**Friends Management**

In this is an application "Friends Management" is a common requirement. The application has features like "Friend Connection", “Subscribe”, “Unsubscribe", “Common Friend List”, “Subscriber List” "Block", "Receive Updates" etc.

**Technology Choice**

1. **Spring Boot**

Spring Boot is user friendly in nature there is no much to write code

1. **Swagger**

**Swagger is used for front end to test the application**

1. **JDBC Template**

**JDBC template are used**

1. **H2 Database (In Memory Database)**
2. **Pivotal**

**Spring Boot**

1. Spring Boot allows easy setup of standalone Spring-based applications.
2. Ideal for spinning up microservices and easy to deploy.
3. Makes data access less of a pain, i.e. JPA mappings through Spring Data.

**Swagger**

1. Swagger is a framework for describing API using a common language that everyone can understand.
2. The Swagger spec standardizes API practices, how to define parameters, paths, responses, models, etc.

**AWS**

1. The Free Tier; which provides enough credit to run an EC2 micro instance 24/7 all month.
2. It comes with S3 storage, EC2 compute hours, Elastic Load Balancer time, and much more.
3. This gives a chance to try out AWS in our software

**Deployment to AWS-EC2**

The Application is deployed on AWS - EC2 instance.It can be accessed via the below url and the path for all the api is /firendsapi [http://ec2-18-216-161-170.us-east-2.compute.amazonaws.com:8080](http://ec2-18-216-161-170.us-east-2.compute.amazonaws.com:8080/)

For example: To access /firends endpoint, the URL should be:

<http://ec2-18-216-161-170.us-east-2.compute.amazonaws.com:8080/firendsapi/friends>

Swagger UI is configured for the app and it is available: <http://ec2-18-216-161-170.us-east-2.compute.amazonaws.com:8080/swagger-ui.html>

Jenkins is configured on EC2 to build and deploy snapshots for the friendmanagement micro service.Trigger the below job so that it will automatically deploy and run the microservice on EC2.

<http://ec2-18-216-161-170.us-east-2.compute.amazonaws.com:9090/job/RELEASE_FRIEND_MGMT/>

Credentials : sudarsan/cg@123

**Deployment using Docker**

First step is to run MySQL in Docker container, use below command.

docker run --name mysql-friend-management-containar -e MYSQL\_ROOT\_PASSWORD=root -e MYSQL\_DATABASE=friend\_management -e MYSQL\_USER=zarvis -e MYSQL\_PASSWORD=password -d mysql:5.6

1. name mysql-friend-management-containar to assign a name to the container.
2. -e to pass environment variables to the container, and create a friend\_management database with a user username and password. This user will be granted superuser permissions for the friend\_management database.
3. -p 3306:3306 to expose MySQL port to the local host.
4. -d to tell Docker to daemonize the container and keep it running.mysql:5.6 to download MySQL 5.6 Server image from Docker public repo if not in your computer yet.

Now that we have defined the Dockerfile, build a docker image for our application. Type the following command from the root directory of the project to build the docker image

$ docker build -t friendmanagement .

Once we have a docker image, run and link with MySQL database container which we already create below command.

$ docker run -p 8080:8080 --name friendmanagement --link mysql-friend-management-containar -d friendmanagement

After that check the logs to make sure Server is running OK

$ docker logs friendmanagement

The application Docker image is also available in Docker Hub : <https://hub.docker.com/r/isudarsan/zarvis-apps/tags/>

Image can be pulled directly from Docker Hub using docker pull isudarsan/zarvis-apps:friendmanagement-0.0.1-SNAPSHOT

**Note : Since the application deployed on AWS-Free Tier, the URLs might not work always :)**

**Spring Boot Admin for monitoring**

Configured spring boot admin for application monitoring, available at : [http://http://ec2-18-188-249-225.us-east-2.compute.amazonaws.com:8093](http://http/ec2-18-188-249-225.us-east-2.compute.amazonaws.com:8093)

**List of REST Endpoints and Explanation**

1. Returns a list of friends of a person.
   * Path : /friends
   * Input :
2. {
3. "email":"abc@example.com"
4. }
   * Sample Output :
5. {
6. "success": true,
7. "friends":[
8. "pqr@gmail.com",
9. "lmn@gmail.com",
10. ],
11. "count":2
12. }
    * Defined Errors :
      + 40000 : Occurs when invalid email provided in the request.
      + 40006 : Occurs when the email address in the request is not valid (Not matched with the Regex)
      + 40004 : Occurs when the person given by the email does not exist
13. Returns list of common friends of two persons
    * Path : /commonfriends
    * Input :
14. {
15. "friends":[
16. "abc@gmail.com",
17. "pqr@gmail.com"
18. ]
19. }
    * Output :
20. {
21. "success": true,
22. "friends":[
23. "lmn@gmail.com",
24. "ijk@gmail.com"
25. ],
26. "count":2
27. }
    * Defined Errors :
      + 40000 : Invalid request.
      + 40006 : Request contains emails are invalid
      + 40004 : Persons given by the email do not exist
      + 40004 : The two email addresses in the input are the same
28. Establish Friendship between two persons
    * Path : /friendrequest
    * Input :
29. {
30. "friends":[
31. "abc@gmail.com",
32. "pqr@gmail.com"
33. ]
34. }
    * Output :
35. {
36. "success": true
37. }
    * Defined Errors :
      + 40000 : Invalid request
      + 40006 : Request contains emails are invalid
      + 40004 : Persons given by the email do not exist
      + 40004 : The two email addresses in the input are the same
      + 40001 : Persons in the input are already friends
      + 40002 : One of the person in the request have blocked the other person
38. Person subscribe to another Person
    * Path : /subscribe
    * Input :
39. {
40. "requestor":"abc@gmail.com",
41. "target":"pqr@gmail.com"
42. }
    * Output :
43. {
44. "success": true
45. }
    * Defined Errors :
      + 40000 : Invalid request
      + 40006 : Request contains emails are invalid
      + 40004 : Persons given by the email do not exist
      + 40004 : The two email addresses in the input are the same
      + 40001 : Persons in the input are already friends
      + 40004 : Duplicate subscription, a person already subscribed to another
46. Person block updates from another Person
    * Path : /block
    * Input :
47. {
48. "requestor":"example@example.com",
49. "target":"example2@example.com"
50. }
    * Output :
51. {
52. "success": true
53. }
    * Defined Errors :
      + 40000 : Invalid request
      + 40006 : Request contains emails are invalid
      + 40004 : Persons given by the email do not exist
      + 40004 : The two email addresses in the input are the same
      + 40001 : Persons in the input are already friends
      + 40002 : The person already blocked updates from the user
54. Post an update which returns a list of emails that will receive the update.
    * Path : /sendupdates
    * Input :
55. {
56. "sender":"abc@gmail.com",
57. "target":"Hello, how are you ! xyz@gmail.com"
58. }
    * Output :
59. {
60. "success": true,
61. "recipients":[
62. "xyz@gmail.com",
63. "pqr@gmail.com"
64. ]
65. }
    * Defined Errors :
      + 40000 : Invalid request
      + 40006 : Request contains emails are invalid
      + 40004 : Persons given by the email do not exist

**Other Errors**

In any case any of the endpoint fails it outputs an error response instead. The error response has the below format

```

{

"success": false,

"errorCode": "5000",

"message":"Unknown Error"

}

```

**Database**

The Database is pre populated with 10 persons for testing purpose, aslo the data can be found from the SQL script file which is placed inside the code repository.

Below is the simple ER Diagram used for the application.

@mentions (<https://docs.docker.com/>) (<https://spring.io/guides/gs/spring-boot-docker/>)