Microservices With JHipster

Jayesh Satish Nikam
Department of Computer Science
State University of New York at Binghamton
jnikam1@binghamton.edu

Abstract- JHipster is a development platform to quickly generate, develop, and deploy modern web applications and microservice architectures.[1] It combines three very successful frameworks in web development: Bootstrap, Angular, and Spring Boot. [4] Microservices are style of architecture that develops one application as a group of small services. Each service runs in its own process. The services communicate with clients, and sometimes one another, using lightweight protocols, often over messaging or HTTP. In this paper we will cover how Microservice work with JHipster.

Keywords: JHipster, Bootstrap, Angular, Spring Boot, Microservices

I. Introduction

Adopting a microservice architecture provides unique opportunities to add failover and resiliency to your systems, so your components can handle load spikes and errors gracefully.[3]

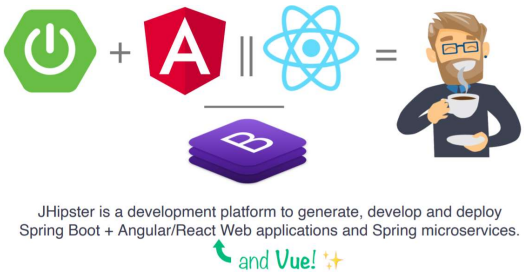

Figure 1. JHipster

Technology has traditionally been organized into technology layers: UI team, database team, operations team. When teams are separated along these lines, even simple changes can lead to a cross team project consuming huge chunks of time and budget. A smart team will optimize around this and choose the lesser of two evils: forcing the logic into whichever application they have access to.[5]

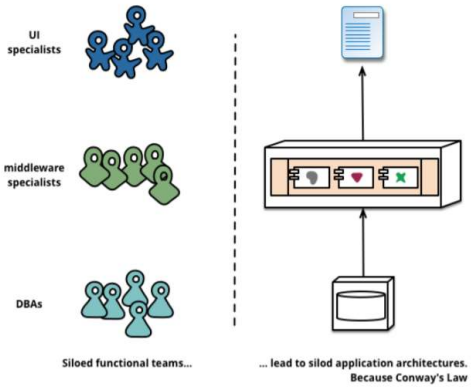

Figure 2. Conway's law

Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure. [5] — Melvin E. Conway

In this paper, I'll show you how to build a microservices architecture with JHipster. As part of this process, we need to generate three applications and running several others. [1]

1. Generate a gateway.
2. Generate a blog microservice.
3. Generate a store microservice.
4. Run the JHipster Registry, Keycloak, and MongoDB.

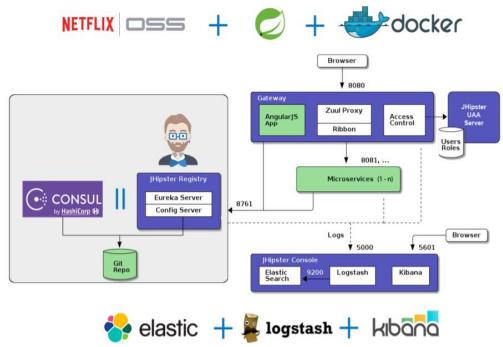

Figure 3. JHipster Microservices architecture

II. Methodology

The first choice we have to select on JHipster is the kind of application we want to generate. There are two choices:

2.0.1. Monolithic: architecture uses a single, one-size-fits-all application, which contains both the front-end code, and the back-end Spring Boot code.[1]

2.0.2 Microservices: architecture splits the front-end and the back-end, so that it's easier for your application to scale and survive infrastructure issues. [1]

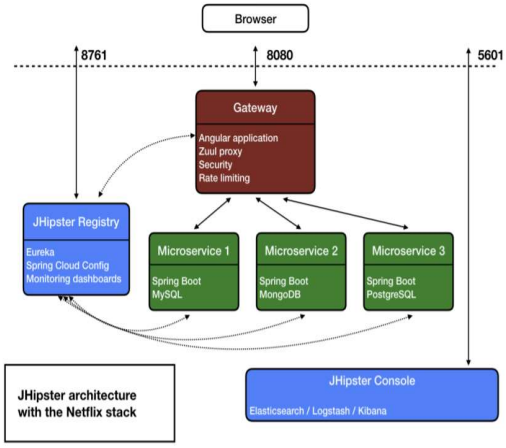

Figure 4. Microservices

We are going to see the Mircoservices part.

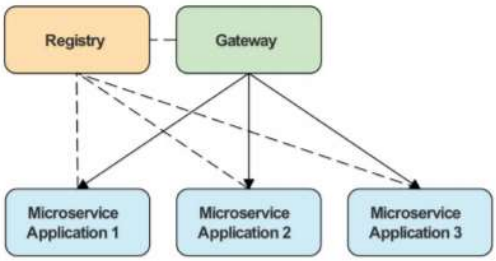

Figure 5. General Overview of Mircoservices

2.1 JHipster Registry

A JHipster Registry is an essential component in the microservices architecture because it connects all the other components with each other and provides the communication between these components. [4]

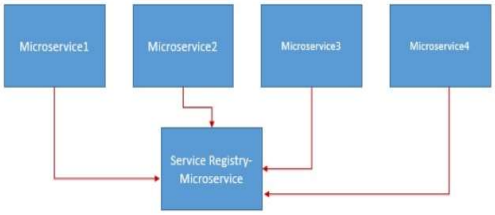

Figure 6. JHipster Registry

2.1.1 Consul

Consul provides service discovery, fault tolerance, vault for key-value storage. This is an alternative to JHipster Registry that can provide both service discovery and configuration store.[4]

2.2 Microservices Application

The microservice application which will provide the backend capabilities through exposing the API. [4]

2.3 Gateway

The microservices gateway is the frontend of the whole system which will include all the APIs of every microservice application in the system. JHipster API Gateway will work with JHipster Registry

for service discovery and load balance HTTP requests using Spring Cloud Load Balancer. [4]

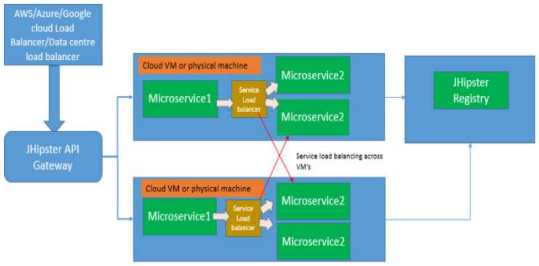

Figure 7. JHipster Gateway

2.4 To Create Microservice

To generate a store microservice, open a terminal window and navigate to the jhipster-microservices-example directory. Create a store directory and run JHipster in it. [1]

```
cd ~/jhipster-microservices-example
mkdir store
cd store
jhipster
```

Figure 8. Commands to run JHipster

Figure 9. Launching JHipster

2.5 JHipster Control Center:

The control center monitors and manages applications. The control center provides the following features:

2.5.1. Instances: Instances provide a list of application instances; when the service gets registered in the JHipster Registry, the service is available in the list of instances. It also provides a dashboard to view the list.

2.5.2 Metrics: The metrics are managed by a Micrometer that tracks system metrics like JVM, HTTP requests, cache usage, and database connections.

2.5.3. Health checks: The health checks are integrated with Spring Actuator to get the status of health of services.

2.5.4. Logs: Logs are integrated with Logback to trace the logs of the running application.

2.5.5. API documentation: API documentation provides Swagger APIn documentation.

2.6.5. Security with JWT: JWT is a stateless security mechanism that can be scaled to multiple servers. This is the default security option for JHipster Microservices. JWT provides Header, Payload, and Signature. All the client session data is stored in the token. No server-side session management is being done in JWT. JWT is supported by OAuth 2.0 for secure transmission. [4]

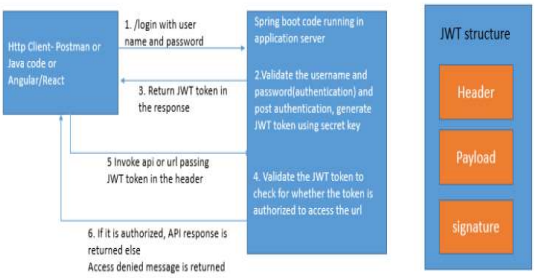

Figure 10. Security with JWT

2.7 Microservice Security with JHipster UAA:

JHipster UAA is user accounting and authoring microservices using OAuth2.

OAuth2 is an authorization framework that can be used for centralized identity management. JHipster UAA provides an OAuth2 endpoint that provides access token based on the authentication. The token provides fine-grained access control to access the Rest based services. UAA is the combination of the authorization server and the resource server. [4]

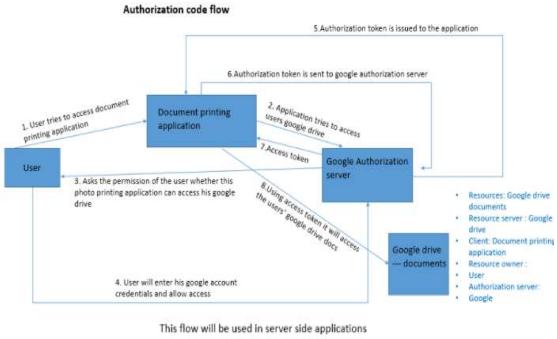

Figure 11. OAuth flow

III. Conclusion

In this project, we have covered how microservices work with JHipster. How to setup Microservices with JHipster. Also, the advantages and methods. The JHipster Registry we have used here is just a basic setup: in production, it must be secured, mainly because it can contain your applications' properties.

However, JHipster makes microservices easy doesn't mean we should use them. Using a microservices architecture is a great way to scale development teams, but if you don't have a large team, Monolith might work better.

IV. Reference

[1] https://www.jhipster.tech/
[2]https://pure.unamur.be/ws/portalfiles/portal/27242657/NUTTINCK_HALIN_Memoire.pdf
[3]https://github.com/axel-halin/Thesis-JHipster
[4]https://www.infoq.com/minibooks/jhipster-mini-book-5/
[5]https://en.wikipedia.org/wiki/Conway%27s_law