:The user requests for an unauthorized file.
Expected output: The file will not be served.

3.File permission errors.
Invocation method: The user attempts to modify a file having restricted permissions set.
Expected output: The file will not be modified.

4.Invalid File request.
Invocation method:The user requests for an invalid file.
Expected output: The file can’t be served to the user.

5.Valid Upload File request.
Invocation method: The user makes a valid request to upload a file.
Expected output: The file will be uploaded successfully.

6.Valid Create File request.
Invocation method: The user makes a valid request to create a file.
Expected output: The file will be created successfully.

7.Valid Delete File request.
Invocation method: The user makes a valid request to delete a file.
Expected output: The file will be deleted successfully.

8.Valid Update File request.
Invocation method: The user makes a valid request to update a file.
Expected output: The file will be updated successfully.

9.Valid Read file request.
Invocation method: The user makes a valid request to read a file.
Expected output: The file will be read successfully.

E)Security Service

1. Authorization:

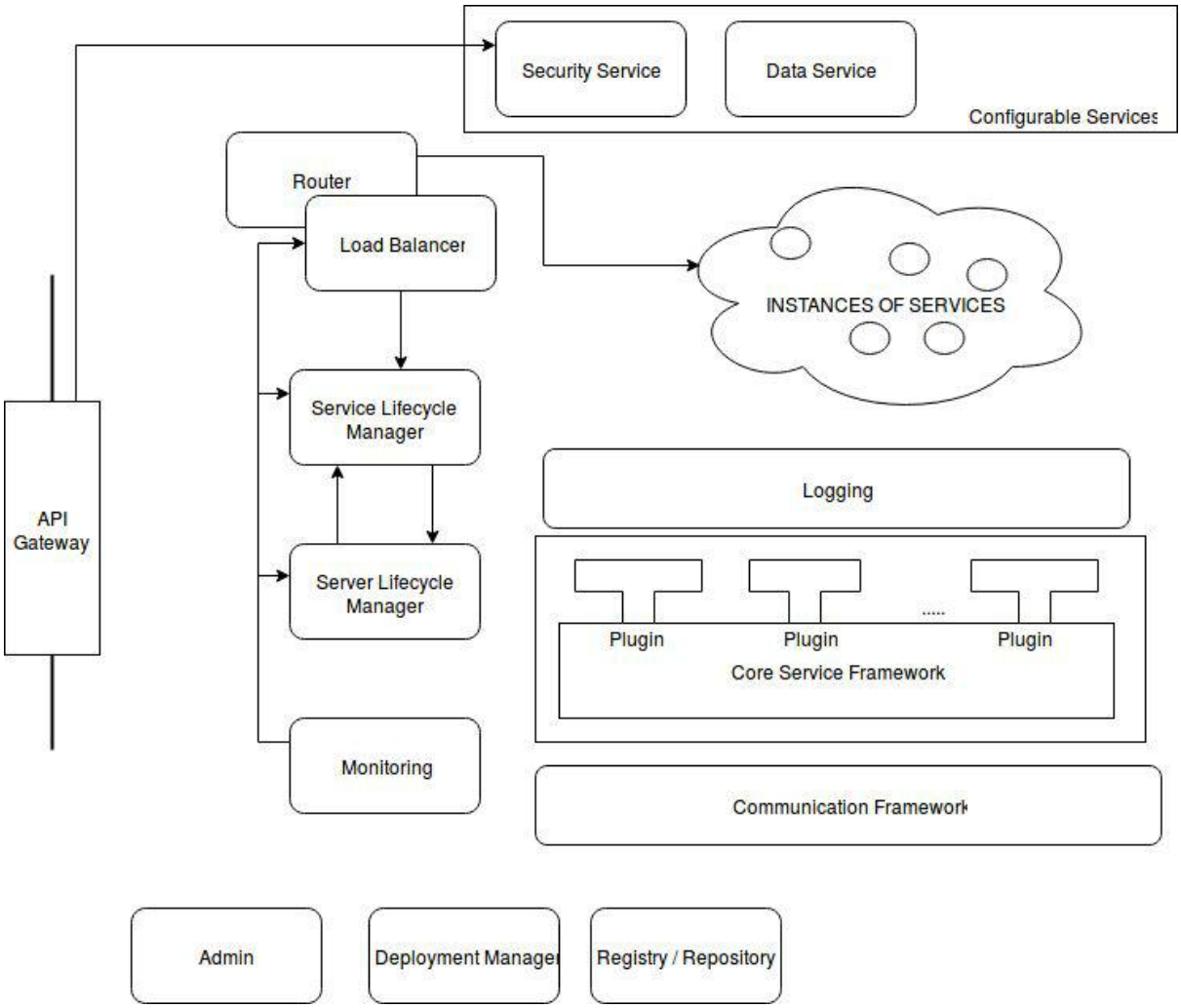
a. Invalid user.
b. Invalid password.
c. Valid user and valid password.

2. Authentication:

a. Request for valid service.
b. Request for invalid service.
c. Valid token exist.
d. Invalid token for service.

4. SOLUTION DESIGN CONSIDERATIONS

4.1 High level design of the project:



4.2 Environment to be used

- The platform should be installable on any 64-bit linux machine/distributed machines.
- The internal messaging will be done through RabbitMQ and thus the RabbitMQ Server will be installed and will be a part of the environment.
- Vagrant and VirtualBox.

4.3 Technologies to be used

- J2EE should be primarily used for the development of the platform.
- Bash Shell scripts can be used to make installation/configuration automated.
- RabbitMQ must be used for MQ implementation, LDAP for security, MySQL as database,NFS as common File System.
- Vagrant from Hashicorp for provisioning virtual machines.
- Virtualbox is used as an hypervisor and provisioner for creating machines

4.4 Overall system flow & interactions

- User will hit the URI which will be accepted at the API gateway. API gateway is a multi-threaded servlet running on Tomcat.

- API gateway will authenticate the user (Based on username and password), get an authentication-token back from Authentication service, then all the further user requests will contain authentication-token in them. And based on the authentication-token and service request the user will be checked for authorization.If the user is authorized the request is pushed into the MQ of the Load Balancer. API gateway will also have a MQ on which it will receive the output from Load Balancer.

- Load Balancer will have two components - Routing and Load Balancing. The Load Balancing component will gather machine as well service level stats from Monitoring module.And based on the machine level stats it will communicate with the Server LifeCycle Manager to provision a new machine or kill a machine. And based on service level stats it communicates with Service LifeCycle Manager to invoke a new instance of the service. The Routing component based on the service request will push the request message into the respective service queue.

- Service LifeCycle manager will receive request from Load Balancer to start a new service instance along with the machine domain name on which the service instance should be started.

- Server life cycle manager will analyse the machine load (from machine level stats from monitoring module) and will assign a machine to be used to invoke another instance of the service to Load Balancer.

- Each machine on the network will have a Machine Agent running on it.The task of the Machine Agent will be to gather service level as well as machine level stats(ram and cpu usage) and send it to Monitoring Module.Machine Agent can also create or kill a service instance(docker container running service instance).

- There will be Service Listener running in a Docker Container. This Service Listener will receive the request from its request queue, parse the json request received and then call the required function with the obtained parameters. Then the response given by that java function will be received by the service listener and will put it in its response queue from where it will reach the user.

- **Utilities -**
  - ➢ **Admin Utility** - The admin module will gather machine as well service level stats from the Monitoring module and display then to the Admin. The Admin module will also have LDAP control. It can add / delete users , change user permissions.
  - ➢ **Logging Utility** -The Logging Utility will create logs at two levels - platform level and service level.A different log file will be created for each service.

- **Deployment Manager** - It is an independent module. It allows the application developer to upload a zip containing three folders - files (containing config/xml files) ,lib (jars needed) and src (containing .java files) into common nfs directory. Now along with the necessary .java files like CoreService Framework, Communication With MQ is compiled.If compilation is successful then the .class files are placed into the folder var/serverless/Services/servicename

4.5 Approach for Common Services framework (to be used by system, configured & developed services)

● Every service (system, configured & developed services) will inherit a core service class. This class will abstract out connectivity details among the services and will store and manage the service configuration details. This will contain methods to send request messages to other services; putting them into respective MQs. It will extract the configuration details corresponding to the user and use them in system service calls.
● Since every service will inherit this class, hence all will automatically get this Functionality.

4.6 Approach for communication & connectivity

● RabbitMQ will be the primary communication channel between every system Component.

4.7 Registry and repository

● Registry will contain the stats of every service (what instances are running on what machines, binary files location of each service in the common file system etc.)
● Repository will contain one directory per user which will in-turn contain one directory per service and will contain all the version-revisions of that service.

4.8 Load Balancing

- Load Balancer will have two components - Routing and Load Balancing.
- The Load Balancing component will gather machine as well service level stats from Monitoring module.And based on the machine level stats it will communicate with the Server LifeCycle Manager to provision a new machine or kill a machine(scaling up and down of machines). And based on service level stats it communicates with Service LifeCycle Manager to invoke a new instance of the service in a docker container.
- The Routing component of the load Balancer is multithreaded (Thread pool is used to handle multiple requests simultaneously). And based on the service request it will push the request message into the respective service queue.

4.9 Wire and file formats

● Each message sent to a MQ will have an authentication token in the msg header. Message body will contain the svc name, function name and its parameters along with their data types. We will also have an identifier(correlation id) for every message and will be included as key of that message in the MQ.

5. USER'S VIEW OF THE SYSTEM

5.1 Executable :

The application builder can download the zip file to setup the platform. The zip file will contain all the dependencies (Tomcat, Rabbit MQ, NFS, MySQL, Kerberos/LDAP etc.) needed for the platform to run.
The application builder must install this zip to setup the platform on his private premise. README contained in the zip will list down all the installation steps needed.

5.2 Setup :

After all dependencies are installed, they must be setup for being able to use them :-

- Tomcat must be setup in a directory where *tomcat* user has the permissions to access. Must change the .ini and config files for Tomcat to work. Must configure the port for Tomcat.
- Rabbit MQ is to be installed on a dedicated machine which should act as the MQ broker. The ip and port of the machine where this broker listens to must be saved to be used by all other machines in the network.
- Database like MySQL must be installed along with an user. The endpoint to where and how to access the database must be defined/stored in a configuration file.
- API Gateway must be setup to serve any HTTP requests. API gateway must be configured to have a single endpoint for the whole platform. All available services would be given a sub-domain of this endpoint.
- LDAP/Kerberos service must be setup to be used against Security service.
- For Repository and registry we must have a Network FIle System, which all machines can access. Setup of this common file system should be done.

5.3 Configure :

Configuration of the platform according to dependencies setup in the above phase must be done here.

- Configuration of how to connect to LDAP/Kerberos server (ip:port) for the configuration of security service.
- Configuration of how Data service will connect to the DB.
- Similarly how email service will connect to the email server.
- Configuration of common services like File service - which directory (NFS or local) the file service will use as its working directory; rabbitMq - configuration of ip and port where the MQ broker is running.
- Configuration of machines which will be used for scaling up and down of the services' instances. They should have access to a common directory having executables of all the services and a bootstrap script which will setup the machine to be used in a distributed fashion.

5.4 Develop :

Application developer can now develop his own services using the base services provided by the platform.

5.5 Package:

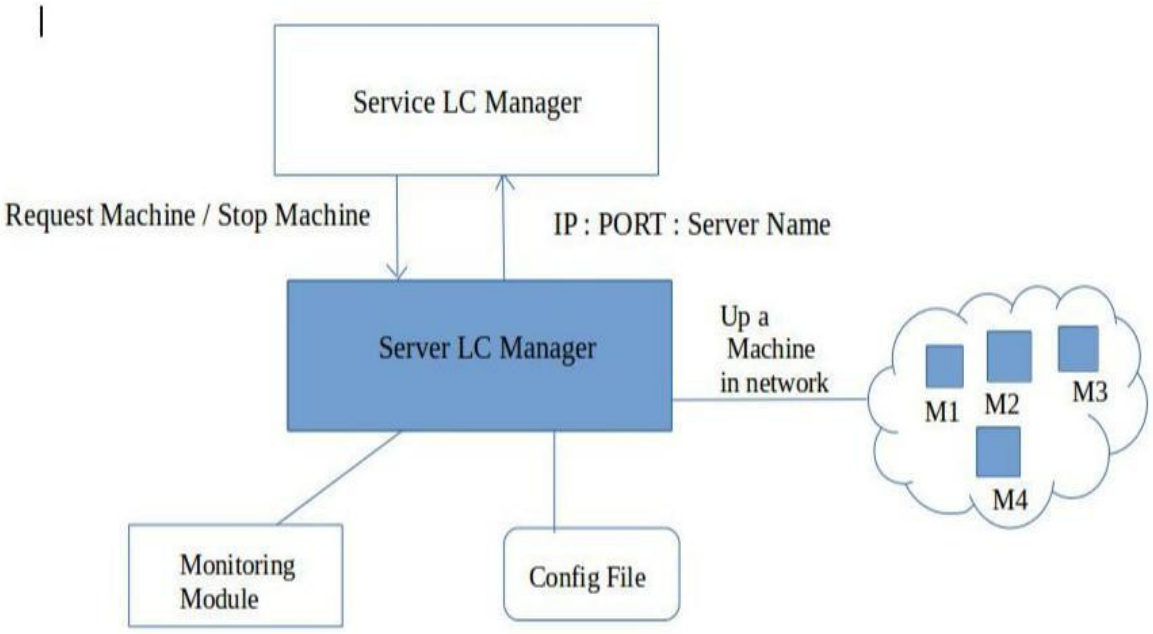
All java class files for services should be bundled up in a jar file. Then alonwith this jar file, we also include all the dependencies and config files needed for the code to run in the zipped folder.

5.6 Deploy :

The JAR should be uploaded at the deployment portal which will return an URI which will be the end-point to use the application.

6. THE MODULES

6.1 Server Lifecycle Manager :



Main functionalities of this module are as follows :

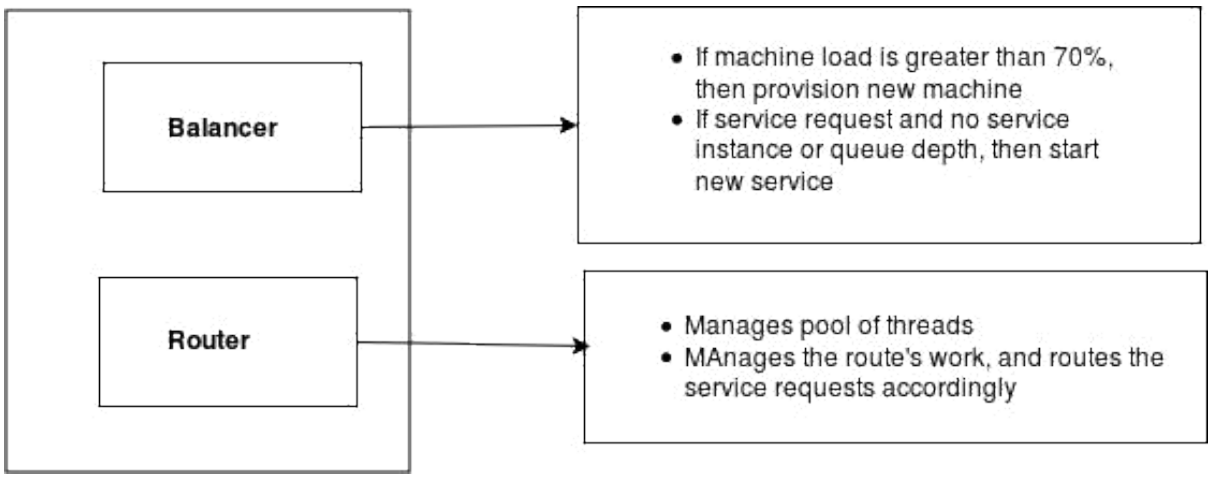
1 . **StartMachine ()**

- Get stats from Monitoring Module
- Get ip and hostname from config file
- Login to machine through Ssh using ip address and hostname
- Install Docker (if required)
- Initialization and BookKeeping
- Run Agent
- Make Agent send its stats to Monitoring Module

2.**StopMachine ()**

- Shutdown the machine
- Thennotify the Load Balancer.

6.2 Load Balancer :



Main functionalities of this module are as follows :

1. **Routing –** Place the message in respective Service MQ

2. **Monitoring –**

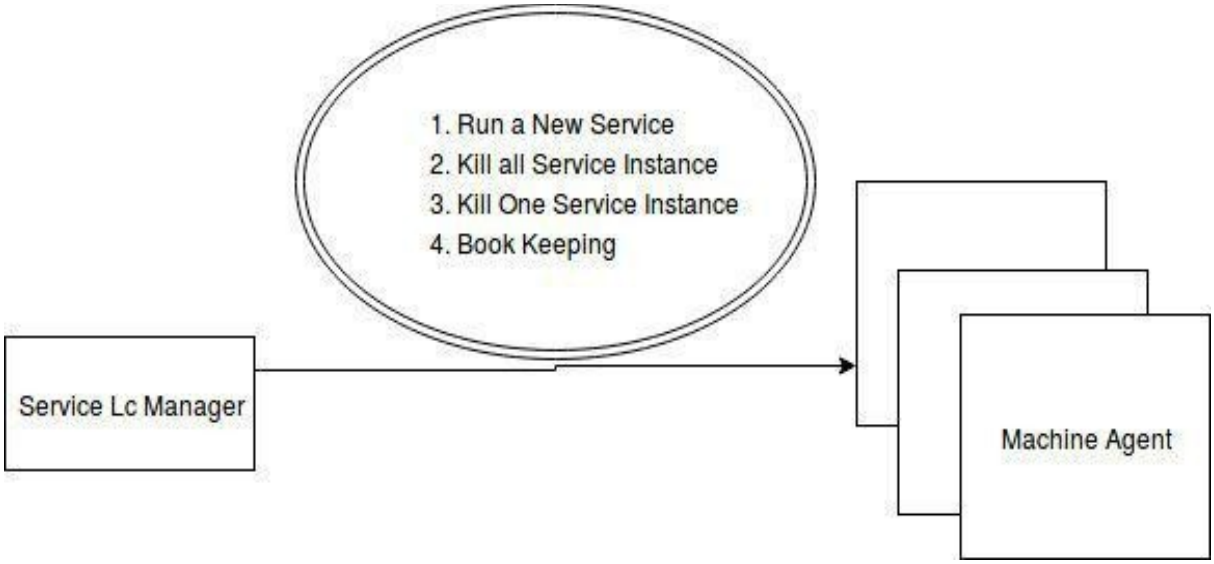
- Gather stats from Monitoring Module.

- And using Rules (Memory Threshold,CPU Threshold,etc) send request to Service LifeCycle Manager or Server LifeCycle Manager.

- The request for (Service LC Manager) can be -
  1. Kill a Service instance.
  2. Kill all Services on a particular machine.
  3. Start a new Service instance.

- The request for (Server LifeCycle Manager) can be -
  1. Start Machine
  2. Stop Machine

6.3 Service LifeCycle Manager :



Main functionalities of this module are as follows :

1.Run a Service instance:

Gather stats from monitoring module and based on the stats, start a service in a docker container on that machine.

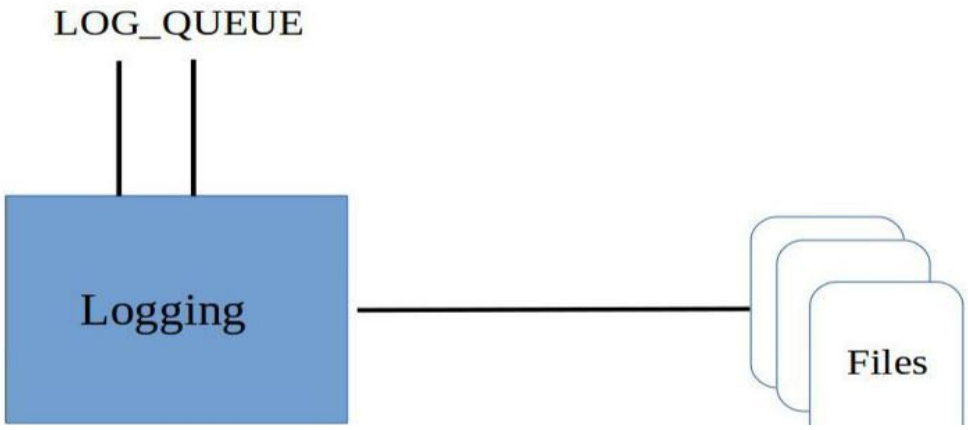
2.Kill all Service instances running on single machine input:(ipadress, hostname)

Get info about services running on that machine from registry. Then kill those service containers.

3.Kill a single Service instance input:(ipadress, hostname, serviceid)

Kill that service instance from docker container using input.

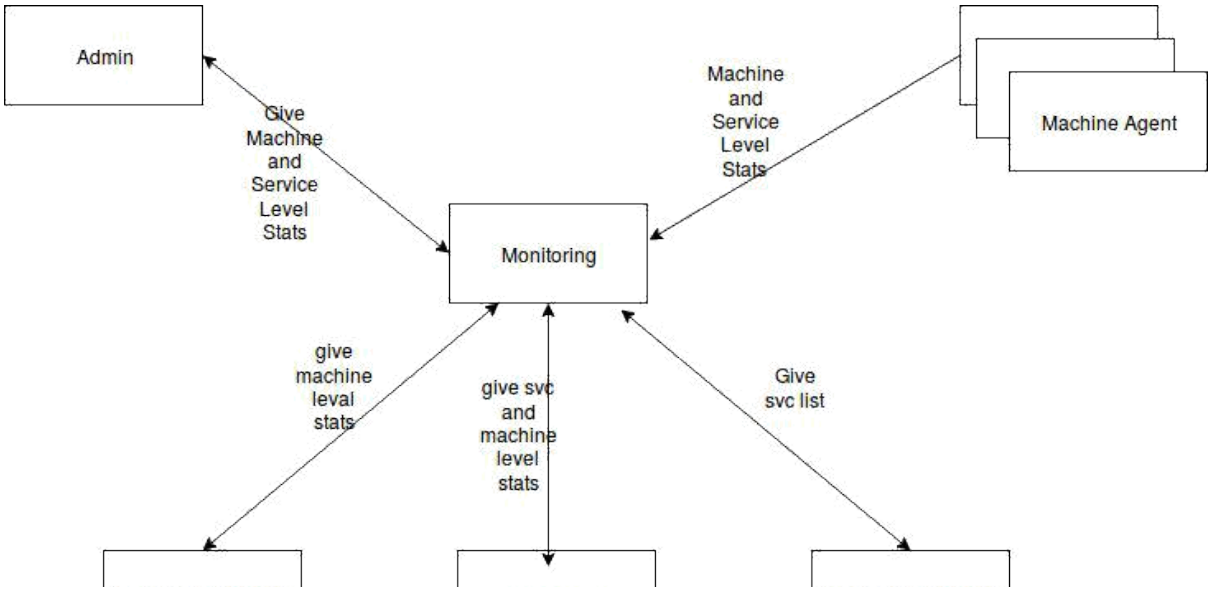
6.4 Logging :



Main functionalities of this module are as follows :

1) Retrieve Latest LogFile From “Log” Directory
2) Check Size of that file
   If size() > 50MB :
   Zip current .logFile
   Delete last modified .log file
   Create new .log File with current timestamp 3.Write Log in that file.
3) Format of .log File, all attributes are of string type :
   <Data severity uid reqid processname serviceid processid invoker_method log>

6.5 Monitoring Service:



Main functionalities of this module are as follows :

1) **getAllSvcLevelStats()** : Give Json Object for all service level stats.
2) **GetAllMachineLevelStats()** : Give Json Object for all machine level stats.
3) **svcList(String host)** : Give all services on specific host machine.
4) **getCompleteStats()** : Give all machine and service level stats.
5) **removeMachine(JSONObject obj)** : Before killing host server LC manager asks to remove stats of that machine.This function clears the stats for given machine.
6) **updateStats(JSONObject obj)** :Update states with data obtained from machine agent of host machine.

6.6 Admin Service:

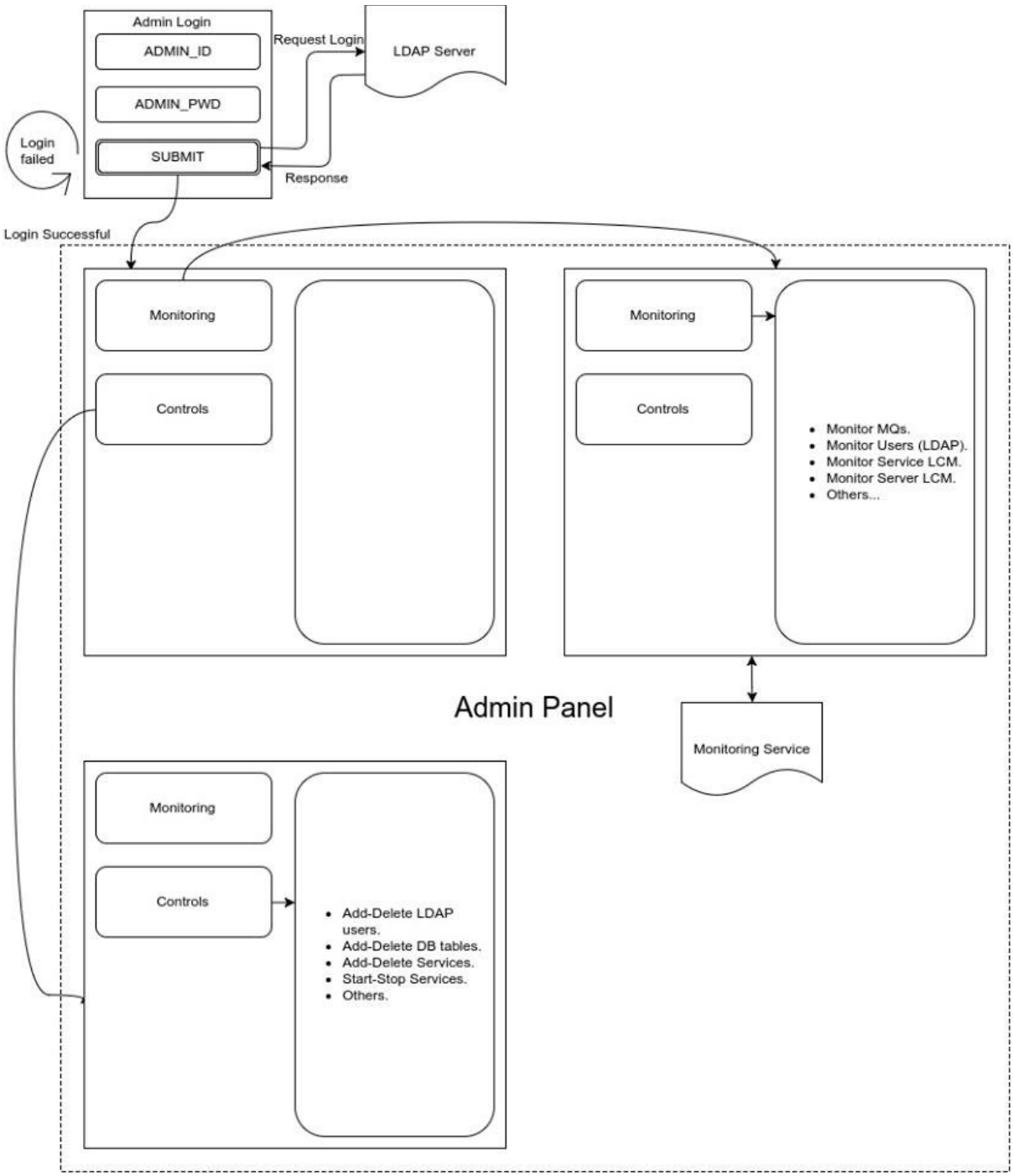
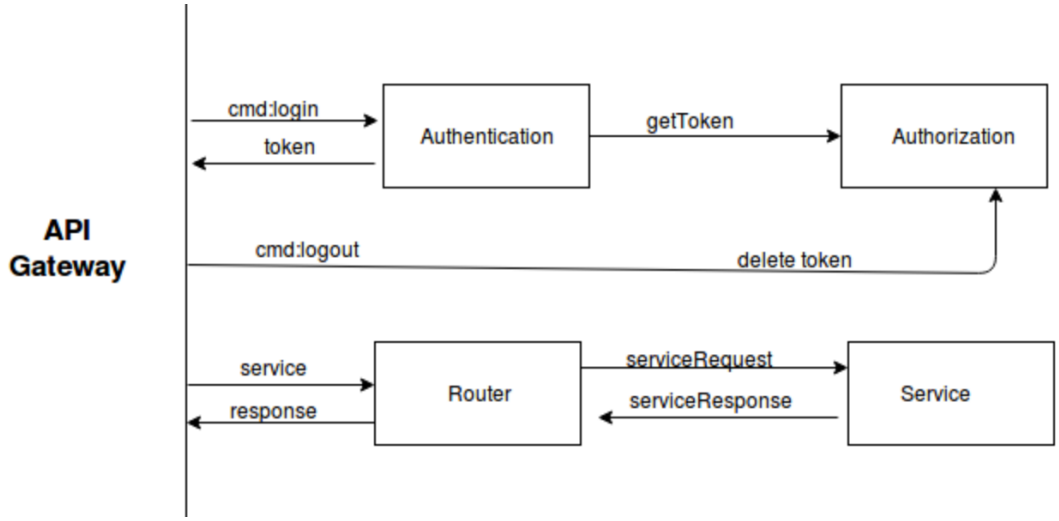


Figure : Admin Console

6.7 API Gateway:

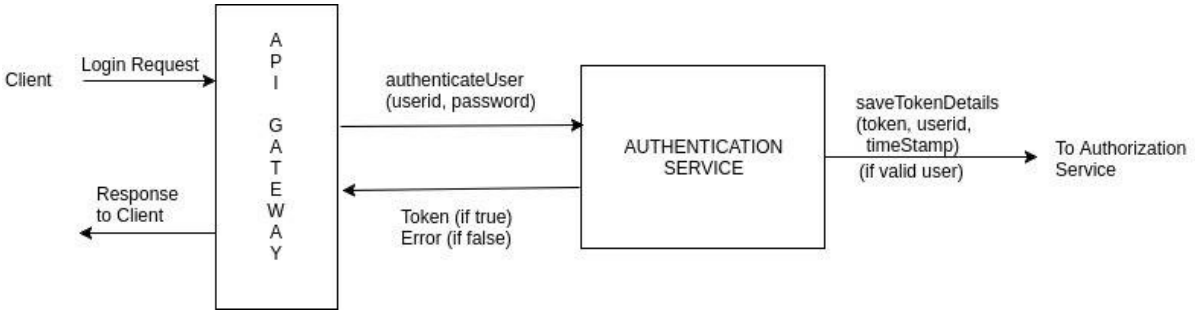


Main functionalities of this module are as follows :

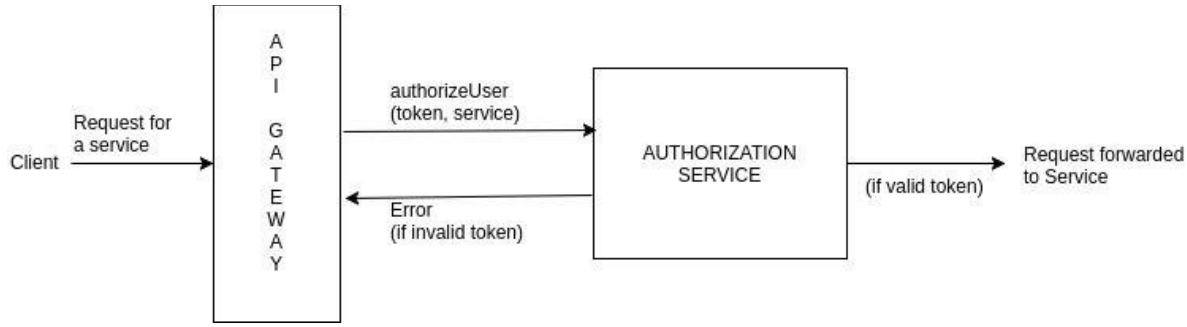
1) **Login :** Call to Authentication service check if user is authorized and returns the token
2) **Logout** : Call to Authentication Service and revoke the token back.
   - It checks if user is authorized or not.
   - If user is authorized then it redirects the request to router.
   - Returns the reply back to user.

6.8 Authentication and Authorization

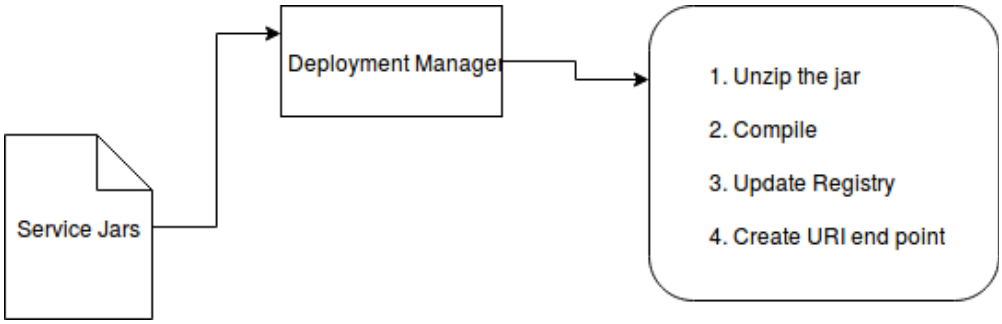
- Authentication :



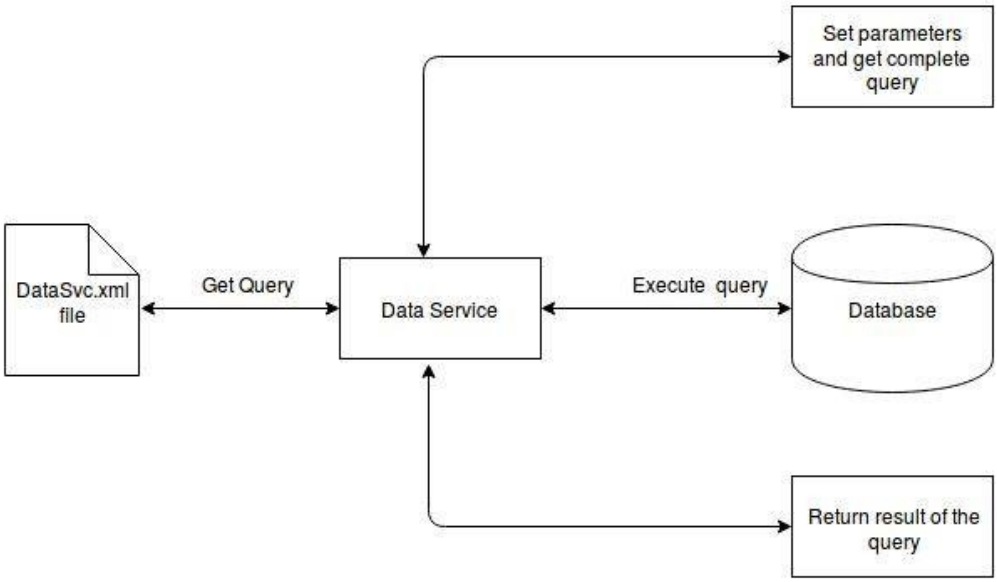
- Authorization :



6.9 Deployment Manager

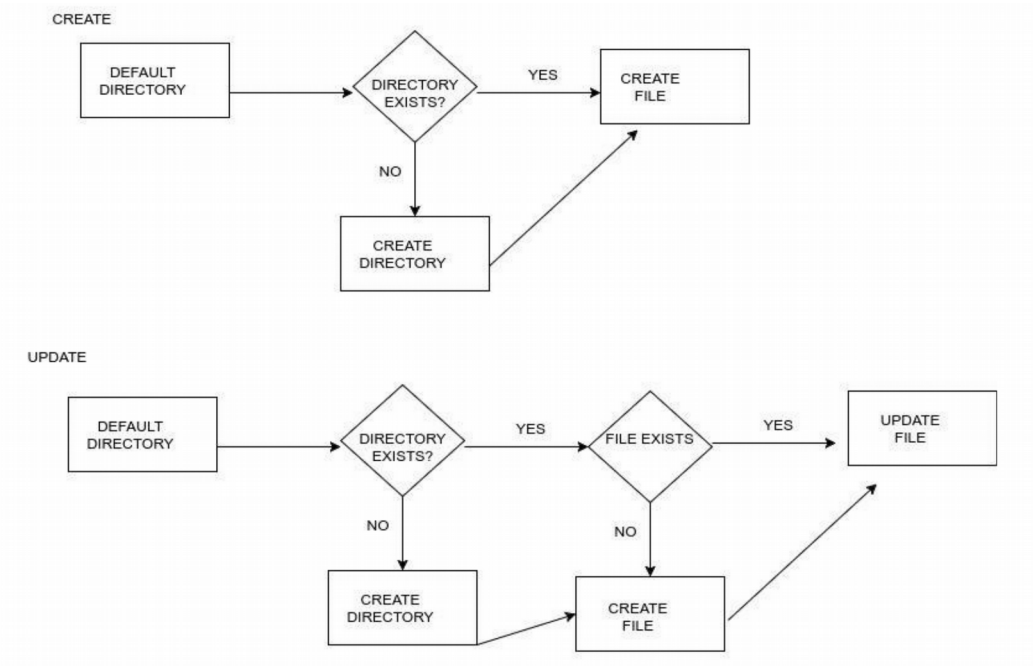


6.10 Data Service



Data Service Block Diagram

6.11 File Service



7. USE CASE:

7.1 Scenario of Weather Service Application :

→ **Purpose :**

The purpose of this application is to give user weather and pollution report,forecasting report or schedule task for user.

→ **Users:**

1) Suggestion Application
   Purpose : Any application that schedule users task based on weather and pollution of location.
2) Any person
   Purpose : User may be any common person who want to get weather information.

7.2 Build Services and Package :

Every service mentioned here should be a different java class. Every class should inherit **coreServiceClass** which contains the wrapper functions to use system services.

- **Climate Service**
  - getWeather(Boolean inCelsius) : Takes parameter : inCelsius to indicate output format
  - getWeather(): default to either °**C or** °**F**
  - getHumidity(): Returns humidity
  - getWindSpeed(): Return wind speed in km/hr.

- **Pollution Service**
  - getPollutionLevel(String cityName) : takes cityName and return pollution level
  - getPollutionLevel(String cityName, int numberofYearsAgo) : Return pollution level of a city in the same month but previous year if the 2nd parameter is 1
  - getTopPollutedCities(int topX) : Returns topX polluted cities currently
  - Boolean alertForCity(String cityName, int pollutionLevelThreshold, String toEmailId) : Should send an alert email to the specified email id, whenever the pollution Level for mentioned city goes beyond given threshold. Should return true if the alert is set successfully.

- **Scheduler Service**
    - getScheduledEntities() :Give schedule of all the scheduled entities.
    - getDailyReport(String cityName,String time, String toEmailId) : Should schedule a daily weather report for given city which should send an email at time mentioned in format ‘hh:mm’ Should return true if report is set successfully.

- **Forecast Service**
    - getForecast(String cityName, int nextNdays): Returns next n days predicted temperature.