Introducción al desarrollo de Microservicios (en java) con docker

Gustavo A. Arellano

@arellano_gus

2021 ©

Agenda

- Presentación del expositor
- Disclaimer
- Introducción
  - Una arquitectura “limpia”
  - Arquitectura REST
    - Stateless, mostly “Serverless”, bajamente acoplada y altamente cohesiva
    - Definición de un URL
    - Verbos HTTP
    - Códigos de error HTTP
    - Payload de entrada y de salida (JSON)
  - Estructura propuesta
- Taller práctico
- Preguntas y respuestas

Presentador

- Gustavo Arellano es Matemático egresado de la Facultad de Ciencias de la UNAM y posee el grado de Maestro en Ciencias, por el Posgrado en Ciencia e Ingeniería de la Computación, UNAM, con especialidad en Ingeniería de Software.
- Gustavo ha trabajado para GE, Apple y Citibank por varios años en EEUU como Ingeniero de Software y en México ha sido Director General Adjunto de Desarrollo para la CNBV y Director Adjunto de desarrollo tecnológico para CONACYT, entre otros cargos directivos en la Administración Pública Federal.
- Posee más de 12 certificaciones en diversos ámbitos como Java, Scrum, Ethical Hacking, Secure Programming & Forensic Investigator. Ha publicado material relacionado con Inteligencia Artificial en la revista internacional arbitrada BMC Bioinformatics. (Antelope)
- Actualmente es Arquitecto en Jefe de la primer consultora CMMI DEV y SVC 2.0 nivel 5 de Latinoamérica: Ultrasist.

Disclaimer

- Este es un taller gratuito sin patrocinadores ni venta de productos o servicios de ningún tipo. Es solo por el placer de compartir.
- A cambio, se les solicita a todos los participantes que difundan este conocimiento igualmente de manera gratuita a quién consideren que le pueda ser de utilidad.
- No soy un “experto” ni vengo a demostrar que sé mas (ni menos) que otra persona. Sólo comparto algo que creo que es útil y muy probablemente habrá mil maneras de hacer lo mismo de mejor manera. Pero esta es la que sé y que me ha funcionado. Toda crítica CONSTRUCTIVA será bienvenida.
- El tema de microservicios es INMENSO y en 4 horas NO se cubrirán todos los aspectos de este paradigma, pero SI se proporcionarán las bases para que cualquier participante pueda, personalmente, extender este conocimiento.
- Se va a grabar y la grabación se hará pública, pero no está permitido comercializarla ni parcial ni totalmente, bajo ninguna circunstancia. Pero puede ser compartida de manera GRATUITA a cualquier organización o individuo que a su vez acepta estar obligado a cumplir con estos mismos términos.

Intro

- Evolución desde la antigüedad hasta ahora:
  - Monolitos (sin capas)
  - Monolitos (con capas)
  - SOA
  - Microservicios
  - Funciones (AWS lambda / Azure functions / google cloud functions, por ejemplo)

```
<%@ page import = "java.io.*,java.util.*,java.sql.*"%>
<%@ page import = "javax.servlet.http.*,javax.servlet.*" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix = "c"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/sql" prefix = "sql"%>

<html>
   <head>
      <title>SELECT Operation</title>
   </head>

   <body>
      <sql:setDataSource var = "snapshot" driver = "com.mysql.jdbc.Driver"
         url = "jdbc:mysql://localhost/TEST"
         user = "root"  password = "pass123"/>

      <sql:query dataSource = "${snapshot}" var = "result">
         SELECT * from Employees;
      </sql:query>

      <table border = "1" width = "100%">
         <tr>
            <th>Emp ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Age</th>
         </tr>

         <c:forEach var = "row" items = "${result.rows}">
            <tr>
               <td><c:out value = "${row.id}"/></td>
               <td><c:out value = "${row.first}"/></td>
               <td><c:out value = "${row.last}"/></td>
               <td><c:out value = "${row.age}"/></td>
            </tr>
         </c:forEach>
      </table>

   </body>
</html>
```

Un monolito sin capas…

Emp ID	First Name	Last Name	Age
100	Zara	Ali	18
101	Mahnaz	Fatma	25
102	Zaid	Khan	30
103	Sumit	Mittal	28

Hagan esto SOLO bajo su propio riesgo, pero debo advertir que es peor que tomar veneno…

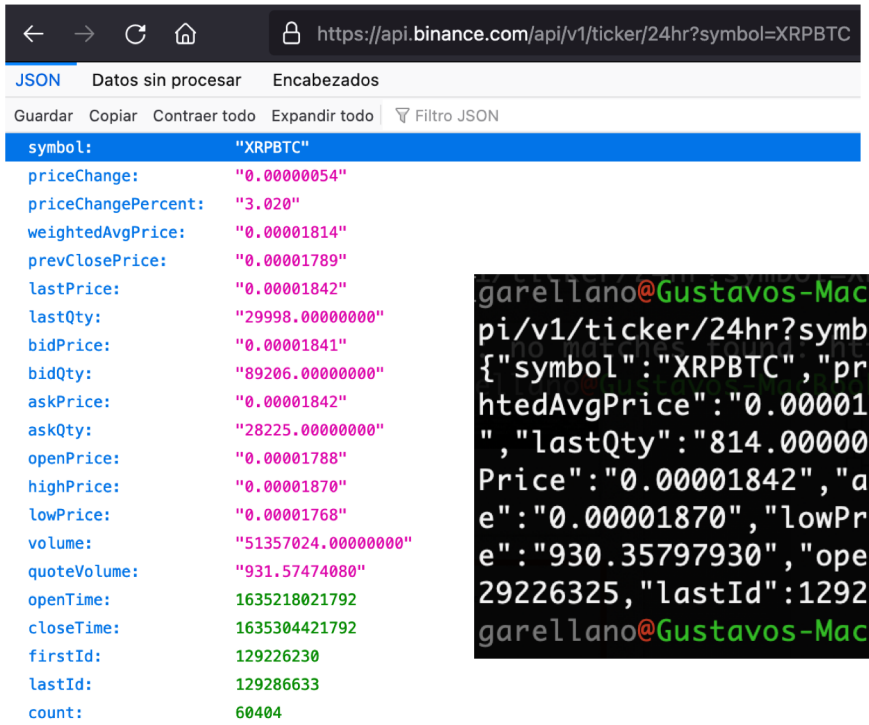
```html
<html>
  <head>
    <title>SELECT Operation</title>
  </head>
    <body>

      <table border = "1" width = "100%">
         <tr>
            <th>Emp ID</th>
            <th>First Name</th>
            <th>Last Name</th>
            <th>Age</th>
         </tr>
        <tr>
            <td>100</td>
            <td>Zara</td>
            <td>All</td>
            <td>18</td>
        </tr>
        <tr>
            <td>101</td>
            <td>Mahnaz</td>
            <td>Fatma</td>
            <td>25</td>
        </tr>
        <tr>
            <td>102</td>
            <td>Zaid</td>
            <td>Kahan</td>
            <td>30</td>
        </tr>
        <tr>
            <td>103</td>
            <td>Sumit</td>
            <td>Mittal</td>
            <td>28</td>
        </tr>
      </table>

    </body>
</html>
```

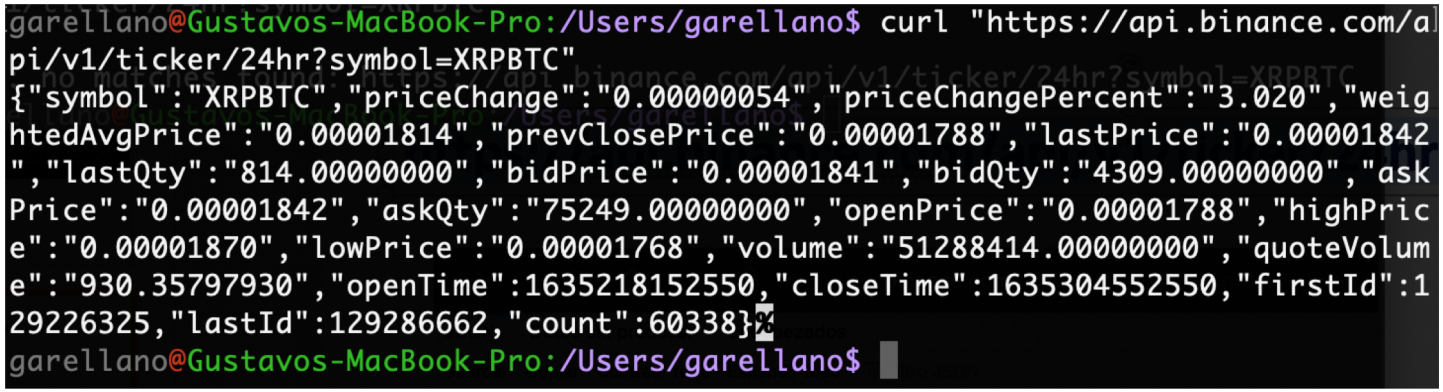
Versus

```
{
  [
    {"id":100, "name": "Zara", "last": "All", "age":18},
    {"id":101, "name": "Mahnaz", "last": "Fatma", "age":25},
    {"id":102, "name": "Zaid", "last": "Kahan", "age":30},
    {"id":103, "name": "Sumit", "last": "Mittal", "age":28}
  ]
}
```

(En Software el fin NO justifica los medios….)

https://api.binance.com/api/v1/ticker/24hr?symbol=XRPBTC





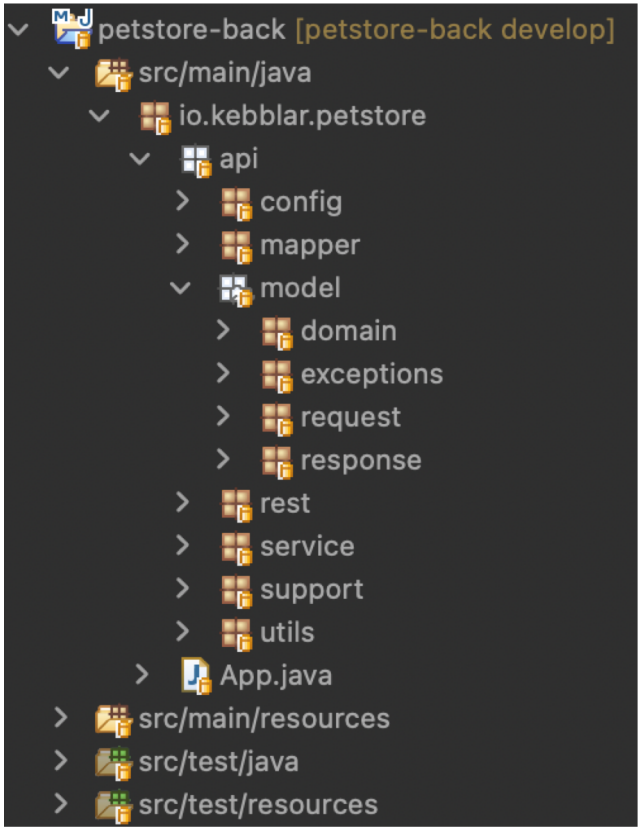
Servicio REST

- Comunicación SIN estado
- Construcción de un URL
- Verbos HTTP (GET, PUT, POST, DELETE,…)
- Códigos de ejecución HTTP (2XX, 3XX, 4XX, 5XX)
- Payload de solicitud
- Payload de respuesta
- Hacer una sola cosa, pero hacerla bien

Cliente reactivo de un servicio REST (Ejemplo)

- index.html
- ok.js

Estructura de un micro servicio



Base de datos

```
docker stop db
docker rm db

docker run -d \
--name db \
--restart unless-stopped \
-p 3306:3306 \
-v /home/ubuntu/work/db:/var/lib/mysql \
-e MYSQL_ROOT_PASSWORD=secreto \
-e TZ=Mexico/Mexico_City \
mariadb

echo ”ok"
```

El directorio **/home/ubuntu/work/db** (o su equivalente)
YA debe de estar creado ANTES de ejecutar el script de la izquierda

La adopción de este paradigma me ha permitido desarrollar Software con muchas mas Calidad que antes porque:

- Ahora mis sistemas “aguantan” mucha mas carga sin caerse como antes.
- Es mucho mas fácil darle mantenimiento a piezas independientes de mis sistemas, que a un todo altamente interconectado.
- Probar cada parte es mas fácil que probar todo junto.
- Puedo tener equipos de personas cuya interacción ahora sólo es “la necesaria” y cada equipo puede desarrollar su parte en el lenguaje mas apropiado sin problemas de interacción con otros equipos y/o lenguajes.
- Puedo “actualizar” una parte del sistema sin tener que “bajar” el sistema completo.
- Puedo “ver en vivo” el resultado de un cambio en mi plataforma de pruebas de manera prácticamente instantánea.

GRACIAS

- Sesión de preguntas y respuestas