**JHipster installation steps:**

1. Install Java 8 from [the Oracle website](http://www.oracle.com/technetwork/java/javase/downloads/index.html).
2. Install Node.js from [the Node.js website](http://nodejs.org/) (prefer an LTS version). This will also install npm, which is the node package manager we are using in the next commands.
3. (Recommended) Update NPM: npm install -g npm
4. Install Yeoman: npm install -g yo
5. Only for AngularJS 1, install Bower: npm install -g bower
6. Only for AngularJS 1, install Gulp: npm install -g gulp-cli (If you have previously installed a version of gulp globally, please run npm rm -g gulp to make sure your old version doesn’t collide with gulp-cli)
7. Install JHipster: npm install -g generator-jhipster

**JHipster provides two architecture styles:**

* A “monolithic” architecture uses a single, one-size-fits-all application, which contains both the front-end AngularJS code, and the back-end Spring Boot code.
* “microservices” architecture splits the front-end and the back-end, so that it’s easier for your application to scale and survive infrastructure issues.

**The JHipster microservices architecture works in the following way:**

* A gateway is a JHipster-generated application (using application type microservice gateway when you generate it) that handles Web traffic, and serves an Angular application. There can be several different gateways, if you want to follow the Backends for Frontends pattern, but that’s not mandatory.
* The JHipster Registry is a runtime application, using the usual JHipster structure, on which all applications registers and get their configuration from. And its source code is available on GitHub under the JHipster organization at [jhipster/jhipster-registry](https://github.com/jhipster/jhipster-registry).
* Microservices are JHipster-generated applications (using application type microservice application when you generate them), that handle REST requests. They are stateless, and several instances of them can be launched in parallel to handle heavy loads.
* The JHipster Console is a monitoring & alerting console, based on the ELK stack.

**Creating micro services application:**

First of all, create an empty directory in which you will create your application:

mkdir myapplication

Go to that directory:

cd myapplication/

To generate your application, type:

yo jhipster

Answer the questions asked by the generator to create an application tailored to your needs.

### Which type of application would you like to create?

### What is the base name of your application?

### What is your default Java package name?

### Which type of authentication would you like to use?

### Which type of database would you like to use?

### Which production database would you like to use?

### Which development database would you like to use?

### Do you want to use Hibernate 2nd level cache?

### Would you like to use Maven or Gradle?

### Which other technologies would you like to use?

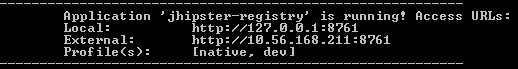
### Which Framework would you like to use for the client?

### Would you like to use the LibSass stylesheet preprocessor for your CSS?

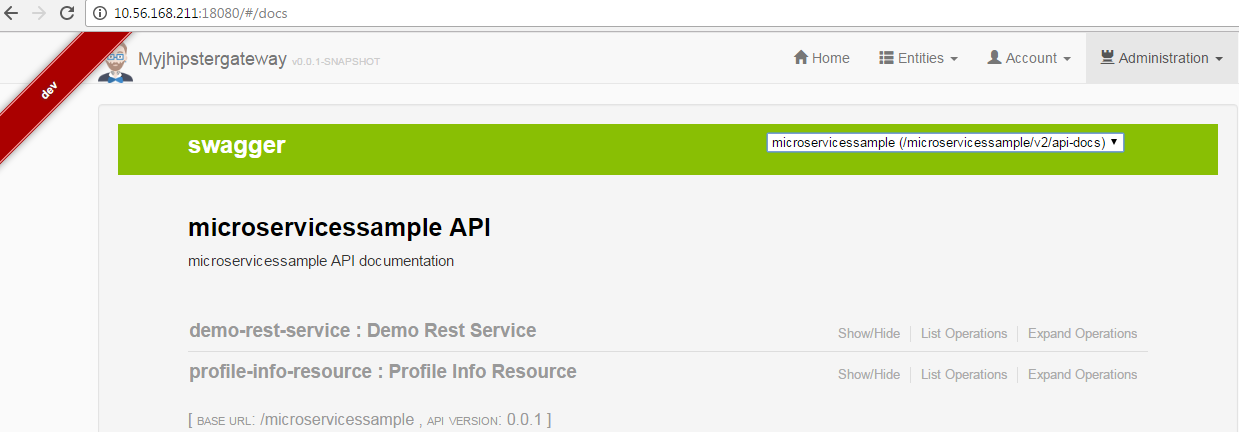
### Would you like to enable translation support with Angular Translate?

### Which testing frameworks would you like to use?

1. Download source code for JHispter Registry from <https://github.com/jhipster/jhipster-registry> and ran that. This will run on port 8761 by default.



1. Created micro services gateway project with all required inputs and chosen JHipster registry (Eureka spring boot) to register this gateway. This will run on the port(**18080**) we specified at the time of creation 

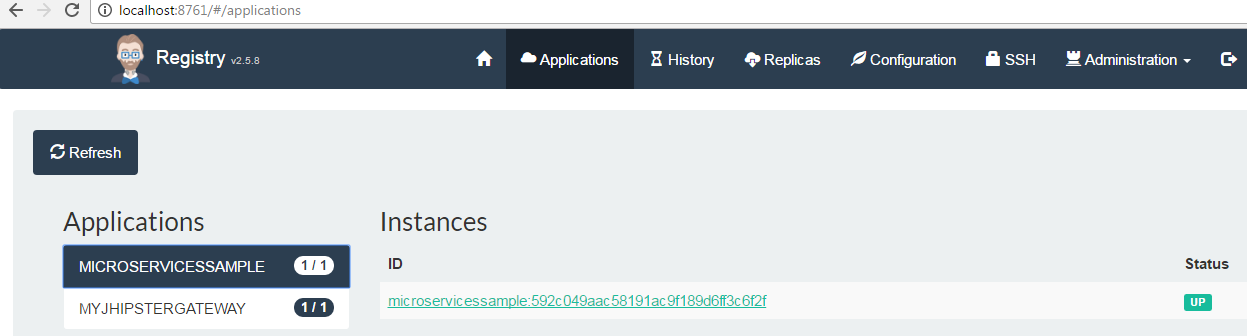


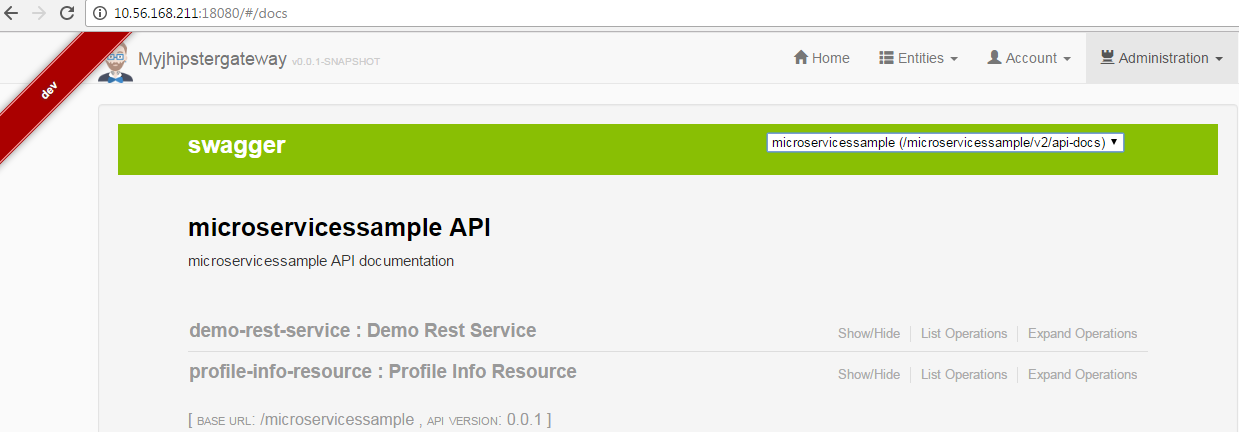
1. Created micro service project with all required inputs and chosen JHipster registry (Eureka spring boot) to register this micro service. This will run on the port we specified at the time of creation.

* After successful creation of micro services run mvnw from the project directory location. It will start the service and deployed on JHipster registry.



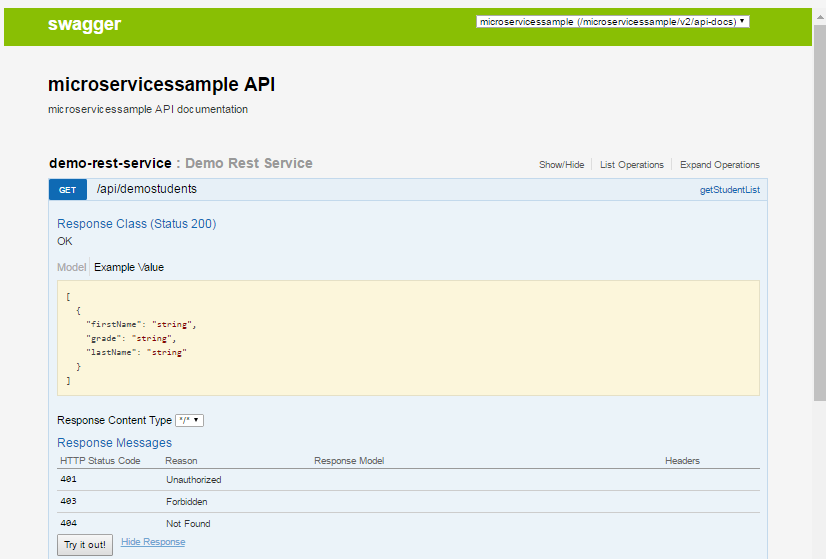
1. This micro service we could find in JHipster registry as well in gateway with default services (profile-info-resource) created by JHipster.

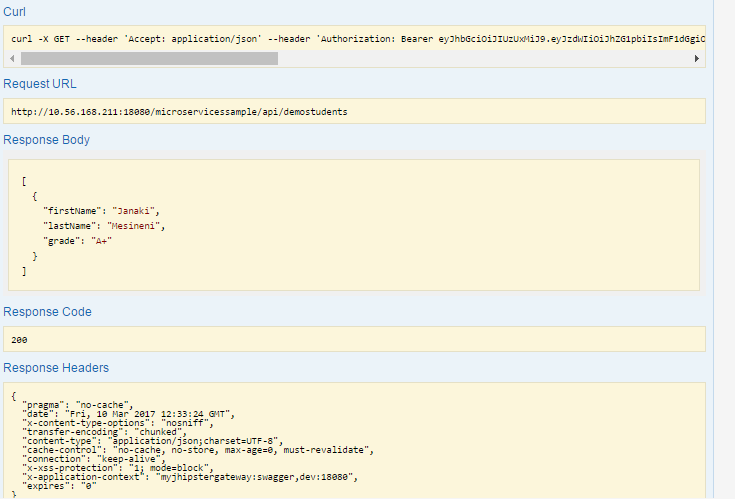




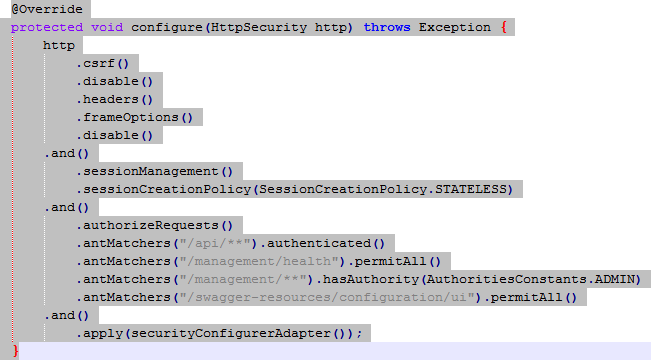
ii. Then I have added an entity and service by following steps given in <https://jhipster.github.io/creating-an-entity/> , <https://jhipster.github.io/creating-a-service/> then deployed again.

I could see this new service changes (demo-rest-service) added in the micro service which was deployed in gateway.





* We can change the accessing permissions by changing D:\Janaki\JhipsterWorkspace\jhipsterloginservice\src\main\java\com\adms\jhipster\config\MicroserviceSecurityConfiguration.java



But this is not advisable (which avoids JWT security mechanism).

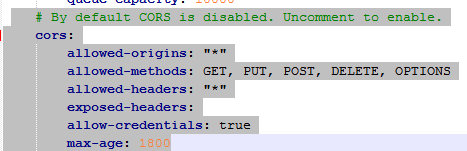
**The JHipster API gateway**

JHipster use to generate an API gateway. This gateway is a normal JHipster application, so you can use the usual JHipster options and development workflows on that project, but it also acts as the entrance to your microservices. More specifically, it provides HTTP routing and load balancing, quality of service, security and API documentation for all microservices.

When the gateways and the microservices are launched, they will register themselves in the registry (using the eureka.client.serviceUrl.defaultZonekey in the src/main/resources/config/application.yml file).

The gateways will automatically proxy all requests to the microservices, using their application name: for example, when microservices app1 is registered, it is available on the gateway on the /app1 URL.

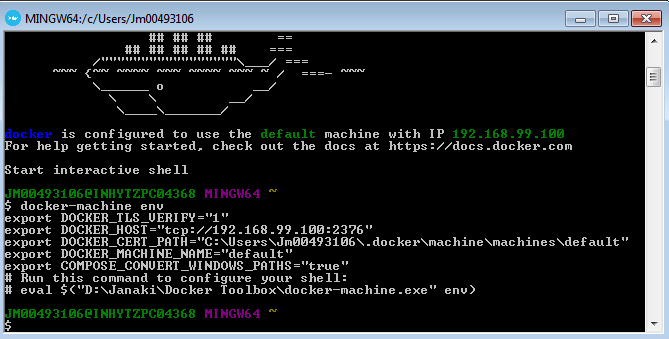
* Micro services gateway will read the services deployed on Registry and generates JWT token to access them.
* To avail the microservices through gateway we need to uncomment CORS properties in gateway D:\Janaki\JhipsterWorkspace\microservices-gateway\src\main\resources\config\application.yml



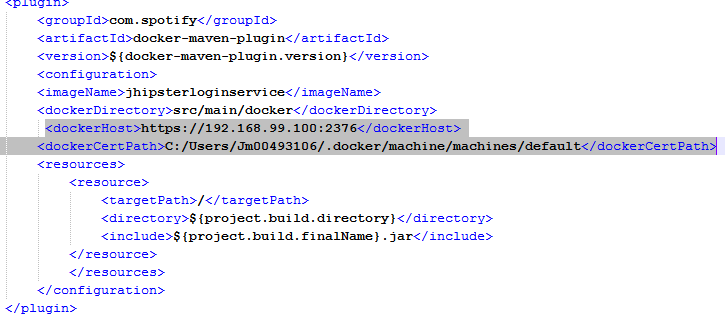
**Deploying micro services on Docker:**

**Docker installation:**

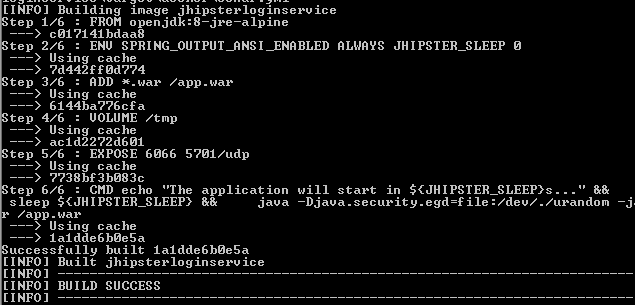
* Docker internally uses Linux VM to run, so we need to enable VM through BOIS settings before installing docker.
* Download [Docker Toolbox](https://docs.docker.com/toolbox/toolbox_install_windows/) for windows version 10 below.
* [InstallDocker](https://docs.docker.com/docker-for-windows/install/#download-docker-for-windows.) for windows 10
* Through docker quick start you could find docker machine configuration as below



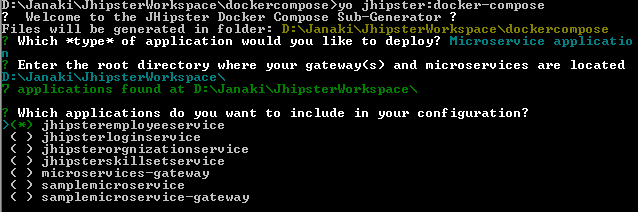
* Specify this docker IP in pom.xml of micro services and gateways, which we want to deploy on docker

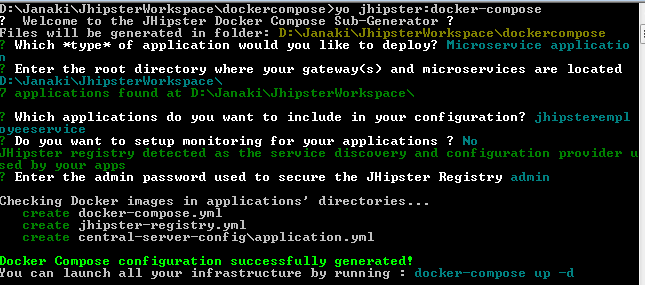


* Create docker image for all micro services and gateway by executing **mvnw package docker:build** in the project directory.
* You could see the below on successful build

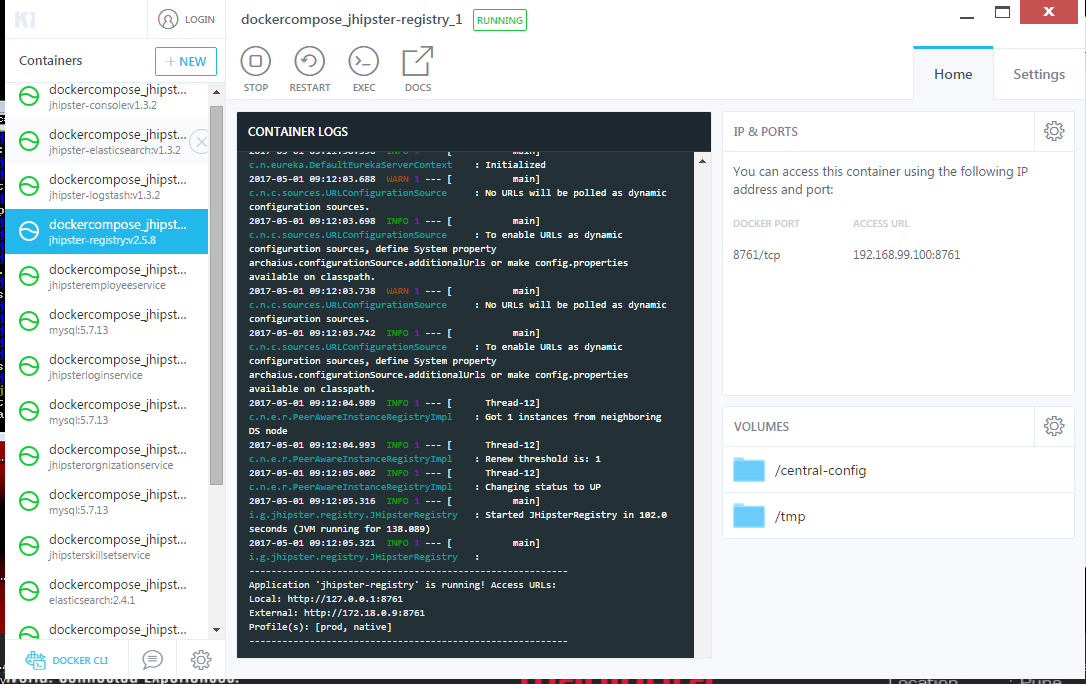


* Once the docker image created, create a directory (ex:docker-compose) and open command prompt, navigate to the directory and run **yo Jhipster:docker-compose** to place all microservices and gateways in one place run them in a single go.
* It will prompt the below questions



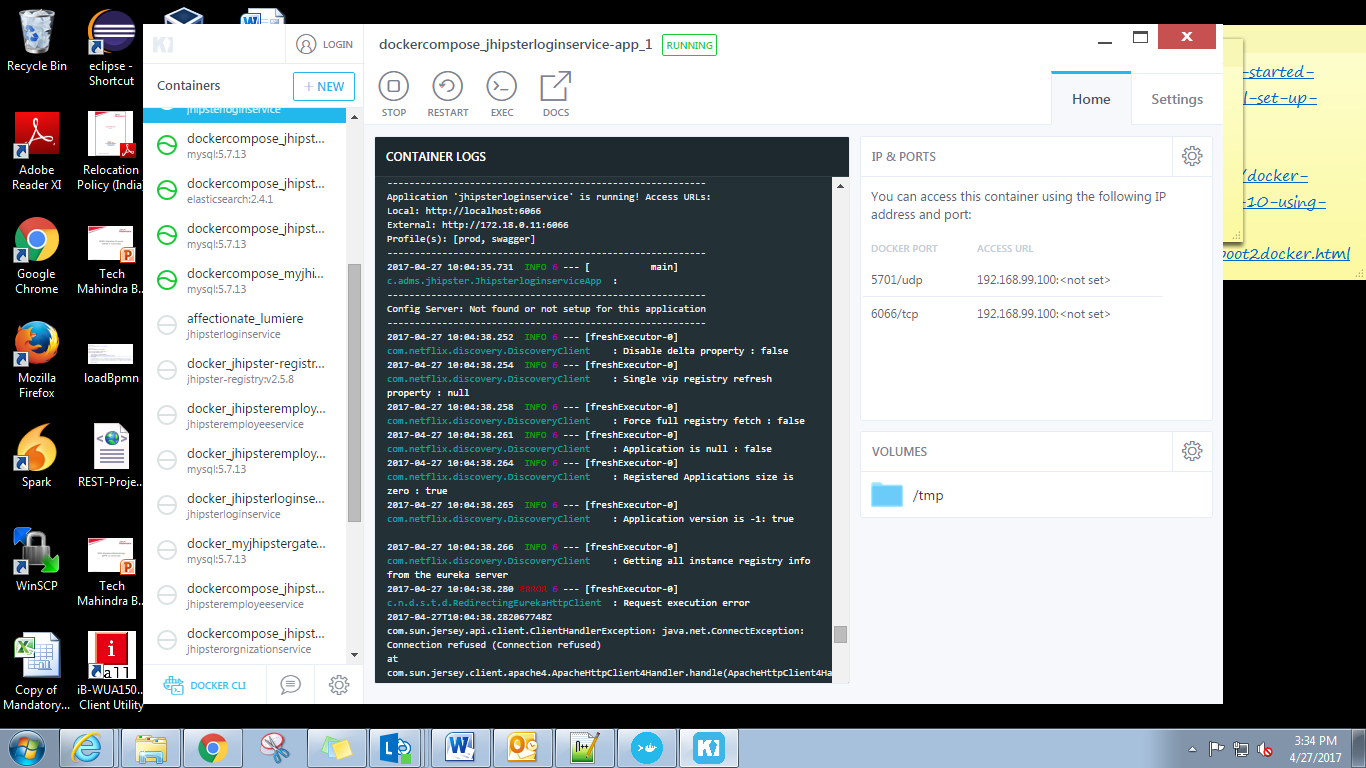


* On successful docker compose configuration run **docker-compose up** which will create the docker image in the container and starts them. You could see this from **Kitematic** s/w given by Docker Toolbox.



**Note**: before running docker-compose up, execute

**@FOR /f "tokens=\*" %i IN ('docker-machine env --shell cmd') DO @%i** to run shell commands on cmd.



**Elasticsearch:** Elasticsearch is an option that adds search capabilities on top of your database.

When the Elasticsearch option is selected:

* Spring Data Elasticsearch is being used, and is automatically configured by Spring Boot ([here is the documentation](http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-nosql.html#boot-features-elasticsearch)).
* The “repository” package has new subpackage, called “search”, that holds all ElastiSearch repositories.
* The “User” entity gets indexed in Elasticsearch, and you can query is using the /api/\_search/users/:query REST endpoint.
* When the [entity sub-generator](https://jhipster.github.io/creating-an-entity/) is used, the generated entity gets automatically indexed by Elasticsearch, and is used in the REST endpoint. Search capabilities are also added to the AngularJS user interface, so you can search your entity in the main CRUD screen.

You can enable HTTP access by adding this configuration in application-dev.yml :

elasticsearch:

properties:

http:

enabled: true

Through kibana we can monitor the elastic search

